AMNS

1.5.0

Generated by Doxygen 1.8.20

Chapter 1

AMNS Library

The AMNS user routines provide services from the AMNS system to user codes

Author

ITM-AMNS, WPCD-AMNS

The goals of the project are depicted in table_amns_goals.

Table 1.1 Goals of the AMNS system

Physics code		AMNS implementation
• Access to AMNS data only via interface – initialization (2) – finalization (2) – querying parameters (2) – setting parameters (2) – getting data (1) • Separation between use of the data and the implementation of the data • Code author doesn't need to become an expert in AMNS • Ensures compatibility between codes		• Only accessed by a set of defined calls • Implementation by AMNS experts • Different versions can be supported • Different implementations possible – Analytic formulae – Table lookup • "Old" versions should always be recoverable (even if wrong) • Should become easier to implement "new" data

The section A quick introduction to AMNS library provides an introduction to using the library.

The section How the library works provides a description of how the library works.

The section Provision of data for the library provides a description of how the data needed by the library is read in and then stored in amns_data IDS's stored under the device name "amns".

Some information about an earlier version of the AMNS system can be found in **?**.

1.1 A quick introduction to AMNS library

The AMNS library provides a standardized method for accessing Atomic, Molecular, Nuclear or Surface data. It does this by providing a library of routines that can be accessed from Fortran, C, Python or Java.

Debugging output for the AMNS library can be enabled by setting the environment variable "IMAS_AMNS_DEBUG" to "yes" ("no" will disable it).

setenv IMAS_AMNS_DEBUG yes

or

export IMAS_AMNS_DEBUG=yes

for csh/tcsh or sh/ksh/bash shells, respectively.

1.1.1 Fortran function calls

The 9 calls to the AMNS system are (using fortran as the example language) :

- amns_module::imas_amns_setup, initialization call for the AMNS package
```
subroutine imas_amns_setup(handle, version, error_status)
    optional version, error_status
    type(amns_handle_type), intent(out) :: handle
    type(amns_version_type), intent(in) :: version
    type(amns_error_type), intent(out) :: error_status
```

- amns_module::imas_amns_query, query routine for the AMNS package
```
subroutine imas_amns_query(handle,query,answer,error_status)
    optional error_status
    type(amns_handle_type), intent(in) :: handle
    type(amns_query_type), intent(in) :: query
    type(amns_answer_type), intent(out) :: answer
    type(amns_error_type), intent(out) :: error_status
```

- amns_module::imas_amns_set, set a parameter for the AMNS package
```
subroutine imas_amns_set(handle,set,error_status)
    optional error_status
    type(amns_handle_type), intent(in) :: handle
    type(amns_set_type), intent(in) :: set
    type(amns_error_type), intent(out) :: error_status
```

- amns_module::imas_amns_finish, finalization call for the AMNS package
```
subroutine imas_amns_finish(handle, error_status)
    optional error_status
    type(amns_handle_type), intent(inout) :: handle
    type(amns_error_type), intent(out) :: error_status
```

- amns_module::imas_amns_setup_table, initialization call for a particular reaction
```
subroutine imas_amns_setup_table(handle, reaction_type, reactant, handle_rx, error_status)
    optional error_status
    type(amns_handle_type), intent(in) :: handle
    type(amns_reaction_type), intent(in) :: reaction_type
    type(amns_reactants_type), intent(in) :: reactant
    type(amns_handle_rx_type), intent(out) :: handle_rx
    type(amns_error_type), intent(out) :: error_status
```

- amns_module::imas_amns_query_table, query routine for a particular reaction

```
subroutine imas_amns_query_table(handle_rx,query,answer,error_status)
    optional error_status
    type(amns_handle_rx_type), intent(in) :: handle_rx
    type(amns_query_type), intent(in) :: query
    type(amns_answer_type), intent(out) :: answer
    type(amns_error_type), intent(out) :: error_status
```

- amns_module::imas_amns_set_table, set a parameter for a particular reaction

```
subroutine imas_amns_set_table(handle_rx,set,error_status)
    optional error_status
    type(amns_handle_rx_type), intent(in) :: handle_rx
    type(amns_set_type), intent(in) :: set
    type(amns_error_type), intent(out) :: error_status
```

- amns_module::imas_amns_finish_table, finalization call for a particular reaction

```
subroutine imas_amns_finish_table(handle_rx, error_status)
    optional error_status
    type(amns_handle_rx_type), intent(inout) :: handle_rx
    type(amns_error_type), intent(out) :: error_status
```

- amns_module::imas_amns_rx, get the rates associated with the input args for a particular reaction

```
interface imas_amns_rx
    module procedure imas_amns_rx_1, imas_amns_rx_2, imas_amns_rx_3
  end interface
  subroutine imas_amns_rx_1(handle_rx,out,arg1,arg2,arg3,error_status)
    optional arg2,arg3,error_status
    type(amns_handle_rx_type), intent(inout) :: handle_rx
    real (kind=r8), intent(out) :: out(:)
    real (kind=r8), intent(in) :: arg1(:),arg2(:),arg3(:)
    type(amns_error_type), intent(out) :: error_status
  subroutine imas_amns_rx_2(handle_rx,out,arg1,arg2,arg3,error_status)
    optional arg2,arg3,error_status
    type(amns_handle_rx_type), intent(inout) :: handle_rx
    real (kind=r8), intent(out) :: out(:,:)
    real (kind=r8), intent(in) :: arg1(:,:),arg2(:,:),arg3(:,:)
    type(amns_error_type), intent(out) :: error_status
  subroutine imas_amns_rx_3(handle_rx,out,arg1,arg2,arg3,error_status)
    optional arg2,arg3,error_status
    type(amns_handle_rx_type), intent(inout) :: handle_rx
    real (kind=r8), intent(out) :: out(:,:,:)
    real (kind=r8), intent(in) :: arg1(:,:,:),arg2(:,:,:),arg3(:,:,:)
    type(amns_error_type), intent(out) :: error_status
```

1.1.2 User Interface Data Structures

A number of data structures are used by the library interface. Some are opaque (i.e. the contents are not of relevance to the user), and some need to be set or read by the user programme.

The two opaque types are handles which are returned by the setup routines and then need to be passed to the other routines:

- amns_handle_type (amns_types::amns_handle_type), used for the database wide routines
- amns_handle_rx_type (amns_types::amns_handle_rx_type), used for the reaction specific routines

In some language bindings these are the basis of classes.

The non-opaque types are:

- amns_error_type (amns_types::amns_error_type), used to indicate if an error occurred and, if so, what the error was
- amns_reaction_type (amns_types::amns_reaction_type), used to indicate the requested reaction
- amns_set_type (amns_types::amns_set_type), used to set an AMNS internal parameter
- amns_query_type (amns_types::amns_query_type), used to query an AMNS internal parameter

- amns_answer_type (amns_types::amns_answer_type), used to contain the answer from an AMNS query
- amns_version_type (amns_types::amns_version_type), used to specify the AMNS version (version number, access layer backend to be used and the username to be used for accessing the data)
- amns_reactants_type (amns_types::amns_reactants_type), used to specify the reactants to a reaction
- amns_reactant_type (amns_types::amns_reactant_type), a sub-component of amns_reactants_type used to characterize the individual reactants

1.1.3 AMNS User Interface Data Queries

The currently available queries for querystring in the call to amns_module::imas_amns_query is

- **version:** Return the version information
- **prop_comment** : information about properties_comment
- **prop_source** : information about properties_source
- **prop_provider** : information about properties_provider
- **prop_creation** : information about properties_creation_date
- **code_name** : information about code_name
- **code_commit** : information about code_commit
- **code_version** : information about code_version
- **code_repository** : information about code_repository

The currently available queries for querystring in the call to amns_module::imas_amns_query_table are

- **source:** source (origin) of the data
- **provider** : information about the data provider
- **citation** : information about the data citation
- **no_of_reactants:** number of reactants involved
- **index:** Not sure what this is
- **filled:** whether the data table has been filled ("Filled" or "Empty")
- **reaction_type:** reaction type
- **reactants:** nuclear charges of reactants
- **version:** information about the version
- **state_label:** label for the charge state (if appropriate)
- **result_unit:** units of the result
- **result_label:** description of the result
- **ndim:** dimensionality of the requested data
- **interp_fun:** number of the interpolating function
- **prop_comment** : information about properties_comment

- **prop_source** : information about properties_source
- **prop_provider** : information about properties_provider
- **prop_creation** : information about properties_creation_date
- **code_name** : information about code_name
- **code_commit** : information about code_commit
- **code_version** : information about code_version
- **code_repository** : information about code_repository

1.1.4 AMNS User Interface Data Setting Options

The currently setting options for setstring in the call to amns_module::imas_amns_set is

- **debug:** enable debugging
- **nodebug:** disable debugging
- **backend=mdsplus:** use the MDSplus backend
- **backend=hdf5:** use the HDF5 backend
- **backend=ascii:** use the ASCII backend

The currently available setting options for setstring in the call to amns_module::imas_amns_set_table is

- **warn:** request warning messages
- **nowarn:** disable warnings
- **debug:** enable debugging
- **nodebug:** disable debugging

1.1.5 C function calls

The 9 calls to the AMNS system are:

- IMAS_AMNS_SETUP, initialization call for the AMNS package
```
void IMAS_AMNS_C_SETUP(void **handle_out, amns_error_type *error_status);
```
- IMAS_AMNS_QUERY, query routine for the AMNS package
```
void IMAS_AMNS_C_QUERY(void *handle_in, amns_query_type *query,
                       amns_answer_type *answer, amns_error_type *error_status)
```
- IMAS_AMNS_SET, set a parameter for the AMNS package
```
void IMAS_AMNS_C_SET(void *handle_in, amns_set_type *set, amns_error_type *error_status);
```
- IMAS_AMNS_FINISH, finalization call for the AMNS package
```
void IMAS_AMNS_C_FINISH(void **handle_inout, amns_error_type *error_status);
```
- IMAS_AMNS_SETUP_TABLE, initialization call for a particular reaction
```
void IMAS_AMNS_C_SETUP_TABLE(void *handle_in, amns_reaction_type *reaction_type,
                             void *reactant_handle_in, void **handle_rx_out,
                             amns_error_type *error_status);
```

- IMAS_AMNS_QUERY_TABLE, query routine for a particular reaction
```
void IMAS_AMNS_C_QUERY_TABLE(void *handle_rx_in, amns_query_type *query,
                             amns_answer_type *answer, amns_error_type *error_status);
```

- IMAS_AMNS_SET_TABLE, set a parameter for a particular reaction
```
void IMAS_AMNS_C_SET_TABLE(void *handle_rx_in, amns_set_type *set,
                           amns_error_type *error_status);
```

- IMAS_AMNS_FINISH_TABLE, finalization call for a particular reaction
```
void IMAS_AMNS_C_FINISH_TABLE(void **handle_rx_inout, amns_error_type *error_status);
```

- IMAS_AMNS_RX, get the rates associated with the input args for a particular reaction
```
void IMAS_AMNS_C_RX_0_A(void *handle_rx_in, double *out,
                        double arg1, amns_error_type *error_status);
void IMAS_AMNS_C_RX_0_B(void *handle_rx_in, double *out,
                        double arg1, double arg2, amns_error_type *error_status);
void IMAS_AMNS_C_RX_0_C(void *handle_rx_in, double *out,
                        double arg1, double arg2, double arg3,
                        amns_error_type *error_status);
void IMAS_AMNS_C_RX_1_A(void *handle_rx_in, int nx, double *out,
                        double *arg1, amns_error_type *error_status);
void IMAS_AMNS_C_RX_1_B(void *handle_rx_in, int nx, double *out,
                        double *arg1, double *arg2, amns_error_type *error_status);
void IMAS_AMNS_C_RX_1_C(void *handle_rx_in, int nx, double *out,
                        double *arg1, double *arg2, double *arg3,
                        amns_error_type *error_status);
void IMAS_AMNS_C_RX_2_A(void *handle_rx_in, int nx, int ny,
                        double *out, double *arg1, amns_error_type *error_status);
void IMAS_AMNS_C_RX_2_B(void *handle_rx_in, int nx, int ny,
                        double *out, double *arg1, double *arg2,
                        amns_error_type *error_status);
void IMAS_AMNS_C_RX_2_C(void *handle_rx_in, int nx, int ny,
                        double *out, double *arg1, double *arg2, double *arg3,
                        amns_error_type *error_status);
void IMAS_AMNS_C_RX_3_A(void *handle_rx_in, int nx, int ny, int nz,
                        double *out, double *arg1, amns_error_type *error_status);
void IMAS_AMNS_C_RX_3_B(void *handle_rx_in, int nx, int ny, int nz,
                        double *out, double *arg1, double *arg2,
                        amns_error_type *error_status);
void IMAS_AMNS_C_RX_3_C(void *handle_rx_in, int nx, int ny, int nz,
                        double *out, double *arg1, double *arg2, double *arg3,
                        amns_error_type *error_status);
```

In addition, service routines are provided for dealing with reactants:
```
void IMAS_AMNS_C_SETUP_REACTANTS(void **reactants_handle_out, char string_in[reaction_length],
                                 int index_in, int n_reactants);
void IMAS_AMNS_C_SET_REACTANT(void *reactants_handle_in, int reactant_index,
                              amns_reactant_type *reactant_in);
void IMAS_AMNS_C_GET_REACTANT(void *reactants_handle_in, int reactant_index,
                              amns_reactant_type *reactant_out);
void IMAS_AMNS_C_FINISH_REACTANTS(void **reactants_handle_inout);
```

1.1.6 Python function calls

The Python interface creates

- amns.Amns class which implements the following methods
  - amns.AMNS::query() to make queries of the AMNS system
  - amns.AMNS::set() to change the settings of the AMNS system
  - amns.AMNS::get_table() to specify which AMNS data is needed (creating a Table)
  - amns.AMNS::finalize() to finish with the AMNS system, freeing up the memory
- amns.Table class which implements the following methods
  - amns.Table::query() to make queries about a specific set of data
  - amns.Table::set() to change the settings for a specific set of data
  - amns.Table::data() to actually get the AMNS data for a specific process
  - amns.Table::finalize() to finish with a specific AMNS Table, freeing up the memory

- amns.Reactants class which implements the following methods
  - amns.Reactants::add() to add a reactant
  - amns.Reactants::test() ???
  - amns.Reactants::value() to return the reactants
- amns.AmnsException (class)

1.1.7 Examples of using the AMNS library from different languages

Five minimal examples of using the AMNS library are given below:

1.1.7.1 Fortran example

```
program minimal
  use ids_types  ! IGNORE
  use amns_types  ! IGNORE
  use amns_module  ! IGNORE
  implicit none
  type (amns_handle_type) :: amns                              ! AMNS global handle
  type (amns_handle_rx_type)  :: amns_rx                       ! AMNS table handle
  type (amns_reaction_type) :: xx_rx
  type (amns_reactants_type) :: species
  type (amns_answer_type) :: answer
  real (kind=ids_real) :: te=100.0_ids_real, ne=1e20_ids_real, rate
! set up the AMNS system  -- here we force to version 1
  call imas_amns_setup(amns, version=amns_version_type(", 1, ", "))
! query to find when the index file was created
  call imas_amns_query(amns, amns_query_type("prop_creation"), answer)
  write(*,*) 'Database created ', trim(answer%string)
! set up reactants/products
  allocate(species%components(4))
  species%components = &
       (/ amns_reactant_type(6, 1, 12, 0), amns_reactant_type(1, 0, 2, 0), &
          amns_reactant_type(6, 0, 12, 1), amns_reactant_type(1, 1, 2, 1) /)
! set up reaction
  xx_rx%string='CX'
! set up table
  call imas_amns_setup_table(amns, xx_rx, species, amns_rx)
! query to find information about the table
  call imas_amns_query_table(amns_rx, amns_query_type("prop_creation"), answer)
  write(*,*) 'Database created ', trim(answer%string)
  call imas_amns_query_table(amns_rx, amns_query_type("citation"), answer)
  write(*,*) 'Citation ', trim(answer%string)
! get results
  call imas_amns_rx(amns_rx, rate, te, ne)
  write(*,*) 'Rate = ', rate
! finish with table
  call imas_amns_finish_table(amns_rx)
! finish with amns
  call imas_amns_finish(amns)
end program minimal
```

1.1.7.2 C example

```
#include "amns_interface.h"
int main(int argc, char *argv[])
{
  void* amns_handle = NULL;
  amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
  void* reactants_handle = NULL;
  amns_c_reactant_type species1  = {.ZN=6, .ZA=1, .MI=12, .LR=0};
  amns_c_reactant_type species2  = {.ZN=1, .ZA=0, .MI=2 , .LR=0};
  amns_c_reactant_type species3  = {.ZN=6, .ZA=0, .MI=12, .LR=1};
  amns_c_reactant_type species4  = {.ZN=1, .ZA=1, .MI=2 , .LR=1};
  amns_c_reaction_type xx_rx  = {.string = "CX"};
  void* amns_cx_handle;
  double rate;
  IMAS_AMNS_CC_SETUP(&amns_handle, &error_stat);
  printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
  IMAS_AMNS_CC_SETUP_REACTANTS(&reactants_handle, "", 0, 4);
  IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 1, &species1);
  IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 2, &species2);
  IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 3, &species3);
```

```
  IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 4, &species4);
  IMAS_AMNS_CC_SETUP_TABLE(amns_handle, &xx_rx, reactants_handle, &amns_cx_handle, &error_stat);
  printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
  IMAS_AMNS_CC_RX_0_B(amns_cx_handle, &rate, 100.0, 1e20, &error_stat);
  printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
  printf("Rate = %25.15e\n", rate);
  IMAS_AMNS_CC_FINISH_TABLE(&amns_cx_handle, &error_stat);
  printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
  IMAS_AMNS_CC_FINISH_REACTANTS(&reactants_handle);
  IMAS_AMNS_CC_FINISH(&amns_handle, &error_stat);
  printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
  return 0;
}
```

1.1.7.3 Python example

```
#! /usr/bin/env python
import amns
import numpy as np
amnsdb = amns.Amns()
r = amns.Reactants()
r.add(6,1,12)
r.add(1,0,2)
r.add(6,0,12,lr=1)
r.add(1,1,2,lr=1)
table = amnsdb.get_table(b"CX", r) # fix error: expected bytes, found str
print("table.no_of_reactants", table.no_of_reactants)
dat = table.data(np.array([100.0]), np.array([1e20]))
print("Rate = \t%.15e" % (dat[0]))
amnsdb.finalize()
```

1.1.7.4 Java example

```
import amns.*;
import amns.type.*;
public class AmnsMinimal {
    public static void main(String[] args) {
        // this is a pointer from C
        long ptrAmnsHandle = 0;
        AmnsErrorType error_stat = new AmnsErrorType();
        // this is a pointer from C
        long ptrReactantsHandle = 0;
        // Definition of species
        AmnsReactantType species1 = new AmnsReactantType();
        species1.ZN = 6;
        species1.ZA = 1;
        species1.MI = 12;
        species1.LR = 0;
        AmnsReactantType species2 = new AmnsReactantType();
        species2.ZN = 1;
        species2.ZA = 0;
        species2.MI = 2;
        species2.LR = 0;
        AmnsReactantType species3 = new AmnsReactantType();
        species3.ZN = 6;
        species3.ZA = 0;
        species3.MI = 12;
        species3.LR = 1;
        AmnsReactantType species4 = new AmnsReactantType();
        species4.ZN = 1;
        species4.ZA = 1;
        species4.MI = 2;
        species4.LR = 1;
        long ptrCXHandle = 0;
        AmnsReactionType reactionType = new AmnsReactionType();
        reactionType.string = "CX";
        reactionType.isotopeResolved = 0;
        double rate = 0;
        // Class to manage Amns calls to low level
        Amns amns = new Amns();
        // Setup
        String imas_amns_debug = System.getenv("IMAS_AMNS_JAVA_DEBUG");
        Boolean amns_debug = (imas_amns_debug != null && !imas_amns_debug.trim().isEmpty() &&
       !imas_amns_debug.equals("no") && !imas_amns_debug.equals("NO") );
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP");
        ptrAmnsHandle = amns.ImasAmnsCCSetup(error_stat);
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " + ptrAmnsHandle);
        if (amns_debug) System.err.println("[JVM] Error.flag= " + error_stat.flag + " Error.string= " +
      error_stat.string);
        int idx = 0;
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP_REACTANTS");
        ptrReactantsHandle = amns.ImasAmnsCcSetupReactantsNumber(idx, 4);
```

```
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      ptrReactantsHandle);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species1");
        amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 1, species1);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species2");
        amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 2, species2);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species3");
        amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 3, species3);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species4");
        amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 4, species4);
        // we are calling now SETUP_TABLE using all the elements above
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP_TABLE");
        ptrCXHandle = amns.ImasAmnsCCSetupTable(ptrAmnsHandle, reactionType
                                              , ptrReactantsHandle
                                              , error_stat);
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_RX_0_B");
        rate = amns.ImasAmnsCCRX0B( ptrCXHandle, 100, 1.0e20, error_stat);
        System.out.println("Value of rate: " + rate);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_TABLE");
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
        ptrCXHandle =amns.ImasAmnsCCFinishTable( ptrCXHandle, error_stat);
        if (amns_debug) System.err.println("[JVM] After IMAS_AMNS_CC_FINISH_TABLE");
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_REACTANTS");
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      ptrReactantsHandle);
        ptrReactantsHandle = amns.ImasAmnsCCFinishReactants(ptrReactantsHandle);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_REACTANTS");
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      ptrReactantsHandle);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH");
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " + ptrAmnsHandle);
        ptrAmnsHandle = amns.ImasAmnsCCFinish(ptrAmnsHandle, error_stat);
        if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH");
        if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " + ptrAmnsHandle);
    }
}
```

1.1.7.5 Matlab example

```
% Make AMNS available in Matlab
% Import Java classes (interface to AMNS)
import amns.type.AmnsAnswerType
import amns.type.AmnsErrorType
import amns.type.AmnsQueryType
import amns.type.AmnsReactantType
import amns.type.AmnsReactionType
import amns.type.AmnsSetType
import amns.Amns
% minimal example in Matlab
% class to manage Amns calls to low level
amnsjava=amns.Amns;
% variables for querying and setting
amns_error   = amns.type.AmnsErrorType;
amns_answer  = amns.type.AmnsAnswerType;
amns_query   = amns.type.AmnsQueryType;
amns_set     = amns.type.AmnsSetType;
% setup AMNS system
ptrAmnsHandle = amnsjava.ImasAmnsCCSetup(amns_error);
if (amns_error.flag ~= 0)
  fprintf ('Error = %s\n' , amns_error.string)
end
amns_query.string = "version";
amnsjava.ImasAmnsCCQuery(ptrAmnsHandle, amns_query, amns_answer, amns_error)
if (amns_error.flag ~= 0)
  fprintf ('Error = %s\n' , amns_error.string)
end
fprintf ('Version %i\n', amns_answer.number)
% setup the reactants and products
species = amns.type.AmnsReactantType;
idx = 0;
ptrReactantsHandle = amnsjava.ImasAmnsCcSetupReactantsNumber(idx, 4);
species.ZN=double(6);  species.ZA=double(1);  species.MI=double(12);  species.LR=0;  % C1+
amnsjava.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 1, species);
species.ZN=double(1);  species.ZA=double(0);  species.MI=double(2);   species.LR=0;  % D0
amnsjava.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 2, species);
species.ZN=double(6);  species.ZA=double(0);  species.MI=double(12);  species.LR=1;  % C0
amnsjava.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 3, species);
species.ZN=double(1);  species.ZA=double(1);  species.MI=double(2);   species4.LR=1; % D1+
amnsjava.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 4, species);
% identify the reaction
reactionType=amns.type.AmnsReactionType;
reactionType.string='CX';
reactionType.isotopeResolved=0;
```

```
% Setup the table
ptrCXHandle = amnsjava.ImasAmnsCCSetupTable(ptrAmnsHandle, reactionType, ptrReactantsHandle, amns_error);
if (amns_error.flag ~= 0)
  fprintf ('Error = %s\n' , amns_error.string)
end
amns_query.string = "reactants";
amnsjava.ImasAmnsCCQueryTable(ptrCXHandle, amns_query, amns_answer, amns_error)
if (amns_error.flag ~= 0)
  fprintf ('Error = %s\n' , amns_error.string)
end
fprintf ('Reactants/Products = %s\n', amns_answer.string)
% calculate the rate at (100, 1e20)
rate = amnsjava.ImasAmnsCCRX0B(ptrCXHandle,100,1.0e20,amns_error);
if (amns_error.flag ~= 0)
  fprintf ('Error = %s\n' , amns_error.string)
end
fprintf ('Rate = %.15e\n' , rate)
% close down
ptrCXHandle =amnsjava.ImasAmnsCCFinishTable( ptrCXHandle, amns_error);
if (amns_error.flag ~= 0)
  fprintf ('Error = %s\n' , amns_error.string)
end
ptrReactantsHandle = amnsjava.ImasAmnsCCFinishReactants(ptrReactantsHandle);
if (amns_error.flag ~= 0)
  fprintf ('Error = %s\n' , amns_error.string)
end
ptrAmnsHandle = amnsjava.ImasAmnsCCFinish(ptrAmnsHandle, amns_error);
if (amns_error.flag ~= 0)
  fprintf ('Error = %s\n' , amns_error.string)
end
```

1.1.8 Some other examples

A number of examples are given below to show the use of the AMNS system

1.1.8.1 Coronal example in Fortran

Calculate the coronal charge state balance as a function of temperature. The README:

```
# An example of using the AMNS library to calculate coronal distributions
- the source code is in src/
- the compiled code, *.o files and some utility sctipts are in obj/
To compile:
  make
To run and analyze series of cases:
  obj/run_cases
```

The source code:

```
program coronal
  use ids_types
  use amns_types
  use amns_module
  implicit none
  integer, parameter :: nt = 120
  type (amns_handle_type) :: amns
  type (amns_handle_rx_type), allocatable :: amns_ei(:), amns_rc(:), amns_lr(:), amns_br(:)
  type (amns_reaction_type) :: xx_rx
  type (amns_reactants_type), allocatable :: species_m1(:), species_p1(:), species_cx(:), species(:)
  type (amns_query_type) :: query
  type (amns_answer_type) :: answer
  type (amns_set_type) :: set
  real (kind=ids_real), allocatable :: te(:), ne(:), na(:), rhs(:)
  real (kind=ids_real), allocatable :: rate_ei(:,:), rate_rc(:,:), rate_lr(:,:), rate_br(:,:)
  real (kind=ids_real), allocatable :: l(:), d(:), u(:)
  real (kind=ids_real) :: zn, mi
  real (kind=ids_real) :: test, norm, line_radiation, recombination_radiation
  character (len=12), allocatable :: state_labels(:)
  integer ns, is, it, isref
  integer :: iargc
  character (len=256) :: arg
! the command line can specify zn and am for the species to be modelled
  if(iargc().le.0) then
     zn=6
     mi=12
     write(*,*)
       '-----------------------------------------------------------------------------------------------'
     write(*,*) 'Assumed carbon --- if not OK specify the nuclear change and atomic mass as arguemnts to the
       program'
```

```
      write(*,*)
        '-------------------------------------------------------------------------------------------------------'
    else
      call getarg(1,arg)
      read(arg,*) zn
      if(iargc().ge.2) then
        call getarg(2,arg)
        read(arg,*) mi
      else
        mi = 2*zn
      write(*,*)
        '-------------------------------------------------------------------------------------------------------'
        write(*,*) 'Approximated the atomic mass as 2 * ZN = ', mi
      write(*,*)
        '-------------------------------------------------------------------------------------------------------'
      endif
    endif
    ns=nint(zn)
! some initializations
    allocate(species_m1(0:ns), species_p1(0:ns), species_cx(0:ns), species(0:ns), amns_ei(0:ns),
         amns_rc(0:ns), amns_lr(0:ns), amns_br(0:ns))
    allocate(state_labels(0:ns))
    allocate(l(0:ns), d(0:ns), u(0:ns), na(0:ns), rhs(0:ns), te(0:nt), ne(0:nt))
    allocate(rate_ei(0:nt,0:ns), rate_rc(0:nt,0:ns), rate_lr(0:nt,0:ns), rate_br(0:nt,0:ns))
    do is=0, ns
      allocate(species_m1(is)%components(4))
      species_m1(is)%components = (/ amns_reactant_type(zn, is, mi, 0), amns_reactant_type(0, -1, 0, 0),
        amns_reactant_type(zn, is-1, mi, 1), amns_reactant_type(0, -1, 0, 1) /)
      allocate(species_p1(is)%components(4))
      species_p1(is)%components = (/ amns_reactant_type(zn, is, mi, 0), amns_reactant_type(0, -1, 0, 0),
        amns_reactant_type(zn, is+1, mi, 1), amns_reactant_type(0, -1, 0, 1) /)
      allocate(species_cx(is)%components(4))
      species_cx(is)%components = (/ amns_reactant_type(zn, is, mi, 0), amns_reactant_type(1,  0, 2, 0),
        amns_reactant_type(zn, is-1, mi, 1), amns_reactant_type(1,  1, 2, 1) /)
      allocate(species(is)%components(2))
      species(is)%components    = (/ amns_reactant_type(zn ,is, mi, 0),
        amns_reactant_type(zn,   is, mi, 1) /)
    enddo
    do it=0,nt
      te(it) = 10.0_ids_real**(it/120.0_ids_real*7.0_ids_real-1.0_ids_real)
    enddo
    ne=1.0e20_ids_real
! set up the AMNS system
    call imas_amns_setup(amns)
    write(*,*) 'Done IMAS_AMNS_SETUP'
! set up tables for ionization
    xx_rx%string='EI'
    do is=0,ns
      if(species_p1(is)%components(1)%za .ne. species_p1(is)%components(1)%zn) then
        call imas_amns_setup_table(amns, xx_rx, species_p1(is), amns_ei(is))
        query%string='state_label'
        call imas_amns_query_table(amns_ei(is),query,answer)
        state_labels(is)=answer%string
      endif
    enddo
! set up tables for line radiation
    xx_rx%string='LR'
    do is=0,ns
      if(species(is)%components(1)%za .ne. species(is)%components(1)%zn) then
        call imas_amns_setup_table(amns, xx_rx, species(is), amns_lr(is))
        query%string='state_label'
        call imas_amns_query_table(amns_lr(is),query,answer)
        state_labels(is)=answer%string
      endif
    enddo
! set up tables for recombination
    xx_rx%string='RC'
    do is=0,ns
      if(species_m1(is)%components(1)%za .ne. 0) then
        call imas_amns_setup_table(amns, xx_rx, species_m1(is), amns_rc(is))
        query%string='state_label'
        call imas_amns_query_table(amns_rc(is),query,answer)
        state_labels(is)=answer%string
      endif
    enddo
! set up tables for recombination radiation
    xx_rx%string='BR'
    do is=0,ns
      if(species(is)%components(1)%za .ne. 0) then
        call imas_amns_setup_table(amns, xx_rx, species(is), amns_br(is))
        query%string='state_label'
        call imas_amns_query_table(amns_br(is),query,answer)
        state_labels(is)=answer%string
      endif
    enddo
    write(*,*) 'Done IMAS_AMNS_SETUP_TABLE'
! interpolate ionization and line radiation data
```

```
  set%string='nowarn'
  do is=0,ns
     if(species(is)%components(1)%za .ne. species(is)%components(1)%zn) then
         call imas_amns_set_table(amns_ei(is),set)
         call imas_amns_rx(amns_ei(is),rate_ei(:,is),te,ne)
         call imas_amns_set_table(amns_lr(is),set)
         call imas_amns_rx(amns_lr(is),rate_lr(:,is),te,ne)
     endif
  enddo
! interpolate recombination and recombination radiation data
  set%string='nowarn'
  do is=0,ns
     if(species(is)%components(1)%za .ne. 0) then
         call imas_amns_set_table(amns_rc(is),set)
         call imas_amns_rx(amns_rc(is),rate_rc(:,is),te,ne)
         call imas_amns_set_table(amns_br(is),set)
         call imas_amns_rx(amns_br(is),rate_br(:,is),te,ne)
     endif
  enddo
  write(*,*) 'Done IMAS_AMNS_RX'
! finalize the tables
  do is=0,ns
     if(species(is)%components(1)%za .ne. species(is)%components(1)%zn) then
        call imas_amns_finish_table(amns_ei(is))
        call imas_amns_finish_table(amns_lr(is))
     endif
  enddo
  do is=0,ns
     if(species(is)%components(1)%za .ne. 0) then
        call imas_amns_finish_table(amns_rc(is))
        call imas_amns_finish_table(amns_br(is))
     endif
  enddo
  write(*,*) 'Done IMAS_AMNS_FINISH_TABLE'
! finalize the system
  call imas_amns_finish(amns)
  write(*,*) 'Done IMAS_AMNS_FINISH'
  open(10,file='coronal.out')
  write(10,'(100a15)') '#te            ',state_labels, &
                       '   LR          ',  &
                       '   BR          '
  do it=0, nt
! set up the matrix
     l(0)=0.0_ids_real
     d(0)=-rate_ei(it,0)
     u(0)=rate_rc(it,1)
     rhs(0)=0.0_ids_real
     do is=1,ns-1
        l(is)=rate_ei(it,is-1)
        d(is)=-rate_ei(it,is)-rate_rc(it,is)
        u(is)=rate_rc(it,is+1)
        rhs(is)=0.0_ids_real
     enddo
     l(ns)=rate_ei(it,ns-1)
     d(ns)=-rate_rc(it,ns)
     u(ns)=0.0_ids_real
     rhs(ns)=0.0_ids_real
! we need to set the value of 1 charge state
     isref=0
     do is=1, ns-1
        if(rate_ei(it,is) .LT. rate_rc(it,is)) then
           exit
        else
           isref=isref+1
        endif
     enddo
!     write(*,*) te(it), isref
     u(isref)=0.0_ids_real
     d(isref)=1.0_ids_real
     l(isref)=0.0_ids_real
     rhs(isref)=1.0_ids_real
! solve (l,d,u) na = rhs
     call solve_tridiag(l,d,u,rhs,na,ns+1)
     na(:) = na(:) / sum(na)
     line_radiation = sum(na(0:ns-1)*rate_lr(it,0:ns-1))
     recombination_radiation = sum(na(1:ns)*rate_br(it,1:ns))
     write(10,'(1p,100g15.6E3)') te(it), na(:), line_radiation, recombination_radiation
! check the answer
     norm=max(maxval(rate_ei(it,0:ns-1) * na(0:ns-1)), maxval(rate_rc(it,1:ns) * na(1:ns)))
     is=0
     test = (-rate_ei(it,is) * na(is) + rate_rc(it,is+1) * na(is+1))/norm
     if(abs(test).gt.1e-15_ids_real) write(*,*) 'LARGE ERROR: ',it, is, test
     do is=1, ns-1
        test = (rate_ei(it,is-1) * na(is-1) - (rate_rc(it,is) + rate_ei(it,is)) * na(is) + rate_rc(it,is+1)
       * na(is+1))/norm
        if(abs(test).gt.1e-15_ids_real) write(*,*) 'LARGE ERROR: ',it, is, test
     enddo
```

```
      is=ns
      test = (rate_ei(it,is-1) * na(is-1) - rate_rc(it,is) * na(is))/norm
      if(abs(test).gt.1e-15_ids_real) write(*,*) 'LARGE ERROR: ',it, is, test
    enddo
    close(10)
end program coronal
subroutine solve_tridiag(a,b,c,v,x,n)
    use ids_types
    implicit none
!       a - sub-diagonal (means it is the diagonal below the main diagonal)
!       b - the main diagonal
!       c - sup-diagonal (means it is the diagonal above the main diagonal)
!       v - right part
!       x - the answer
!       n - number of equations

    integer,intent(in) :: n
    real(kind=ids_real),dimension(n),intent(in) :: a,b,c,v
    real(kind=ids_real),dimension(n),intent(out) :: x
    real(kind=ids_real),dimension(n) :: bp,vp
    real(kind=ids_real) :: m
    integer i

! Make copies of the b and v variables so that they are unaltered by this sub
    bp(1) = b(1)
    vp(1) = v(1)

    !The first pass (setting coefficients):
    firstpass: do i = 2,n
       m = a(i)/bp(i-1)
       bp(i) = b(i) - m*c(i-1)
       vp(i) = v(i) - m*vp(i-1)
    end do firstpass

    x(n) = vp(n)/bp(n)
    !The second pass (back-substition)
    backsub:do i = n-1, 1, -1
       x(i) = (vp(i) - c(i)*x(i+1))/bp(i)
    end do backsub

end subroutine solve_tridiag
```

and the Makefile used to compile and run the code:

```
# attempt to work out which compiler we are using
ifndef FC
ifeq (${IMASENV_COMP_VENDOR}, intel)
FC = ifort
else ifeq (${IMASENV_COMP_VENDOR}, gcc)
FC = gfortran
else
# fallback to gfortran
FC = gfortran
endif
endif
# set some locations
OBJECTCODE=obj
VPATH=src
# use pkg-config to determine the INCLUDE and LIBS paths
INCLUDE        += ${shell pkg-config --cflags amns imas-${FC}}
LIBS           += ${shell pkg-config --libs amns imas-${FC}}
# by default run the code, compiling it if necessary
run_coronal: ${OBJECTCODE}/coronal
        @time $^ ${ARGS}
${OBJECTCODE}/coronal : ${OBJECTCODE}/coronal.o
        ${FC} ${FCOPTS} -o $@ $^ ${LIBS}
${OBJECTCODE}/%.o : %.f90
        @mkdir -p ${OBJECTCODE}
        ${FC} ${FCOPTS} -I${OBJECTCODE} ${INCLUDE} -c $< -o $@
# provide help
help:
        @echo Compile and run the coronal example using the ${FC} compiler
        @echo By default C is assumed
        @echo To run for W, do
        @echo '    make ARGS="74 183.84"'
        @echo To plot the coronal densities, do
        @echo '    obj/plot_densities'
        @echo To plot the radiation, do
        @echo '    obj/plot_radiation'
        @echo To run many of the species, do
        @echo '    obj/run_cases'
# clean up
clean:
        -rm ${OBJECTCODE}/*.o ${OBJECTCODE}/coronal
# create the TAGS file for emacs
tags:
        rm TAGS ; etags src/*.[fF]90 src/*.c
```

1.1.8.2 Coronal example in Python

```
# Calculate and plot coronal equilibrium quantities
 - coronal.py -- library of coronal routines
 - coronal_radiation_efficiency.py -- produces a plot to compare the radiation efficiency of various
       elements
 - coronal_info.py -- produces plots of average charge, radiation efficiency and fractional abundance
 - coronal_charge_state_edge.py -- produces a plot of the average charge state of various species for
       pedestal/SOL conditions (3e19 m^-2, 1 -- 10000 eV)
 - coronal_comparison_N+Ne.py -- various coronal plots comparing N & Ne
---
To compare tables produced by the fortran version with that produced by the python version:
import numpy as np
F = np.loadtxt('../coronal_f90/coronal.out')
P = np.loadtxt('../coronal_py/coronal_ne=1.00e+20.out')
M = (F > 1e-100) & (P > 1e-100)
print('Maximum relative deviation for values > 1e-100 = %.2e' % (np.max(np.abs(2*(F-P)/(F+P))[M])))
```

A python library to work with coronal distributions (coronal):

```
'''
Tools to work with the coronal distribution
David.Coster@ipp.mpg.de
'''
import numpy as np
import amns
def te_ne (te, ne):
    ''' Given 1d arrays of te and ne, return the appropriate outer product arrays of TE and NE '''
    TE=np.array(te.repeat(len(ne)).reshape([len(te), len(ne)]),order='F')
    NE=np.array(ne.repeat(len(te)).reshape([len(ne), len(te)]).transpose(),order='F')
    return TE, NE
def TDMASolve(a, b, c, d):
    ''' Solve the tridiagonal system '''
    n = len(d) # n is the numbers of rows, a and c has length n-1
    bl = b.copy()
    dl = d.copy()
    for i in range(n-1):
        dl[i+1] -= dl[i] * a[i+1] / bl[i]
        bl[i+1] -= c[i] * a[i+1] / bl[i]
    for i in reversed(range(n-1)):
        dl[i] -= dl[i+1] * c[i] / bl[i+1]
    return dl / bl # return the solution
def point(ei, rc):
    ''' return the coronal fractional densities determined by the balance of ionization and recombination '''
    n=len(ei)
    l = np.zeros(n)
    d = np.zeros(n)
    u = np.zeros(n)
    na = np.zeros(n)
    l[0] = 0.0
    d[0] = -ei[0]
    u[0] = rc[1]
    na[0] = 0.0
    l[1:-1] = ei[0:-2]
    d[1:-1] = -ei[1:-1] - rc[1:-1]
    u[1:-1] = rc[2:]
    na[1:-1] = 0.0
    l[-1] = ei[-2]
    d[-1] = -rc[-1]
    u[-1] = 0.0
    na[-1] = 0.0
    try:
        isref = list(ei[:] < rc[:]).index(True)
    except:
        isref = len(ei[:])-1
    l[isref] = 0.0
    d[isref] = 1.0
    u[isref] = 0.0
    na[isref] = 1.0
    na = TDMASolve(l,d,u,na)
    na = na / na.sum()
    return na
def distribution (rates, te, ne):
    '''
    inputs: rates dictionary, te and ne
    outputs: a dictionary containing:
      the coronal fractional densities
      average charge
      line radiation efficiency
      recombination (including Bremsstrahlung) radiation efficiency
    '''
    n = len(rates['EI'])
    nte = te.shape[0]
    nne = ne.shape[1]
    ei = np.zeros([n, nte, nne])
    rc = np.zeros([n, nte, nne])
    br = np.zeros([n, nte, nne])
    lr = np.zeros([n, nte, nne])
```

```
    na = np.zeros([n, nte, nne])
    for i in range(n):
        ei[i,:,:] = rates['EI'][i].data(te,ne)
        rc[i,:,:] = rates['RC'][i].data(te,ne)
        br[i,:,:] = rates['BR'][i].data(te,ne)
        lr[i,:,:] = rates['LR'][i].data(te,ne)
    for iit in range(nte):
        for iin in range(nne):
            na[:, iit, iin] = point(ei[:, iit, iin], rc[:, iit, iin])
    ZA=(np.arange(n).repeat(nte).repeat(nne)).reshape([n,nte,nne])
    Z=(na[:,:,:]*ZA).sum(axis=0)/na[:,:,:].sum(axis=0)
    LR_rad = (na*lr).sum(axis=0)*ne
    BR_rad = (na*br).sum(axis=0)*ne
    return dict(na=na, Z=Z, LR_rad=LR_rad, BR_rad=BR_rad)
def rates(ZN):
    ''' return the rates dictionary for the specified nuclear charge '''
    reactantsRC=[]
    reactantsEI=[]
    reactantsRD=[]
    for i in range(ZN+1):
        reactantsRC.append(amns.Reactants())
        reactantsRC[i].add(ZN,i,0)
        reactantsRC[i].add(0,-1,0)
        reactantsRC[i].add(ZN,i-1,0,lr=1)
        reactantsRC[i].add(0,-1,0,lr=1)
        reactantsEI.append(amns.Reactants())
        reactantsEI[i].add(ZN,i,0)
        reactantsEI[i].add(0,-1,0)
        reactantsEI[i].add(ZN,i+1,0,lr=1)
        reactantsEI[i].add(0,-1,0,lr=1)
        reactantsRD.append(amns.Reactants())
        reactantsRD[i].add(ZN,i,0)
        reactantsRD[i].add(ZN,i,0,lr=1)
    amnsdb = amns.Amns()
    print('Setting up tables ...')
    vals=dict(SP=[], EI=[], RC=[], BR=[], LR=[])
    for i in range(ZN+1):
        vals['EI'].append(amnsdb.get_table("EI", reactantsEI[i]))
        vals['RC'].append(amnsdb.get_table("RC", reactantsRC[i]))
        vals['BR'].append(amnsdb.get_table("BR", reactantsRD[i]))
        vals['LR'].append(amnsdb.get_table("LR", reactantsRD[i]))
        vals['SP'].append(vals['EI'][i].state_label)
    return vals
```

An example of using the library to calculate radiation efficiencies:

```
#! /usr/bin/env python
"""
Calculate the coronal radiation efficiencies
David.Coster@ipp.mpg.de
"""
import coronal
import numpy as np
import matplotlib.pylab as plt
from matplotlib.colors import LogNorm
# we will consider the following species
L = [1,2,3,4,6,7,10,18,36,54,74]
rates = []
for i in L:
    print(i)
    rates.append(coronal.rates(i))
te, ne = coronal.te_ne(np.logspace(-1,6,num=121), np.array([1e20]))
dist = []
for r in rates:
    print(r['SP'][0][:-1])
    dist.append(coronal.distribution(r, te, ne))
plt.ion()
plt.clf()
for i, r, d in zip(L, rates, dist):
    print(i)
    plt.loglog(te[:,0],(d['LR_rad'][:,0]+d['BR_rad'][:,0])/ne[0,0], label=r['SP'][0][:-1], lw=3)
plt.xlabel('Te [$eV$]')
plt.ylabel('Radiation Efficiency')
plt.ylim(1e-37,1e-30)
plt.title('ne = %s' % (ne[0,0],))
plt.legend(loc=0)
plt.savefig('coronal_radiation_efficiency.pdf')
plt.savefig('coronal_radiation_efficiency.png')
```

An example of using the library to calculate average charge, radiation efficiencies and fractional state abundance:

```
#! /usr/bin/env python
"""
Calculate coronal data
David.Coster@ipp.mpg.de
"""
import coronal
```

```
import numpy as np
import matplotlib.pylab as plt
from matplotlib.colors import LogNorm
import argparse
parser = argparse.ArgumentParser(description="Provide coronal information", epilog
="""
""", formatter_class=argparse.RawTextHelpFormatter)
parser.add_argument("--nuclear_charge", "-z", type=int, help="Nuclear charge", default=74)
args=parser.parse_args()
rates = coronal.rates(args.nuclear_charge)
te, ne = coronal.te_ne(np.logspace(-1,6,num=121), np.array([1e18, 1e19, 1e20, 1e21]))
dist = coronal.distribution(rates, te, ne)
for i, D in enumerate(ne[0,:]):
    np.savetxt('coronal_info_ne=%.2e.out' % (ne[0,i]), np.concatenate((te[:,i:i+1], dist['na'][:,:,i].T,
       dist['LR_rad'][:,i:i+1]/ne[0,i], dist['BR_rad'][:,i:i+1]/ne[0,i]), axis=1), header='te '+'
       '.join(rates['SP'])+' LR BR')
plt.ion()
plt.clf()
plt.clf()
plt.semilogx(te[:,0], dist['Z'])
plt.xlabel('Te [$eV$]')
plt.ylabel('Average charge')
plt.legend(loc=0, labels=ne[0,:])
plt.title('Average charge for %s with varying ne' % (rates['SP'][0][:-1],))
plt.savefig('coronal_info_average_charge.pdf')
plt.savefig('coronal_info_average_charge.png')
plt.clf()
plt.loglog(te,(dist['LR_rad']+dist['BR_rad'])/ne, lw=3)
plt.xlabel('Te [$eV$]')
plt.ylabel('Radiation efficiency')
plt.ylim(1e-37,1e-29)
plt.title('Radiation efficiency for %s with varying ne' % (rates['SP'][0][:-1],))
plt.legend(loc=0, labels=ne[0,:])
plt.savefig('coronal_info_radiation_efficiency.pdf')
plt.savefig('coronal_info_radiation_efficiency.png')
for i, D in enumerate(ne[0,:]):
    plt.clf()
    plt.loglog(te[:,0], dist['na'][:,:,i].T)
    plt.xlabel('Te [$eV$]')
    plt.ylabel('Fractional abundance')
    plt.ylim(1e-3,1)
    plt.title('Fractional abundance for %s with ne = %.2e' % (rates['SP'][0][:-1], ne[0,i]))
    plt.legend(loc='lower center', fontsize=5, labels=rates['SP'], ncol=9)
    plt.savefig('coronal_info_fractional_abundance_ne=%.2e.pdf' % (ne[0,i]))
    plt.savefig('coronal_info_fractional_abundance_ne=%.2e.png' % (ne[0,i]))
```

1.1.9 Verification suite

amns_verify.py contains a python program that can be used to verify (at least in part) the AMNS library.

```
Verify we have a functioning AMNS system
========================================
To do this we run amns_verify.py and then check how many cases have a
maximum relative error larger than 1%:
    ./amns_verify.py > ! amns_verify.pylog
    cat amns_verify.pylog | awk '/^Maximum/&&$7>0.01{print $5,$7}'
Currently (2019-06-19) there should only be one:
    REF/plt89_w_01.dat 15.4588587315
where we have changed to a different set of W data.
Plots will be produced in 'FIG/'
Reference data is in 'REF/'
As a side product of running the amns_verify.py, a LaTeX file "amns_verify.tex" is produced.  This can be
      converted to PDF by doing
    pdflatex amns_verify
and this will produce amns_verify.pdf which will document for a subset
of the reactions the agreement between the reference data (from REF/)
and the AMNS data, using the plots stored in FIG/.
```

The program also contains examples of accessing a number of different reactions.

```
#! /usr/bin/env python
import os
if not os.environ.get('DISPLAY'):
    # we can still plot, even without having a DISPLAY
    import matplotlib as mpl
    mpl.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import colors, ticker
import amns
import numpy as np
import math as m
def plot(x1,y1,x2,y2,xlabel,ylabel,title,file=None,line1='bo-',line2='r+-'):
    height=210/25.4
```

```
    width=297/25.4
    plt.figure(1, figsize=(width,height))
    plt.clf()
    try:
        plt.loglog(x1, y1, line1, label='AMNS') # ValueError: Data has no positive values, and therefore can
       not be log-scaled.
        plt.loglog(x2, y2, line2, label='REF')
    except ValueError:
        print("Warning: Data has no positive values, not plotting log: %s, %s" % (title ,file))
        plt.plot(x1, y1, line1, label='AMNS')
        plt.plot(x2, y2, line2, label='REF')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    y1_max=m.ceil(m.log10(np.nanmax(y1)))
    try:
        y1_min=m.floor(m.log10(np.nanmin(y1))) # ValueError: math domain error
    except ValueError:
        print("Warning: math domain error on (%e): %s, %s" % (np.nanmin(y1), title ,file))
        y1_min=m.floor(m.log10(np.nanmin(y1+1e-32)))
    y1_min=max(y1_max-20,y1_min)
    plt.ylim(10**y1_min,10**y1_max)
    plt.legend(loc=0)
    if (file is None):
        plt.show()
    else:
        try:
            plt.savefig(file, papertype='a4', orientation='landscape')
        except:
            plt.savefig(file)
def adas(zn, za, mi, reac, ref, file=None, texfile=None):
    nx=[101,1]
    x = np.loadtxt(ref)
    nxr=list(nx) ; nxr.reverse()
    te=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
    ne=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
    r = amns.Reactants()
    if reac in ['RC']:
        r.add(zn,za,mi)
        r.add(0,-1,0)
        r.add(zn,za-1,mi,lr=1)
        r.add(0,-1,0,lr=1)
    elif reac in ['EI']:
        r.add(zn,za,mi)
        r.add(0,-1,0)
        r.add(zn,za+1,mi,lr=1)
        r.add(0,-1,0,lr=1)
    elif reac in ['CX']:
        r.add(zn,za,mi)
        r.add(1,0,2)
        r.add(zn,za-1,mi,lr=1)
        r.add(1,1,2,lr=1)
    elif reac in ['BR', 'LR', 'ZE', 'ZE2', 'EIP']:
        r.add(zn,za,mi)
        r.add(zn,za,mi,lr=1)
    else:
        raise ValueError('Invalid option %s' % (reac))
    table = amnsdb.get_table(reac.encode('UTF-8'), r)
    res = table.data(te, ne)
    plot(te[:,0], res[:,0], x[:,0], x[:,2], 'Te', '%s [%s]' % (table.result_label, table.result_unit), '%s
       for ne=%s, ZN=%s, ZA=%s, MI=%s (%s)' % (reac, ne[0,0], zn, za, mi, table.state_label), file)
    max_rel_err = np.max(np.abs((table.data(x[:,0].copy(), x[:,1].copy()) - x[:,2]) / x[:,2]))
    print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
    if file:
        if texfile:
            print('', file=texfile)
            print('\\subsection{%s of %s}' % (table.result_label.replace('^','\\^').replace('_','\\_'),
       table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
            print('', file=texfile)
            print('Comparison of AMNS with %s' % (ref.replace('_','\\_'),) , file=texfile)
            print('', file=texfile)
            print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
       file=texfile)
            print('', file=texfile)
            print('Maximum relative error for %s = %10.03g' % (ref.replace('_','\\_'), max_rel_err),
       file=texfile)
            print('', file=texfile)
def nuclear_HB(r1, r2, p1, p2, reac,ref, file=None, texfile=None):
    nx=[101,1]
    x = np.loadtxt(ref)
    E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
    r = amns.Reactants()
    r.add(r1[0],r1[1],r1[2])
    r.add(r2[0],r2[1],r2[2])
    r.add(p1[0],p1[1],p1[2],lr=1)
    r.add(p2[0],p2[1],p2[2],lr=1)
    table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
```

```
    res = table.data(E)
    plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label, table.result_unit),
       '%s' % (reac,), file)
    max_rel_err = np.max(np.abs((table.data(x[:,0].copy()) - x[:,1]) / x[:,1]))
    print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
    if file:
        if texfile:
            print('', file=texfile)
            print('\\subsection{%s}' % (table.result_label.replace('^','\\^').replace('_','\\_'),),
       file=texfile)
            print('', file=texfile)
            print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
       file=texfile)
            print('', file=texfile)
            print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
       file=texfile)
            print('', file=texfile)
            print('Maximum relative error for %s = %10.03g' % (ref.replace('^','\\^').replace('_','\\_'),
       max_rel_err), file=texfile)
            print('', file=texfile)
def nuclear_HB_tt(r1, r2, p1, p2, reac, ref, file=None, texfile=None):
    nx=[101,1]
    x = np.loadtxt(ref)
    E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10).repeat(nx[1]).reshape(nx),order='F')
    height=210/25.4
    width=297/25.4
    plt.figure(1, figsize=(width,height))
    r = amns.Reactants()
    r.add(r1[0],r1[1],r1[2])
    r.add(r2[0],r2[1],r2[2])
    r.add(p1[0],p1[1],p1[2],lr=1)
    r.add(p2[0],p2[1],p2[2],lr=1)
    table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
    res = table.data(E)
    plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Ti', '%s [%s]' % (table.result_label, table.result_unit), '%s' %
       (reac,), file)
    max_rel_err = np.max(np.abs((table.data(x[:,0].copy()) - x[:,1]) / x[:,1]))
    print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
    if file:
        if texfile:
            print('', file=texfile)
            print('\\subsection{%s of %s}' % (table.result_label.replace('^','\\^').replace('_','\\_'),
       table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
            print('', file=texfile)
            print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
       file=texfile)
            print('', file=texfile)
            print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
       file=texfile)
            print('', file=texfile)
            print('Maximum relative error for %s = %10.03g' % (ref.replace('^','\\^').replace('_','\\_'),
       max_rel_err), file=texfile)
            print('', file=texfile)
def nuclear_HB_bt(r1, r2, p1, p2, reac, ref, file=None, texfile=None):
    nx=[101,1]
    x = np.loadtxt(ref)
    nxr=list(nx) ; nxr.reverse()
    Ti=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
    E=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
    r = amns.Reactants()
    r.add(r1[0],r1[1],r1[2])
    r.add(r2[0],r2[1],r2[2])
    r.add(p1[0],p1[1],p1[2],lr=1)
    r.add(p2[0],p2[1],p2[2],lr=1)
    table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
    res = table.data(Ti, E)
    plot(Ti[:,0], res[:,0], x[:,0], x[:,2], 'Ti', '%s [%s]' % (table.result_label, table.result_unit), '%s
       for E=%s' % (reac, E[0,0]), file)
    max_rel_err = np.max(np.abs((table.data(x[:,0].copy(), x[:,1].copy()) - x[:,2]) / x[:,2]))
    if file:
        if texfile:
            print('', file=texfile)
            print('\\subsection{%s of %s}' % (table.result_label.replace('^','\\^').replace('_','\\_'),
       table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
            print('', file=texfile)
            print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
       file=texfile)
            print('', file=texfile)
            print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
       file=texfile)
            print('', file=texfile)
            print('Maximum relative error for %s = %10.03g' % (ref.replace('^','\\^').replace('_','\\_'),
       max_rel_err), file=texfile)
            print('', file=texfile)
def total_cross_section_EL(zn,za,mi,reac,ref,file=None):
    nx=[101,1]
    x = np.loadtxt(ref)
```

```
    E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
    r = amns.Reactants()
    r.add(zn,za,mi)
    r.add(zn,za,mi,lr=1)
    table = amnsdb.get_table(reac.encode('UTF-8'), r)
    res = table.data(E)
    plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label, table.result_unit),
       '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn, za, mi), file)
def differential_cross_section_EL(zn,za,mi,reac,ref,file=None):
    nx=[101,1]
    x = np.loadtxt(ref)
    nxr=list(nx) ; nxr.reverse()
    E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
    angle=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
    r = amns.Reactants()
    r.add(zn,za,mi)
    r.add(zn,za,mi,lr=1)
    table = amnsdb.get_table(reac.encode('UTF-8'), r)
    res = table.data(angle,E)
    plot(E[:,0], res[:,0], x[:,0], x[:,2], 'Energy', '%s [%s]' % (table.result_label, table.result_unit),
       '%s for angle=%s, ZN=%s, ZA=%s, MI=%s' % (reac, angle[0,0], zn, za, mi), file)
def rct(zn,za,mi,reac,ref,file=None):
    nx=[101,1]
    x = np.loadtxt(ref)
    E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/6.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
    r = amns.Reactants()
    r.add(zn,za,mi)
    r.add(zn,0.0,mi)
    r.add(zn,za-1,mi,lr=1)
    r.add(zn,1.0,mi,lr=1)
    table = amnsdb.get_table(reac.encode('UTF-8'), r)
    res = table.data(E)
    plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label, table.result_unit),
       '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn, za, mi), file, line2='r*')
def sputter(zn_w,za_w,mi_w,zn_p,za_p,mi_p,reac,ref,file=None):
    nx=[101,1]
    x = np.loadtxt(ref, skiprows=6)
    nxr=list(nx) ; nxr.reverse()
    E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
    A=np.array(np.array([0.0]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
    r = amns.Reactants()
    r.add(zn_w,za_w,mi_w)
    r.add(zn_p,za_p,mi_p)
    r.add(zn_w,za_w,mi_w, lr=1)
    table = amnsdb.get_table(reac.encode('UTF-8'), r)
    res = table.data(E,A)
    plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label, table.result_unit),
       '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn_p, za_p, mi_p), file, line2='r*')
def reflect(zn_w,za_w,mi_w,zn_p,za_p,mi_p,reac,ref,file=None):
    nx=[101,1]
    x = np.loadtxt(ref, skiprows=6)
    nxr=list(nx) ; nxr.reverse()
    E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
    A=np.array(np.array([0.0]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
    r = amns.Reactants()
    r.add(zn_w,za_w,mi_w)
    r.add(zn_p,za_p,mi_p)
    r.add(zn_p,za_p,mi_p, lr=1)
    table = amnsdb.get_table(reac.encode('UTF-8'), r)
    res = table.data(E,A)
    plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label, table.result_unit),
       '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn_p, za_p, mi_p), file, line2='r*')
amnsdb = amns.Amns()
if not os.path.exists('FIG'):
    os.makedirs('FIG')
texfile=open('amns_verify.tex', 'w')
print("'\\documentclass[a4wide,10pt]{article}
\\usepackage[pdftex]{graphicx}
\\usepackage[a4paper,includeheadfoot,margin=2.54cm]{geometry}
\\usepackage{parskip,hyperref,multicol}  %% paralist,longtable,booktabs,spverbatim
\\title{AMNS Verification Certificate}
\\author{David.Coster@ipp.mpg.de on behalf of the AMNS Team}
\\begin{document}
\maketitle
\\tableofcontents
\\newpage
\\begin{multicols}{2}"', file=texfile)
print('', file=texfile)
print('\\section{Atomic rate coefficients using ADAS}', file=texfile)
print('', file=texfile)
adas( 6,0,0, "EI",  'REF/scd96_c0.dat', 'FIG/EI.png',  texfile )
adas( 6,1,0, "RC",  'REF/acd96_c1.dat', 'FIG/RC.png',  texfile )
adas( 6,1,0, "CX",  'REF/ccd96_c1.dat', 'FIG/CX.png',  texfile )
adas( 6,1,0, "BR",  'REF/prb96_c1.dat', 'FIG/BR.png',  texfile )
adas( 6,0,0, "LR",  'REF/plt96_c0.dat', 'FIG/LR.png',  texfile )
adas( 6,1,0, "EIP", 'REF/ecd96_c1.dat', 'FIG/EIP.png', texfile )
adas( 74,0,0, "LR",  'REF/plt89_w_01.dat', 'FIG/W_LR.png', texfile )
```

```
print('', file=texfile)
print('\\newpage', file=texfile)
print('\\section{Nuclear cross sections}', file=texfile)
print('', file=texfile)
nuclear_HB( (1,0,2), (1,0,2), (1,0,1), (1,0,3), "NUC_BB", 'REF/ref_D(D,p)T.dat',       'FIG/D(D,p)T.png'
        , texfile )
nuclear_HB( (1,0,2), (1,0,2), (0,0,1), (2,0,3), "NUC_BB", 'REF/ref_D(D,n)^3He.dat',    'FIG/D(D,n)^3He.png'
        , texfile )
nuclear_HB( (1,0,2), (1,0,3), (0,0,1), (2,0,4), "NUC_BB", 'REF/ref_D(T,n)^4He.dat',    'FIG/D(T,n)^4He.png'
        , texfile )
nuclear_HB( (1,0,2), (2,0,3), (1,0,1), (2,0,4), "NUC_BB", 'REF/ref_D(^3He,p)^4He.dat',
       'FIG/D(^3He,p)^4He.png', texfile )
print('', file=texfile)
print('\\newpage', file=texfile)
print('\\section{Nuclear thermal-thermal rate coefficients}', file=texfile)
print('', file=texfile)
nuclear_HB_tt( (1,0,2), (1,0,2), (1,0,1), (1,0,3), 'NUC_TT', 'REF/ref_tt_D(D,p)T_Ti.dat',
       'FIG/tt_D(D,p)T.png'      , texfile )
nuclear_HB_tt( (1,0,2), (1,0,2), (0,0,1), (2,0,3), 'NUC_TT', 'REF/ref_tt_D(D,n)^3He_Ti.dat',
       'FIG/tt_D(D,n)^3He.png'   , texfile )
nuclear_HB_tt( (1,0,2), (1,0,3), (0,0,1), (2,0,4), 'NUC_TT', 'REF/ref_tt_D(T,n)^4He_Ti.dat',
       'FIG/tt_D(T,n)^4He.png'   , texfile )
nuclear_HB_tt( (1,0,2), (2,0,3), (1,0,1), (2,0,4), 'NUC_TT', 'REF/ref_tt_D(^3He,p)^4He_Ti.dat',
       'FIG/tt_D(^3He,p)^4He.png', texfile )
print('', file=texfile)
print('\\newpage', file=texfile)
print('\\section{Nuclear Beam-target rate-coefficients}', file=texfile)
print('', file=texfile)
nuclear_HB_bt( (1,0,2), (1,0,2), (1,0,1), (1,0,3), 'NUC_BT', 'REF/ref_bt_3.dat', 'FIG/bt_D(D,p)T.png'      ,
       texfile )
nuclear_HB_bt( (1,0,2), (1,0,2), (0,0,1), (2,0,3), 'NUC_BT', 'REF/ref_bt_4.dat', 'FIG/bt_D(D,n)^3He.png'   ,
       texfile )
nuclear_HB_bt( (1,0,2), (1,0,3), (0,0,1), (2,0,4), 'NUC_BT', 'REF/ref_bt_1.dat', 'FIG/bt_D(T,n)^4He.png'   ,
       texfile )
nuclear_HB_bt( (1,0,3), (1,0,2), (0,0,1), (2,0,4), 'NUC_BT', 'REF/ref_bt_2.dat', 'FIG/bt_T(D,n)^4He.png'   ,
       texfile )
nuclear_HB_bt( (2,0,3), (1,0,2), (1,0,1), (2,0,4), 'NUC_BT', 'REF/ref_bt_6.dat', 'FIG/bt_^3He(D,p)^4He.png',
       texfile )
nuclear_HB_bt( (1,0,2), (2,0,3), (1,0,1), (2,0,4), 'NUC_BT', 'REF/ref_bt_5.dat', 'FIG/bt_D(^3He,p)^4He.png',
       texfile )
#total_cross_section_EL       ( 74,0,184, 'EL',  'REF/W-total-elastic-cross-section.dat',
       'FIG/W_EL.png' )
#differential_cross_section_EL( 74,0,184, 'dEL', 'REF/W-angular-diff-elastic-cross-section_90.dat',
       'FIG/W_dEL.png' )
#rct(  2,1,  4, 'RCT', 'REF/RCT_He.dat', 'FIG/RCT_He.png' )
#rct( 10,1, 20, 'RCT', 'REF/RCT_Ne.dat', 'FIG/RCT_Ne.png' )
#rct( 18,1, 40, 'RCT', 'REF/RCT_Ar.dat', 'FIG/RCT_Ar.png' )
#rct( 36,1, 84, 'RCT', 'REF/RCT_Kr.dat', 'FIG/RCT_Kr.png' )
#rct( 54,1,131, 'RCT', 'REF/RCT_Xe.dat', 'FIG/RCT_Xe.png' )
#sputter(74, 0, 184, 1, 0, 2, 'SPUT', 'REF/sputter_dw.y', 'FIG/sputter_W_by_D.png' )
#reflect(74, 0, 184, 1, 0, 2, 'REFL', 'REF/reflect_dw.rn', 'FIG/reflect_D_on_W.png' )
amnsdb.finalize()
print("'\\end{multicols}
\\end{document}"', file=texfile)
```

1.1.10 Extending the AMNS Interface

Note that current interface IMAS_AMNS_RX supports data that can depend on 1, 2 or 3 arguments.

Extending this to more arguments would involve:

- adding additional optional arguments to IMAS_AMNS_RX_0, IMAS_AMNS_RX_1, IMAS_AMNS_RX_2 and IMAS_AMNS_RX_3
- modifying the call to interpol in these routines
- modifying interpol in data_suport if going beyond 4 arguments
- creating additional interpolation routines if going beyond quadrilinear interpolation
- providing the equivalent routines in amns_module_isoc.F90
- providing the equivalent interfaces in amns_interface.h equivalent to IMAS_AMNS_C_RX_0_A
- providing the glue code equivalent in amns_interface.h equivalent to IMAS_AMNS_CC_RX_0_A

- providing the equivalent routines and interfaces for Python (in cython/)

The arguments and output all have to match in size and rank, and can be scalars, 1d, 2d or 3d arrays.

Extending this to higher rank output and arguments involves:

- providing IMAS_AMNS_RX_4, IMAS_AMNS_RX_5 etc. in amns_module.F90
- providing the equivalent routines in amns_module_isoc.F90
- providing the equivalent interfaces in amns_interface.h equivalent to IMAS_AMNS_C_RX_0_A
- providing the glue code equivalent in amns_interface.h equivalent to IMAS_AMNS_CC_RX_0_A
- providing the equivalent routines and interfaces for Python (in cython/)

1.2 How the library works

The AMNS system consists of 4 components

- the AMNS library
- the data needed by the AMNS library stored in the "amns_data" IDS under the device name "amns", either in the user's own database or in the local public database
- the "amns_driver" program that fills the IDS's
- the various data files that might be needed by the "amns_driver" program

The library is built on a Fortran interface that provides 9 entry points of access.

- Four of the routines are used to initialize, query, set parameters and finalize the entire AMNS system,
- Four are used to perform the same operation for a "table" which represents some form of Atomic, Molecular, Nuclear or Surface data.
- The remaining routine provides the actual data. Wrappers are provided for C, Python and Java to access these routines.

AMNS data is returned as a function of 1, 2 or 3 arguments (more can be implemented if needed). Currently supported methods are

- linear, bilinear or trilinear interpolation within a table where the data is provided by the relevant IDS
- the calling of an identified analytic function whose parameters may be supplied by the relevant IDS

The AMNS initialization call returns either a handle (Fortran, C) or a class reference (Python).

This handle or class reference can then be used to set or query parameters associated with the AMNS system. They can also be used to request a table reference for a particular reaction, again either a handle or a class reference is returned.

A particular reaction is requested by passing

- the set of reactants and products (see amns_types::amns_reactants_type) and the examples in each of the languages (Fortran example, C example, Python example and Java example)
- a reaction type (see amns_types::amns_reaction_type) which contains a string identifying the reaction and an integer indicating whether the reaction is isotope resolved (*e.g.* a nuclear reaction) or not (*e.g.* most atomic reactions)

The table handle or class reference can be used to set or query the table, or to request results for a particular reaction evaluated at one or more positions in the abstract space associated with the particular reaction (*e.g.* electron temperature and density), with the request allowing for 0D, 1D, 2D or 3D data.

Internally the fortran library opens shot 1, run 0 of the "amns" device stored in the database of:

- the user running the program, defaulting to the public database if the user has no "amns" data
- *unless* the amns_module::imas_amns_setup initialization call overrides this by specifying a different user in the amns_types::amns_version_type argument (in the user field)

and then finds information in the "release" field of the amns_data IDS to determine which shot/run "amns" IDS to read. It opens and reads this amns_data IDS to determine if the requested reaction is available for the specified reactants and products. If it is, it stores the relevant information in the table handle object. In particular, for the table handle, information about the table or function dimensionality as well as particulars about the function evaluation or table interpolation.

Important to note that while the request for a table setup involves disk access to the amns IDS's, use of the table handle once obtained does not.

1.2.1 How to find what AMNS data is available

Information about the AMNS data that is locally available can be running the amns_dump_index.py Python program which dumps information from the index block and the shot/run's pointed to by the index block.

If the AMNS system has been installed in the recommended way, then this program can be run by

```
$AMNS_PREFIX/share/examples/py/amns_dump_index.py
```

for data stored by the user running the program. To find data stored under "public" use

```
$AMNS_PREFIX/share/examples/py/amns_dump_index.py -u public
```

Help on running the program can be find using the "-h" option.

Even more information about the AMNS data available on your local system can be found using the amns_scan.py Python program. This can be run by

```
$AMNS_PREFIX/share/examples/py/amns_scan.py
```

and by default will produce AMNS data stored by the user running the program. To find data stored under "public" use

```
$AMNS_PREFIX/share/examples/py/amns_scan.py -u public
```

The resultant $*$.tex file needs to be processed using pdflatex. Note that the LaTeX package **spverbatim** is required.

Help on running the program can be find using the "-h" option.

1.2.2 More information about AMNS internals

Todo Provide information about some of the internals including the interpolation routines as well as the use of sorting.

1.3 Provision of data for the library

The "amns" IDS's are filled by the fortran program amns_driver.f90. This is driven by a file "amns_driver.data" which contains information about each entry that will be filled. This contains blocks of data for each nuclear charge and mass, and then identifies what data is available. For ADAS ADF11 data, information about each reaction that will be part of the AMNS system is given. For nuclear data, an identifier ("NUCLEAR") is provided and this then triggers a call in the amns_driver program to add the nuclear data that is stored internally.

1.3.1 File used to drive the AMNS data ingestion process

The format for the input file is:

```
this file describes the format of the amns_driver.data file.
the first line identifies the location of the adas adf11 data.
Then follows blocks of the form:
species
<zn> <am>
<process>
END
where
<zn> is the nuclear charge
<am> is the number of nucleons, or 0 if isotope effects are not relevant
<process> can be one or more of adf11, nuclear, elastic_total, elastic_differential, rct, surfacesputter or
      surfacereflection
each of these blocks has optional lines following and is terminated by END.
for adf11, the following line should contain
<zn> <amu> <nr> <'ID'>
where
  <zn> = nuclear charge
  <amu> = atomic mass
  <nr> = number of adas reactions
  <'ID'> = species name(in quotes)
Then for each adas reaction, a line should presented of the form
<rx> <ic> <'FILE'> <'DESCRIPTION'> <'UNITS'> <rt> <dim>
where
  <rx> = reaction label(rc, ei, cx, br, lr, ze, ze2, eip)
  <ic> = adas class of data: 1-acd, 2-scd, 3-ccd, 4-prb, 5-prc, 6-qcd, 7-xcd, 8-plt, 9-pls, 10-zcd, 11-ycd,
      12-ecd
  <'FILE'> = subdirectory/filename(e.g. 'acd96/acd96_he.dat')
  <'DESCRIPTION'> = descriptor of process(e.g. 'Recombination')
  <'UNITS'> = units of reaction(e.g. 'm^{3} s^{-1}')
  <rt> = result_transform(0 => none, 1 => 10**)
  <dim> = dimensionality(always 2 so far)
for nuclear there is no follow on line.
for the surface cpo there are(currently) two types of data:
surfacereflection  -> eckstein reflection yield fit formula parameters
surfacesputter  -> eckstein sputter yield fit formula parameters
each are followed by an input file to be written to the cpo
see the file 'amns_driver.data' for an example.
```

and the current version of the file is:

```
'../../data/atomic/adas/adf11/'
SPECIES
1 0
ADF11
1       1.00794       8 'H'
RC      1  'acd12/acd12_h.dat'          'Recombination'                              'm^{3} s^{-1}'  1
       2
EI      2  'scd12/scd12_h.dat'          'Electron Impact Ionisation'                 'm^{3} s^{-1}'  1
       2
CX      3  'ccd96/ccd96_h.dat'          'CX recombination coeffts'                   'm^{3} s^{-1}'  1
       2
BR      4  'prb12/prb12_h.dat'          'Recomb/brems power coeffts'                 'W m^{3}'       1
       2
LR      8  'plt12/plt12_h.dat'          'Line radiation'                             'W m^{3}'       1
       2
```

```
ZE     10  'zcd96/zcd96_h.dat'          'Effective Charge'                          'e'             0
       2
ZE2    11  'ycd96/ycd96_h.dat'          'Effective Square Charge'                   'e^{2}'         0
       2
EIP    12  'ecd96/ecd96_h.dat'          'Effective Ionisation Potential'            'eV'            0
       2
BMS
END
SPECIES
1 2
NUCLEAR
END
SPECIES
1 3
NUCLEAR
END
SPECIES
2 0
ADF11
2       4.002602      8 'He'
RC      1  'acd96/acd96_he.dat'          'Recombination'                             'm^{3} s^{-1}'  1
       2
EI      2  'scd96/scd96_he.dat'          'Electron Impact Ionisation'                'm^{3} s^{-1}'  1
       2
CX      3  'ccd96/ccd96_he.dat'          'CX recombination coeffts'                  'm^{3} s^{-1}'  1
       2
BR      4  'prb96/prb96_he.dat'          'Recomb/brems power coeffts'                'W m^{3}'       1
       2
LR      8  'plt96/plt96_he.dat'          'Line radiation'                            'W m^{3}'       1
       2
ZE     10  'zcd96/zcd96_he.dat'          'Effective Charge'                          'e'             0
       2
ZE2    11  'ycd96/ycd96_he.dat'          'Effective Square Charge'                   'e^{2}'         0
       2
EIP    12  'ecd96/ecd96_he.dat'          'Effective Ionisation Potential'            'eV'            0
       2
END
SPECIES
2 3
NUCLEAR
END
SPECIES
3 0
ADF11
3       6.941         8 'Li'
RC      1  'acd96/acd96_li.dat'          'Recombination'                             'm^{3} s^{-1}'  1
       2
EI      2  'scd96/scd96_li.dat'          'Electron Impact Ionisation'                'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_li.dat'          'CX recombination coeffts'                  'm^{3} s^{-1}'  1
       2
BR      4  'prb96/prb96_li.dat'          'Recomb/brems power coeffts'                'W m^{3}'       1
       2
LR      8  'plt96/plt96_li.dat'          'Line radiation'                            'W m^{3}'       1
       2
ZE     10  'zcd96/zcd96_li.dat'          'Effective Charge'                          'e'             0
       2
ZE2    11  'ycd96/ycd96_li.dat'          'Effective Square Charge'                   'e^{2}'         0
       2
EIP    12  'ecd96/ecd96_li.dat'          'Effective Ionisation Potential'            'eV'            0
       2
END
SPECIES
4 0
ADF11
4       9.012182      8 'Be'
RC      1  'acd96/acd96_be.dat'          'Recombination'                             'm^{3} s^{-1}'  1
       2
EI      2  'scd96/scd96_be.dat'          'Electron Impact Ionisation'                'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_be.dat'          'CX recombination coeffts'                  'm^{3} s^{-1}'  1
       2
BR      4  'prb96/prb96_be.dat'          'Recomb/brems power coeffts'                'W m^{3}'       1
       2
LR      8  'plt96/plt96_be.dat'          'Line radiation'                            'W m^{3}'       1
       2
ZE     10  'zcd96/zcd96_be.dat'          'Effective Charge'                          'e'             0
       2
ZE2    11  'ycd96/ycd96_be.dat'          'Effective Square Charge'                   'e^{2}'         0
       2
EIP    12  'ecd96/ecd96_be.dat'          'Effective Ionisation Potential'            'eV'            0
       2
SURFACESPUTTER
'../../../data/surface/block_syield_be.dat'
SURFACEREFLECTION
'../../../data/surface/block_ryield_be.dat'
END
```

```
SPECIES
5 0
ADF11
5       10.811        8 'B'
RC      1  'acd89/acd89_b.dat'           'Recombination'                                'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_b.dat'           'Electron Impact Ionisation'                   'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_b.dat'           'CX recombination coeffts'                     'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_b.dat'           'Recomb/brems power coeffts'                   'W m^{3}'       1
       2
LR      8  'plt89/plt89_b.dat'           'Line radiation'                               'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_b.dat'           'Effective Charge'                             'e'             0
       2
ZE2    11  'ycd89/ycd89_b.dat'           'Effective Square Charge'                      'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_b.dat'           'Effective Ionisation Potential'               'eV'            0
       2
SURFACESPUTTER
'../../../data/surface/block_syield_b.dat'
SURFACEREFLECTION
'../../../data/surface/block_ryield_b.dat'
END
SPECIES
6 0
ADF11
6       12.011        8 'C'
RC      1  'acd96/acd96_c.dat'           'Recombination'                                'm^{3} s^{-1}'  1
       2
EI      2  'scd96/scd96_c.dat'           'Electron Impact Ionisation'                   'm^{3} s^{-1}'  1
       2
CX      3  'ccd96/ccd96_c.dat'           'CX recombination coeffts'                     'm^{3} s^{-1}'  1
       2
BR      4  'prb96/prb96_c.dat'           'Recomb/brems power coeffts'                   'W m^{3}'       1
       2
LR      8  'plt96/plt96_c.dat'           'Line radiation'                               'W m^{3}'       1
       2
ZE     10  'zcd96/zcd96_c.dat'           'Effective Charge'                             'e'             0
       2
ZE2    11  'ycd96/ycd96_c.dat'           'Effective Square Charge'                      'e^{2}'         0
       2
EIP    12  'ecd96/ecd96_c.dat'           'Effective Ionisation Potential'               'eV'            0
       2
SURFACESPUTTER
'../../../data/surface/block_syield_c.dat'
SURFACEREFLECTION
'../../../data/surface/block_ryield_c.dat'
END
SPECIES
7 0
ADF11
7       14.00674      8 'N'
RC      1  'acd96/acd96_n.dat'  'Recombination'                'm^{3} s^{-1}'  1 2
EI      2  'scd96/scd96_n.dat'  'Electron Impact Ionisation'    'm^{3} s^{-1}'  1 2
CX      3  'ccd89/ccd89_n.dat'  'CX recombination coeffts'      'm^{3} s^{-1}'  1 2
BR      4  'prb96/prb96_n.dat'  'Recomb/brems power coeffts'    'W m^{3}'       1 2
LR      8  'plt96/plt96_n.dat'  'Line radiation'                'W m^{3}'       1 2
ZE     10  'zcd96/zcd96_n.dat'  'Effective Charge'              'e'             0 2
ZE2    11  'ycd96/ycd96_n.dat'  'Effective Square Charge'                'e^{2}'         0 2
EIP    12  'ecd96/ecd96_n.dat'  'Effective Ionisation Potential'        'eV'            0 2
END
SPECIES
8 0
ADF11
8       15.9994       8 'O'
RC      1  'acd96/acd96_o.dat'  'Recombination'                                        'm^{3} s^{-1}'  1
       2
EI      2  'scd96/scd96_o.dat'  'Electron Impact Ionisation'                           'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_o.dat'  'CX recombination coeffts'                             'm^{3} s^{-1}'  1
       2
BR      4  'prb96/prb96_o.dat'  'Recomb/brems power coeffts'                           'W m^{3}'       1
       2
LR      8  'plt96/plt96_o.dat'  'Line radiation'                                       'W m^{3}'       1
       2
ZE     10  'zcd96/zcd96_o.dat'  'Effective Charge'                                     'e'             0
       2
ZE2    11  'ycd96/ycd96_o.dat'  'Effective Square Charge'                              'e^{2}'         0
       2
EIP    12  'ecd96/ecd96_o.dat'  'Effective Ionisation Potential'                       'eV'            0
       2
END
SPECIES
9 0
ADF11
```

```
9       18.9984032    8 'F'
RC      1  'acd89/acd89_f.dat'  'Recombination'                                  'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_f.dat'  'Electron Impact Ionisation'                     'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_f.dat'  'CX recombination coeffts'                       'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_f.dat'  'Recomb/brems power coeffts'                     'W m^{3}'       1
       2
LR      8  'plt89/plt89_f.dat'  'Line radiation'                                 'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_f.dat'  'Effective Charge'                               'e'             0
       2
ZE2    11  'ycd89/ycd89_f.dat'  'Effective Square Charge'                        'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_f.dat'  'Effective Ionisation Potential'                 'eV'            0
       2
END
SPECIES
10 0
ADF11
10      20.1797       8 'Ne'
RC      1  'acd96/acd96_ne.dat'  'Recombination'                                 'm^{3} s^{-1}'  1
       2
EI      2  'scd96/scd96_ne.dat'  'Electron Impact Ionisation'                    'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_ne.dat'  'CX recombination coeffts'                      'm^{3} s^{-1}'  1
       2
BR      4  'prb96/prb96_ne.dat'  'Recomb/brems power coeffts'                    'W m^{3}'       1
       2
LR      8  'plt96/plt96_ne.dat'  'Line radiation'                                'W m^{3}'       1
       2
ZE     10  'zcd96/zcd96_ne.dat'  'Effective Charge'                              'e'             0
       2
ZE2    11  'ycd96/ycd96_ne.dat'  'Effective Square Charge'                       'e^{2}'         0
       2
EIP    12  'ecd96/ecd96_ne.dat'  'Effective Ionisation Potential'                'eV'            0
       2
END
SPECIES
13 0
ADF11
13      26.981539     8 'Al'
RC      1  'acd89/acd89_al.dat'          'Recombination'                         'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_al.dat'          'Electron Impact Ionisation'            'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_al.dat'          'CX recombination coeffts'              'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_al.dat'          'Recomb/brems power coeffts'            'W m^{3}'       1
       2
LR      8  'plt89/plt89_al.dat'          'Line radiation'                        'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_al.dat'          'Effective Charge'                      'e'             0
       2
ZE2    11  'ycd89/ycd89_al.dat'          'Effective Square Charge'               'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_al.dat'          'Effective Ionisation Potential'        'eV'            0
       2
END
SPECIES
14 0
ADF11
14      28.0855       8 'Si'
RC      1  'acd89/acd89_si.dat'          'Recombination'                         'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_si.dat'          'Electron Impact Ionisation'            'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_si.dat'          'CX recombination coeffts'              'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_si.dat'          'Recomb/brems power coeffts'            'W m^{3}'       1
       2
LR      8  'plt89/plt89_si.dat'          'Line radiation'                        'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_si.dat'          'Effective Charge'                      'e'             0
       2
ZE2    11  'ycd89/ycd89_si.dat'          'Effective Square Charge'               'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_si.dat'          'Effective Ionisation Potential'        'eV'            0
       2
END
SPECIES
16 0
ADF11
16      32.066        8 'S'
RC      1  'acd89/acd89_s.dat'           'Recombination'                         'm^{3} s^{-1}'  1
       2
```

```
EI      2  'scd89/scd89_s.dat'          'Electron Impact Ionisation'                 'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_s.dat'          'CX recombination coeffts'                   'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_s.dat'          'Recomb/brems power coeffts'                 'W m^{3}'       1
       2
LR      8  'plt89/plt89_s.dat'          'Line radiation'                             'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_s.dat'          'Effective Charge'                           'e'             0
       2
ZE2    11  'ycd89/ycd89_s.dat'          'Effective Square Charge'                    'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_s.dat'          'Effective Ionisation Potential'             'eV'            0
       2
END
SPECIES
17 0
ADF11
17      35.4527       8 'Cl'
RC      1  'acd89/acd89_cl.dat'          'Recombination'                              'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_cl.dat'          'Electron Impact Ionisation'                 'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_cl.dat'          'CX recombination coeffts'                   'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_cl.dat'          'Recomb/brems power coeffts'                 'W m^{3}'       1
       2
LR      8  'plt89/plt89_cl.dat'          'Line radiation'                             'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_cl.dat'          'Effective Charge'                           'e'             0
       2
ZE2    11  'ycd89/ycd89_cl.dat'          'Effective Square Charge'                    'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_cl.dat'          'Effective Ionisation Potential'             'eV'            0
       2
END
SPECIES
18 0
ADF11
18      39.948        8 'Ar'
RC      1  'acd89/acd89_ar.dat'          'Recombination'                              'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_ar.dat'          'Electron Impact Ionisation'                 'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_ar.dat'          'CX recombination coeffts'                   'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_ar.dat'          'Recomb/brems power coeffts'                 'W m^{3}'       1
       2
LR      8  'plt89/plt89_ar.dat'          'Line radiation'                             'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_ar.dat'          'Effective Charge'                           'e'             0
       2
ZE2    11  'ycd89/ycd89_ar.dat'          'Effective Square Charge'                    'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_ar.dat'          'Effective Ionisation Potential'             'eV'            0
       2
END
SPECIES
24 0
ADF11
24      51.9961       8 'Cr'
RC      1  'acd89/acd89_cr.dat'          'Recombination'                              'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_cr.dat'          'Electron Impact Ionisation'                 'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_cr.dat'          'CX recombination coeffts'                   'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_cr.dat'          'Recomb/brems power coeffts'                 'W m^{3}'       1
       2
LR      8  'plt89/plt89_cr.dat'          'Line radiation'                             'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_cr.dat'          'Effective Charge'                           'e'             0
       2
ZE2    11  'ycd89/ycd89_cr.dat'          'Effective Square Charge'                    'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_cr.dat'          'Effective Ionisation Potential'             'eV'            0
       2
END
SPECIES
26 0
ADF11
26      55.847        8 'Fe'
RC      1  'acd89/acd89_fe.dat'          'Recombination'                              'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_fe.dat'          'Electron Impact Ionisation'                 'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_fe.dat'          'CX recombination coeffts'                   'm^{3} s^{-1}'  1
```

```
       2
BR      4  'prb89/prb89_fe.dat'          'Recomb/brems power coeffts'                'W m^{3}'       1
       2
LR      8  'plt89/plt89_fe.dat'          'Line radiation'                            'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_fe.dat'          'Effective Charge'                          'e'             0
       2
ZE2    11  'ycd89/ycd89_fe.dat'          'Effective Square Charge'                   'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_fe.dat'          'Effective Ionisation Potential'            'eV'            0
       2
END
SPECIES
28 0
ADF11
28      58.6934       8 'Ni'
RC      1  'acd89/acd89_ni.dat'          'Recombination'                             'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_ni.dat'          'Electron Impact Ionisation'                'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_ni.dat'          'CX recombination coeffts'                  'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_ni.dat'          'Recomb/brems power coeffts'                'W m^{3}'       1
       2
LR      8  'plt89/plt89_ni.dat'          'Line radiation'                            'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_ni.dat'          'Effective Charge'                          'e'             0
       2
ZE2    11  'ycd89/ycd89_ni.dat'          'Effective Square Charge'                   'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_ni.dat'          'Effective Ionisation Potential'            'eV'            0
       2
END
SPECIES
29 0
ADF11
29      63.546        8 'Cu'
RC      1  'acd89/acd89_cu.dat'          'Recombination'                             'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_cu.dat'          'Electron Impact Ionisation'                'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_cu.dat'          'CX recombination coeffts'                  'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_cu.dat'          'Recomb/brems power coeffts'                'W m^{3}'       1
       2
LR      8  'plt89/plt89_cu.dat'          'Line radiation'                            'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_cu.dat'          'Effective Charge'                          'e'             0
       2
ZE2    11  'ycd89/ycd89_cu.dat'          'Effective Square Charge'                   'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_cu.dat'          'Effective Ionisation Potential'            'eV'            0
       2
END
SPECIES
32 0
ADF11
32      72.61         8 'Ge'
RC      1  'acd89/acd89_ge.dat'          'Recombination'                             'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_ge.dat'          'Electron Impact Ionisation'                'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_ge.dat'          'CX recombination coeffts'                  'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_ge.dat'          'Recomb/brems power coeffts'                'W m^{3}'       1
       2
LR      8  'plt89/plt89_ge.dat'          'Line radiation'                            'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_ge.dat'          'Effective Charge'                          'e'             0
       2
ZE2    11  'ycd89/ycd89_ge.dat'          'Effective Square Charge'                   'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_ge.dat'          'Effective Ionisation Potential'            'eV'            0
       2
END
SPECIES
36 0
ADF11
36      83.80         8 'Kr'
RC      1  'acd89/acd89_kr.dat'          'Recombination'                             'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_kr.dat'          'Electron Impact Ionisation'                'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_kr.dat'          'CX recombination coeffts'                  'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_kr.dat'          'Recomb/brems power coeffts'                'W m^{3}'       1
       2
```

```
LR      8  'plt89/plt89_kr.dat'          'Line radiation'                          'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_kr.dat'          'Effective Charge'                        'e'             0
       2
ZE2    11  'ycd89/ycd89_kr.dat'          'Effective Square Charge'                 'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_kr.dat'          'Effective Ionisation Potential'          'eV'            0
       2
END
SPECIES
42 0
ADF11
42      95.94         8 'Mo'
RC      1  'acd89/acd89_mo.dat'          'Recombination'                           'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_mo.dat'          'Electron Impact Ionisation'              'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_mo.dat'          'CX recombination coeffts'                'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_mo.dat'          'Recomb/brems power coeffts'              'W m^{3}'       1
       2
LR      8  'plt89/plt89_mo.dat'          'Line radiation'                          'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_mo.dat'          'Effective Charge'                        'e'             0
       2
ZE2    11  'ycd89/ycd89_mo.dat'          'Effective Square Charge'                 'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_mo.dat'          'Effective Ionisation Potential'          'eV'            0
       2
END
SPECIES
54 0
ADF11
54      131.29        8 'Xe'
RC      1  'acd89/acd89_xe.dat'          'Recombination'                           'm^{3} s^{-1}'  1
       2
EI      2  'scd89/scd89_xe.dat'          'Electron Impact Ionisation'              'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_xe.dat'          'CX recombination coeffts'                'm^{3} s^{-1}'  1
       2
BR      4  'prb89/prb89_xe.dat'          'Recomb/brems power coeffts'              'W m^{3}'       1
       2
LR      8  'plt89/plt89_xe.dat'          'Line radiation'                          'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_xe.dat'          'Effective Charge'                        'e'             0
       2
ZE2    11  'ycd89/ycd89_xe.dat'          'Effective Square Charge'                 'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_xe.dat'          'Effective Ionisation Potential'          'eV'            0
       2
END
SPECIES
74 0
ADF11
74      183.84        8 'W'
RC      1  'acd50/acd50_w.dat'           'Recombination'                           'm^{3} s^{-1}'  1
       2
EI      2  'scd50/scd50_w.dat'           'Electron Impact Ionisation'              'm^{3} s^{-1}'  1
       2
CX      3  'ccd89/ccd89_w.dat'           'CX recombination coeffts'                'm^{3} s^{-1}'  1
       2
BR      4  'prb50/prb50_w.dat'           'Recomb/brems power coeffts'              'W m^{3}'       1
       2
LR      8  'plt42/plt42_w.dat'           'Line radiation'                          'W m^{3}'       1
       2
ZE     10  'zcd89/zcd89_w.dat'           'Effective Charge'                        'e'             0
       2
ZE2    11  'ycd89/ycd89_w.dat'           'Effective Square Charge'                 'e^{2}'         0
       2
EIP    12  'ecd89/ecd89_w.dat'           'Effective Ionisation Potential'          'eV'            0
       2
END
```

Chapter 2

Calculate and plot coronal equilibrium quantities

- coronal.py – library of coronal routines
- coronal_radiation_efficiency.py – produces a plot to compare the radiation efficiency of various elements
- coronal_info.py – produces plots of average charge, radiation efficiency and fractional abundance
- coronal_charge_state_edge.py – produces a plot of the average charge state of various species for pedestal/SOL conditions (3e19 m^-2, 1 – 10000 eV)
- coronal_comparison_N+Ne.py – various coronal plots comparing N & Ne

---

To compare tables produced by the fortran version with that produced by the python version:
import numpy as np F = np.loadtxt('../coronal_f90/coronal.out') P = np.loadtxt('../coronal_py/coronal_ne=1.↩
00e+20.out') M = (F > 1e-100) & (P > 1e-100)
print('Maximum relative deviation for values > 1e-100 = %.2e' % (np.max(np.abs(2∗(F-P)/(F+P))[M])))

Chapter 3

Example python programs

Programs reading the amns IDS's directly:

amns_dump_index.py : dumps the index block identifying what AMNS data is available (can specify another user's database with the "--user" argument)
amns_scan.py : prepares a latex document describing the available AMNS data in the amns IDS's (for the public and user databases) by scanning both the index and data amns IDS's

Programs using the AMNS library

amns_test.py : demonstrates using the AMNS library to access atomic data
amns_nuclear.py : demonstrates using the AMNS library to access nuclear data
amns_nuclear_densities.py : implement a 0D model of the densities of H, D, T, 3-He, 4-He and n involved in thermonuclear reactions

Chapter 4

This is a minimal test example for Java

to run the example: ./AmnsMinimal or, if the CLASSPATH is set to include the AMNS jar file: java -cp .:$CLASS↩
PATH AmnsMinimal
environment variables:

- IMAS_AMNS_JAVA_DEBUG yes OR no enable extra debugging output for the java example
- IMAS_AMNS_DEBUG yes OR no enable debugging of the AMNS system

Chapter 5

Minimal example for matlab

We have something that works but is based on some hacks.
In order for the matlab test case to work, two files (javaclasspath.txt and javalibrarypath.txt) need to be present, either in the run directory or in the user's matlab preference directory.
These need to contain the path to the AMNS jar file, and the path to the directory containing the libamnsjni.so file, and are obtained by autoconfig/automake magic from the equivalent $*$.in files.
To run the test case:
make installcheck-local
or, directly:
matlab -nosplash -nodesktop -r "run testminimal.m; quit"

Chapter 6

Verify we have a functioning AMNS system

To do this we run amns_verify.py and then check how many cases have a maximum relative error larger than 1%:

```
./amns_verify.py > ! amns_verify.pylog
cat amns_verify.pylog | awk '/^Maximum/&&$7>0.01{print $5,$7}'
```

Currently (2019-06-19) there should only be one:

```
REF/plt89_w_01.dat 15.4588587315
```

where we have changed to a different set of W data.
Plots will be produced in 'FIG/'
Reference data is in 'REF/'
As a side product of running the amns_verify.py, a LaTeX file "amns_verify.tex" is produced. This can be converted to PDF by doing

```
pdflatex amns_verify
```

and this will produce amns_verify.pdf which will document for a subset of the reactions the agreement between the reference data (from REF/) and the AMNS data, using the plots stored in FIG/.

Chapter 7

Todo List

page AMNS Library

Provide information about some of the internals including the interpolation routines as well as the use of sorting.

Subprogram amns_external_functions::sputter_data_1004 (function_parameters, energy_arr, angle_arr, yield_arr, with_warnings, fun_err)

provide more information

Subprogram amns_external_functions::reflect_data_1005 (function_parameters, energy_arr, angle_arr, refl_arr, with_warnings, fun_err)

provide more information

Subprogram amns_external_functions::nuclear_data_1006 (function_parameters, x, f, with_warnings, fun_err)

provide more information

Type amns_module::imas_amns_rx

Update this interface for 4D and 5D

Subprogram eckstein_yields::etf (M1, M2, Z1, Z2)

provide more information

Subprogram eckstein_yields::omegal (epsl)

provide more information

Subprogram eckstein_yields::sn (epsl)

provide more information

Subprogram eckstein_yields::seyield (E0, M1, M2, Z1, Z2, q, lambda, u, ETh)

provide more information

Subprogram eckstein_yields::sayield (E0, theta, f, b, c, Esp)

provide more information

Subprogram eckstein_yields::reyieldlight (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

provide more information

Subprogram eckstein_yields::reyieldself (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

provide more information

Subprogram eckstein_yields::reyield (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

provide more information

Subprogram eckstein_yields::rayield (angledeg, C1, C2, C3, C4)

provide more information

Chapter 8

Module Index

8.1 Modules

Here is a list of all modules:

Chapter 9

Modules Index

9.1 Modules List

Here is a list of all modules with brief descriptions:

Chapter 10

Data Type Index

10.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 11

Data Type Index

11.1 Data Types List

Here are the data types with brief descriptions:

Chapter 12

File Index

12.1 File List

Here is a list of all files with brief descriptions:

Chapter 13

Module Documentation

13.1 AMNS Directly Called External Functions/Subroutines

External functions/subroutines to be used directly by the ITM AMNS interface.

Functions/Subroutines

- subroutine amns_external_functions::nuclear_data_1001 (function_parameters, x, f, with_warnings, fun_err)

  AMNS External function ...
- subroutine amns_external_functions::nuclear_data_1002 (function_parameters, Tin, f, with_warnings, fun↩ _err)

  AMNS External function ...
- subroutine amns_external_functions::rct_data_1003 (function_parameters, Tin, f, with_warnings, fun_err)

  AMNS External function ...
- subroutine amns_external_functions::sputter_data_1004 (function_parameters, energy_arr, angle_arr, yield_arr, with_warnings, fun_err)

  AMNS External function ...
- subroutine amns_external_functions::reflect_data_1005 (function_parameters, energy_arr, angle_arr, refl_↩ arr, with_warnings, fun_err)

  AMNS External function ...
- subroutine amns_external_functions::nuclear_data_1006 (function_parameters, x, f, with_warnings, fun_err)

  AMNS External function ...

13.1.1 Detailed Description

External functions/subroutines to be used directly by the ITM AMNS interface.

Author

David Coster and others

13.1.2 Function/Subroutine Documentation

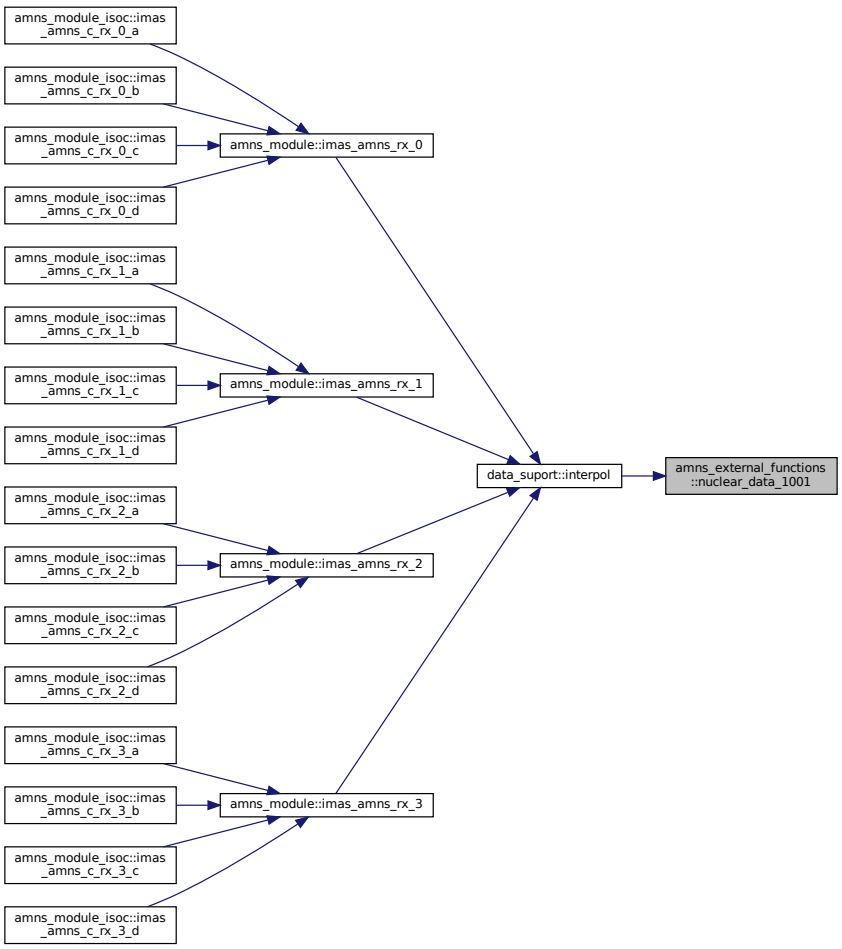
13.1.2.1 nuclear_data_1001()

```
subroutine amns_external_functions::nuclear_data_1001 (
            real (ids_real), dimension(:,:), intent(in) function_parameters,
            real (ids_real), dimension(:), intent(in) x,
            real (ids_real), dimension(:), intent(out) f,
            logical, intent(in) with_warnings,
            type (fun_err_t), intent(out) fun_err )
```

AMNS External function ...
Bosch and Hale "Improved formulas for fusion cross-sections and thermal reactivities" (See doi=10.1088/0029-5515/32/4/I07) **?** and corrigendum **?**
Definition at line 25 of file amns_external_functions.f90.

```
00026     implicit none
00027     real (ids_real),  intent(in)  :: function_parameters(:,:), x(:)
00028     real (ids_real),  intent(out) :: f(:)
00029     logical,           intent(in)  :: with_warnings
00030     type (fun_err_t),  intent(out) :: fun_err
00031     integer :: i, j
00032     real (ids_real) :: e, s
00033
00034     fun_err%ierr = 0
00035     fun_err%cerr = ''
00036
00037     if(size(function_parameters,1).ne.12) then
00038        write(fun_err%cerr,*) "nuclear_data_1001: Expected the length of 'function_parameters' to be 12
      in 'nuclear_data_1001' and not ", &
00039             size(function_parameters)
00040        fun_err%ierr = -1
00041        return
00042     endif
00043     if(size(x).ne.size(f)) then
00044        write(fun_err%cerr,*) "nuclear_data_1001: Expected the input and output vectors to be the same
      size in 'nuclear_data_1001'"
00045        fun_err%ierr = -1
00046        return
00047     endif
00048
00049     do i=1, size(x)
00050        e = x(i) * 1.0e-3_ids_real
00051        j=1
00052        if(e.lt.function_parameters(11,j) .and. with_warnings) then
00053           write(*,*) 'extrapolating  below the desired range'
00054        endif
00055        do while(e.gt.function_parameters(12,j).and. &
00056             j.lt.size(function_parameters,2))
00057           j=j+1
00058        enddo
00059        if(e.gt.function_parameters(12,j)) then
00060           if (with_warnings) then
00061              write(*,*) 'extrapolating  above the desired range'
00062              write(*,*) j, x(i), function_parameters(12,j)
00063              write(*,*) 'Taking the boundary value when calculating S!'
00064           endif
00065           e = function_parameters(12,j)
00066        endif
00067        s = ( function_parameters(2,j) &
00068             & + e*(function_parameters(3,j) &
00069             & + e*(function_parameters(4,j) &
00070             & + e*( function_parameters(5,j) &
00071             & +e*function_parameters(6,j) ) ) ) ) &
00072             & /( 1 + e*(function_parameters(7,j)&
00073             &  + e*(function_parameters(8,j) &
00074             & + e*( function_parameters(9,j) &
00075             & +e*function_parameters(10,j) ) ) ) )
00076        ! We clamp the energy for S (the nuclear fusion probability).
00077        ! However, there is no need to clamp the energy in denominator.
00078        ! The denominator roughly describes the probability of the reactants to
00079        ! overcome the Coulomb barrier.
00080        e = x(i) * 1.0e-3_ids_real
00081        f(i) = s / (e * exp(function_parameters(1,j)/sqrt(e))) * 1.0e-31_ids_real
00082     enddo
```

Here is the caller graph for this function:



13.1.2.2 nuclear_data_1002()

```
subroutine amns_external_functions::nuclear_data_1002 (
            real (ids_real), dimension(:,:), intent(in) function_parameters,
            real (ids_real), dimension(:), intent(in) Tin,
            real (ids_real), dimension(:), intent(out) f,
            logical, intent(in) with_warnings,
            type (fun_err_t), intent(out) fun_err )
```

AMNS External function ...

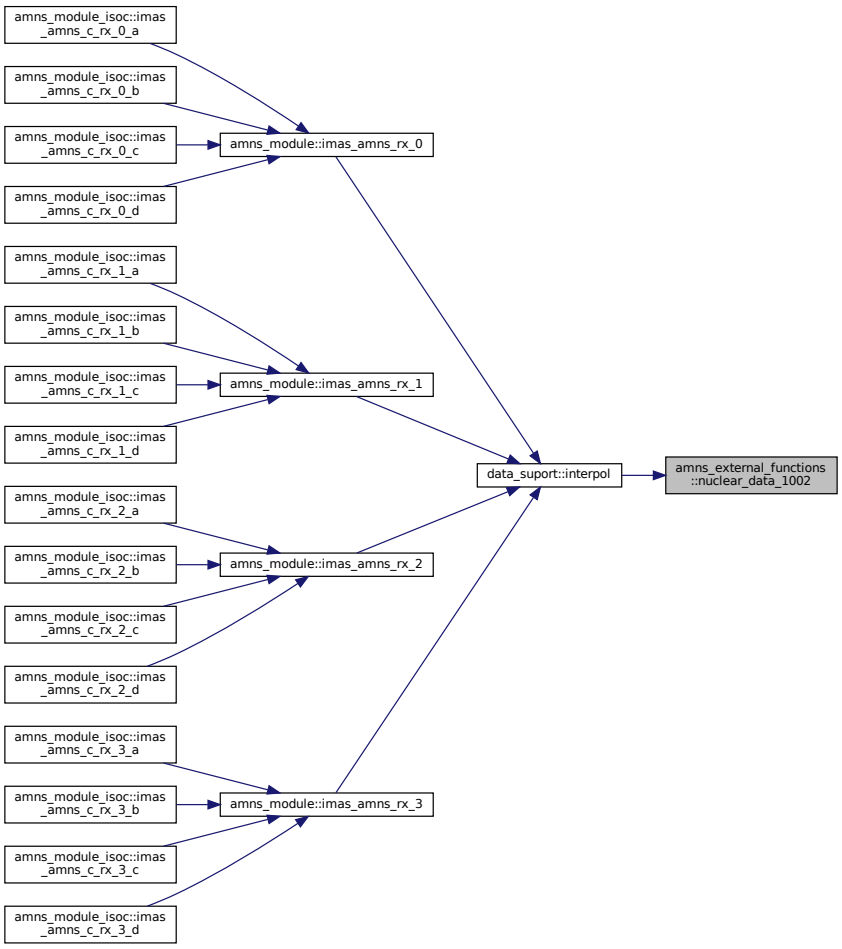
Bosch and Hale "Improved formulas for fusion cross-sections and thermal reactivities" (See doi=10.1088/0029-5515/32/4/I07) **?** and corrigendum **?**

Definition at line 91 of file amns_external_functions.f90.

```
00092     implicit none
00093     real (ids_real), intent(in)  :: function_parameters(:,:), Tin(:)
00094     real (ids_real), intent(out) :: f(:)
00095     logical,           intent(in)  :: with_warnings
00096     type (fun_err_t),  intent(out) :: fun_err
00097     integer :: iT ! index for the different energy vaulues
00098     integer :: iR ! index for possible different temperature ranges
00099
00100     real (ids_real) :: T
00101     real (ids_real) :: theta,xi,mrc2,BG
00102     real (ids_real) :: C(1:7)
00103
```

```
00104     fun_err%ierr = 0
00105     fun_err%cerr = ”
00106
00107     if(size(function_parameters,1).ne.11) then
00108        write(fun_err%cerr,*) "nuclear_data_1002: Expected the length of ’function_parameters’ to be 11
       in ’nuclear_data_1002’ and not ", &
00109             size(function_parameters)
00110        fun_err%ierr = -1
00111        return
00112     endif
00113     if(size(tin).ne.size(f)) then
00114        write(fun_err%cerr,*) "nuclear_data_1002: Expected the input and output vectors to be the same
       size in ’nuclear_data_1002’"
00115        fun_err%ierr = -1
00116        return
00117     endif
00118
00119     do it=1, size(tin)
00120        t=tin(it)*1e-3 ! temperature in keV
00121        ir=1
00122        if(t.lt.function_parameters(10,ir) .and. with_warnings) then
00123           write(*,*) ’extrapolating  below the desired range’
00124        endif
00125        do while(t.gt.function_parameters(11,ir).and. &
00126             ir.lt.size(function_parameters,2))
00127           ir=ir+1
00128        enddo
00129        if(t.gt.function_parameters(11,ir) .and. with_warnings) then
00130           write(*,*) ’extrapolating  above the desired range’
00131 ! dpc fix: changed 12 to 11
00132           write(*,*) ir, tin(it), function_parameters(11,ir)
00133        endif
00134
00135        bg  =function_parameters(1,   ir)
00136        mrc2=function_parameters(2,   ir)
00137        c   =function_parameters(3:9, ir)
00138        ! (/Tmin,Tmax/) = function_parameters(10:11, iR)
00139
00140        theta=t/(1-t*(c(2)+t*(c(4)+t*c(6)))&
00141            /(1+t*(c(3)+t*(c(5)+t*c(7)))))
00142        xi=(bg**2/(4*theta))**(1.0_ids_real/3)
00143        f(it) = c(1)*theta*sqrt(xi/(mrc2*t**3))&
00144            *exp(-3*xi)*1.e-6 ! from cm^3/s to m^3s/
00145     end do
00146
```

Here is the caller graph for this function:



13.1.2.3 nuclear_data_1006()

```
subroutine amns_external_functions::nuclear_data_1006 (
            real (ids_real), dimension(:,:), intent(in) function_parameters,
            real (ids_real), dimension(:), intent(in) x,
            real (ids_real), dimension(:), intent(out) f,
            logical, intent(in) with_warnings,
            type (fun_err_t), intent(out) fun_err )
```

AMNS External function ...
This provides T-T cross-section data developed by David Coster

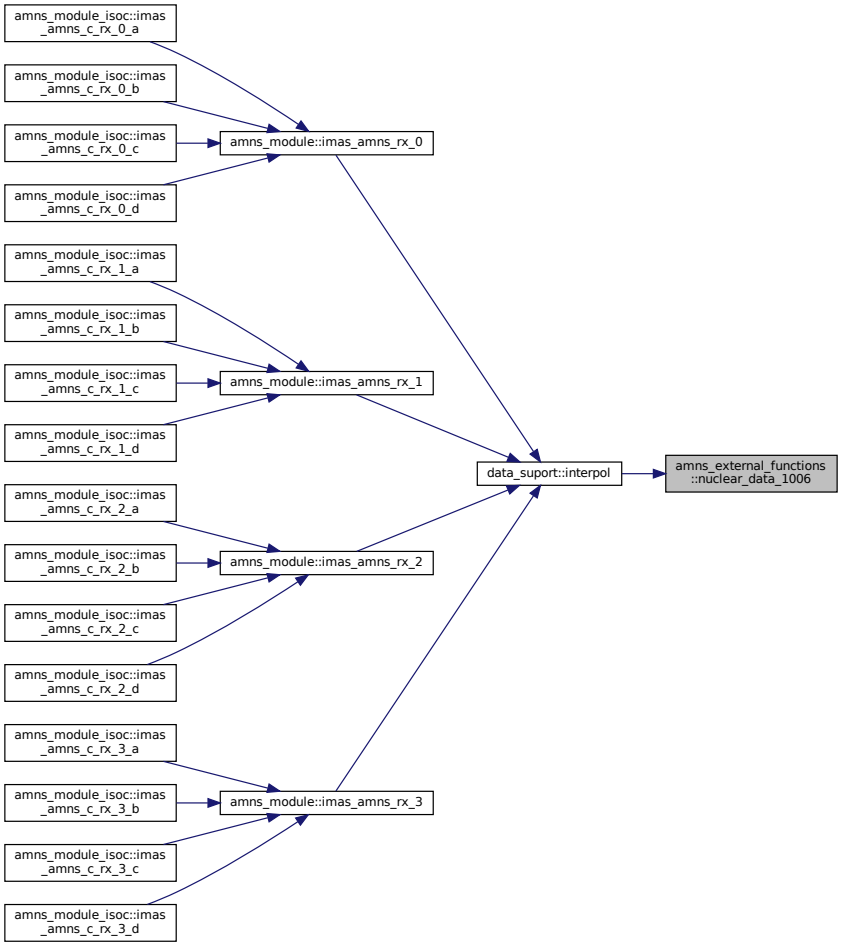
Todo provide more information

Definition at line 443 of file amns_external_functions.f90.

```
00444     implicit none
00445     real (ids_real),   intent(in)  :: function_parameters(:,:), x(:)
00446     real (ids_real),   intent(out) :: f(:)
00447     logical,           intent(in)  :: with_warnings
00448     type (fun_err_t),  intent(out) :: fun_err
00449     integer :: i, j
00450     real (ids_real) :: e, s
00451
00452     fun_err%ierr = 0
00453     fun_err%cerr = ”
```

```
00454
00455     if(size(function_parameters,1).ne.14) then
00456        write(fun_err%cerr,*) "nuclear_data_1006: Expected the length of 'function_parameters' to be 14
      in 'nuclear_data_1006' and not ", &
00457             size(function_parameters)
00458        fun_err%ierr = -1
00459        return
00460     endif
00461     if(size(x).ne.size(f)) then
00462        write(fun_err%cerr,*) "nuclear_data_1006: Expected the input and output vectors to be the same
      size in 'nuclear_data_1006'"
00463        fun_err%ierr = -1
00464        return
00465     endif
00466
00467     do i=1, size(x)
00468        e = x(i) * 1.0e-3_ids_real
00469        j=1
00470        if(e.lt.function_parameters(13,j) .and. with_warnings) then
00471           write(*,*) 'extrapolating  below the desired range'
00472        endif
00473        do while(e.gt.function_parameters(14,j).and. &
00474             j.lt.size(function_parameters,2))
00475           j=j+1
00476        enddo
00477        if(e.gt.function_parameters(14,j)) then
00478           if (with_warnings) then
00479              write(*,*) 'extrapolating  above the desired range'
00480              write(*,*) j, x(i), function_parameters(12,j)
00481              write(*,*) 'Taking the boundary value when calculating S!'
00482           endif
00483           e = function_parameters(14,j)
00484        endif
00485        s = ( function_parameters(2,j) &
00486             & + e*(function_parameters(3,j) &
00487             & + e*(function_parameters(4,j) &
00488             & + e*(function_parameters(5,j) &
00489             & + e*(function_parameters(6,j) &
00490             & + e* function_parameters(7,j) ) ) ) ) ) &
00491             & /( 1.0_ids_real &
00492             & + e*(function_parameters(8,j) &
00493             & + e*(function_parameters(9,j) &
00494             & + e*(function_parameters(10,j) &
00495             & + e*(function_parameters(11,j) &
00496             & + e* function_parameters(12,j) ) ) ) ) )
00497        ! We clamp the energy for S (the nuclear fusion probability).
00498        ! However, there is no need to clamp the energy in denominator.
00499        ! The denominator roughly describes the probability of the reactants to
00500        ! overcome the Coulomb barrier.
00501        e = x(i) * 1.0e-3_ids_real
00502        f(i) = s / (e * exp(function_parameters(1,j)/sqrt(e))) * 1.0e-31_ids_real
00503     enddo
```

Here is the caller graph for this function:



13.1.2.4 rct_data_1003()

```
subroutine amns_external_functions::rct_data_1003 (
            real (ids_real), dimension(:), intent(in) function_parameters,
            real (ids_real), dimension(:), intent(in) Tin,
            real (ids_real), dimension(:), intent(out) f,
            logical, intent(in) with_warnings,
            type (fun_err_t), intent(out) fun_err )
```

AMNS External function ...

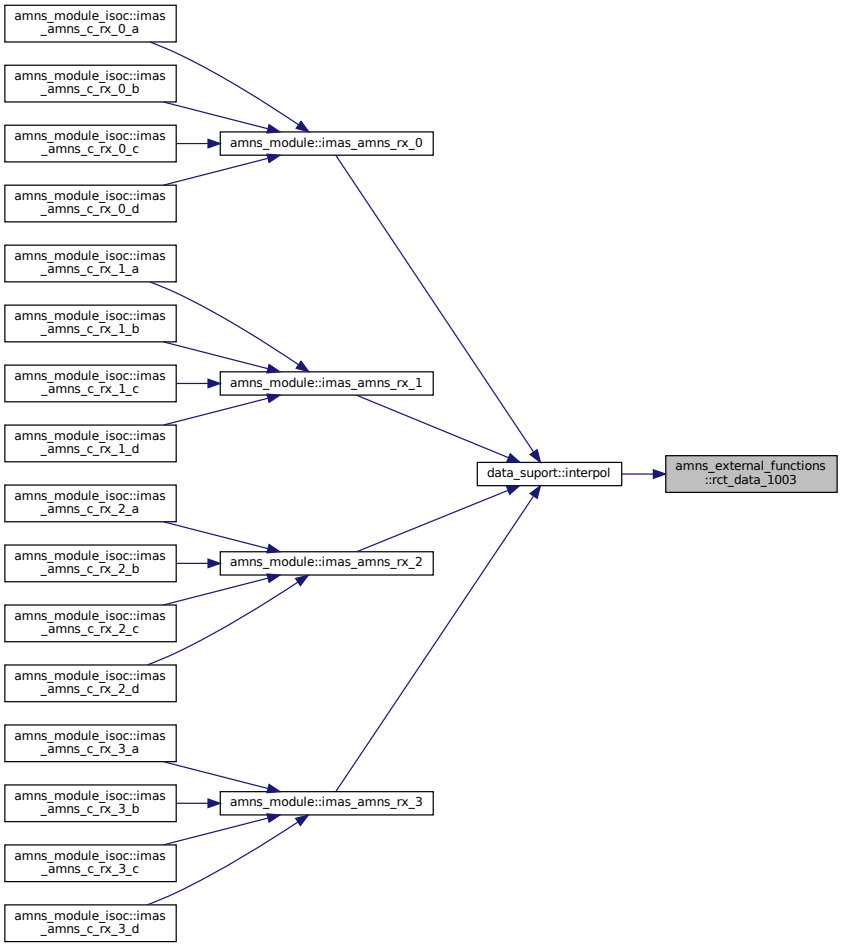
See S.A. Maiorov, O.F. Petrov, V.E. Fortov: "Calculation of resonant charge exchange cross-sections of ions Rubidium, Cesium, Mercury and noble gases", EPS 2007, http://epsppd.epfl.ch/Warsaw/pdf/P2_↩115.pdf

Definition at line 156 of file amns_external_functions.f90.

```
00157     implicit none
00158     real (ids_real), intent(in)  :: function_parameters(:), Tin(:)
00159     real (ids_real), intent(out) :: f(:)
00160     logical,         intent(in)  :: with_warnings
00161     type (fun_err_t), intent(out) :: fun_err
00162     integer                :: iT
00163
00164     fun_err%ierr = 0
00165     fun_err%cerr = ”
00166
00167     if(size(function_parameters,1).ne.2) then
```

```
00168         write(fun_err%cerr,*) "rct_data_1003: Expected the length of 'function_parameters' to be 2 in
      'rct_data_1003' and not ", &
00169              size(function_parameters)
00170         fun_err%ierr = -1
00171         return
00172       endif
00173       if(size(tin).ne.size(f)) then
00174         write(fun_err%cerr,*) "rct_data_1003: Expected the input and output vectors to be the same size
      in 'rct_data_1003'"
00175         fun_err%ierr = -1
00176         return
00177       endif
00178
00179       do it=1, size(tin)
00180         f(it) = function_parameters(1)*(1.0_ids_real + function_parameters(2)*
      log(1.0_ids_real/tin(it)))**2*1e-20_ids_real
00181       end do
00182
```

Here is the caller graph for this function:



13.1.2.5 reflect_data_1005()

```
subroutine amns_external_functions::reflect_data_1005 (
            real (ids_real), dimension(:,:), intent(in) function_parameters,
            real (ids_real), dimension(:), intent(in) energy_arr,
            real (ids_real), dimension(:), intent(in) angle_arr,
            real (ids_real), dimension(:), intent(out) refl_arr,
            logical, intent(in) with_warnings,
```

```
            type (fun_err_t), intent(out) fun_err )
```

AMNS External function ...
this is derived from the routine reflyield from K. Schmid
See ...

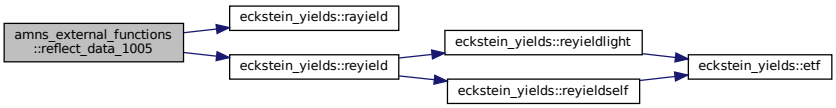
Todo provide more information

Definition at line 296 of file amns_external_functions.f90.

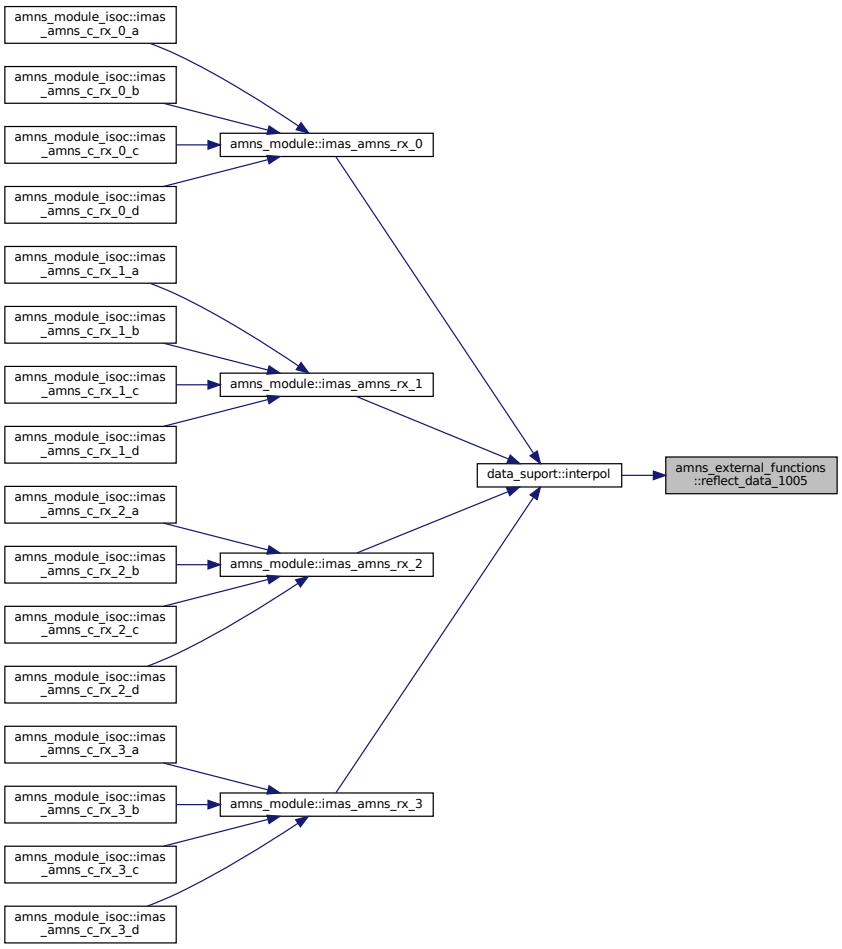
```
00297     use eckstein_yields
00298     implicit none
00299     real (ids_real), intent(in)  :: function_parameters(:,:), energy_arr(:), angle_arr(:)
00300     real (ids_real), intent(out) :: refl_arr(:)
00301     logical,           intent(in)  :: with_warnings
00302     type (fun_err_t),  intent(out) :: fun_err
00303
00304     real (ids_real)              :: energy, angle, refl
00305     real(ids_real)               :: M1, M2, Z1, Z2, A1, A2, A3, A4, E1, ESP, ANGLEE0, ANGLEE1, C1, C2,
      C3, C4
00306     real(ids_real)               :: C1FROM, C2FROM, C3FROM, C4FROM
00307     real(ids_real)               :: C1TO, C2TO, C3TO, C4TO
00308     real(ids_real)               :: matchanglee0, matchanglee1
00309     integer                 :: numtab, ianglee0, numpars, boundianglee0, boundianglee1, ipars
00310     integer                 :: iD, iA
00311     logical, parameter      :: debug=.true.
00312
00313     integer                 :: have_angle_data
00314
00315     fun_err%ierr = 0
00316     fun_err%cerr = ''
00317
00318     numtab = ubound(function_parameters, 2)
00319     do id = lbound(energy_arr,1), ubound(energy_arr,1)
00320
00321        energy = energy_arr(id)
00322        angle  = angle_arr(id)
00323
00324        matchanglee0 = 0.0
00325        boundianglee0 = -1
00326        boundianglee1 = -1
00327        have_angle_data = -1
00328        do ia = lbound(function_parameters, 2), ubound(function_parameters, 2)
00329           matchanglee1 = function_parameters(11, ia)
00330           if(debug .and. with_warnings) then
00331              write(*,*) matchanglee0, energy, matchanglee1
00332           endif
00333           if((matchanglee0.le.energy).and.(matchanglee1.ge.energy)) then
00334              if (ia .gt. 1) then
00335                 if(ia .lt. numtab) then
00336                    boundianglee0 = ia - 1
00337                    boundianglee1 = ia
00338                 else
00339                    boundianglee0 = numtab
00340                    boundianglee1 = numtab
00341                 end if
00342              else
00343                 boundianglee0 = 1
00344                 boundianglee1 = 1
00345              end if
00346              have_angle_data = 1
00347              exit
00348           end if
00349           if (matchanglee1 < matchanglee0)  then
00350              if (matchanglee1 .lt. 0) then
00351                 if(debug .and. with_warnings) then
00352                    write(*,*) 'WARNING::reflyield-> no angular reflection yield data available,
      returning value for perpendicular impact'
00353                 endif
00354                 have_angle_data = -1
00355                 exit
00356              else
00357                 write(fun_err%cerr,*) 'reflect_data_1005: reflyield-> angular dependence energies are
      not sorted, rebuild the database'
00358                 fun_err%ierr = -1
00359                 return
00360              end if
00361           endif
00362           matchanglee0 = matchanglee1
00363        end do
00364
00365        if((boundianglee0.lt.0).or.(boundianglee1.lt.0)) then
00366           if (matchanglee1.lt.energy) then
00367              boundianglee0 = numtab
00368              boundianglee1 = numtab
00369           else
00370              boundianglee0 = 1
```

```
00371             boundianglee1 = 1
00372          end if
00373          if (debug .and. with_warnings) then
00374             write(*,*) 'reflect_data_1005: No Bound found for: ', energy, ' angle: ', angle, 'using
       boundianglee0 = ', boundianglee0, 'and boundianglee1 = ',boundianglee1
00375          end if
00376       end if
00377
00378       z1  = function_parameters( 1, boundianglee0)
00379       z2  = function_parameters( 2, boundianglee0)
00380       m1  = function_parameters( 3, boundianglee0)
00381       m2  = function_parameters( 4, boundianglee0)
00382       a1  = function_parameters( 5, boundianglee0)
00383       a2  = function_parameters( 6, boundianglee0)
00384       a3  = function_parameters( 7, boundianglee0)
00385       a4  = function_parameters( 8, boundianglee0)
00386       e1  = function_parameters( 9, boundianglee0)
00387       esp = function_parameters(10, boundianglee0)
00388
00389       if ((angle.lt.1.0).or.(have_angle_data.eq.-1)) then
00390          if (debug .and. with_warnings) then
00391             write(*,*) 'angle = ',angle, ' is small or no angular data available: using perpendicular
       impact model with energy dependence'
00392          end if
00393          refl = reyield(energy, m1, m2, z1, z2, a1, a2, a3, a4)
00394       else
00395          if (debug .and. with_warnings) then
00396             write(*,*) 'angle = ',angle, ' is large using angular dependent model and interpolation
       of energy dependence'
00397          end if
00398          ! Interplolate the bounding table entries
00399          if(boundianglee0.lt.boundianglee1) then
00400             anglee0 = function_parameters(11, boundianglee0)
00401             anglee1 = function_parameters(11, boundianglee1)
00402             if(debug .and. with_warnings) then
00403                write(*,*) 'Interpolating angular dependent reflection yield for energy' ,energy,'
       between values for energies ', anglee0, ' and ', anglee1
00404             end if
00405             c1from = function_parameters(12, boundianglee0)
00406             c2from = function_parameters(13, boundianglee0)
00407             c3from = function_parameters(14, boundianglee0)
00408             c4from = function_parameters(15, boundianglee0)
00409
00410             c1to = function_parameters(12, boundianglee1)
00411             c2to = function_parameters(13, boundianglee1)
00412             c3to = function_parameters(14, boundianglee1)
00413             c4to = function_parameters(15, boundianglee1)
00414
00415             c1 = c1from + ((energy - anglee0) * (c1to - c1from) / (anglee1 - anglee0))
00416             c2 = c2from + ((energy - anglee0) * (c2to - c2from) / (anglee1 - anglee0))
00417             c3 = c3from + ((energy - anglee0) * (c3to - c3from) / (anglee1 - anglee0))
00418             c4 = c4from + ((energy - anglee0) * (c4to - c4from) / (anglee1 - anglee0))
00419          else
00420             anglee0 = function_parameters(11, boundianglee0)
00421             if(debug .and. with_warnings) then
00422                write(*,*) 'Energy: ',energy,'  for requested angle: ',angle,' not within db bounds
       using anglular dependent value at energy = ', anglee0
00423             end if
00424             c1 = function_parameters(12, boundianglee0)
00425             c2 = function_parameters(13, boundianglee0)
00426             c3 = function_parameters(14, boundianglee0)
00427             c4 = function_parameters(15, boundianglee0)
00428          end if
00429          refl = rayield(angle, c1, c2, c3, c4)
00430       end if
00431
00432       refl_arr(id) = min(1.0_ids_real, refl)
00433
00434    enddo
00435
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.1.2.6 sputter_data_1004()

```
subroutine amns_external_functions::sputter_data_1004 (
            real (ids_real), dimension(:,:), intent(in) function_parameters,
            real (ids_real), dimension(:), intent(in) energy_arr,
            real (ids_real), dimension(:), intent(in) angle_arr,
            real (ids_real), dimension(:), intent(out) yield_arr,
            logical, intent(in) with_warnings,
            type (fun_err_t), intent(out) fun_err )
```

AMNS External function ...
this is derived from the routine sputteryield from K. Schmid
See ...

Todo provide more information

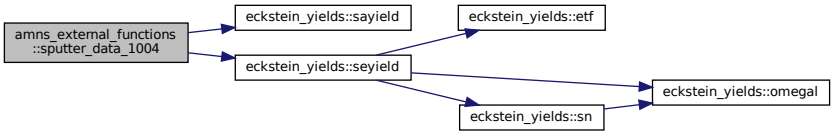
Definition at line 192 of file amns_external_functions.f90.

```
00193     use eckstein_yields
00194     implicit none
00195     real (ids_real), intent(in)  :: function_parameters(:,:), energy_arr(:), angle_arr(:)
00196     real (ids_real), intent(out) :: yield_arr(:)
00197     logical,           intent(in)  :: with_warnings
00198     type (fun_err_t),  intent(out) :: fun_err
00199
```

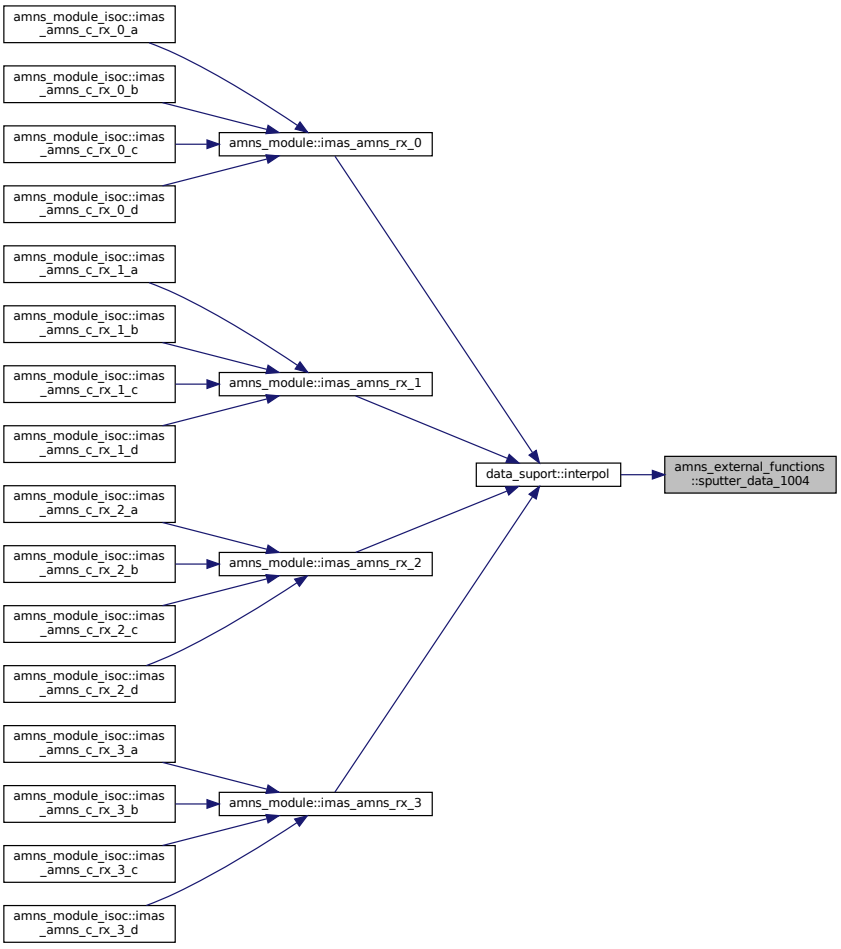
```
00200     real (ids_real)              :: energy, angle, yield
00201     real(ids_real)               :: matchanglee0, matchanglee1
00202     real(ids_real)               :: M1, M2, Z1, Z2, q, lambda, mu, ETh, f,b,c,esp
00203     integer                 :: numtab, ianglee0, numpars, boundianglee0, ipars
00204     integer                 :: iD, iA
00205     logical, parameter      :: debug=.true.
00206
00207     integer                 :: have_angle_data
00208
00209     fun_err%ierr = 0
00210     fun_err%cerr = ''
00211
00212     numtab = ubound(function_parameters, 2)
00213     if(debug .and. with_warnings) then
00214        write(*,*) numtab
00215     endif
00216     do id = lbound(energy_arr,1), ubound(energy_arr,1)
00217
00218        energy = energy_arr(id)
00219        angle  = angle_arr(id)
00220
00221        if(debug .and. with_warnings) then
00222           write(*,*) energy, angle
00223        endif
00224
00225        matchanglee0 = 0.0
00226        boundianglee0 = -1
00227        have_angle_data = -1
00228        do ia = lbound(function_parameters, 2), ubound(function_parameters, 2)
00229           matchanglee1 = function_parameters(3, ia)
00230           if(debug .and. with_warnings) then
00231              write(*,*) matchanglee0, energy, matchanglee1
00232           endif
00233           if((matchanglee0.le.energy).and.(matchanglee1.ge.energy)) then
00234              boundianglee0 = ia
00235              have_angle_data = 1
00236              exit
00237           end if
00238           if (matchanglee1 < matchanglee0)  then
00239              if (matchanglee1 < matchanglee0)  then
00240                 if(debug .and. with_warnings) then
00241                    write(*, *) 'WARNING::sputteryield-> no angular reflection yield data available,
    returning value for perpendicular impact'
00242                 endif
00243                 have_angle_data = -1
00244                 exit
00245              else
00246                 write(fun_err%cerr,*) 'sputter_data_1004: sputteryield-> angular dependence energies
    are not sorted, rebuild the database'
00247                 fun_err%ierr = -1
00248                 return
00249              endif
00250           end if
00251           matchanglee0 = matchanglee1
00252        end do
00253
00254        if(boundianglee0.lt.0) then
00255           if (debug .and. with_warnings) then
00256              write(*,*) 'sputter_data_1004: No Bound found for: ', energy, ' angle: ', angle, 'using
    boundianglee0 = ', numtab
00257           end if
00258           boundianglee0 = ubound(function_parameters, 2)
00259        end if
00260
00261        m1     = function_parameters( 4, boundianglee0)
00262        m2     = function_parameters( 5, boundianglee0)
00263        z1     = function_parameters( 1, boundianglee0)
00264        z2     = function_parameters( 2, boundianglee0)
00265        q      = function_parameters( 6, boundianglee0)
00266        lambda = function_parameters( 9, boundianglee0)
00267        eth    = function_parameters( 8, boundianglee0)
00268        mu     = function_parameters( 7, boundianglee0)
00269        f      = function_parameters(10, boundianglee0)
00270        b      = function_parameters(11, boundianglee0)
00271        c      = function_parameters(12, boundianglee0)
00272        esp    = function_parameters(13, boundianglee0)
00273
00274        yield = 0.0
00275        if (energy .gt. eth) then
00276           if (have_angle_data .gt. 0) then
00277              yield = seyield(energy, m1, m2, z1, z2, q, lambda, mu, eth) * sayield(energy, angle, f,
    b, c, esp)
00278           else
00279              yield = seyield(energy, m1, m2, z1, z2, q, lambda, mu, eth)
00280           endif
00281        end if
00282
```

```
00283         yield_arr(id) = yield
00284
00285       enddo
00286
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.2 AMNS Callable Fortran Subroutines

Module implementing the IMAS AMNS interface.

Data Types

- interface amns_module::imas_amns_rx

  get the rates associated with the input args for a particular reaction (generic interface for 1d, 2d & 3d arrays)

Functions/Subroutines

- subroutine amns_module::imas_amns_setup (handle, version, error_status)

  initialization call for the AMNS package
- subroutine amns_module::imas_amns_setup_table (handle, reaction_type, reactants, handle_rx, error_↩ status)

  initialization call for a particular reaction
- subroutine amns_module::imas_amns_finish (handle, error_status)

  finalization call for the AMNS package
- subroutine amns_module::imas_amns_finish_table (handle_rx, error_status)

  finalization call for a particular reaction
- subroutine amns_module::imas_amns_query (handle, query, answer, error_status)

  query routine for the AMNS package
- subroutine amns_module::imas_amns_query_table (handle_rx, query, answer, error_status)

  query routine for a particular reaction
- subroutine amns_module::imas_amns_set (handle, set, error_status)

  set a parameter for the AMNS package
- subroutine amns_module::imas_amns_set_table (handle_rx, set, error_status)

  set a parameter for a particular reaction

13.2.1 Detailed Description

Module implementing the IMAS AMNS interface.

Author

David Coster, updated David Tskhakaya (small updates in: IMAS_AMNS_SET_TABLE, IMAS_AMNS_FIN↩ ISH)

13.2.2 Function/Subroutine Documentation

13.2.2.1 imas_amns_finish()

```
subroutine amns_module::imas_amns_finish (
            type(amns_handle_type), intent(inout) handle,
            type(amns_error_type), intent(out), optional error_status )
```

finalization call for the AMNS package

Parameters

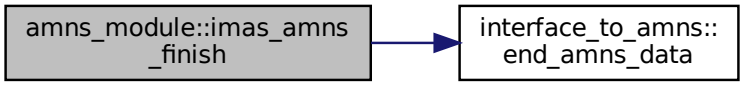
in	*handle*	Opaque handle from call to IMAS_AMNS_SETUP
out	*error_status*	If an error occurs, this will be set (optional)

Definition at line 383 of file amns_module.f90.

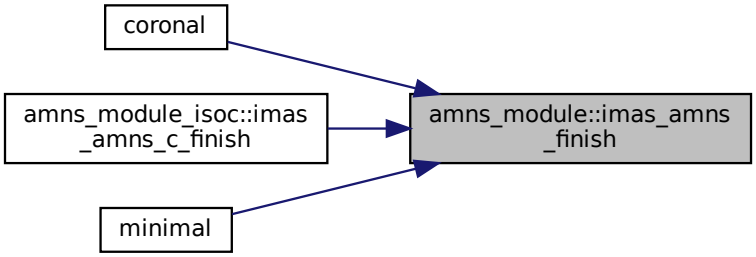
```
00384     use ids_types  ! IGNORE
00385     use amns_types
```

```
00386     use amns_utility
00387     use interface_to_amns
00388 !    use deallocate_structures ! IGNORE
00389     implicit none
00390     optional error_status
00391     type(amns_handle_type), intent(inout) :: handle
00392     type(amns_error_type), intent(out) :: error_status
00393
00394     if(present(error_status)) then
00395         error_status%flag=.false.
00396         error_status%string="No error"
00397     end if
00398
00399     if(handle%debug) write(*,*) 'IMAS_AMNS_FINISH start'
00400     if(handle%initialized) then
00401        handle%initialized=.false.
00402     else
00403        if(handle%debug) write(*,*) 'IMAS_AMNS_FINISH: Attempt to FINISH an uninitialized case'
00404     endif
00405     if(present(error_status)) then
00406        if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE: requested error_status response'
00407     endif
00408     call end_amns_data
00409     call ids_deallocate(amns00)
00410     if(handle%debug) write(*,*) 'IMAS_AMNS_FINISH end'
00411
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.2.2.2 imas_amns_finish_table()

```
subroutine amns_module::imas_amns_finish_table (
             type(amns_handle_rx_type), intent(inout) handle_rx,
             type(amns_error_type), intent(out), optional error_status )
```

finalization call for a particular reaction

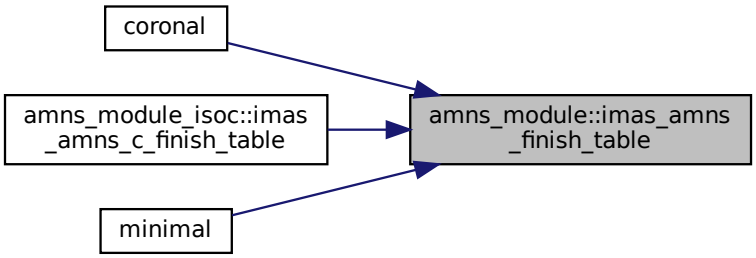
Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*error_status*	If an error occurs, this will be set (optional)

Definition at line 419 of file amns_module.f90.

```
00420     use ids_types  ! IGNORE
00421     use amns_types
00422     implicit none
00423     optional error_status
00424     type(amns_handle_rx_type), intent(inout) :: handle_rx
00425     type(amns_error_type), intent(out) :: error_status
00426
00427     if(present(error_status)) then
00428         error_status%flag=.false.
00429         error_status%string="No error"
00430     end if
00431
00432     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_FINISH_TABLE start'
00433     if(handle_rx%initialized) then
00434        handle_rx%initialized=.false.
00435        if(allocated(handle_rx%components)) then
00436           deallocate(handle_rx%components)
00437        endif
00438        handle_rx%no_of_reactants=0
00439        if(allocated(handle_rx%string)) then
00440           deallocate(handle_rx%string)
00441        endif
00442        call delete(handle_rx%grid)
00443     else
00444        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_FINISH_TABLE: Attempt to FINISH an uninitialized
      table'
00445     endif
00446     if(present(error_status)) then
00447        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_FINISH_TABLE: requested error_status response'
00448     endif
00449     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_FINISH_TABLE end'
00450
```

Here is the caller graph for this function:



13.2.2.3 imas_amns_query()

```
subroutine amns_module::imas_amns_query (
             type(amns_handle_type), intent(in) handle,
             type(amns_query_type), intent(in) query,
             type(amns_answer_type), intent(out) answer,
             type(amns_error_type), intent(out), optional error_status )
```

query routine for the AMNS package

Parameters

in	*handle*	Opaque handle from call to IMAS_AMNS_SETUP

Parameters

in	*query*	Specifies the query
out	*answer*	Returns the answer
out	*error_status*	If an error occurs, this will be set (optional)

Remarks

Possible queries are:

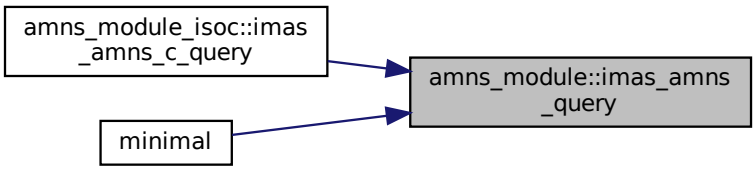
- **version** : information about the version
- **prop_comment** : information about properties_comment
- **prop_source** : information about properties_source
- **prop_provider** : information about properties_provider
- **prop_creation** : information about properties_creation_date
- **code_name** : information about code_name
- **code_commit** : information about code_commit
- **code_version** : information about code_version
- **code_repository** : information about code_repository

Definition at line 470 of file amns_module.f90.

```
00471     use ids_types  ! IGNORE
00472     use amns_types
00473     implicit none
00474     optional error_status
00475     type(amns_handle_type), intent(in) :: handle
00476     type(amns_query_type), intent(in) :: query
00477     type(amns_answer_type), intent(out) :: answer
00478     type(amns_error_type), intent(out) :: error_status
00479
00480     if(present(error_status)) then
00481         error_status%flag=.false.
00482         error_status%string="No error"
00483     end if
00484
00485     if(handle%debug) write(*,*) 'IMAS_AMNS_QUERY start'
00486     if(handle%debug) write(*,*) 'IMAS_AMNS_QUERY: query = ',trim(query%string)
00487 ! The choices are documented in the doxygen block above
00488     select case (query%string)
00489     case ("version")                ! \property version: information about the version
00490        answer%string=handle%version%string
00491        answer%number=handle%version%number
00492     case ("prop_comment")               ! \property properties_comment: information about the version
00493        answer%string=handle%properties_comment
00494        answer%number=0
00495     case ("prop_source")               ! \property properties_source: information about the version
00496        answer%string=handle%properties_source
00497        answer%number=0
00498     case ("prop_provider")               ! \property properties_provider: information about the
    version
00499        answer%string=handle%properties_provider
00500        answer%number=0
00501     case ("prop_creation")               ! \property properties_creation_date: information about the
    version
00502        answer%string=handle%properties_creation_date
00503        answer%number=0
00504     case ("code_name")               ! \property code_name: information about the version
00505        answer%string=handle%code_name
00506        answer%number=0
00507     case ("code_commit")               ! \property code_commit: information about the version
00508        answer%string=handle%code_commit
00509        answer%number=0
00510     case ("code_version")               ! \property code_version: information about the version
00511        answer%string=handle%code_version
00512        answer%number=0
00513     case ("code_repository")               ! \property code_repository: information about the version
00514        answer%string=handle%code_repository
00515        answer%number=0
00516     case default
00517        answer%string='Not implemented yet'
00518     end select
00519     if(present(error_status)) then
00520        if(handle%debug) write(*,*) 'IMAS_AMNS_QUERY: requested error_status response'
00521     endif
```

```
00522     if(handle%debug) write(*,*) 'IMAS_AMNS_QUERY end'
00523
```

Here is the caller graph for this function:



13.2.2.4 imas_amns_query_table()

```
subroutine amns_module::imas_amns_query_table (
             type(amns_handle_rx_type), intent(in) handle_rx,
             type(amns_query_type), intent(in) query,
             type(amns_answer_type), intent(out) answer,
             type(amns_error_type), intent(out), optional error_status )
```

query routine for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
in	*query*	Specifies the query
out	*answer*	Returns the answer
out	*error_status*	If an error occurs, this will be set (optional)

Remarks

Possible queries are:

- **source** : information about the data source
- **provider** : information about the data provider
- **citation** : information about the data citation
- **no_of_reactants** : number of reactants and products
- **index** :
- **filled** : whether AMNS data exists
- **reaction_type** : type of the reaction (value originally passed to IMAS_AMNS_SETUP_TABLE in reaction_typestring)
- **reactants** : information about the reactants and products
- **coordinates** : information about the coordinates
- **version** : information about the version
- **state_label** : label of the charge state
- **result_unit** : units of the result
- **result_label** : label of the result
- **ndim** : dimensionality of the requested data

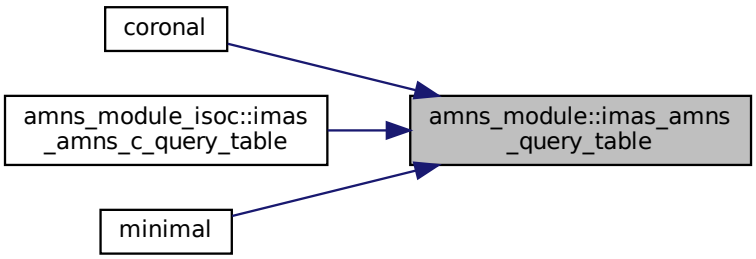
- **interp_fun** : number of the interpolating function
- **prop_comment** : information about properties_comment
- **prop_source** : information about properties_source
- **prop_provider** : information about properties_provider
- **prop_creation** : information about properties_creation_date
- **code_name** : information about code_name
- **code_commit** : information about code_commit
- **code_version** : information about code_version
- **code_repository** : information about code_repository

Definition at line 557 of file amns_module.f90.

```
00558     use ids_types  ! IGNORE
00559     use amns_types
00560     use amns_utility
00561     implicit none
00562     optional error_status
00563     type(amns_handle_rx_type), intent(in) :: handle_rx
00564     type(amns_query_type), intent(in) :: query
00565     type(amns_answer_type), intent(out) :: answer
00566     type(amns_error_type), intent(out) :: error_status
00567     integer ir
00568
00569     if(present(error_status)) then
00570         error_status%flag=.false.
00571         error_status%string="No error"
00572     end if
00573
00574     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_QUERY_TABLE start'
00575     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_QUERY_TABLE: query = ',trim(query%string)
00576 ! The choices are documented in the doxygen block above
00577     select case (query%string)
00578     case ("source")
00579        answer%string=handle_rx%source
00580        answer%number=0
00581     case ("provider")
00582        answer%string=handle_rx%provider
00583        answer%number=0
00584     case ("citation")
00585        answer%string=handle_rx%citation
00586        answer%number=0
00587     case ("no_of_reactants")
00588        answer%string=string(handle_rx%no_of_reactants)
00589        answer%number=handle_rx%no_of_reactants
00590     case ("index")
00591        answer%string=string(handle_rx%index)
00592        answer%number=handle_rx%index
00593     case ("filled")
00594        if(handle_rx%filled) then
00595           answer%string= 'Filled'
00596        else
00597           answer%string= 'Empty'
00598        endif
00599     case ("reaction_type")
00600        answer%string= handle_rx%reaction_type
00601     case ("reactants")
00602        answer%string=""
00603        do ir=1,handle_rx%no_of_reactants
00604           answer%string= trim(answer%string) // " " // &
00605                trim(string(nint(handle_rx%components(ir)%ZN))) // "/" // &
00606                trim(string(nint(handle_rx%components(ir)%ZA))) // "/" // &
00607                trim(string(nint(handle_rx%components(ir)%MI))) // "/" // &
00608                trim(string(handle_rx%components(ir)%LR))
00609        enddo
00610        answer%string= adjustl(answer%string)
00611     case ("coordinates")
00612        answer%string=""
00613        do ir=1,handle_rx%grid%ndim
00614           answer%string= trim(answer%string) // " " // &
00615                trim(handle_rx%grid%axe(ir)%label) // "/" // &
00616                trim(handle_rx%grid%axe(ir)%units)
00617        enddo
00618        answer%string= adjustl(answer%string)
00619     case ("version")
00620        answer%string=""
00621        answer%number=handle_rx%version%number
00622     case ("state_label")
00623        answer%string=handle_rx%grid%state_label
00624        answer%number=-1
00625     case ("result_unit")
```

```
00626        answer%string=handle_rx%grid%result_unit
00627        answer%number=-1
00628     case ("result_label")
00629        answer%string=handle_rx%grid%result_label
00630        answer%number=-1
00631     case ("ndim")
00632        answer%string=string(handle_rx%grid%ndim)
00633        answer%number=handle_rx%grid%ndim
00634     case ("interp_fun")
00635        answer%string=string(handle_rx%grid%interpol_function)
00636        answer%number=handle_rx%grid%ndim
00637     case ("prop_comment")                ! \property properties_comment: information about the version
00638        answer%string=handle_rx%properties_comment
00639        answer%number=0
00640     case ("prop_source")                ! \property properties_source: information about the version
00641        answer%string=handle_rx%properties_source
00642        answer%number=0
00643     case ("prop_provider")                ! \property properties_provider: information about the
       version
00644        answer%string=handle_rx%properties_provider
00645        answer%number=0
00646     case ("prop_creation")                ! \property properties_creation_date: information about the
       version
00647        answer%string=handle_rx%properties_creation_date
00648        answer%number=0
00649     case ("code_name")                ! \property code_name: information about the version
00650        answer%string=handle_rx%code_name
00651        answer%number=0
00652     case ("code_commit")                ! \property code_commit: information about the version
00653        answer%string=handle_rx%code_commit
00654        answer%number=0
00655     case ("code_version")                ! \property code_version: information about the version
00656        answer%string=handle_rx%code_version
00657        answer%number=0
00658     case ("code_repository")                ! \property code_repository: information about the version
00659        answer%string=handle_rx%code_repository
00660        answer%number=0
00661     case default
00662        answer%string='Not implemented yet'
00663     end select
00664     if(present(error_status)) then
00665        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_QUERY_TABLE: requested error_status response'
00666     endif
00667     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_QUERY_TABLE end'
00668
```

Here is the caller graph for this function:



13.2.2.5 imas_amns_set()

```
subroutine amns_module::imas_amns_set (
             type(amns_handle_type), intent(inout) handle,
             type(amns_set_type), intent(in) set,
             type(amns_error_type), intent(out), optional error_status )
```

set a parameter for the AMNS package

Parameters

in	*handle*	Opaque handle from call to IMAS_AMNS_SETUP
in	*set*	Specifies the setting to be applied
out	*error_status*	If an error occurs, this will be set (optional)

Remarks

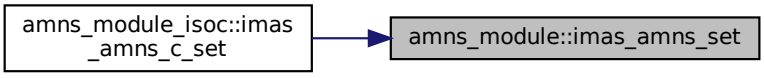
Possible settings are:

- **debug** : enable debugging
- **nodebug** : disable debugging
- **backend=mdsplus** : use the MDSplus backend
- **backend=hdf5** : use the HDF5 backend
- **backend=ascii** : use the ASCII backend

Definition at line 683 of file amns_module.f90.

```
00684     use ids_types  ! IGNORE
00685     use amns_types
00686     implicit none
00687     optional error_status
00688     type(amns_handle_type), intent(inout) :: handle
00689     type(amns_set_type), intent(in) :: set
00690     type(amns_error_type), intent(out) :: error_status
00691
00692     if(present(error_status)) then
00693         error_status%flag=.false.
00694         error_status%string="No error"
00695     end if
00696
00697     if(handle%debug) write(*,*) 'IMAS_AMNS_SET start'
00698     if(handle%debug) write(*,*) 'IMAS_AMNS_SET: set = ',trim(set%string)
00699     if(present(error_status)) then
00700        if(handle%debug) write(*,*) 'IMAS_AMNS_SET_TABLE: requested error_status response'
00701     endif
00702 ! The choices are documented in the doxygen block above
00703     select case(set%string)
00704     case ("debug")
00705        handle%debug=.true.
00706     case ("nodebug")
00707        handle%debug=.false.
00708     case ("backend=mdsplus")
00709        handle%version%backend='mdsplus'
00710     case ("backend=hdf5")
00711        handle%version%backend='hdf5'
00712     case ("backend=ascii")
00713        handle%version%backend='ascii'
00714     case default
00715        write(*,*) "IMAS_AMNS_SET: not yet implemented ", trim(set%string)
00716     end select
00717     if(handle%debug) write(*,*) 'IMAS_AMNS_SET end'
00718
```

Here is the caller graph for this function:



13.2.2.6 imas_amns_set_table()

```
subroutine amns_module::imas_amns_set_table (
             type(amns_handle_rx_type), intent(inout) handle_rx,
```

```
type(amns_set_type), intent(in) set,
type(amns_error_type), intent(out), optional error_status )
```

set a parameter for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
in	*set*	Specifies the setting to be applied
out	*error_status*	If an error occurs, this will be set (optional)

Remarks

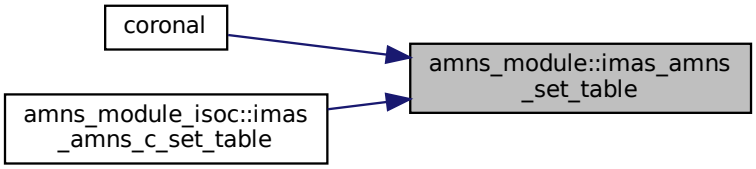
Possible settings are:

- **warn** : request warning messages
- **nowarn** : disable warnings
- **debug** : enable debuggibg
- **nodebug** : disable debugging

Definition at line 732 of file amns_module.f90.

```
00733     use ids_types  ! IGNORE
00734     use amns_types
00735     implicit none
00736     optional error_status
00737     type(amns_handle_rx_type), intent(inout) :: handle_rx
00738     type(amns_set_type), intent(in) :: set
00739     type(amns_error_type), intent(out) :: error_status
00740
00741     if(present(error_status)) then
00742         error_status%flag=.false.
00743         error_status%string="No error"
00744     end if
00745
00746     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SET_TABLE start'
00747     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SET_TABLE: set = ',trim(set%string)
00748 ! The choices are documented in the doxygen block above
00749     select case (set%string)
00750     case ("warn")
00751        if(handle_rx%filled) then
00752           call set_option(handle_rx%grid,.true.)
00753        else
00754           write(*,*) 'IMAS_AMNS_SET_TABLE: Attempt to set WARN using an unfilled table'
00755        endif
00756     case ("nowarn")
00757        if(handle_rx%filled) then
00758           call set_option(handle_rx%grid,.false.)
00759        else
00760           write(*,*) 'IMAS_AMNS_SET_TABLE: Attempt to set NOWARN using an unfilled table'
00761        endif
00762     case ("debug")
00763        handle_rx%debug=.true.
00764     case ("nodebug")
00765        handle_rx%debug=.false.
00766     case default
00767        write(*,*) 'IMAS_AMNS_SET_TABLE: not implemeted yet '
00768     end select
00769     if(present(error_status)) then
00770        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SET_TABLE: requested error_status response'
00771     endif
00772     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SET_TABLE end'
00773
```

Here is the caller graph for this function:



13.2.2.7 imas_amns_setup()

```
subroutine amns_module::imas_amns_setup (
            type(amns_handle_type), intent(out) handle,
            type(amns_version_type), intent(in), optional version,
            type(amns_error_type), intent(out), optional error_status )
```

initialization call for the AMNS package

Parameters

out	*handle*	Opaque handle to be used for further calls
in	*version*	Requested version (optional)
out	*error_status*	If an error occurs, this will be set (optional)

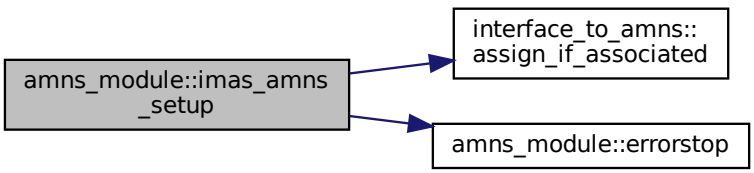
Definition at line 53 of file amns_module.f90.

```
00054     use ids_types  ! IGNORE
00055 !    use amns_types
00056     use amns_utility
00057     use ids_schemas   ! IGNORE
00058     use ids_routines  ! IGNORE
00059     use interface_to_amns
00060 !    use read_structures ! IGNORE
00061     implicit none
00062     optional version, error_status
00063     type(amns_handle_type), intent(out) :: handle
00064     type(amns_version_type), intent(in) :: version
00065     type(amns_error_type), intent(out)  :: error_status
00066
00067     type(amns_version_type) :: default_version
00068
00069 #ifdef AL
00070 !NLL    integer :: imas_open_env, imas_open_hdf5, imas_create_env, imas_create_hdf5
00071     integer :: imas_status
00072 #endif
00073     integer :: idx
00074     integer :: shot, run
00075     character (len=256) :: file
00076     character(len=3)    :: treename = 'ids'
00077     character(len=9)    :: amnspath = 'amns_data'
00078     character(STRMAXLEN) :: uri
00079     character(len=32) :: amns_debug_env
00080     logical :: amns_debug
00081
00082     write(*,'(a)') 'AMNS Library code version "' // trim(git_version_amns) // '"'
00083
00084     if(present(error_status)) then
00085         error_status%flag=.false.
00086         error_status%string="No error"
00087     end if
00088
00089     call getenv('IMAS_AMNS_DEBUG', amns_debug_env)
00090     amns_debug = amns_debug_env.eq."yes" .or. amns_debug_env.eq."YES" .or. amns_debug_env.eq."Yes"
00091     if (amns_debug) handle%debug = .true.
00092     amns_debug = amns_debug_env.eq."no" .or. amns_debug_env.eq."NO" .or. amns_debug_env.eq."No"
```

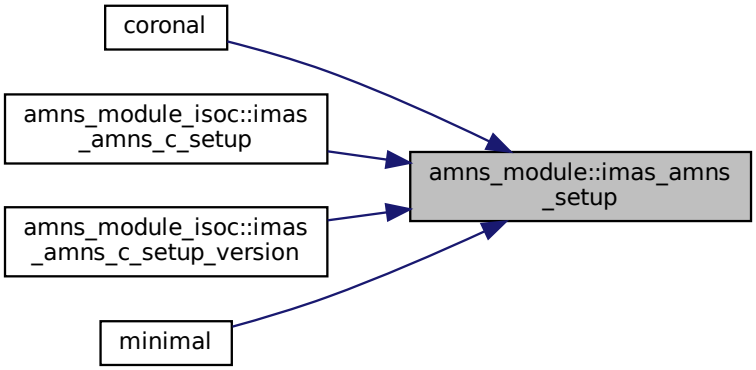
```
00093     if (amns_debug) handle%debug = .false.
00094
00095     if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP start'
00096
00097     ! by default, we get all data from system
00098     ! then, we can overwrite it using values passed as arguments
00099     default_version%string='DEFAULT'
00100     if(handle%debug) call getenv('USER', user)
00101     default_version%user = user
00102     call getenv('IMAS_VERSION', ds_version)
00103     if(handle%debug) write(*,*) 'Using DATAVERSION ', trim(ds_version)
00104
00105     if(present(version)) then
00106        if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP: requested database = ',trim(version%string),
      version%number
00107        handle%version=version
00108        version_no = version%number
00109        if(version%user .eq. '') then
00110           handle%version%USER = user
00111           if(handle%debug) write(*,*) 'Reset USER to ', trim(user)
00112        endif
00113     else
00114        handle%version=default_version
00115        version_no = -1
00116     endif
00117     if(handle%version%backend .eq. '') then
00118        handle%version%backend = backend
00119     endif
00120
00121 ! get the index information from 0/1
00122     shot=0
00123     run=1
00124     if(handle%version%backend.eq.'ascii') then
00125 !       write(file,'(a,"_",i6.6,"_",i6.6,".IDS")') 'amns',shot,run
00126 !       write(*,*) 'Reading data from ', trim(file), ' for ', shot, run
00127 !       call open_read_file(1, trim(file))
00128 !       call read_ids(amns00, 'amns')
00129 !       call close_read_file
00130        call errorstop('Not supported in the IDS version')
00131 #ifdef AL
00132     elseif(handle%version%backend.eq.'hdf5') then
00133 !       imas_status = imas_open_hdf5(treename, shot, run, idx)
00134 !       write(*,*) 'IMAS_STATUS, IDX = ', imas_status, idx
00135 !       call ids_get(idx, amnspath, amns00)
00136 !       call imas_close(idx)
00137        call errorstop('HDF5 not supported in the IDS version')
00138     elseif(handle%version%backend.eq.'mdsplus') then
00139        if(handle%debug) write(*,*) 'Attempting to access data from ',  shot, run,  &
00140             trim(handle%version%USER), ' amns ', trim(ds_version)
00141 !NLL       imas_status = imas_open_env(treename, shot, run, idx, &
00142 !NLL            trim(handle%version%USER), 'amns', trim(ds_version))
00143        !call ual_begin_pulse_action(MDSPLUS_BACKEND, shot, run, &
00144        !     trim(handle%version%USER), 'amns', trim(ds_version), idx)
00145        !call ual_open_pulse(idx, OPEN_PULSE, '', imas_status)
00146        call al_build_uri_from_legacy_parameters(mdsplus_backend, shot, run, &
00147             trim(handle%version%USER), 'amns', trim(ds_version), '', uri, imas_status)
00148        call al_begin_dataentry_action(uri, open_pulse, idx, imas_status)
00149        if(handle%debug) write(*,*) 'IMAS_STATUS, IDX = ', imas_status, idx
00150        if(imas_status.eq.-1) then
00151           if(handle%debug) write(0,*) 'Failure opening IDS!'
00152           ! stop 1
00153        endif
00154        if(imas_status.ne.0) then
00155           handle%version%USER = 'public'
00156           if(handle%debug) write(*,*) 'Attempting to access data from ',  shot, run,  &
00157                trim(handle%version%USER), ' amns ', trim(ds_version)
00158 !NLL           imas_status = imas_open_env(treename, shot, run, idx, &
00159 !NLL                trim(handle%version%USER), 'amns', trim(ds_version))
00160           !call ual_begin_pulse_action(MDSPLUS_BACKEND, shot, run, &
00161           !     trim(handle%version%USER), 'amns', trim(ds_version), idx)
00162           !call ual_open_pulse(idx, OPEN_PULSE, '', imas_status)
00163           call al_build_uri_from_legacy_parameters(mdsplus_backend, shot, run, &
00164                trim(handle%version%USER), 'amns', trim(ds_version), '', uri, imas_status)
00165           call al_begin_dataentry_action(uri, open_pulse, idx, imas_status)
00166
00167           if(handle%debug) write(*,*) 'IMAS_STATUS, IDX = ', imas_status, idx
00168           if(imas_status.ne.0) then
00169              if(present(error_status)) then
00170                 error_status%flag=.true.
00171                 error_status%string='Could not find any AMNS data, even under "public"'
00172                 return
00173              else
00174                 call errorstop('Could not find any AMNS data, even under "public"')
00175              end if
00176           endif
00177        endif
00178        call ids_get(idx, amnspath, amns00)
```

```
00179        call imas_close(idx)
00180 #else
00181     else
00182        write(*,*) 'The version of the AMNS routines without the AL was compiled'
00183        write(*,*) 'Use the ascii backend'
00184        if(present(error_status)) then
00185           error_status%flag=.true.
00186           error_status%string='UAL called in a non-AL version of the code'
00187           return
00188        else
00189           call errorstop('UAL called in a non-AL version of the code')
00190        end if
00191 #endif
00192     endif
00193     handle%properties_comment = assign_if_associated(amns00%ids_properties%comment, "Not specified in
       the AMNS IDS")
00194     handle%properties_source = assign_if_associated(amns00%ids_properties%source, "Not specified in
       the AMNS IDS")
00195     handle%properties_provider = assign_if_associated(amns00%ids_properties%provider, "Not specified
       in the AMNS IDS")
00196     handle%properties_creation_date = assign_if_associated(amns00%ids_properties%creation_date, "Not
       specified in the AMNS IDS")
00197     handle%code_name = assign_if_associated(amns00%code%name, "Not specified in the AMNS IDS")
00198     handle%code_commit = assign_if_associated(amns00%code%commit, "Not specified in the AMNS IDS")
00199     handle%code_version = assign_if_associated(amns00%code%version, "Not specified in the AMNS IDS")
00200     handle%code_repository = assign_if_associated(amns00%code%repository, "Not specified in the AMNS
       IDS")
00201
00202     if(.not.associated(amns00%release)) then
00203        if(present(error_status)) then
00204           error_status%flag=.true.
00205           error_status%string="No version information in INDEX shot"
00206           return
00207        else
00208           call errorstop("No version information in INDEX shot")
00209        end if
00210     endif
00211     if(version_no.le.0) then
00212        version_no=size(amns00%release)
00213        if(handle%debug) write(*,*) 'Found global version # ', version_no
00214     else
00215        if(version_no.gt.size(amns00%release)) then
00216           write(*,*) 'Requested version out of range ', version_no, ' > ', size(amns00%release)
00217           if(present(error_status)) then
00218              error_status%flag=.true.
00219              error_status%string="Requested version out of range"
00220              return
00221           else
00222              call errorstop('Requested version out of range')
00223           end if
00224        endif
00225     endif
00226     handle%version%string = trim(handle%version%USER) // ': ' // trim(handle%version%backend)
00227     handle%version%number = version_no
00228
00229     if(present(error_status)) then
00230        if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP: requested error_status response'
00231     endif
00232     handle%initialized=.true.
00233     if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP end'
00234
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.2.2.8 imas_amns_setup_table()

```
subroutine amns_module::imas_amns_setup_table (
             type(amns_handle_type), intent(in) handle,
             type(amns_reaction_type), intent(in) reaction_type,
             type(amns_reactants_type), intent(in) reactants,
             type(amns_handle_rx_type), intent(out) handle_rx,
             type(amns_error_type), intent(out), optional error_status )
```

initialization call for a particular reaction

Parameters

in	*handle*	Opaque handle from call to IMAS_AMNS_SETUP
in	*reaction_type*	Specifier for the reaction
in	*reactants*	Specifier for the reactants and products
out	*handle_rx*	Opaque handle to be passed to table routine calls
out	*error_status*	If an error occurs, this will be set (optional)

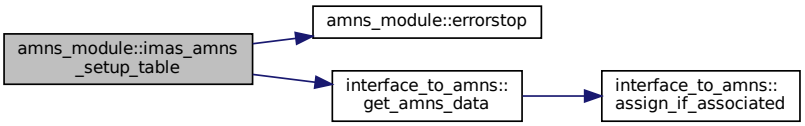
Definition at line 245 of file amns_module.f90.

```
00246     use ids_types  ! IGNORE
00247     use amns_types
00248     use amns_utility
00249     use interface_to_amns
00250     ! do not remove the comment!!!
00251     use ids_schemas   ! IGNORE
00252     use ids_routines  ! IGNORE
00253
00254     implicit none
00255     optional error_status
00256     type(amns_handle_type), intent(in) :: handle
00257     type(amns_reaction_type), intent(in) :: reaction_type
00258     type(amns_reactants_type), intent(in) :: reactants
00259     type(amns_handle_rx_type), intent(out) :: handle_rx
00260     type(amns_error_type), intent(out) :: error_status
00261
00262     integer :: no_of_reactants, i, is
00263     integer :: izn, izm, shot, run, ierr
00264     character*128 :: data_file, error_description
00265     integer :: iversion, nrelease, irelease
00266
00267     if(present(error_status)) then
00268        error_status%flag=.false.
```

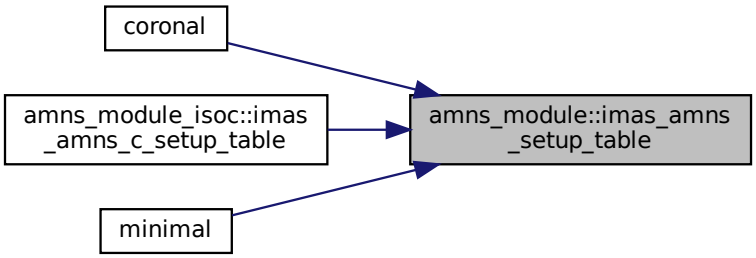
```
00269        error_status%string="No error"
00270     end if
00271
00272     handle_rx%debug= handle%debug
00273     handle_rx%reaction_type= reaction_type%string
00274     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE start'
00275     izn=0
00276     izm=0
00277     if(allocated(reactants%components)) then
00278        no_of_reactants=size(reactants%components)
00279        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE: number of reactants = ',no_of_reactants
00280        do i=1,no_of_reactants
00281 ! only reactants, not products
00282           if(reactants%components(i)%lr .eq. 0 .and. &
00283                (reactants%components(i)%int_specifier .ge. 0 .or.
      reactants%components(i)%int_specifier .eq. ids_int_invalid)) then
00284              if(nint(reactants%components(i)%ZN) .gt. izn) then
00285                 izn=nint(reactants%components(i)%ZN)
00286                 if(reaction_type%isotope_resolved .ne. 0) izm=nint(reactants%components(i)%MI)
00287              else if(nint(reactants%components(i)%ZN) .eq. izn .and. reaction_type%isotope_resolved
      .ne. 0) then
00288                 izm=max(izm,nint(reactants%components(i)%MI))
00289              endif
00290           endif
00291           if(handle_rx%debug) &
00292                write(*,'(a,i3,3(1x,f6.2),i2)') 'IMAS_AMNS_SETUP_TABLE: reactant#, ZN, ZA, MI, LR = ',
      &
00293                i,reactants%components(i)%ZN, &
00294                reactants%components(i)%ZA, &
00295                reactants%components(i)%MI, &
00296                reactants%components(i)%lr
00297        enddo
00298        handle_rx%no_of_reactants=no_of_reactants
00299        allocate(handle_rx%components(no_of_reactants))
00300        handle_rx%components=reactants%components
00301     else
00302        no_of_reactants=0
00303        handle_rx%no_of_reactants=0
00304     endif
00305     if(allocated(reactants%string)) then
00306        allocate(handle_rx%string(size(reactants%string)))
00307        handle_rx%string=reactants%string
00308     endif
00309     if(present(error_status)) then
00310        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE: requested error_status response'
00311     endif
00312 ! find local version number for shot
00313     shot=izn+1000*izm
00314     if(shot.eq.0) then
00315        if(present(error_status)) then
00316           error_status%flag=.true.
00317           return
00318        else
00319           write(*,*) 'Could not calculate the AMNS shot number'
00320           if(present(error_status)) then
00321              error_status%flag=.true.
00322              error_status%string="Could not calculate the AMNS shot number"
00323              return
00324           else
00325              call errorstop('Could not calculate the AMNS shot number')
00326           end if
00327        endif
00328     endif
00329     run=0
00330     iversion=handle%version%number
00331     if(associated(amns00%release(iversion)%data_entry)) then
00332        nrelease=size(amns00%release(iversion)%data_entry)
00333        do irelease=1, nrelease   ! loop over data in a version
00334           if(amns00%release(iversion)%data_entry(irelease)%shot .eq. shot) then
00335              run=amns00%release(iversion)%data_entry(irelease)%run
00336              exit
00337           endif
00338        enddo
00339     endif
00340     if(run.eq.0) then
00341        if(present(error_status)) then
00342           error_status%flag=.true.
00343           write(error_status%string,*) 'No data found for ', shot, ' in version ', iversion
00344           return
00345        else
00346           write(*,*) 'No data found for ', shot, ' in version ', iversion
00347           call errorstop('No data found')
00348        endif
00349     else
00350        if(handle_rx%debug) write(*,*) 'Found local version ', run, ' for case = ', shot
00351        handle_rx%version%number = run
00352        ! now handle the various reaction possibilities
```

```
00353
00354        call get_amns_data(reaction_type, reactants, handle_rx%grid, &
00355             handle_rx%properties_comment, handle_rx%properties_source, &
00356             handle_rx%properties_provider, handle_rx%properties_creation_date, &
00357             handle_rx%code_name, handle_rx%code_commit, &
00358             handle_rx%code_version, handle_rx%code_repository, &
00359             handle_rx%source, handle_rx%provider, handle_rx%citation, &
00360             shot, run, handle%version%backend, handle%version%user, ds_version, &
00361             ierr, error_description, handle_rx%debug)
00362        if(ierr.eq.0) then
00363           handle_rx%filled=.true.
00364        else
00365           if(present(error_status)) then
00366              error_status%flag=.true.
00367              error_status%string="'get_amns_data' returned an error - " // error_description
00368              return
00369           else
00370              stop "'get_amns_data' returned an error"
00371           end if
00372        endif
00373        handle_rx%initialized=.true.
00374        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE end'
00375     endif
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.3 AMNS Callable C Subroutines

Module implementing the ITM AMNS interface for C codes.

Functions/Subroutines

- subroutine amns_module_isoc::imas_amns_c_setup (handle, error_status_fc)

  provides IMAS_AMNS_C_SETUP by calling IMAS_AMNS_SETUP
- subroutine amns_module_isoc::imas_amns_c_setup_version (handle, version_fc, error_status_fc)

  provides IMAS_AMNS_C_SETUP by calling IMAS_AMNS_SETUP with version argument
- subroutine amns_module_isoc::imas_amns_c_finish (handle, error_status_fc)

  provides IMAS_AMNS_C_FINISH by calling IMAS_AMNS_FINISH
- subroutine amns_module_isoc::imas_amns_c_finish_table (handle_rx, error_status_fc)

  provides IMAS_AMNS_C_FINISH_TABLE by calling IMAS_AMNS_FINISH_TABLE
- subroutine amns_module_isoc::imas_amns_c_set (handle, set_fc, error_status_fc)

  provides IMAS_AMNS_C_SET by calling IMAS_AMNS_SET
- subroutine amns_module_isoc::imas_amns_c_query (handle, query_fc, answer_fc, error_status_fc)

  provides IMAS_AMNS_C_QUERY by calling IMAS_AMNS_QUERY
- subroutine amns_module_isoc::imas_amns_c_setup_table (handle, reaction_type_fc, reactant, handle_rx, error_status_fc)

  provides IMAS_AMNS_C_SETUP_TABLE by calling IMAS_AMNS_SETUP_TABLE
- subroutine amns_module_isoc::imas_amns_c_query_table (handle_rx, query_fc, answer_fc, error_status↩_fc)

  provides IMAS_AMNS_C_QUERY_TABLE by calling IMAS_AMNS_QUERY_TABLE
- subroutine amns_module_isoc::imas_amns_c_set_table (handle_rx, set_fc, error_status_fc)

  provides IMAS_AMNS_C_SET_TABLE by calling IMAS_AMNS_SET_TABLE
- subroutine amns_module_isoc::imas_amns_c_rx_0_a (handle_rx, out, arg1, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_0_b (handle_rx, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_0_c (handle_rx, out, arg1, arg2, arg3, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_0_d (handle_rx, out, arg1, arg2, arg3, arg4, error_status↩_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_1_a (handle_rx, nx, out, arg1, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_1_b (handle_rx, nx, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_1_c (handle_rx, nx, out, arg1, arg2, arg3, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_1_d (handle_rx, nx, out, arg1, arg2, arg3, arg4, error_↩status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_2_a (handle_rx, nx, ny, out, arg1, error_status_fc)

get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine amns_module_isoc::imas_amns_c_rx_2_b (handle_rx, nx, ny, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine amns_module_isoc::imas_amns_c_rx_2_c (handle_rx, nx, ny, out, arg1, arg2, arg3, error_↩ status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine amns_module_isoc::imas_amns_c_rx_2_d (handle_rx, nx, ny, out, arg1, arg2, arg3, arg4, error↩ _status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine amns_module_isoc::imas_amns_c_rx_3_a (handle_rx, nx, ny, nz, out, arg1, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine amns_module_isoc::imas_amns_c_rx_3_b (handle_rx, nx, ny, nz, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into three separate subroutine for 1,2 or 3 arguments.

- subroutine amns_module_isoc::imas_amns_c_rx_3_c (handle_rx, nx, ny, nz, out, arg1, arg2, arg3, error_↩ status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into three separate subroutine for 1,2 or 3 arguments.

- subroutine amns_module_isoc::imas_amns_c_rx_3_d (handle_rx, nx, ny, nz, out, arg1, arg2, arg3, arg4, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine amns_module_isoc::imas_amns_c_setup_reactants (reactants_handle, string, index, n_↩ reactants)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

- subroutine amns_module_isoc::imas_amns_c_set_reactant (reactants_handle, reactant_index, reactant)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

- subroutine amns_module_isoc::imas_amns_c_get_reactant (reactants_handle, reactant_index, reactant)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

- subroutine amns_module_isoc::imas_amns_c_finish_reactants (reactants_handle)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

13.3.1 Detailed Description

Module implementing the ITM AMNS interface for C codes.

Author

Hajo Klingshirn

13.3.2 Function/Subroutine Documentation

13.3.2.1 imas_amns_c_finish()

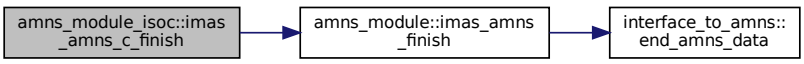
```
subroutine amns_module_isoc::imas_amns_c_finish (
             type(c_ptr), intent(inout) handle,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

provides IMAS_AMNS_C_FINISH by calling IMAS_AMNS_FINISH

Definition at line 94 of file amns_module_isoc.f90.

```
00095     type(c_ptr), intent(inout) :: handle
00096     type(amns_fc_error_type), intent(out) :: error_status_fc
00097
00098     ! internal
00099     type(amns_handle_type), pointer :: lhandle
00100     type(amns_error_type) :: error_status
00101
00102     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH: enter"
00103     call c_f_pointer(handle, lhandle)
00104     call imas_amns_finish(lhandle, error_status)
00105     deallocate(lhandle)
00106     handle = c_null_ptr
00107     error_status_fc%flag = error_status%flag
00108     error_status_fc%string = copy(error_status%string)
00109     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH: return"
```

Here is the call graph for this function:



13.3.2.2 imas_amns_c_finish_reactants()

```
subroutine amns_module_isoc::imas_amns_c_finish_reactants (
             type(c_ptr), intent(inout) reactants_handle )
```

Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

Definition at line 679 of file amns_module_isoc.f90.

```
00680     type(c_ptr), intent(inout) :: reactants_handle
00681
00682     ! internal
00683     type(amns_reactants_type), pointer :: lreactants
00684
00685     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH_REACTANTS: enter"
00686     call c_f_pointer(reactants_handle, lreactants)
00687     deallocate(lreactants%components)
00688     deallocate(lreactants)
00689     reactants_handle = c_null_ptr
00690     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH_REACTANTS: return"
```

13.3.2.3 imas_amns_c_finish_table()

```
subroutine amns_module_isoc::imas_amns_c_finish_table (
             type(c_ptr), intent(inout) handle_rx,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

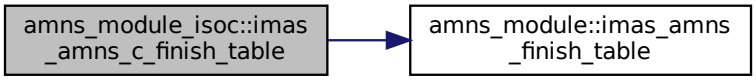
provides IMAS_AMNS_C_FINISH_TABLE by calling IMAS_AMNS_FINISH_TABLE

Definition at line 114 of file amns_module_isoc.f90.

```
00115     type(c_ptr), intent(inout) :: handle_rx
00116     type(amns_fc_error_type), intent(out) :: error_status_fc
00117
00118     ! internal
00119     type(amns_handle_rx_type), pointer :: lhandle_rx
00120     type(amns_error_type) :: error_status
00121
00122     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH_TABLE: enter"
00123     call c_f_pointer(handle_rx, lhandle_rx)
00124     call imas_amns_finish_table(lhandle_rx, error_status)
```

```
00125     deallocate(lhandle_rx)
00126     handle_rx = c_null_ptr
00127     error_status_fc%flag = error_status%flag
00128     error_status_fc%string = copy(error_status%string)
00129     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH_TABLE: return"
```

Here is the call graph for this function:



13.3.2.4 imas_amns_c_get_reactant()

```
subroutine amns_module_isoc::imas_amns_c_get_reactant (
             type(c_ptr), intent(in), value reactants_handle,
             integer(c_int), intent(in), value reactant_index,
             type(amns_reactant_type), intent(out) reactant )
```

Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

Definition at line 661 of file amns_module_isoc.f90.

```
00662     type(c_ptr), intent(in), value :: reactants_handle
00663     integer(c_int), intent(in), value :: reactant_index
00664     type(amns_reactant_type), intent(out) :: reactant
00665
00666     ! internal
00667     type(amns_reactants_type), pointer :: lreactants
00668
00669     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_GET_REACTANT: enter"
00670     call c_f_pointer(reactants_handle, lreactants)
00671     reactant = lreactants%components(reactant_index)
00672     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_GET_REACTANT: return"
```

13.3.2.5 imas_amns_c_query()

```
subroutine amns_module_isoc::imas_amns_c_query (
             type(c_ptr), intent(in), value handle,
             type(amns_fc_query_type), intent(in) query_fc,
             type(amns_fc_answer_type), intent(out) answer_fc,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

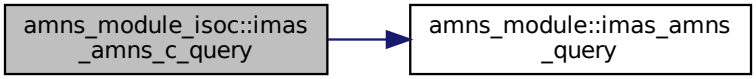
provides IMAS_AMNS_C_QUERY by calling IMAS_AMNS_QUERY

Definition at line 157 of file amns_module_isoc.f90.

```
00158     type(c_ptr), intent(in), value :: handle
00159     type(amns_fc_query_type), intent(in) :: query_fc
00160     type(amns_fc_answer_type), intent(out) :: answer_fc
00161     type(amns_fc_error_type), intent(out) :: error_status_fc
00162
00163     ! internal
00164     type(amns_handle_type), pointer :: lhandle
00165     type(amns_query_type) :: query
00166     type(amns_answer_type) :: answer
00167     type(amns_error_type) :: error_status
00168
00169     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: enter"
00170     call c_f_pointer(handle, lhandle)
00171     query%string = copy(query_fc%string)
00172     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: query%string ='"//query%string//"'"
00173     call c_f_pointer(handle, lhandle)
00174     call imas_amns_query(lhandle,query,answer,error_status)
00175     answer_fc%string = copy(answer%string)
00176     answer_fc%number = answer%number
00177     error_status_fc%flag = error_status%flag
00178     error_status_fc%string = copy(error_status%string)
```

```
00179     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: answer%string ='"//answer%string//"'"
00180     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: answer%number =", answer%number
00181     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: return"
```

Here is the call graph for this function:



13.3.2.6 imas_amns_c_query_table()

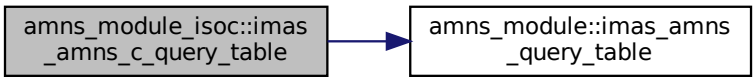
```
subroutine amns_module_isoc::imas_amns_c_query_table (
             type(c_ptr), intent(in), value handle_rx,
             type(amns_fc_query_type), intent(in) query_fc,
             type(amns_fc_answer_type), intent(out) answer_fc,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

provides IMAS_AMNS_C_QUERY_TABLE by calling IMAS_AMNS_QUERY_TABLE

Definition at line 215 of file amns_module_isoc.f90.

```
00216     type(c_ptr), intent(in), value :: handle_rx
00217     type(amns_fc_query_type), intent(in) :: query_fc
00218     type(amns_fc_answer_type), intent(out) :: answer_fc
00219     type(amns_fc_error_type), intent(out) :: error_status_fc
00220
00221     ! internal
00222     type(amns_handle_rx_type), pointer :: lhandle_rx
00223     type(amns_query_type) :: query
00224     type(amns_answer_type) :: answer
00225     type(amns_error_type) :: error_status
00226
00227     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY_TABLE: enter"
00228     call c_f_pointer(handle_rx, lhandle_rx)
00229     query%string = copy(query_fc%string)
00230     call imas_amns_query_table(lhandle_rx,query,answer,error_status)
00231     answer_fc%string = copy(answer%string)
00232     answer_fc%number = answer%number
00233     error_status_fc%flag = error_status%flag
00234     error_status_fc%string = copy(error_status%string)
00235     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY_TABLE: return"
```

Here is the call graph for this function:



13.3.2.7 imas_amns_c_rx_0_a()

```
subroutine amns_module_isoc::imas_amns_c_rx_0_a (
             type(c_ptr), intent(in), value handle_rx,
             real (kind=ids_real), intent(out) out,
```

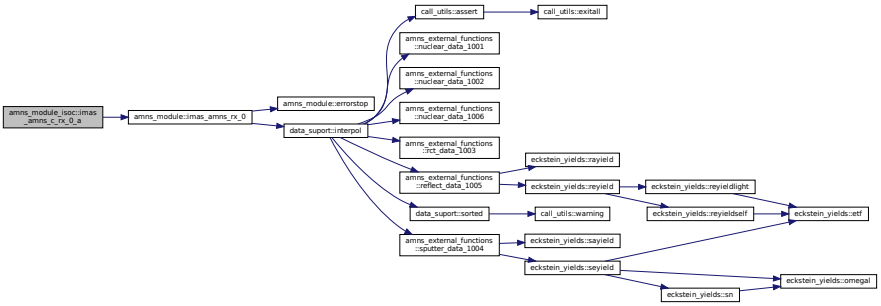
```
            real (kind=ids_real), intent(in), value arg1,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 263 of file amns_module_isoc.f90.

```
00264     type(c_ptr), intent(in), value :: handle_rx
00265     real (kind=ids_real), intent(out) :: out
00266     real (kind=ids_real), intent(in), value :: arg1
00267     type(amns_fc_error_type), intent(out) :: error_status_fc
00268
00269     ! internal
00270     type(amns_handle_rx_type), pointer :: lhandle_rx
00271     type(amns_error_type) :: error_status
00272
00273     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_A: enter"
00274     call c_f_pointer(handle_rx, lhandle_rx)
00275     call imas_amns_rx_0(lhandle_rx, out, arg1, error_status=error_status)
00276     error_status_fc%flag = error_status%flag
00277     error_status_fc%string = copy(error_status%string)
00278     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_A: return"
```

Here is the call graph for this function:



13.3.2.8 imas_amns_c_rx_0_b()

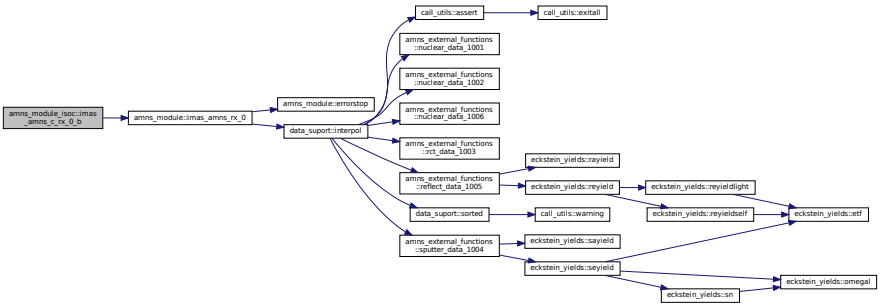
```
subroutine amns_module_isoc::imas_amns_c_rx_0_b (
            type(c_ptr), intent(in), value handle_rx,
            real (kind=ids_real), intent(out) out,
            real (kind=ids_real), intent(in), value arg1,
            real (kind=ids_real), intent(in), value arg2,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 284 of file amns_module_isoc.f90.

```
00285     type(c_ptr), intent(in), value :: handle_rx
00286     real (kind=ids_real), intent(out) :: out
00287     real (kind=ids_real), intent(in), value :: arg1,arg2
00288     type(amns_fc_error_type), intent(out) :: error_status_fc
00289
00290     ! internal
00291     type(amns_handle_rx_type), pointer :: lhandle_rx
00292     type(amns_error_type) :: error_status
00293
00294     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_B: enter"
00295     call c_f_pointer(handle_rx, lhandle_rx)
00296     call imas_amns_rx_0(lhandle_rx, out, arg1, arg2=arg2, error_status=error_status)
00297     error_status_fc%flag = error_status%flag
00298     error_status_fc%string = copy(error_status%string)
00299     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_B: return"
```

Here is the call graph for this function:



13.3.2.9 imas_amns_c_rx_0_c()

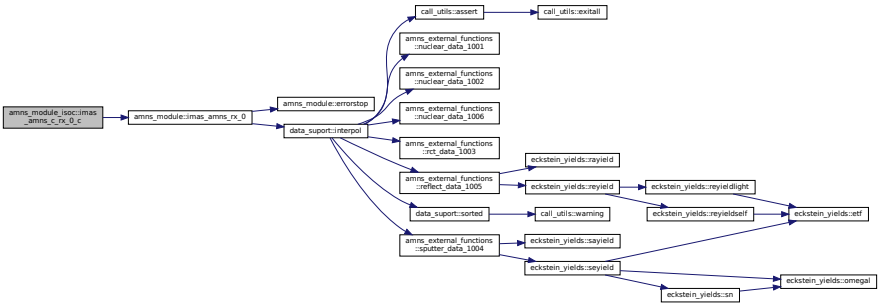
```
subroutine amns_module_isoc::imas_amns_c_rx_0_c (
            type(c_ptr), intent(in), value handle_rx,
            real (kind=ids_real), intent(out) out,
            real (kind=ids_real), intent(in), value arg1,
            real (kind=ids_real), intent(in), value arg2,
            real (kind=ids_real), intent(in), value arg3,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 305 of file amns_module_isoc.f90.

```
00306     type(c_ptr), intent(in), value :: handle_rx
00307     real (kind=ids_real), intent(out) :: out
00308     real (kind=ids_real), intent(in), value :: arg1,arg2,arg3
00309     type(amns_fc_error_type), intent(out) :: error_status_fc
00310
00311     ! internal
00312     type(amns_handle_rx_type), pointer :: lhandle_rx
00313     type(amns_error_type) :: error_status
00314
00315     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_C: enter"
00316     call c_f_pointer(handle_rx, lhandle_rx)
00317     call imas_amns_rx_0(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, error_status=error_status)
00318     error_status_fc%flag = error_status%flag
00319     error_status_fc%string = copy(error_status%string)
00320     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_C: return"
```

Here is the call graph for this function:



13.3.2.10 imas_amns_c_rx_0_d()

```
subroutine amns_module_isoc::imas_amns_c_rx_0_d (
```

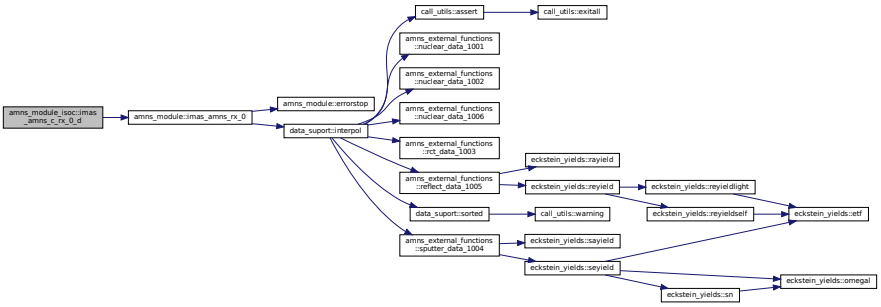
```
            type(c_ptr), intent(in), value handle_rx,
            real (kind=ids_real), intent(out) out,
            real (kind=ids_real), intent(in), value arg1,
            real (kind=ids_real), intent(in), value arg2,
            real (kind=ids_real), intent(in), value arg3,
            real (kind=ids_real), intent(in), value arg4,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 326 of file amns_module_isoc.f90.

```
00327     type(c_ptr), intent(in), value :: handle_rx
00328     real (kind=ids_real), intent(out) :: out
00329     real (kind=ids_real), intent(in), value :: arg1,arg2,arg3,arg4
00330     type(amns_fc_error_type), intent(out) :: error_status_fc
00331
00332     ! internal
00333     type(amns_handle_rx_type), pointer :: lhandle_rx
00334     type(amns_error_type) :: error_status
00335
00336     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_D: enter"
00337     call c_f_pointer(handle_rx, lhandle_rx)
00338     call imas_amns_rx_0(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, arg4=arg4,
      error_status=error_status)
00339     error_status_fc%flag = error_status%flag
00340     error_status_fc%string = copy(error_status%string)
00341     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_D: return"
```

Here is the call graph for this function:



13.3.2.11 imas_amns_c_rx_1_a()

```
subroutine amns_module_isoc::imas_amns_c_rx_1_a (
            type(c_ptr), intent(in), value handle_rx,
            integer(c_int), intent(in), value nx,
            real (c_double), dimension(0:nx-1), intent(out) out,
            real (c_double), dimension(0:nx-1), intent(in) arg1,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

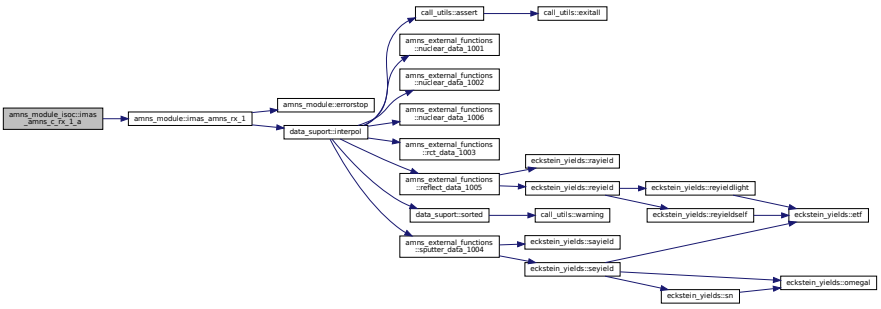
get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 348 of file amns_module_isoc.f90.

```
00349     type(c_ptr), intent(in), value :: handle_rx
00350     integer(c_int), intent(in), value :: nx
00351     real (c_double), intent(out) :: out(0:nx-1)
00352     real (c_double), intent(in) :: arg1(0:nx-1)
00353     type(amns_fc_error_type), intent(out) :: error_status_fc
00354
00355     ! internal
00356     type(amns_handle_rx_type), pointer :: lhandle_rx
00357     type(amns_error_type) :: error_status
00358
00359     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_A: enter"
00360     call c_f_pointer(handle_rx, lhandle_rx)
00361     call imas_amns_rx_1(lhandle_rx, out, arg1, error_status=error_status)
00362     error_status_fc%flag = error_status%flag
00363     error_status_fc%string = copy(error_status%string)
```

```
00364     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_A: return"
```

Here is the call graph for this function:



13.3.2.12 imas_amns_c_rx_1_b()

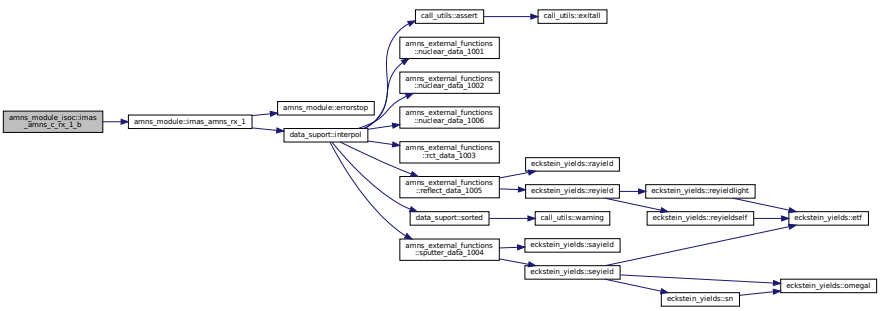
```
subroutine amns_module_isoc::imas_amns_c_rx_1_b (
             type(c_ptr), intent(in), value handle_rx,
             integer(c_int), intent(in), value nx,
             real (c_double), dimension(0:nx-1), intent(out) out,
             real (c_double), dimension(0:nx-1), intent(in) arg1,
             real (c_double), dimension(0:nx-1), intent(in) arg2,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 370 of file amns_module_isoc.f90.

```
00371     type(c_ptr), intent(in), value :: handle_rx
00372     integer(c_int), intent(in), value :: nx
00373     real (c_double), intent(out) :: out(0:nx-1)
00374     real (c_double), intent(in) :: arg1(0:nx-1), arg2(0:nx-1)
00375     type(amns_fc_error_type), intent(out) :: error_status_fc
00376
00377     ! internal
00378     type(amns_handle_rx_type), pointer :: lhandle_rx
00379     type(amns_error_type) :: error_status
00380
00381     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_B: enter"
00382     call c_f_pointer(handle_rx, lhandle_rx)
00383     call imas_amns_rx_1(lhandle_rx, out, arg1, arg2=arg2, error_status=error_status)
00384     error_status_fc%flag = error_status%flag
00385     error_status_fc%string = copy(error_status%string)
00386     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_B: return"
```

Here is the call graph for this function:



13.3.2.13 imas_amns_c_rx_1_c()

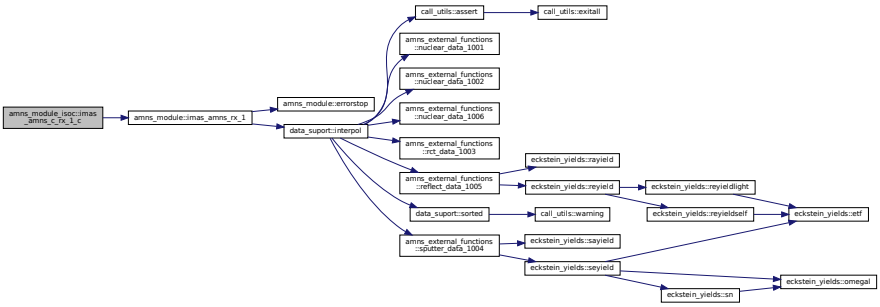
```
subroutine amns_module_isoc::imas_amns_c_rx_1_c (
            type(c_ptr), intent(in), value handle_rx,
            integer(c_int), intent(in), value nx,
            real (c_double), dimension(0:nx-1), intent(out) out,
            real (c_double), dimension(0:nx-1), intent(in) arg1,
            real (c_double), dimension(0:nx-1), intent(in) arg2,
            real (c_double), dimension(0:nx-1), intent(in) arg3,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 392 of file amns_module_isoc.f90.

```
00393     type(c_ptr), intent(in), value :: handle_rx
00394     integer(c_int), intent(in), value :: nx
00395     real (c_double), intent(out) :: out(0:nx-1)
00396     real (c_double), intent(in) :: arg1(0:nx-1), arg2(0:nx-1), arg3(0:nx-1)
00397     type(amns_fc_error_type), intent(out) :: error_status_fc
00398
00399     ! internal
00400     type(amns_handle_rx_type), pointer :: lhandle_rx
00401     type(amns_error_type) :: error_status
00402
00403     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_C: enter"
00404     call c_f_pointer(handle_rx, lhandle_rx)
00405     call imas_amns_rx_1(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, error_status=error_status)
00406     error_status_fc%flag = error_status%flag
00407     error_status_fc%string = copy(error_status%string)
00408     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_C: return"
```

Here is the call graph for this function:



13.3.2.14 imas_amns_c_rx_1_d()

```
subroutine amns_module_isoc::imas_amns_c_rx_1_d (
            type(c_ptr), intent(in), value handle_rx,
            integer(c_int), intent(in), value nx,
            real (c_double), dimension(0:nx-1), intent(out) out,
            real (c_double), dimension(0:nx-1), intent(in) arg1,
            real (c_double), dimension(0:nx-1), intent(in) arg2,
            real (c_double), dimension(0:nx-1), intent(in) arg3,
            real (c_double), dimension(0:nx-1), intent(in) arg4,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

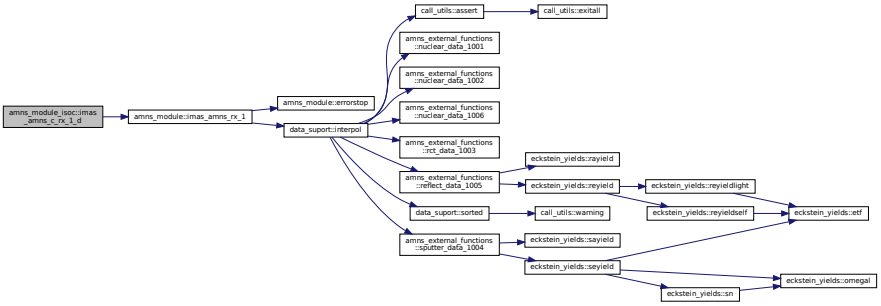
get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 414 of file amns_module_isoc.f90.

```
00415     type(c_ptr), intent(in), value :: handle_rx
00416     integer(c_int), intent(in), value :: nx
00417     real (c_double), intent(out) :: out(0:nx-1)
00418     real (c_double), intent(in) :: arg1(0:nx-1), arg2(0:nx-1), arg3(0:nx-1), arg4(0:nx-1)
00419     type(amns_fc_error_type), intent(out) :: error_status_fc
00420
```

```
00421     ! internal
00422     type(amns_handle_rx_type), pointer :: lhandle_rx
00423     type(amns_error_type) :: error_status
00424
00425     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_D: enter"
00426     call c_f_pointer(handle_rx, lhandle_rx)
00427     call imas_amns_rx_1(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, arg4=arg4,
      error_status=error_status)
00428     error_status_fc%flag = error_status%flag
00429     error_status_fc%string = copy(error_status%string)
00430     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_D: return"
```

Here is the call graph for this function:



13.3.2.15 imas_amns_c_rx_2_a()

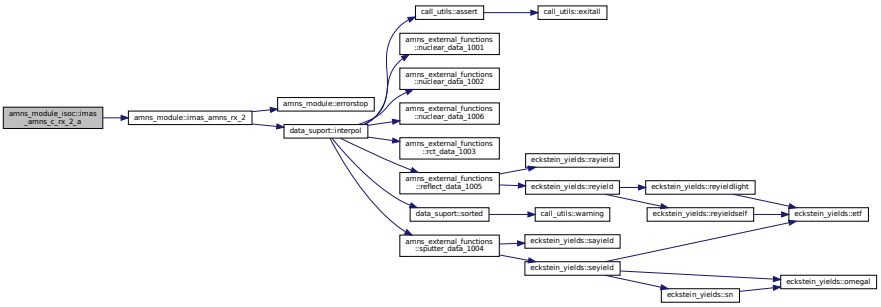
```
subroutine amns_module_isoc::imas_amns_c_rx_2_a (
            type(c_ptr), intent(in), value handle_rx,
            integer(c_int), intent(in), value nx,
            integer(c_int), intent(in), value ny,
            real (c_double), dimension(0:nx-1,0:ny-1), intent(out) out,
            real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg1,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 437 of file amns_module_isoc.f90.

```
00438     type(c_ptr), intent(in), value :: handle_rx
00439     integer(c_int), intent(in), value :: nx, ny
00440     real (c_double), intent(out) :: out(0:nx-1,0:ny-1)
00441     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1)
00442     type(amns_fc_error_type), intent(out) :: error_status_fc
00443
00444     ! internal
00445     type(amns_handle_rx_type), pointer :: lhandle_rx
00446     type(amns_error_type) :: error_status
00447
00448     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_A: enter"
00449     call c_f_pointer(handle_rx, lhandle_rx)
00450     call imas_amns_rx_2(lhandle_rx, out, arg1, error_status=error_status)
00451     error_status_fc%flag = error_status%flag
00452     error_status_fc%string = copy(error_status%string)
00453     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_A: return"
```

Here is the call graph for this function:



13.3.2.16 imas_amns_c_rx_2_b()

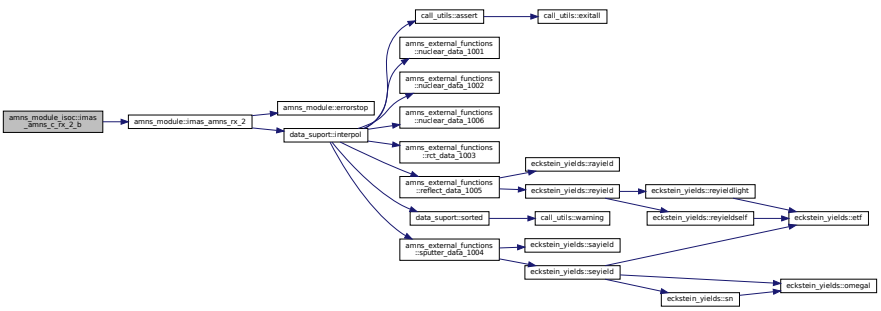
```
subroutine amns_module_isoc::imas_amns_c_rx_2_b (
             type(c_ptr), intent(in), value handle_rx,
             integer(c_int), intent(in), value nx,
             integer(c_int), intent(in), value ny,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(out) out,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg1,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg2,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 459 of file amns_module_isoc.f90.

```
00460     type(c_ptr), intent(in), value :: handle_rx
00461     integer(c_int), intent(in), value :: nx, ny
00462     real (c_double), intent(out) :: out(0:nx-1,0:ny-1)
00463     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1), arg2(0:nx-1,0:ny-1)
00464     type(amns_fc_error_type), intent(out) :: error_status_fc
00465
00466     ! internal
00467     type(amns_handle_rx_type), pointer :: lhandle_rx
00468     type(amns_error_type) :: error_status
00469
00470     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_B: enter"
00471     call c_f_pointer(handle_rx, lhandle_rx)
00472     call imas_amns_rx_2(lhandle_rx, out, arg1, arg2=arg2, error_status=error_status)
00473     error_status_fc%flag = error_status%flag
00474     error_status_fc%string = copy(error_status%string)
00475     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_B: return"
```

Here is the call graph for this function:



13.3.2.17 imas_amns_c_rx_2_c()

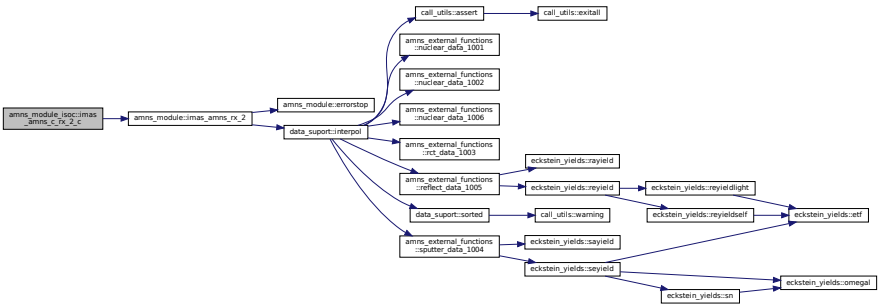
```
subroutine amns_module_isoc::imas_amns_c_rx_2_c (
             type(c_ptr), intent(in), value handle_rx,
             integer(c_int), intent(in), value nx,
             integer(c_int), intent(in), value ny,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(out) out,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg1,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg2,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg3,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 481 of file amns_module_isoc.f90.

```
00482     type(c_ptr), intent(in), value :: handle_rx
00483     integer(c_int), intent(in), value :: nx, ny
00484     real (c_double), intent(out) :: out(0:nx-1,0:ny-1)
00485     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1), arg2(0:nx-1,0:ny-1), arg3(0:nx-1,0:ny-1)
00486     type(amns_fc_error_type), intent(out) :: error_status_fc
00487
00488     ! internal
00489     type(amns_handle_rx_type), pointer :: lhandle_rx
00490     type(amns_error_type) :: error_status
00491
00492     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_C: enter"
00493     call c_f_pointer(handle_rx, lhandle_rx)
00494     call imas_amns_rx_2(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, error_status=error_status)
00495     error_status_fc%flag = error_status%flag
00496     error_status_fc%string = copy(error_status%string)
00497     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_C: return"
```

Here is the call graph for this function:



13.3.2.18 imas_amns_c_rx_2_d()

```
subroutine amns_module_isoc::imas_amns_c_rx_2_d (
             type(c_ptr), intent(in), value handle_rx,
             integer(c_int), intent(in), value nx,
             integer(c_int), intent(in), value ny,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(out) out,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg1,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg2,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg3,
             real (c_double), dimension(0:nx-1,0:ny-1), intent(in) arg4,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

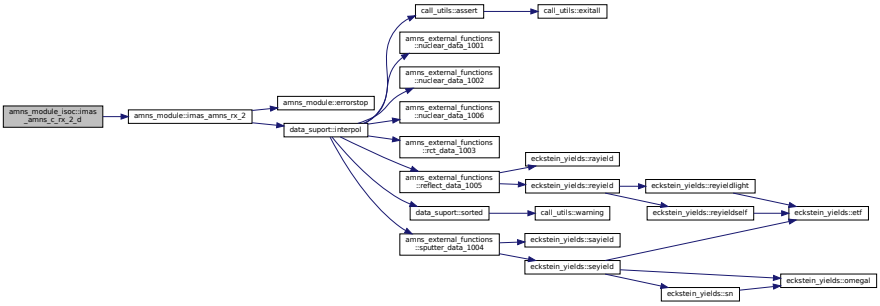
get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 503 of file amns_module_isoc.f90.

```
00504     type(c_ptr), intent(in), value :: handle_rx
00505     integer(c_int), intent(in), value :: nx, ny
00506     real (c_double), intent(out) :: out(0:nx-1,0:ny-1)
```

```
00507     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1), arg2(0:nx-1,0:ny-1), arg3(0:nx-1,0:ny-1),
      arg4(0:nx-1,0:ny-1)
00508     type(amns_fc_error_type), intent(out) :: error_status_fc
00509
00510     ! internal
00511     type(amns_handle_rx_type), pointer :: lhandle_rx
00512     type(amns_error_type) :: error_status
00513
00514     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_D: enter"
00515     call c_f_pointer(handle_rx, lhandle_rx)
00516     call imas_amns_rx_2(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, arg4=arg4,
      error_status=error_status)
00517     error_status_fc%flag = error_status%flag
00518     error_status_fc%string = copy(error_status%string)
00519     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_D: return"
```

Here is the call graph for this function:



13.3.2.19 imas_amns_c_rx_3_a()

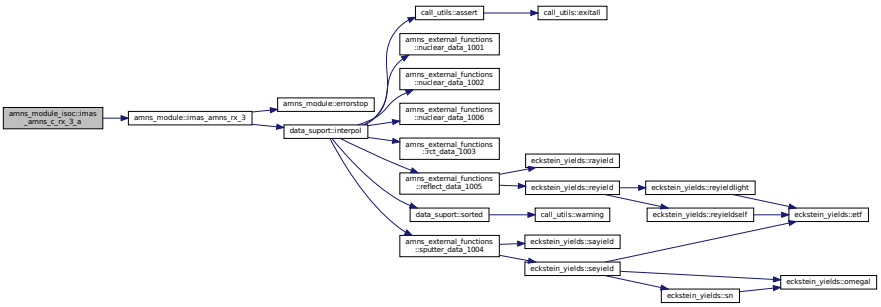
```
subroutine amns_module_isoc::imas_amns_c_rx_3_a (
             type(c_ptr), intent(in), value handle_rx,
             integer(c_int), intent(in), value nx,
             integer(c_int), intent(in), value ny,
             integer(c_int), intent(in), value nz,
             real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(out) out,
             real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg1,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (3d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 526 of file amns_module_isoc.f90.

```
00527     type(c_ptr), intent(in), value :: handle_rx
00528     integer(c_int), intent(in), value :: nx, ny, nz
00529     real (c_double), intent(out) :: out(0:nx-1,0:ny-1,0:nz-1)
00530     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1,0:nz-1)
00531     type(amns_fc_error_type), intent(out) :: error_status_fc
00532
00533     ! internal
00534     type(amns_handle_rx_type), pointer :: lhandle_rx
00535     type(amns_error_type) :: error_status
00536
00537     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_A: enter"
00538     call c_f_pointer(handle_rx, lhandle_rx)
00539     call imas_amns_rx_3(lhandle_rx, out, arg1, error_status=error_status)
00540     error_status_fc%flag = error_status%flag
00541     error_status_fc%string = copy(error_status%string)
00542     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_A: return"
```

Here is the call graph for this function:



13.3.2.20 imas_amns_c_rx_3_b()

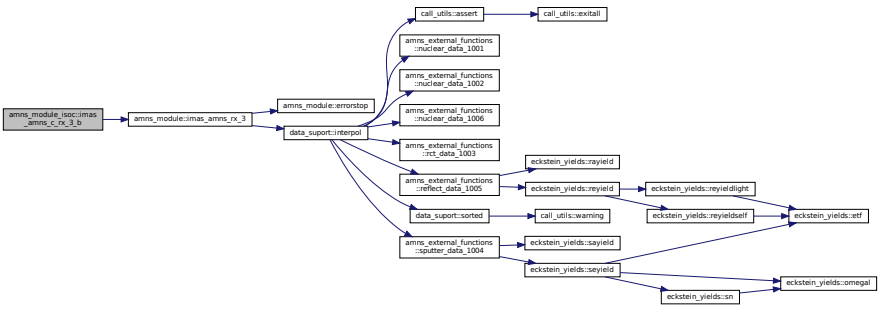
```
subroutine amns_module_isoc::imas_amns_c_rx_3_b (
            type(c_ptr), intent(in), value handle_rx,
            integer(c_int), intent(in), value nx,
            integer(c_int), intent(in), value ny,
            integer(c_int), intent(in), value nz,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(out) out,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg1,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg2,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (3d) args for a particular reaction Expanded into three separate subroutine for 1,2 or 3 arguments.

Definition at line 548 of file amns_module_isoc.f90.

```
00549     type(c_ptr), intent(in), value :: handle_rx
00550     integer(c_int), intent(in), value :: nx, ny, nz
00551     real (c_double), intent(out) :: out(0:nx-1,0:ny-1,0:nz-1)
00552     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1,0:nz-1), arg2(0:nx-1,0:ny-1,0:nz-1)
00553     type(amns_fc_error_type), intent(out) :: error_status_fc
00554
00555     ! internal
00556     type(amns_handle_rx_type), pointer :: lhandle_rx
00557     type(amns_error_type) :: error_status
00558
00559     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_B: enter"
00560     call c_f_pointer(handle_rx, lhandle_rx)
00561     call imas_amns_rx_3(lhandle_rx, out, arg1, arg2=arg2, error_status=error_status)
00562     error_status_fc%flag = error_status%flag
00563     error_status_fc%string = copy(error_status%string)
00564     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_B: return"
```

Here is the call graph for this function:



13.3.2.21 imas_amns_c_rx_3_c()

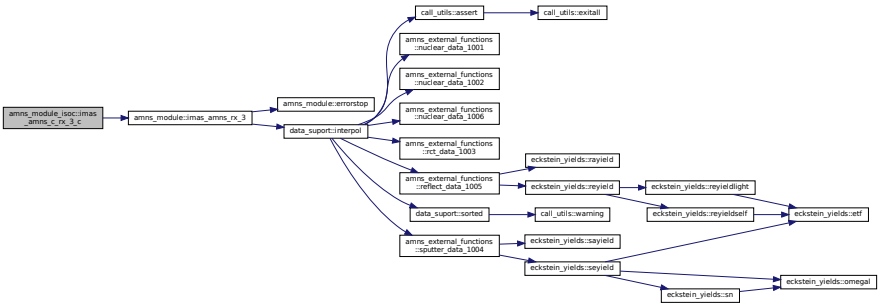
```
subroutine amns_module_isoc::imas_amns_c_rx_3_c (
            type(c_ptr), intent(in), value handle_rx,
            integer(c_int), intent(in), value nx,
            integer(c_int), intent(in), value ny,
            integer(c_int), intent(in), value nz,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(out) out,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg1,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg2,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg3,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (3d) args for a particular reaction Expanded into three separate subroutine for 1,2 or 3 arguments.

Definition at line 570 of file amns_module_isoc.f90.

```
00571     type(c_ptr), intent(in), value :: handle_rx
00572     integer(c_int), intent(in), value :: nx, ny, nz
00573     real (c_double), intent(out) :: out(0:nx-1,0:ny-1,0:nz-1)
00574     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1,0:nz-1), arg2(0:nx-1,0:ny-1,0:nz-1),
      arg3(0:nx-1,0:ny-1,0:nz-1)
00575     type(amns_fc_error_type), intent(out) :: error_status_fc
00576
00577     ! internal
00578     type(amns_handle_rx_type), pointer :: lhandle_rx
00579     type(amns_error_type) :: error_status
00580
00581     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_C: enter"
00582     call c_f_pointer(handle_rx, lhandle_rx)
00583     call imas_amns_rx_3(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, error_status=error_status)
00584     error_status_fc%flag = error_status%flag
00585     error_status_fc%string = copy(error_status%string)
00586     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_C: return"
```

Here is the call graph for this function:



13.3.2.22 imas_amns_c_rx_3_d()

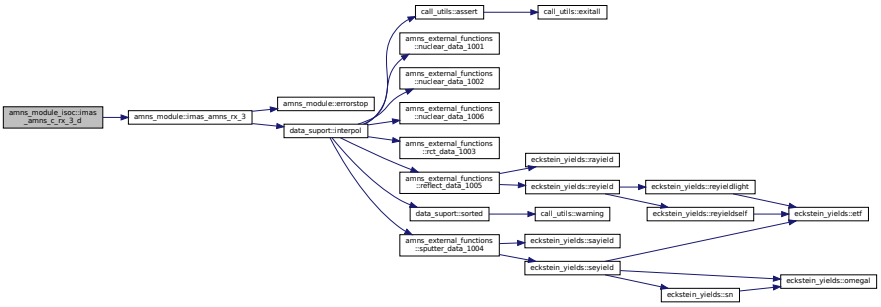
```
subroutine amns_module_isoc::imas_amns_c_rx_3_d (
            type(c_ptr), intent(in), value handle_rx,
            integer(c_int), intent(in), value nx,
            integer(c_int), intent(in), value ny,
            integer(c_int), intent(in), value nz,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(out) out,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg1,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg2,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg3,
            real (c_double), dimension(0:nx-1,0:ny-1,0:nz-1), intent(in) arg4,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

get the rates associated with the input (3d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

Definition at line 592 of file amns_module_isoc.f90.

```
00593     type(c_ptr), intent(in), value :: handle_rx
00594     integer(c_int), intent(in), value :: nx, ny, nz
00595     real (c_double), intent(out) :: out(0:nx-1,0:ny-1,0:nz-1)
00596     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1,0:nz-1), arg2(0:nx-1,0:ny-1,0:nz-1),
      arg3(0:nx-1,0:ny-1,0:nz-1), arg4(0:nx-1,0:ny-1,0:nz-1)
00597     type(amns_fc_error_type), intent(out) :: error_status_fc
00598
00599     ! internal
00600     type(amns_handle_rx_type), pointer :: lhandle_rx
00601     type(amns_error_type) :: error_status
00602
00603     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_D: enter"
00604     call c_f_pointer(handle_rx, lhandle_rx)
00605     call imas_amns_rx_3(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, arg4=arg4,
      error_status=error_status)
00606     error_status_fc%flag = error_status%flag
00607     error_status_fc%string = copy(error_status%string)
00608     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_D: return"
```

Here is the call graph for this function:



13.3.2.23 imas_amns_c_set()

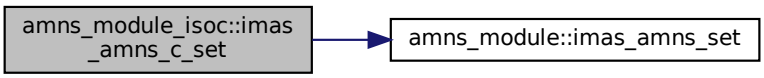
```
subroutine amns_module_isoc::imas_amns_c_set (
            type(c_ptr), intent(in), value handle,
            type(amns_fc_set_type), intent(in) set_fc,
            type(amns_fc_error_type), intent(out) error_status_fc )
```

provides IMAS_AMNS_C_SET by calling IMAS_AMNS_SET

Definition at line 134 of file amns_module_isoc.f90.

```
00135     type(c_ptr), intent(in), value :: handle
00136     type(amns_fc_set_type), intent(in) :: set_fc
00137     type(amns_fc_error_type), intent(out) :: error_status_fc
00138
00139     ! internal
00140     type(amns_handle_type), pointer :: lhandle
00141     type(amns_set_type) :: set
00142     type(amns_error_type) :: error_status
00143
00144     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET: enter"
00145     call c_f_pointer(handle, lhandle)
00146     set%string = copy(set_fc%string)
00147     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET: set%string ='"//set%string//"'"
00148     call c_f_pointer(handle, lhandle)
00149     call imas_amns_set(lhandle, set, error_status)
00150     error_status_fc%flag = error_status%flag
00151     error_status_fc%string = copy(error_status%string)
00152     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET: return"
```

Here is the call graph for this function:



13.3.2.24 imas_amns_c_set_reactant()

```
subroutine amns_module_isoc::imas_amns_c_set_reactant (
             type(c_ptr), intent(in), value reactants_handle,
             integer(c_int), intent(in), value reactant_index,
             type(amns_reactant_type), intent(in) reactant )
```

Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

Definition at line 644 of file amns_module_isoc.f90.

```
00645     type(c_ptr), intent(in), value :: reactants_handle
00646     integer(c_int), intent(in), value :: reactant_index
00647     type(amns_reactant_type), intent(in) :: reactant
00648
00649     ! internal
00650     type(amns_reactants_type), pointer :: lreactants
00651
00652     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET_REACTANT: enter"
00653     call c_f_pointer(reactants_handle, lreactants)
00654     lreactants%components(reactant_index) = reactant
00655     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET_REACTANT: return"
```

13.3.2.25 imas_amns_c_set_table()

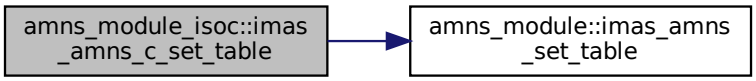
```
subroutine amns_module_isoc::imas_amns_c_set_table (
             type(c_ptr), intent(in), value handle_rx,
             type(amns_fc_set_type), intent(in) set_fc,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

provides IMAS_AMNS_C_SET_TABLE by calling IMAS_AMNS_SET_TABLE

Definition at line 240 of file amns_module_isoc.f90.

```
00241     type(c_ptr), intent(in), value :: handle_rx
00242     type(amns_fc_set_type), intent(in) :: set_fc
00243     type(amns_fc_error_type), intent(out) :: error_status_fc
00244
00245     ! internal
00246     type(amns_handle_rx_type), pointer :: lhandle_rx
00247     type(amns_set_type) :: set
00248     type(amns_error_type) :: error_status
00249
00250     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET_TABLE: enter"
00251     call c_f_pointer(handle_rx, lhandle_rx)
00252     set%string = copy(set_fc%string)
00253     call imas_amns_set_table(lhandle_rx,set,error_status)
00254     error_status_fc%flag = error_status%flag
00255     error_status_fc%string = copy(error_status%string)
00256     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET_TABLE: return"
```

Here is the call graph for this function:



13.3.2.26 imas_amns_c_setup()

```
subroutine amns_module_isoc::imas_amns_c_setup (
             type(c_ptr), intent(out) handle,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

provides IMAS_AMNS_C_SETUP by calling IMAS_AMNS_SETUP

Definition at line 46 of file amns_module_isoc.f90.

```
00047     type(c_ptr), intent(out) :: handle
00048     type(amns_fc_error_type), intent(out) :: error_status_fc
00049
00050     ! internal
00051     type(amns_handle_type), pointer :: lhandle
00052     type(amns_error_type) :: error_status
00053
00054     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP: enter"
00055     allocate(lhandle)
00056     !lhandle%debug = IMAS_AMNS_C_DEBUG
00057     call imas_amns_setup(lhandle, error_status=error_status)
00058     !lhandle%debug = IMAS_AMNS_C_DEBUG ! this due to issue in IMAS_AMNS_SETUP that lhandle is
       intent(out)
00059     handle = c_loc(lhandle)
00060     error_status_fc%flag = error_status%flag
00061     error_status_fc%string = copy(error_status%string)
00062     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP: return"
```

Here is the call graph for this function:

13.3.2.27 imas_amns_c_setup_reactants()

```
subroutine amns_module_isoc::imas_amns_c_setup_reactants (
             type(c_ptr), intent(out) reactants_handle,
             character(kind=c_char), dimension(*), intent(in) string,
             integer(c_int), intent(in), value index,
             integer(c_int), intent(in), value n_reactants )
```

Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

Definition at line 615 of file amns_module_isoc.f90.

```
00616     type(c_ptr), intent(out) :: reactants_handle
00617     character(kind=c_char), intent(in) :: string(*)
00618     integer(c_int), intent(in), value :: index, n_reactants
00619     integer i
```

```
00620
00621     ! internal
00622     type(amns_reactants_type), pointer :: lreactants
00623
00624     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_REACTANTS: enter, string"
00625     allocate(lreactants)
00626     allocate(lreactants%components(n_reactants))
00627     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_REACTANTS: ", lreactants%string
00628     ! FIXME: %string is an array, provide access functions
00629     allocate(lreactants%string(1))
00630     lreactants%string(1) = ”
00631     do i=1, len(lreactants%string(1))
00632        if(string(i) .EQ. c_null_char) exit
00633        lreactants%string(1)(i:i) = string(i)
00634     end do
00635     lreactants%index = index
00636
00637     reactants_handle = c_loc(lreactants)
00638     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_REACTANTS: return"
```

13.3.2.28 imas_amns_c_setup_table()

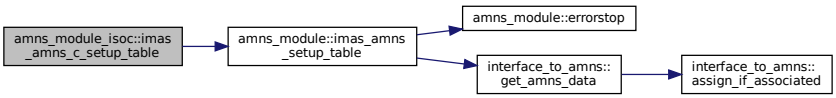
```
subroutine amns_module_isoc::imas_amns_c_setup_table (
             type(c_ptr), intent(in), value handle,
             type(amns_fc_reaction_type), intent(in) reaction_type_fc,
             type(c_ptr), intent(in), value reactant,
             type(c_ptr), intent(out) handle_rx,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

provides IMAS_AMNS_C_SETUP_TABLE by calling IMAS_AMNS_SETUP_TABLE

Definition at line 186 of file amns_module_isoc.f90.

```
00187     type(c_ptr), intent(in), value :: handle
00188     type(amns_fc_reaction_type), intent(in) :: reaction_type_fc
00189     type(c_ptr), intent(in), value :: reactant
00190     type(c_ptr), intent(out) :: handle_rx
00191     type(amns_fc_error_type), intent(out) :: error_status_fc
00192
00193     ! internal
00194     type(amns_handle_type), pointer :: lhandle
00195     type(amns_reactants_type), pointer :: lreactant
00196     type(amns_handle_rx_type), pointer :: lhandle_rx
00197     type(amns_reaction_type) :: reaction_type
00198     type(amns_error_type) :: error_status
00199
00200     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_TABLE: enter"
00201     call c_f_pointer(handle, lhandle)
00202     call c_f_pointer(reactant, lreactant)
00203     allocate(lhandle_rx)
00204     reaction_type%string = copy(reaction_type_fc%string)
00205     reaction_type%isotope_resolved = reaction_type_fc%isotope_resolved
00206     call imas_amns_setup_table(lhandle, reaction_type, lreactant, lhandle_rx, error_status)
00207     handle_rx = c_loc(lhandle_rx)
00208     error_status_fc%flag = error_status%flag
00209     error_status_fc%string = copy(error_status%string)
00210     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_TABLE: return"
```

Here is the call graph for this function:



13.3.2.29 imas_amns_c_setup_version()

```
subroutine amns_module_isoc::imas_amns_c_setup_version (
             type(c_ptr), intent(out) handle,
             type(amns_fc_version_type), intent(in) version_fc,
             type(amns_fc_error_type), intent(out) error_status_fc )
```

provides IMAS_AMNS_C_SETUP by calling IMAS_AMNS_SETUP with version argument
Definition at line 67 of file amns_module_isoc.f90.

```
00068     type(c_ptr), intent(out) :: handle
00069     type(amns_fc_version_type), intent(in) :: version_fc
00070     type(amns_fc_error_type), intent(out) :: error_status_fc
00071
00072     ! internal
00073     type(amns_handle_type), pointer :: lhandle
00074     type(amns_version_type) :: version
00075     type(amns_error_type) :: error_status
00076
00077     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_VERSION: enter"
00078     allocate(lhandle)
00079     !lhandle%debug = IMAS_AMNS_C_DEBUG
00080     version%string = copy(version_fc%string)
00081     version%number = version_fc%number
00082     version%backend = copy(version_fc%backend)
00083     version%user = copy(version_fc%user)
00084     call imas_amns_setup(lhandle, version, error_status=error_status)
00085     !lhandle%debug = IMAS_AMNS_C_DEBUG ! this due to issue in IMAS_AMNS_SETUP that lhandle is
      intent(out)
00086     error_status_fc%flag = error_status%flag
00087     error_status_fc%string = copy(error_status%string)
00088     handle = c_loc(lhandle)
00089     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_VERSION: return"
```

Here is the call graph for this function:

13.4 AMNS Fortran Types

Data types used for the ITM AMNS interface.

Data Types

- type amns_types::amns_version_type

  Type for specifying the AMNS version (not interoperable)
- type amns_types::amns_fc_version_type

  Type for specifying the AMNS version (interoperable with c)
- type amns_types::amns_reactant_type

  Type for indicating a single reactant or product when using the AMNS interface.
- type amns_types::amns_reactants_type

  Type for indicating the reactants when using the AMNS interface NOT interoperable with C.
- type amns_types::amns_handle_type

  type for the AMNS handle (opaque for user codes) NOT interoperable with C.
- type amns_types::amns_handle_rx_type

  Type for the AMNS RX handle (opaque for user codes) NOT interoperable with C.
- type amns_types::amns_error_type

  Type for error returns from the AMNS interface (not interoperable)
- type amns_types::amns_fc_error_type

  Type for error returns from the AMNS interface (interoperable with c)
- type amns_types::amns_reaction_type

  Type used for specifying reactions when using the AMNS interface (not interoperable)
- type amns_types::amns_fc_reaction_type

  Type used for specifying reactions when using the AMNS interface (interoperable with c)
- type amns_types::amns_set_type

  Type for setting parameters in the AMNS package (not interoperable)
- type amns_types::amns_fc_set_type

  Type for setting parameters in the AMNS package (interoperable with c)
- type amns_types::amns_query_type

  Type for querying parameters in the AMNS package (not interoperable)
- type amns_types::amns_fc_query_type

  Type for querying parameters in the AMNS package (interoperable with c)
- type amns_types::amns_answer_type

  Type for answers from queries in the AMNS package (not interoperable)
- type amns_types::amns_fc_answer_type

  Type for answers from queries in the AMNS package (interoperable with c)

Variables

- integer, parameter amns_types::version_length =32

  used to specify the maximum length of the string version number
- integer, parameter amns_types::set_length =32

  maximum length of the string passed by the set calls
- integer, parameter amns_types::reaction_length =16

  maximum length for specifying a reaction type
- integer, parameter amns_types::query_length =16

  maximum length for a query argument
- integer, parameter amns_types::answer_length =128

  maximum length of an answer

13.4.1 Detailed Description

Data types used for the ITM AMNS interface.

Author

David Coster

13.4.2 Variable Documentation

13.4.2.1 answer_length

```
integer, parameter amns_types::answer_length =128
```

maximum length of an answer
Definition at line 40 of file amns_types.f90.

```
00040   integer, parameter :: answer_length=128
```

13.4.2.2 query_length

```
integer, parameter amns_types::query_length =16
```

maximum length for a query argument
Definition at line 37 of file amns_types.f90.

```
00037   integer, parameter :: query_length=16
```

13.4.2.3 reaction_length

```
integer, parameter amns_types::reaction_length =16
```

maximum length for specifying a reaction type
Definition at line 34 of file amns_types.f90.

```
00034   integer, parameter :: reaction_length=16
```

13.4.2.4 set_length

```
integer, parameter amns_types::set_length =32
```

maximum length of the string passed by the set calls
Definition at line 31 of file amns_types.f90.

```
00031   integer, parameter :: set_length=32
```

13.4.2.5 version_length

```
integer, parameter amns_types::version_length =32
```

used to specify the maximum length of the string version number
Definition at line 28 of file amns_types.f90.

```
00028   integer, parameter :: version_length=32
```

13.5 AMNS Internal Implementation of Interpolation Grids

data_suport module from Silvio Gori's grid package

Modules

- module call_utils

  utils module from Silvio Gori's grid package
- module f90_kind

  f90_kind module from Silvio Gori's grid package
- module strings

  strings module from Silvio Gori's grid package
- module unit_h

  unit_h module from Silvio Gori's grid package

13.5.1 Detailed Description

data_suport module from Silvio Gori's grid package

Author

Silvio Gori, updated by David Tskhakaya (updated grit_t, new grid(),..., deleted reset_grid∗, get_grid∗, read↩ _grid∗)

13.6 AMNS External Utility Functions/Subroutines

External functions/subroutines called by the ITM AMNS External Functions/Subroutines.

Functions/Subroutines

- real(ids_real) function eckstein_yields::etf (M1, M2, Z1, Z2)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::omegal (epsl)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::sn (epsl)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::seyield (E0, M1, M2, Z1, Z2, q, lambda, u, ETh)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::sayield (E0, theta, f, b, c, Esp)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::reyieldlight (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::reyieldself (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::reyield (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::rayield (angledeg, C1, C2, C3, C4)

  AMNS External utility function ...

13.6.1 Detailed Description

External functions/subroutines called by the ITM AMNS External Functions/Subroutines.

Author

David Coster and others

13.6.2 Function/Subroutine Documentation

13.6.2.1 etf()

```
real(ids_real) function eckstein_yields::etf (
            real(ids_real) M1,
            real(ids_real) M2,
            real(ids_real) Z1,
            real(ids_real) Z2 )
```

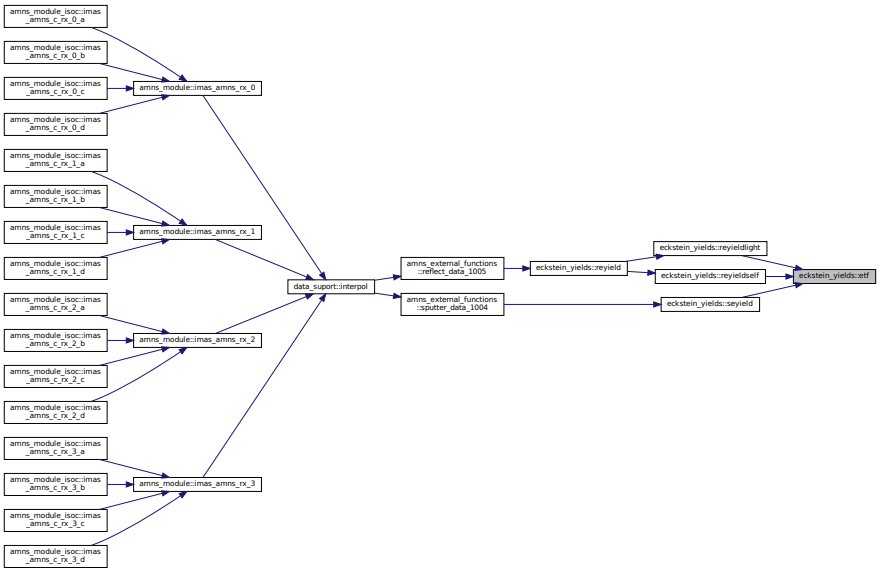
AMNS External utility function ...
See ...

Todo provide more information

Definition at line 22 of file eckstein_yields.f90.

```
00023     implicit none
00024     real(ids_real) :: etf, M1, M2, Z1, Z2
00025     etf = (30.74*(m1 + m2)*z1*sqrt(z1**0.6666666666666666 + z2**0.6666666666666666)*z2)/m2
```

Here is the caller graph for this function:



13.6.2.2 omegal()

```
real(ids_real) function eckstein_yields::omegal (
            real(ids_real) epsl )
```

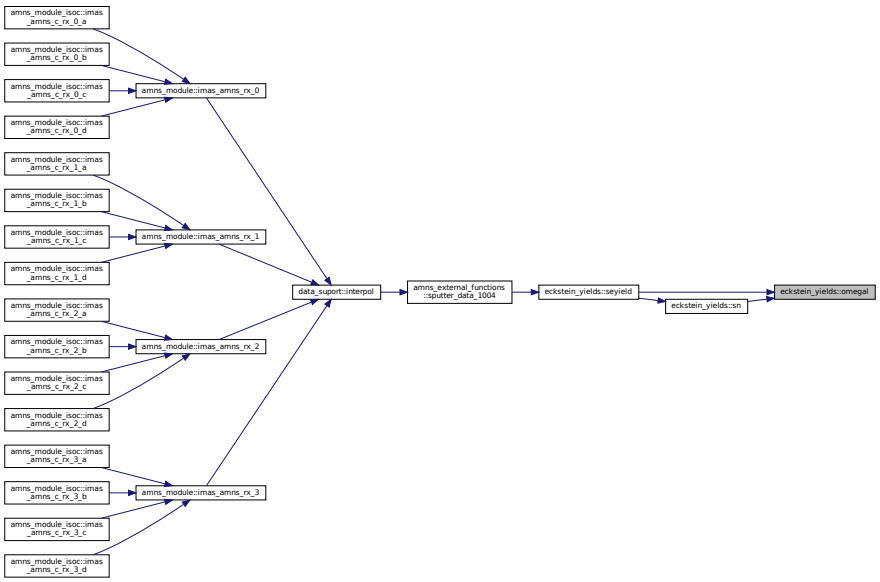
AMNS External utility function ...
See ...

Todo provide more information

Definition at line 33 of file eckstein_yields.f90.

```
00034     implicit none
00035     real(ids_real) :: omegaL, epsl
00036     omegal = 0.008*(epsl**0.1504) + 0.1728*sqrt(epsl) + epsl
```

Here is the caller graph for this function:



13.6.2.3 rayield()

```
real(ids_real) function eckstein_yields::rayield (
            real(ids_real) angledeg,
            real(ids_real) C1,
            real(ids_real) C2,
            real(ids_real) C3,
            real(ids_real) C4 )
```

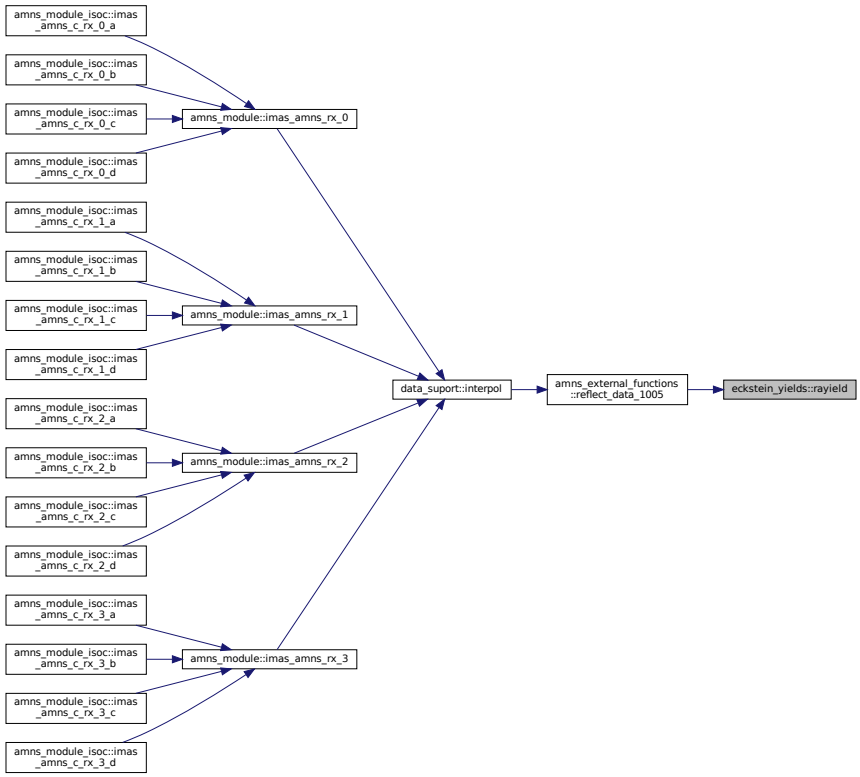
AMNS External utility function ...
See ...

Todo provide more information

Definition at line 136 of file eckstein_yields.f90.

```
00137     implicit none
00138     real(ids_real) :: rayield, angledeg, C1, C2, C3, C4
00139
00140     rayield = c1 + c2 * tanh(c3 * angledeg * degtorad + c4)
00141
```

Here is the caller graph for this function:



13.6.2.4 reyield()

```
real(ids_real) function eckstein_yields::reyield (
            real(ids_real) E0,
            real(ids_real) M1,
            real(ids_real) M2,
            real(ids_real) Z1,
            real(ids_real) Z2,
```

```
            real(ids_real) A1,
            real(ids_real) A2,
            real(ids_real) A3,
            real(ids_real) A4 )
```

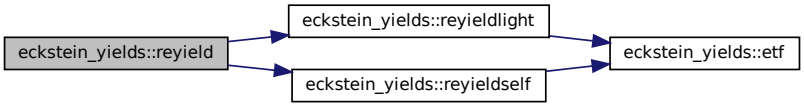
AMNS External utility function ...
See ...

Todo provide more information

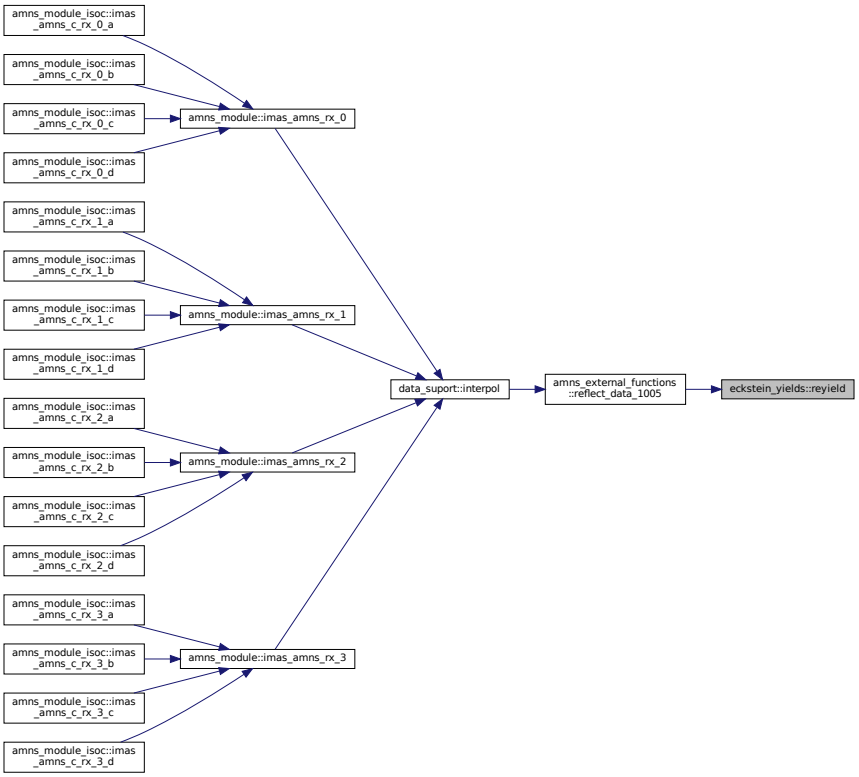
Definition at line 119 of file eckstein_yields.f90.

```
00120     implicit none
00121     real(ids_real) :: reyield, E0, M1, M2, Z1, Z2, A1, A2, A3, A4, eps
00122     if (m1 .ge. m2) then
00123 !       write(*,*) 'M1 = ', M1, ' is >= M2 = ',M2,' Using  ryieldself'
00124        reyield = reyieldself(e0, m1, m2, z1, z2, a1, a2, a3, a4)
00125     else
00126 !       write(*,*) 'M1 = ', M1, ' is <  M2 = ',M2,' Using  ryieldlight'
00127        reyield = reyieldlight(e0, m1, m2, z1, z2, a1, a2, a3, a4)
00128     end if
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.6.2.5 reyieldlight()

```
real(ids_real) function eckstein_yields::reyieldlight (
            real(ids_real) E0,
            real(ids_real) M1,
            real(ids_real) M2,
            real(ids_real) Z1,
            real(ids_real) Z2,
            real(ids_real) A1,
            real(ids_real) A2,
            real(ids_real) A3,
            real(ids_real) A4 )
```

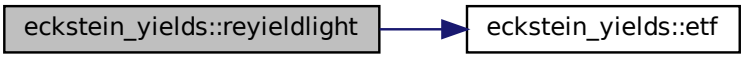
AMNS External utility function ...
See ...

Todo provide more information

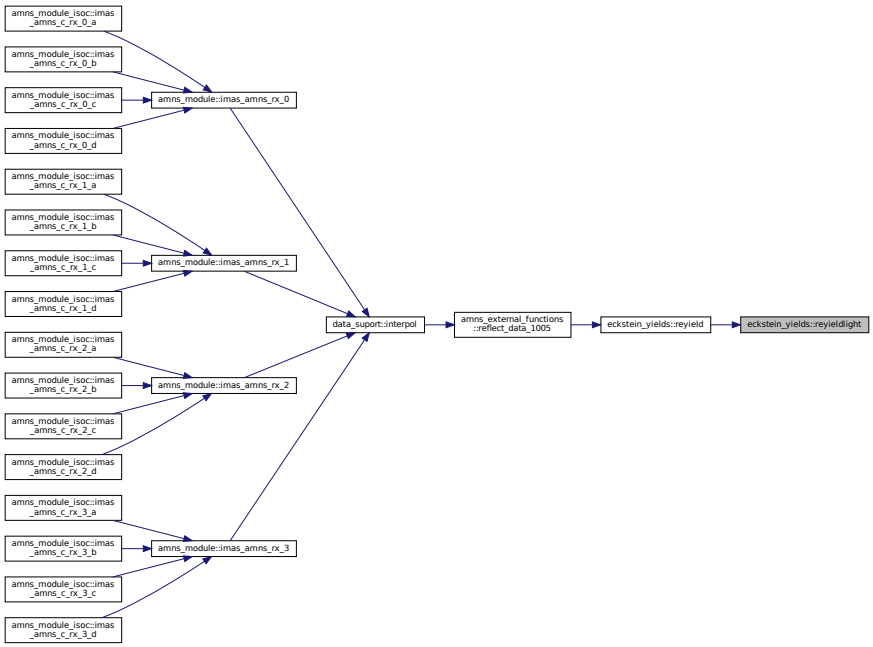
Definition at line 85 of file eckstein_yields.f90.

```
00086     implicit none
00087     real(ids_real) :: reyieldlight, E0, M1, M2, Z1, Z2, A1, A2, A3, A4, eps
00088
00089     eps = e0/etf(m1, m2, z1, z2)
00090
00091     reyieldlight = (a1 * eps**a2) / (1.0 + a3 * eps**a4)
00092
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.6.2.6 reyieldself()

```
real(ids_real) function eckstein_yields::reyieldself (
              real(ids_real) E0,
              real(ids_real) M1,
              real(ids_real) M2,
              real(ids_real) Z1,
              real(ids_real) Z2,
              real(ids_real) A1,
              real(ids_real) A2,
              real(ids_real) A3,
              real(ids_real) A4 )
```

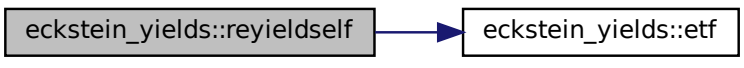
AMNS External utility function ...
See ...

Todo provide more information

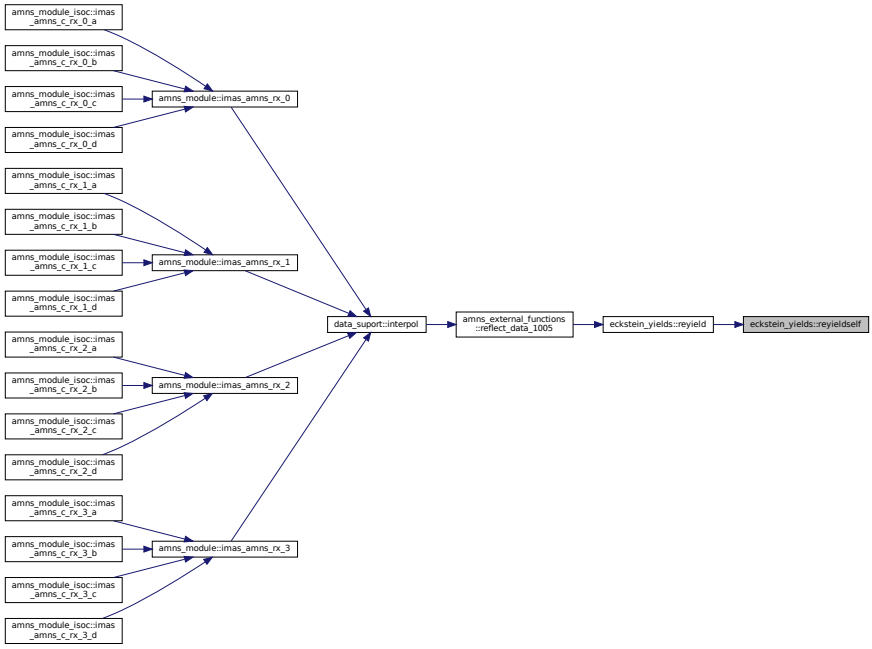
Definition at line 100 of file eckstein_yields.f90.

```
00101     implicit none
00102     real(ids_real) :: reyieldself, E0, M1, M2, Z1, Z2, A1, A2, A3, A4, eps
00103
00104     eps = e0/etf(m1, m2, z1, z2)
00105
00106     if(a3 * eps**a4 .lt. 700) then  ! prevent overflow DPC
00107        reyieldself = exp(a1*eps**a2)/(1.0 + exp(a3 * eps**a4))
00108     else
00109        reyieldself = exp(a1*eps**a2 - a3 * eps**a4)
00110     endif
00111
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.6.2.7 sayield()

```
real(ids_real) function eckstein_yields::sayield (
             real(ids_real) E0,
             real(ids_real) theta,
             real(ids_real) f,
             real(ids_real) b,
             real(ids_real) c,
             real(ids_real) Esp )
```

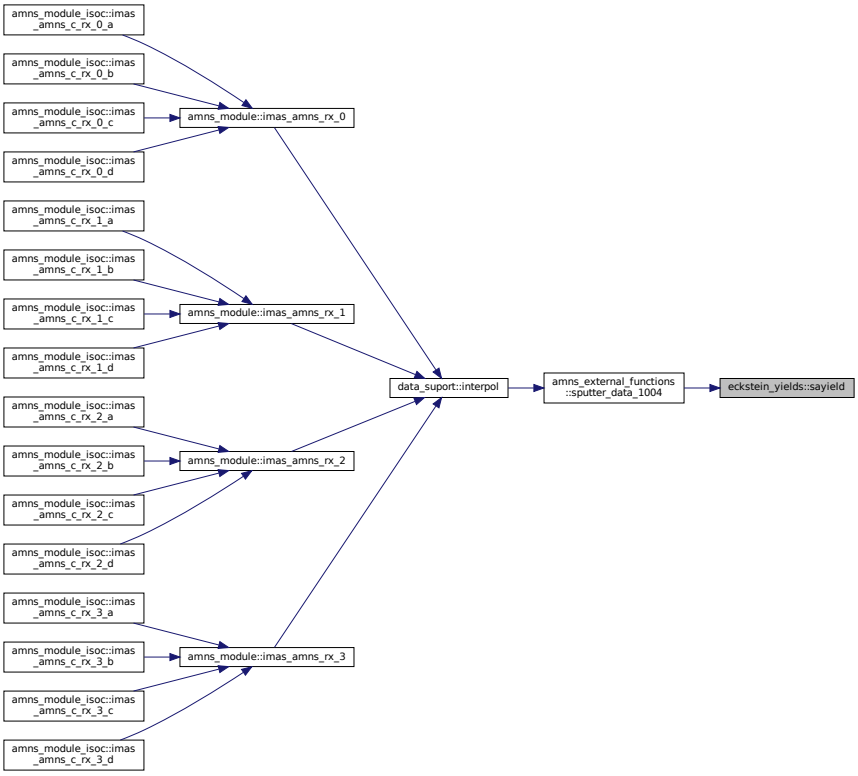
AMNS External utility function ...
See ...

Todo provide more information

Definition at line 70 of file eckstein_yields.f90.

```
00071     implicit none
00072     real(ids_real) :: sayield, E0, theta, thetarad, f, b, c, Esp
00073     real(ids_real) :: thetastar, cosfact
00074     thetarad = theta * degtorad
00075     thetastar = mathpi - acos(sqrt(1/(1 + e0/esp)))
00076     cosfact = cos(((mathpi/2.)**c)*((thetarad/thetastar)**c))
00077     sayield = exp(b*(1 - 1/cosfact))/(cosfact**f)
```

Here is the caller graph for this function:



13.6.2.8 seyield()

```
real(ids_real) function eckstein_yields::seyield (
            real(ids_real) E0,
            real(ids_real) M1,
            real(ids_real) M2,
            real(ids_real) Z1,
            real(ids_real) Z2,
            real(ids_real) q,
            real(ids_real) lambda,
            real(ids_real) u,
            real(ids_real) ETh )
```

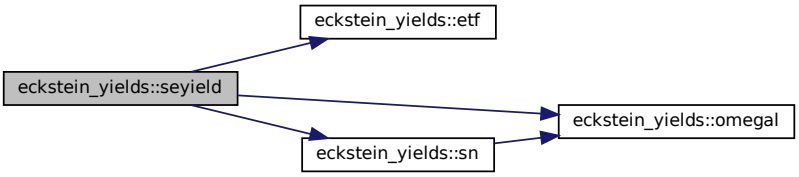
AMNS External utility function ...
See ...

Todo provide more information

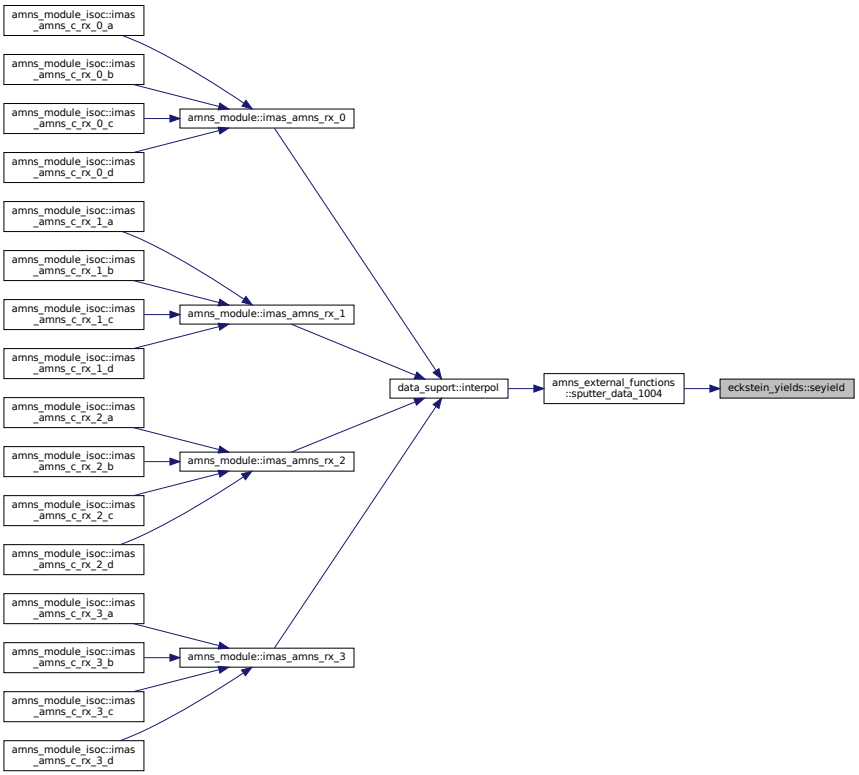
Definition at line 55 of file eckstein_yields.f90.

```
00056     implicit none
00057     real(ids_real) :: seyield, E0, M1, M2, Z1, Z2, q, lambda, u, ETh, epsl, snucl, omega, etoeth
00058     epsl = e0/etf(m1, m2, z1, z2)
00059     snucl = sn(epsl)
00060     omega = omegal(epsl)
00061     etoeth =  ((e0/eth) - 1.0)**u
00062     seyield = (etoeth*q*snucl)/(etoeth + (lambda/omega))
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.6.2.9 sn()

```
real(ids_real) function eckstein_yields::sn (
            real(ids_real) epsl )
```

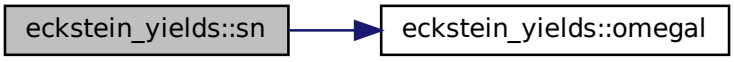
AMNS External utility function ...
See ...

Todo provide more information

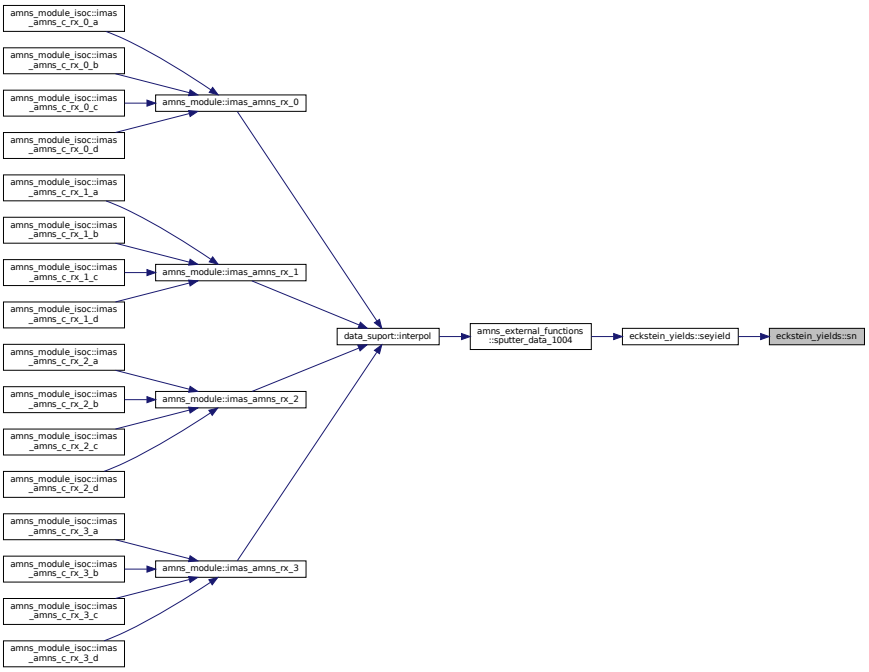
Definition at line 44 of file eckstein_yields.f90.

```
00045     implicit none
00046     real(ids_real) :: sn, epsl
00047     sn = (0.5*log(1 + 1.2288*epsl))/omegal(epsl)
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.7 AMNS Utility Routines

Module implementing a generalized interface to AMNS IDSs.

Modules

- module amns_utility

  Module implementing various utility functions for the AMNS interface.

Functions/Subroutines

- character ∗24 function amns_utility::int_to_string (int)

  convert an integer to a string

- subroutine interface_to_amns::get_amns_data (reaction_type, reactants, grid, properties_comment, properties_source, properties_provider, properties_creation_date, code_name, code_commit, code_version, code_repository, source, provider, citation, shot, run, backend, user, ds_version, ierr, error_description, debug)

  transfer data from the IDS to the internal data structure

- character(len=answer_length) function interface_to_amns::assign_if_associated (ids_str, def_str)

  utility to assign if associated

- subroutine interface_to_amns::end_amns_data

  deallocate idss

13.7.1 Detailed Description

Module implementing a generalized interface to AMNS IDSs.

Author

David Coster, updated by David Tskhakaya \ (updates: get_adas_data --> get_amns_data)

13.7.2 Function/Subroutine Documentation

13.7.2.1 assign_if_associated()

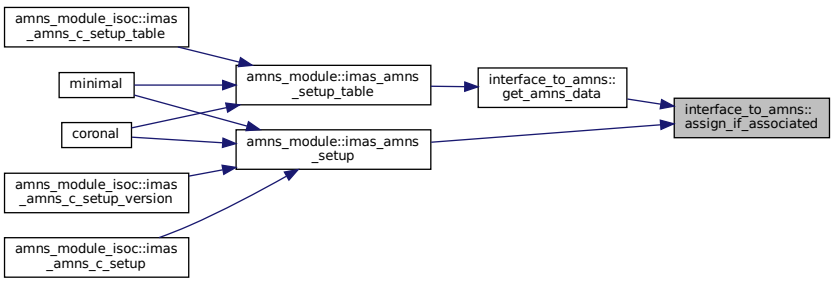
```
character(len=answer_length) function interface_to_amns::assign_if_associated (
            character(len=*), dimension(:), intent(in), pointer ids_str,
            character(len=*), intent(in) def_str )
```

utility to assign if associated

Definition at line 504 of file interface_to_amns.f90.

```
00505     implicit none
00506     character(len=*), dimension(:), pointer, intent(in) :: ids_str
00507     character(len=*), intent(in) :: def_str
00508     character(len=answer_length) :: assign_if_associated
00509     if(associated(ids_str)) then
00510        if(size(ids_str).ge.1) then
00511           assign_if_associated = ids_str(1)
00512        else
00513           assign_if_associated = def_str
00514        endif
00515     else
00516        assign_if_associated= def_str
00517     endif
```

Here is the caller graph for this function:



13.7.2.2 end_amns_data()

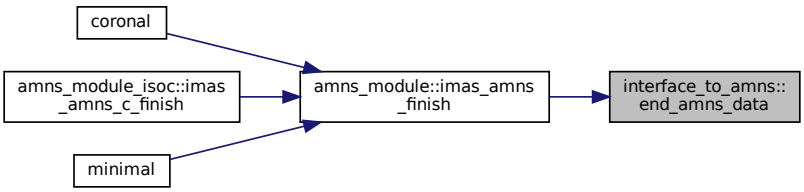
```
subroutine interface_to_amns::end_amns_data
```

deallocate idss

Definition at line 523 of file interface_to_amns.f90.

```
00524
00525 !    use deallocate_structures ! IGNORE
00526     implicit none
00527
00528     type (amns_ids_list), pointer :: work, next => null()
00529
00530     work => first
00531
00532     do while(associated(work))
00533        next => work%next
00534        call ids_deallocate(work%amns_ids)
00535        deallocate(work)
00536        work => next
00537     enddo
00538
00539     first => null()
00540
```

Here is the caller graph for this function:



13.7.2.3 get_amns_data()

```
subroutine interface_to_amns::get_amns_data (
             type(amns_reaction_type), intent(in) reaction_type,
             type(amns_reactants_type), intent(in) reactants,
             type (grid_t), intent(out) grid,
             character*(answer_length), intent(out) properties_comment,
             character*(answer_length), intent(out) properties_source,
             character*(answer_length), intent(out) properties_provider,
```

```
            character*(answer_length), intent(out) properties_creation_date,
            character*(answer_length), intent(out) code_name,
            character*(answer_length), intent(out) code_commit,
            character*(answer_length), intent(out) code_version,
            character*(answer_length), intent(out) code_repository,
            character*(answer_length), intent(out) source,
            character*(answer_length), intent(out) provider,
            character*(answer_length), intent(out) citation,
            integer shot,
            integer run,
            character (len=*) backend,
            character (len=*) user,
            character (len=*) ds_version,
            integer ierr,
            character*128 error_description,
            logical debug )
```

transfer data from the IDS to the internal data structure
inputs: reaction_type, reactants, shot, run, backend, user, ds_version outputs: grid, source
Definition at line 30 of file interface_to_amns.f90.

```
00038     use ids_types  ! IGNORE
00039     use amns_utility
00040     use f90_kind
00041 ! do not remove the comment!!!
00042     use ids_schemas   ! IGNORE
00043     use ids_routines  ! IGNORE
00044 !    use read_structures ! IGNORE
00045     implicit none
00046
00047     type (grid_t), intent(out) :: grid
00048     character*(answer_length), intent(out) :: &
00049          properties_comment, properties_source, &
00050          properties_provider, properties_creation_date, &
00051          code_name, code_commit, &
00052          code_version, code_repository, &
00053          source, provider, citation
00054     type(amns_reaction_type), intent(in) :: reaction_type
00055     type(amns_reactants_type), intent(in) :: reactants
00056     real (kind=ids_real) :: zn, za
00057     character (len=*) :: backend, user, ds_version
00058     integer :: ierr
00059     character*128 :: error_description
00060     logical :: debug
00061     integer :: iza, nz, nproc, i, k, j, isearch, ndim, ncoord
00062     type (amns_ids_list), pointer :: last, work => null()
00063 ! for calling AMNS IDS
00064     character(len=3)::      treename
00065     integer ::              idx, shot, run
00066     logical :: found, done, match
00067     character (len=256) :: file
00068     integer :: nreac, nprod, nlhs, nrhs, ireac
00069     real (kind=ids_real) :: zn_ids, za_ids
00070     integer :: ids_status
00071     character(STRMAXLEN):: uri
00072
00073     ierr=0
00074     error_description=”
00075     iza = -1
00076
00077     work => first
00078     last => first
00079     found = .false.
00080 !!    write(*,*) associated(first), associated(last), associated(work)
00081     do while(associated(work))
00082 !!       write(*,*) work%shot, shot
00083        if(work%shot.eq.shot.and.work%run.eq.run) then
00084           found=.true.
00085           exit
00086        endif
00087        last => work
00088        work => work%next
00089 !!       write(*,*) associated(first), associated(last), associated(work)
00090     enddo
00091 !!    write(*,*) associated(first), associated(last), associated(work)
00092     if(.not.found) then
00093        if(.not.associated(first)) then
00094           allocate(first)
00095 !!       write(*,*) first%shot, first%run, associated(first%prev), associated(first%next)
00096           work => first
```

```
00097 !!       write(*,*) associated(first), associated(work), work%shot
00098        else if(.not.associated(last%next)) then
00099           allocate(last%next)
00100           work => last%next
00101           work%prev => last
00102        endif
00103        if(work%shot.eq.0) then
00104           treename = 'ids'
00105           if(backend.eq.'ascii') then
00106 !             write(file,'(a,"_",i6.6,"_",i6.6,".IDS")') 'amns',shot,run
00107 !             if(debug) write(*,*) 'Reading data from ', trim(file), ' for ', shot, run
00108 !             call open_read_file(1, trim(file))
00109 !             call read_ids(work%amns_ids, 'amns')
00110 !             call close_read_file
00111           else
00112 #ifdef AL
00113              if(backend.eq.'mdsplus') then
00114                 if(debug) write(*,*) 'Using mdsplus backend'
00115                 !               call
       imas_open_env(treename,shot,run,idx,trim(USER),'amns',ds_version)
00116                 !call al_begin_pulse_action(MDSPLUS_BACKEND, shot, run, &
00117                 !     trim(USER), 'amns', trim(ds_version), idx)
00118                 !call al_open_pulse(idx, OPEN_PULSE, '', ids_status)
00119                 call al_build_uri_from_legacy_parameters(mdsplus_backend, shot, run, &
00120                      trim(user), 'amns', trim(ds_version), '', uri, ids_status)
00121                 call al_begin_dataentry_action(uri, open_pulse, idx, ids_status)
00122
00123              elseif(backend.eq.'hdf5') then
00124                 if(debug) write(*,*) 'Using hdf5 backend'
00125 !                call imas_open_hdf5(treename,shot,run,idx)
00126                 stop 'HFD5 nor currently supported'
00127              else
00128                 error_description = 'Backend not yet supported: ' // trim(backend)
00129                 write(*,*) error_description
00130                 ierr=1
00131                 return
00132              endif
00133              call ids_get(idx,'amns_data',work%amns_ids)
00134              call imas_close(idx)
00135 #else
00136              error_description = 'The AMNS routines were compiled without AL support. ' // &
00137                   'Use the ascii backend'
00138              write(*,*) error_description
00139              ierr=2
00140              return
00141 #endif
00142           endif
00143           work%shot=shot
00144           work%run=run
00145        endif
00146     endif
00147
00148     nproc= SIZE(work%amns_ids%process)
00149
00150     k=0
00151     do isearch = 1, nproc
00152        if (debug) write(*,'(a,i4)') 'Considering IDS process ', isearch
00153 ! first look for the right process
00154        if(reaction_type%string .ne. work%amns_ids%process(isearch)%label(1)) cycle
00155 ! found the right process, now search for the right reaction pattern
00156 ! first calculate the number of reactants and products in the database
00157        if(associated(work%amns_ids%process(isearch)%reactants)) then
00158           nreac = size(work%amns_ids%process(isearch)%reactants)
00159        else
00160           nreac = 0
00161        endif
00162        if(associated(work%amns_ids%process(isearch)%products)) then
00163           nprod = size(work%amns_ids%process(isearch)%products)
00164        else
00165           nprod = 0
00166        endif
00167 ! then calculate the number of reactants and products for the requested reaction
00168        nlhs=0
00169        nrhs=0
00170        if(allocated(reactants%components)) then
00171           do ireac = 1, size(reactants%components)
00172              if(reactants%components(ireac)%lr .eq. 0) then
00173                 if (nrhs .ne. 0) then
00174                    error_description = 'Reactants should precede products'
00175                    write (*,*) error_description
00176                    ierr=3
00177                    return
00178                 endif
00179                 nlhs = nlhs + 1
00180              else if(reactants%components(ireac)%lr .eq. 1) then
00181                 nrhs = nrhs + 1
00182              else
```

```
00183                write(error_description,*) 'Unknown lr option ', reactants%components(ireac)%lr, &
00184                     ' for component ', ireac
00185                write(*,*) error_description
00186                ierr=4
00187                return
00188             endif
00189          enddo
00190       endif
00191 ! the numbers should match
00192       if (debug) write(*,'(a,2i4,a,2i4)') &
00193            'nreac, nprod  IDS: ', nreac, nprod, &
00194            '  REQUEST: ', nlhs, nrhs
00195       if(nreac .ne. nlhs) cycle
00196       if(nprod .ne. nrhs) cycle
00197       match = .true.
00198 ! check the reactants
00199       do ireac = 1, nreac
00200          if (debug) write(*,'(a,i4)') 'Considering reactant ', ireac
00201 ! now check that the zn's match
00202          zn_ids = 0
00203          if(associated(work%amns_ids%process(isearch)%reactants(ireac)%element)) then
00204             if(size(work%amns_ids%process(isearch)%reactants(ireac)%element) .eq. 1) then
00205 !               if(work%amns_ids%process(isearch)%reactants(ireac)%element(1)%multiplicity .eq. 1)
     then
00206                   zn_ids = work%amns_ids%process(isearch)%reactants(ireac)%element(1)%z_n
00207 !                else
00208 !                   write(*,*) 'Case with multiplicity != 1 not yet coded'
00209 !                   write(*,*) ireac,
     work%amns_ids%process(isearch)%reactants(ireac)%element(1)%multiplicity
00210 !                   stop  'ERROR' ! for the moment
00211 !                endif
00212             else
00213                error_description = 'Case with number of elements != 1 not yet coded'
00214                write(*,*) error_description
00215                ierr=5
00216                return
00217             endif
00218          else
00219             error_description = 'Must have at least one constituent'
00220             write(*,*) error_description
00221             ierr=6
00222             return
00223          endif
00224          if (debug) write(*,'(a,f7.3,a,f7.3)') &
00225               'zn  IDS: ', zn_ids, &
00226               '  REQUEST: ', reactants%components(ireac)%zn
00227          match = match .and. (nint(reactants%components(ireac)%zn) .eq. nint(zn_ids))
00228          if (.not. match) cycle
00229 ! now check that the za's match
00230          za_ids = 0
00231          if(work%amns_ids%process(isearch)%reactants(ireac)%relative_charge .eq. -1) then
00232 ! charge is not relevant
00233             if (debug) write(*,'(a)') &
00234                  'za not relevant'
00235          else if(work%amns_ids%process(isearch)%reactants(ireac)%relative_charge .eq. 0) then
00236 ! charge is absolute
00237             if (debug) write(*,'(a,f7.3,a,f7.3)') &
00238                  'za (absolute charge)  IDS: ',
     work%amns_ids%process(isearch)%reactants(ireac)%charge, &
00239                  '  REQUEST: ', reactants%components(ireac)%za
00240             match = match .and. (work%amns_ids%process(isearch)%reactants(ireac)%charge .eq.
     reactants%components(ireac)%za)
00241          else if(work%amns_ids%process(isearch)%reactants(ireac)%relative_charge .eq. 1 ) then
00242 ! charge is relative
00243 ! ???
00244             iza = reactants%components(ireac)%za + 1
00245             if (debug) write(*,'(a,f7.3,a,f7.3)') &
00246                  'za (relative charge)  IDS: ',
     work%amns_ids%process(isearch)%reactants(ireac)%charge, &
00247                  '  REQUEST: ', reactants%components(ireac)%za
00248          else
00249             write(error_description,*) 'Unrecognized value for relative charge ',
     work%amns_ids%process(isearch)%reactants(ireac)%relative_charge
00250             write(*,*) error_description
00251             ierr=7
00252             return
00253          endif
00254          if (.not. match) cycle
00255 ! now check that the am's match
00256          if(reaction_type%isotope_resolved .eq. 1) then
00257             if(debug) write(*,'(a,f7.3,a,f7.3)') &
00258                  'am  IDS: ', work%amns_ids%process(isearch)%reactants(ireac)%mass, &
00259                  '  REQUEST: ', reactants%components(ireac)%mi
00260             if(nint(work%amns_ids%process(isearch)%reactants(ireac)%mass) .ne. 0 .and.
     nint(reactants%components(ireac)%mi) .ne. 0) then
00261                match = match .and. (nint(work%amns_ids%process(isearch)%reactants(ireac)%mass) .eq.
     nint(reactants%components(ireac)%mi))
```

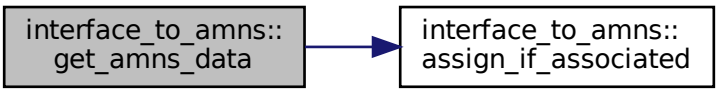
```
00262             endif
00263          endif
00264          if (.not. match) cycle
00265       enddo
00266       if (.not. match) cycle
00267 ! check the products
00268       do ireac = 1, nprod
00269          if (debug) write(*,'(a,i4)') 'Considering product ', ireac
00270 ! now check that the zn's match
00271          zn_ids = 0
00272          if(associated(work%amns_ids%process(isearch)%products(ireac)%element)) then
00273             if(size(work%amns_ids%process(isearch)%products(ireac)%element) .eq. 1) then
00274 !               if(work%amns_ids%process(isearch)%products(ireac)%element(1)%multiplicity .eq. 1)
      then
00275                  zn_ids = work%amns_ids%process(isearch)%products(ireac)%element(1)%z_n
00276 !               else
00277 !                  write(*,*) 'Case with multiplicity != 1 not yet coded'
00278 !                  stop  'ERROR' ! for the moment
00279 !               endif
00280             else
00281                error_description = 'Case with number of elements != 1 not yet coded'
00282                write(*,*) error_description
00283                ierr=8
00284                return
00285             endif
00286          else
00287             error_description = 'Must have at least one constituent'
00288             write(*,*) error_description
00289             ierr=9
00290             return
00291          endif
00292          if (debug) write(*,'(a,f7.3,a,f7.3)') &
00293               'zn  IDS: ', zn_ids, &
00294               '  REQUEST: ', reactants%components(ireac+nlhs)%zn
00295          match = match .and. (nint(reactants%components(ireac+nlhs)%zn) .eq. nint(zn_ids))
00296 ! now check that the za's match
00297          za_ids = 0
00298          if(work%amns_ids%process(isearch)%products(ireac)%relative_charge .eq. -1) then
00299 ! charge is not relevant
00300             if (debug) write(*,'(a)') &
00301                  'za not relevant'
00302          else if(work%amns_ids%process(isearch)%products(ireac)%relative_charge .eq. 0) then
00303 ! charge is absolute
00304             if (debug) write(*,'(a,f7.3,a,f7.3)') &
00305                  'za (absolute charge)  IDS: ',
      work%amns_ids%process(isearch)%products(ireac)%charge, &
00306                  '  REQUEST: ', reactants%components(ireac+nlhs)%za
00307             match = match .and. (work%amns_ids%process(isearch)%products(ireac)%charge .eq.  &
00308                  reactants%components(ireac+nlhs)%za)
00309          else if(work%amns_ids%process(isearch)%products(ireac)%relative_charge .eq. 1 ) then
00310 ! charge is relative
00311 ! ???
00312             if (debug) write(*,'(a,f7.3,a,f7.3)') &
00313                  'za (relative charge)  IDS: ',
      work%amns_ids%process(isearch)%products(ireac)%charge, &
00314                  '  REQUEST: ', reactants%components(ireac+nlhs)%za
00315          else
00316             write(error_description,*) 'Unrecognized value for relative charge ',
      work%amns_ids%process(isearch)%reactants(ireac)%relative_charge
00317             write(*,*) error_description
00318             ierr=10
00319             return
00320          endif
00321          if (.not. match) cycle
00322 ! now check that the am's match
00323          if(reaction_type%isotope_resolved .eq. 1) then
00324             if(debug) write(*,'(a,f7.3,a,f7.3)') &
00325                  'am  IDS: ', work%amns_ids%process(isearch)%products(ireac)%mass, &
00326                  '  REQUEST: ', reactants%components(ireac+nlhs)%mi
00327             if(nint(work%amns_ids%process(isearch)%products(ireac)%mass) .ne. 0 .and.
      nint(reactants%components(ireac+nlhs)%mi) .ne. 0) then
00328                match = match .and. (nint(work%amns_ids%process(isearch)%products(ireac)%mass) .eq.
      nint(reactants%components(ireac+nlhs)%mi))
00329             endif
00330          endif
00331          if (.not. match) cycle
00332       enddo
00333       if (.not. match) cycle
00334       match = .true.
00335       k = isearch
00336       exit
00337    enddo
00338    if(k .eq. 0) then
00339       write(error_description,*) 'GET_AMNS_DATA: Case ',trim(reaction_type%string),' not yet
      implemented'
00340       if(debug) then
00341          write(*,'(a)') trim(error_description)
```

```
00342           do isearch = 1, nproc
00343              write(*,'(a,i4,a,a)') 'Process ', isearch, ' = ' ,
     trim(work%amns_ids%process(isearch)%label(1))
00344           enddo
00345        endif
00346        ierr=11
00347        return
00348     endif
00349
00350     ndim = work%amns_ids%process(k)%table_dimension         ! dimensionality of the requested matrix
00351     ncoord=work%amns_ids%process(k)%coordinate_index
00352     if(iza .eq. -1) iza=1
00353     if(size(work%amns_ids%process(k)%charge_state) .eq. 1) iza=1
00354     if(iza .gt. size(work%amns_ids%process(k)%charge_state)) then
00355        error_description = 'iza out of bounds'
00356        write(*,*) error_description
00357        ierr=12
00358        return
00359     endif
00360
00361 ! some of the 1d functions require 2d tables
00362     if(ndim.eq.1) then
00363        if(.not.associated(work%amns_ids%process(k)%charge_state(iza)%table_1d)) then
00364           if(debug) write(*,*) 'Special treatment for 1d functions requiring a 2d table'
00365           if(associated(work%amns_ids%process(k)%charge_state(iza)%table_2d)) then
00366              ndim=2
00367           endif
00368        endif
00369     endif
00370
00371     done=.true.
00372     select case (ndim)
00373     case(1)
00374        if(ncoord.gt.0) then
00375           call new_grid(grid, &
00376                work%amns_ids%process(k)%charge_state(iza)%table_1d(:), &
00377                work%amns_ids%coordinate_system(ncoord)%coordinate(1)%values(:))
00378        else
00379           call new_grid(grid, &
00380                work%amns_ids%process(k)%charge_state(iza)%table_1d(:))
00381        endif
00382     case(2)
00383        if(ncoord.gt.0) then
00384            call new_grid(grid, &
00385              work%amns_ids%process(k)%charge_state(iza)%table_2d(:,:), &
00386              work%amns_ids%coordinate_system(ncoord)%coordinate(1)%values(:), &
00387              work%amns_ids%coordinate_system(ncoord)%coordinate(2)%values(:))
00388        else
00389           call new_grid(grid, &
00390             work%amns_ids%process(k)%charge_state(iza)%table_2d(:,:))
00391        endif
00392     case(3)
00393        if(ncoord.gt.0) then
00394           call new_grid(grid, &
00395             work%amns_ids%process(k)%charge_state(iza)%table_3d(:,:,:), &
00396             work%amns_ids%coordinate_system(ncoord)%coordinate(1)%values(:), &
00397             work%amns_ids%coordinate_system(ncoord)%coordinate(2)%values(:), &
00398             work%amns_ids%coordinate_system(ncoord)%coordinate(3)%values(:))
00399        else
00400           call new_grid(grid, &
00401             work%amns_ids%process(k)%charge_state(iza)%table_3d(:,:,:))
00402        endif
00403     case(4)
00404        if(ncoord.gt.0) then
00405           call new_grid(grid, &
00406             work%amns_ids%process(k)%charge_state(iza)%table_4d(:,:,:,:), &
00407             work%amns_ids%coordinate_system(ncoord)%coordinate(1)%values(:), &
00408             work%amns_ids%coordinate_system(ncoord)%coordinate(2)%values(:), &
00409             work%amns_ids%coordinate_system(ncoord)%coordinate(3)%values(:), &
00410             work%amns_ids%coordinate_system(ncoord)%coordinate(4)%values(:))
00411        else
00412           call new_grid(grid, &
00413             work%amns_ids%process(k)%charge_state(iza)%table_4d(:,:,:,:))
00414        endif
00415     case(5)
00416        done=.false.
00417        write(*,*) '0D and 5D can be implemented later'
00418     end select
00419
00420     if(done) then
00421        grid%ndim=work%amns_ids%process(k)%table_dimension
00422        if(work%amns_ids%process(k)%result_transformation == 0) then
00423           grid%is_lin= .true.
00424        elseif(work%amns_ids%process(k)%result_transformation == 1) then
00425           grid%is_log= .true.
00426        else
00427           grid%interpol_function = work%amns_ids%process(k)%result_transformation
```

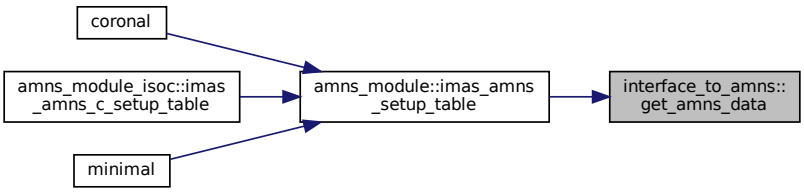
```
00428       endif
00429
00430 ! work%amns_ids%process(k)%label is an array of elements divided into 132 bytes
00431 ! long strings. But state_label inside grid is just 132 character long string
00432 ! we will take first element from label to fit it
00433
00434       if(associated(work%amns_ids%process(k)%charge_state)) then
00435          if(size(work%amns_ids%process(k)%charge_state).ge.iza) then
00436             grid%state_label=assign_if_associated(work%amns_ids%process(k)%charge_state(iza)%label,
      ")
00437          endif
00438       else
00439          grid%state_label="
00440       endif
00441       grid%result_label=assign_if_associated(work%amns_ids%process(k)%result_label, ")
00442       grid%result_unit=assign_if_associated(work%amns_ids%process(k)%result_units, ")
00443       if(ncoord.gt.0) then
00444          do j=1,ndim
00445             if(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%transformation == 0) then
00446                grid%axe(j)%is_lin = .true.
00447             elseif(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%transformation == 1) then
00448                grid%axe(j)%is_log = .true.
00449             else
00450                write(*,*) 'The coordinate transformation is unknown, for process',
      trim(reaction_type%string)
00451             endif
00452             if(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%spacing == 0) then
00453                grid%axe(j)%is_equi = .false.
00454             elseif(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%spacing == 1) then
00455                grid%axe(j)%is_equi = .true.
00456             else
00457                write(*,*) 'The coordinate spacing is unknown, for process',
      trim(reaction_type%string)
00458             endif
00459             grid%axe(j)%lbound= lbound(grid%axe(j)%x, 1)
00460             if(associated(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%extrapolation_type))
      then
00461                if(size(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%extrapolation_type) .eq.
      2) then
00462                   grid%axe(j)%extrapolation_type(:) =
      work%amns_ids%coordinate_system(ncoord)%coordinate(j)%extrapolation_type(:)
00463                else
00464                   write(*,*) 'ERROR: coordinate extrapolation_type not of length 2 in amns_data, 0
      assumed!'
00465                   grid%axe(j)%extrapolation_type = 0
00466                endif
00467             else
00468                write(*,*) 'ERROR: coordinate extrapolation_type not associated in amns_data, 0
      assumed!'
00469                grid%axe(j)%extrapolation_type = 0
00470             endif
00471             if(associated(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%label)) then
00472                grid%axe(j)%label = work%amns_ids%coordinate_system(ncoord)%coordinate(j)%label(1)
00473             endif
00474             if(associated(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%units)) then
00475                grid%axe(j)%units = work%amns_ids%coordinate_system(ncoord)%coordinate(j)%units(1)
00476             endif
00477             if(associated(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%value_labels)) then
00478
      allocate(grid%axe(j)%value_labels(size(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%value_labels)))
00479                grid%axe(j)%value_labels(:) =
      work%amns_ids%coordinate_system(ncoord)%coordinate(j)%value_labels(:)
00480             endif
00481
00482          enddo
00483       endif
00484    endif
00485
00486    properties_comment = assign_if_associated(work%amns_ids%ids_properties%comment, "Not specified in
      the AMNS IDS")
00487    properties_source = assign_if_associated(work%amns_ids%ids_properties%source, "Not specified in
      the AMNS IDS")
00488    properties_provider = assign_if_associated(work%amns_ids%ids_properties%provider, "Not specified
      in the AMNS IDS")
00489    properties_creation_date = assign_if_associated(work%amns_ids%ids_properties%creation_date, "Not
      specified in the AMNS IDS")
00490    code_name = assign_if_associated(work%amns_ids%code%name, "Not specified in the AMNS IDS")
00491    code_commit = assign_if_associated(work%amns_ids%code%commit, "Not specified in the AMNS IDS")
00492    code_version = assign_if_associated(work%amns_ids%code%version, "Not specified in the AMNS IDS")
00493    code_repository = assign_if_associated(work%amns_ids%code%repository, "Not specified in the AMNS
      IDS")
00494
00495    source = assign_if_associated(work%amns_ids%process(k)%source, "Not specified in the AMNS IDS")
00496    provider = assign_if_associated(work%amns_ids%process(k)%provider, "Not specified in the AMNS
      IDS")
00497    citation = assign_if_associated(work%amns_ids%process(k)%citation, "Not specified in the AMNS
      IDS")
```

```
00498
```

Here is the call graph for this function:



Here is the caller graph for this function:



13.7.2.4 int_to_string()

```
character*24 function amns_utility::int_to_string (
             integer, intent(in) int )
```

convert an integer to a string

Definition at line 23 of file amns_utility.f90.

```
00024     integer, intent(in) :: int
00025     write(int_to_string,'(I24)') int
00026     int_to_string=adjustl(int_to_string)
```

13.8 AMNS Internal Utility Functions/Subroutines

External routine used by AMNS.
External routine used by AMNS.

13.9 AMNS C Types and Prototypes

AMNS C Interface.

Classes

- struct amns_version_type

  Type for specifying the AMNS version ("interoperable" version)
- struct amns_c_version_type

  Type for specifying the AMNS version ("C" version)
- struct amns_reactant_type

  Type for indicating a single reactant or product when using the AMNS interface.
- struct amns_error_type

  Type for error returns from the AMNS interface ("interoperable" version)
- struct amns_c_error_type

  Type for error returns from the AMNS interface ("C" version)
- struct amns_reaction_type

  Type used for specifying reactions when using the AMNS interface ("interoperable" version)
- struct amns_c_reaction_type

  Type used for specifying reactions when using the AMNS interface ("C" version)
- struct amns_set_type

  Type for setting parameters in the AMNS package ("interoperable" version)
- struct amns_c_set_type

  Type for setting parameters in the AMNS package ("C" version)
- struct amns_query_type

  Type for querying parameters in the AMNS package ("interoperable" version)
- struct amns_c_query_type

  Type for querying parameters in the AMNS package ("C" version)
- struct amns_answer_type

  Type for answers from queries in the AMNS package ("interoperable" version)
- struct amns_c_answer_type

  Type for answers from queries in the AMNS package ("C" version)

13.9.1 Detailed Description

AMNS C Interface.
AMNS Interface Created based on amns_types.F90 & amns_module_isoc.F90. C structs are only defined for C-interoperable derived Fortran types. All other AMNS types have to be treated as opaque handle types of type void*.
For every interoperable derived type, the following things are provided: -an interoperable C struct with name amns↩_(name)_type -a "convenience" C struct with char* instead of fixed-size strings with name amns_c_(name)_type -two conversion routines to convert between the interoperable and "convenience" type with name

- amns_(name)_type_f2c : converts interoperable to convenience
- amns_(name)_type_c2f : converts convenience to interoperable (These conversion routines only exist for structs containing strings)

Furthermore, every AMNS routine exists in two versions: -the interoperable Fortran routine with name IMAS_AM↩NS_C_(name), with the interoperable types as arguments -a convenience routine with name IMAS_AMNS_CC_↩(name) (CC = C convenience), with the convenience structs as arguments
Typical C programs will make use of both the convenience structs and routines.

Version

"$Id: amns_interface.h 502 2015-10-15 12:23:45Z dpc $"

13.10 Minimal example program in Fortran showing the use of the AMNS library

Minimal Fortran test example.
Minimal Fortran test example.

Author

David Coster

13.11 Minimal example program in C showing the use of the AMNS library

Minimal C test example.
Minimal C test example.

Chapter 14

Module Documentation

14.1 amns Namespace Reference

Namespaces

- package type

Classes

- class Amns
- class AmnsException
- class Reactants
- class Table

14.2 Package amns.type

Classes

- class AmnsAnswerType
- class AmnsErrorType
- class AmnsQueryType
- class AmnsReactantType
- class AmnsReactionType
- class AmnsSetType

14.3 amns_adas Module Reference

Functions/Subroutines

- subroutine adas_amns (amns, root_in, zn_in, amn_in, nreac_in, sp_in, lu_read)

  produce the AMNS database from ADAS data

14.3.1 Function/Subroutine Documentation

14.3.1.1 adas_amns()

```
subroutine amns_adas::adas_amns (
             type (ids_amns_data) amns,
             character (len=256) root_in,
             integer zn_in,
             real (ids_real) amn_in,
```

```
integer nreac_in,
character (len=10) sp_in,
integer lu_read )
```

produce the AMNS database from ADAS data
This sets data enties for the species specified in adas_amns.data, using the ADAS year specified in column 3 of that file.
Currently supported are

- RC: Recombination (acd)
- EI: Electron Impact Ionisation (scd)
- CX: CX recombination coeffts (ccd)
- BR: Recomb/brems power coeffts (prb)
- LR: Line radiation (plt)
- ZE: Effective Charge (zcd)
- ZE2: Effective Square Charge (ycd)
- EIP: Effective Ionisation Potential (ecd)

Author

O'Mullane, Eriksson, Coster

Definition at line 23 of file amns_adas.f90.

```
00024
00025     !-----------------------------------------------------------------------
00026     !
00027     !  SUBROUTINE : adas_amns
00028     !
00029     !  PURPOSE    : to read ADAS adf11 data for carbon and to output into
00030     !               AMNS datastructure format.
00031     !
00032     !  NOTES      : (1) Demonstration project
00033     !
00034     !  ROUTINES:
00035     !          routine    source    brief description
00036     !          ------------------------------------------------------------
00037     !          xxdata_11  adas      read contents of adf11 dataset
00038     !          i4unit     adas      fetch unit number for output of messages
00039     !          i4fctn     adas      convert string to integer form
00040     !          xfelem     adas      return element name given nuclear charge
00041     !          xxword     adas      extract position of number in buffer
00042     !          xxslen     adas      find string less front and tail blanks
00043     !          xxcase     adas      convert a string to upper or lower case
00044     !          xxrptn     adas      analyse an adf11 file partition block
00045     !
00046     !
00047     !
00048     !
00049     ! VERSION : 1.0
00050     ! DATE    : 24-03-2010
00051     ! MODIFIED: Martin O'Mullane
00052     !           - First version.
00053     !
00054     ! DATE    : 07-07-2010
00055     ! MODIFIED: Lars-Goran Eriksson
00056     !               - Extended to eight processes for Carbon and more complete
00057     !                 entries in the data structure.
00058     !
00059     ! DATE    : 2010-12-14
00060     ! MODIFIED: David Coster
00061     !               - Generalized
00062     !
00063     !-----------------------------------------------------------------------
00064
00065     use ids_schemas   ! IGNORE
00066     use ids_types       ! IGNORE
00067     use codata, only: &
00068          & e_mass_in_amu => electron_mass_in_u, &
00069          & d_mass_in_amu => deuteron_mass_in_u
00070
00071     implicit none
00072     !-----------------------------------------------------------------------
```

```
00073
00074     ! AMNS
00075
00076     type (ids_amns_data)                 :: amns
00077
00078     real (ids_real), parameter :: zero = 0.0
00079     real (ids_real), parameter :: log_zero = -300.0
00080
00081     ! xxdata_11 variables
00082
00083     integer, parameter               :: ISDIMD = 200, izdimd = 92, itdimd = 50 , &
00084          iddimd = 40,  ndptn  = 128, ndptnl = 4 , &
00085          ndptnc = 256, ndcnct = 100
00086
00087     integer                          :: IZ0       , IS1MIN    , IS1MAX,  IBLMX , &
00088          ISMAX     , ITMAX     , IDMAX ,  NPTNL , &
00089          NCNCT     , iclass
00090     integer                          :: idmax1, idmax2, idmax3,  idmax4
00091     integer                          :: idmax5, idmax6, idmax7,  idmax8
00092     integer                          :: itmax1, itmax2, itmax3, itmax4
00093     integer                          :: itmax5, itmax6, itmax7, itmax8
00094     real (ids_real)                   :: DNR_AMS
00095     character(len=12)                :: DNR_ELE
00096     logical                          :: LRES      , LSTAN     , LPTN
00097     integer, dimension(NDPTNL)       :: nptn      , IPTNLA
00098     integer, dimension(NDPTNL,NDPTN) :: NPTNC     , IPTNA
00099     integer, dimension(NDPTNL,NDPTN,NDPTNC) :: IPTNCA
00100     integer, dimension(NDCNCT)       :: ICNCTV
00101     integer, dimension(ISDIMD)       :: ISPPR     , ISPBR     , ISSTGR
00102     real (ids_real), dimension(IDDIMD):: ddens
00103     real (ids_real), dimension(ITDIMD):: dtev
00104     real (ids_real), dimension(ISDIMD,ITDIMD,IDDIMD):: drcof
00105
00106     ! program variables
00107
00108     character (len=256)              :: root_in
00109     character (len=256)              :: file
00110     integer                          :: zn_in
00111     integer                          :: nreac_in
00112     character (len=10)               :: sp_in
00113     character (len=11)               :: sp_loc, sp_up
00114     real (ids_real)                   :: amn_in
00115     integer                          :: lu_read
00116
00117     integer                          :: npoints, iz, iunit, iostat, nz, iproc, nprocs, idim
00118     integer                          :: is, ireac
00119     integer                          :: ncoord
00120
00121     ncoord = 0
00122
00123 !-----------------------------------------------------------------------
00124     allocate (amns%time(1))
00125     amns%time=0.0
00126 !    allocate (amns%version(1))
00127 !    allocate (amns%source(1))
00128 !-----------------------------------------------------------------------
00129
00130     sp_loc = sp_in
00131     if(index(sp_loc,'_').gt.0) sp_loc=trim(sp_loc) // '_'
00132     sp_up = upcase(sp_loc)
00133
00134 !    amns%version = 'v0'
00135 !    amns%source  = 'ADAS'
00136
00137     ! Element information
00138
00139     amns%z_n      = real(zn_in)
00140     amns%a        = amn_in
00141
00142
00143 !-----------------------
00144 ! Allocate the structure
00145 !-----------------------
00146
00147 ! Number of processes
00148     nprocs = nreac_in
00149
00150 ! Number of charge states
00151     nz    = zn_in+1
00152
00153     allocate(amns%process(nprocs))
00154 !    allocate(amns%process(nprocs))
00155     allocate(amns%coordinate_system(nprocs))
00156     do iproc=1, nprocs
00157        allocate(amns%process(iproc)%label(1))
00158        allocate(amns%process(iproc)%charge_state(nz))
00159 !       allocate(amns%process(iproc)%zmax(nz))
```

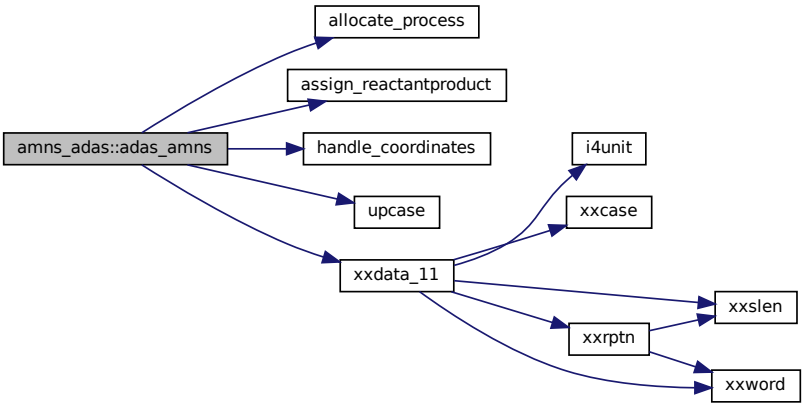
```
00160        do is=1, nz
00161           allocate(amns%process(iproc)%charge_state(is)%label(1))
00162        enddo
00163        allocate(amns%process(iproc)%result_label(1))
00164        allocate(amns%process(iproc)%result_units(1))
00165        allocate(amns%process(iproc)%source(1))
00166        allocate(amns%process(iproc)%provider(1))
00167        allocate(amns%process(iproc)%citation(1))
00168 !       allocate(amns%process(iproc)%table(nz))
00169     enddo
00170
00171 !-----------------------------
00172 ! Fill the structure with data
00173 !-----------------------------
00174
00175     do iproc=1, nprocs
00176        amns%process(iproc)%charge_state(1)%z_min    = 0
00177        amns%process(iproc)%charge_state(1)%z_max    = 0
00178        amns%process(iproc)%charge_state(1)%label(1) = trim(sp_loc) // '0'
00179        do is=1, nz-1
00180           amns%process(iproc)%charge_state(is+1)%z_min = is
00181           amns%process(iproc)%charge_state(is+1)%z_max = is
00182           if(is.lt.10) then
00183              write(amns%process(iproc)%charge_state(is+1)%label(1),'(a,i1,''+'')') trim(sp_loc), is
00184           else
00185              write(amns%process(iproc)%charge_state(is+1)%label(1),'(a,i2,''+'')') trim(sp_loc), is
00186           endif
00187        enddo
00188     enddo
00189
00190 !-----------------------------------
00191 ! Read data for the various processes
00192 !-----------------------------------
00193
00194     iunit = 67
00195
00196     do ireac = 1, nreac_in
00197
00198        read(lu_read, *) amns%process(ireac)%label(1), iclass, file, &
00199             amns%process(ireac)%result_label(1), amns%process(ireac)%result_units(1), &
00200             amns%process(ireac)%result_transformation, amns%process(ireac)%table_dimension
00201
00202        file = trim(root_in) // trim(file)
00203
00204        write(*,*) 'Opening ', trim(file)
00205        open(unit=iunit, file=trim(adjustl(file)), action='read', status='old', iostat=iostat)
00206
00207        call xxdata_11(iunit , iclass ,            &
00208             isdimd , iddimd , itdimd ,            &
00209             ndptnl , ndptn  , ndptnc , ndcnct ,   &
00210             iz0    , is1min , is1max ,            &
00211             nptnl  , nptn   , nptnc  ,            &
00212             iptnla , iptna  , iptnca ,            &
00213             ncnct  , icnctv ,                     &
00214             iblmx  , ismax  , dnr_ele, dnr_ams,   &
00215             isppr  , ispbr  , isstgr ,            &
00216             idmax  , itmax  ,                     &
00217             ddens  , dtev   , drcof ,             &
00218             lres   , lstan  , lptn                )
00219
00220        close(unit=iunit)
00221
00222        write(*,*) 'Closed ', trim(file)
00223
00224        amns%process(ireac)%source   = file    ! Add again later
00225        amns%process(ireac)%provider = 'David.Coster@ipp.mpg.de on behalf of AMNS Team'
00226        amns%process(ireac)%citation = 'http://www.adas.ac.uk/'
00227        amns%process(ireac)%coordinate_index  = handle_coordinates(ncoord, dtev(1:itmax),
     ddens(1:idmax))
00228
00229        select case (amns%process(ireac)%result_transformation)
00230        case (0)
00231           do iz=1, nz
00232              allocate(amns%process(ireac)%charge_state(iz)%table_2d(itmax,idmax))
00233              amns%process(ireac)%charge_state(iz)%table_2d = zero
00234           enddo
00235        case (1)
00236           do iz=1, nz
00237              allocate(amns%process(ireac)%charge_state(iz)%table_2d(itmax,idmax))
00238              amns%process(ireac)%charge_state(iz)%table_2d = log_zero
00239           enddo
00240        case default
00241           write(*,*) 'Case result_trans', amns%process(ireac)%result_transformation, ' not coded'
00242           stop 'Not coded'
00243        end select
00244
00245        select case (iclass)
```

```
00246        case (1, 3, 4)
00247           do iz = 1, nz-1
00248              amns%process(ireac)%charge_state(iz+1)%table_2d(:,:) = -6.0 + drcof(iz,1:itmax,1:idmax)
00249           end do
00250        case (2, 8)
00251           do iz = 1, nz-1
00252              amns%process(ireac)%charge_state(iz  )%table_2d(:,:) = -6.0 + drcof(iz,1:itmax,1:idmax)
00253           end do
00254        case (10, 11, 12)
00255           do iz = 1, nz
00256              amns%process(ireac)%charge_state(iz  )%table_2d(:,:) =         +
       drcof(iz,1:itmax,1:idmax)
00257           end do
00258        case default
00259           write(*,*) 'Case iclass ', iclass, ' not coded'
00260           stop 'Not Coded'
00261        end select
00262
00263
00264 ! Reactant and product are 4.10b extensions of the ITM, they are not present in IMAS yet
00265        select case (iclass)
00266        case (1) ! RC / acd
00267           call allocate_process(amns%process(ireac), 2, 2)
00268           call assign_reactantproduct(amns%process(ireac)%reactants(1), &
00269                trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00270           call assign_reactantproduct(amns%process(ireac)%reactants(2), &
00271                'e', 0, e_mass_in_amu, 2, 0, -1.0d+0)
00272           call assign_reactantproduct(amns%process(ireac)%products(1), &
00273                trim(sp_loc), zn_in, amn_in, 1, 1, -1.0d+0)
00274           call assign_reactantproduct(amns%process(ireac)%products(2), &
00275                'e', 0, e_mass_in_amu, 1, 0, -1.0d+0)
00276        case (2) ! EI / scd
00277           call allocate_process(amns%process(ireac), 2, 2)
00278           call assign_reactantproduct(amns%process(ireac)%reactants(1), &
00279                trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00280           call assign_reactantproduct(amns%process(ireac)%reactants(2), &
00281                'e', 0, e_mass_in_amu, 1, 0, -1.0d+0)
00282           call assign_reactantproduct(amns%process(ireac)%products(1), &
00283                trim(sp_loc), zn_in, amn_in, 1, 1, +1.0d+0)
00284           call assign_reactantproduct(amns%process(ireac)%products(2), &
00285                'e', 0, e_mass_in_amu, 2, 0, -1.0d+0)
00286        case (3) ! CX / ccd
00287           call allocate_process(amns%process(ireac), 2, 2)
00288           call assign_reactantproduct(amns%process(ireac)%reactants(1), &
00289                trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00290           call assign_reactantproduct(amns%process(ireac)%reactants(2), &
00291                'H|D|T', 1, d_mass_in_amu, 1, 0, 0.0d+0)
00292           call assign_reactantproduct(amns%process(ireac)%products(1), &
00293                trim(sp_loc), zn_in, amn_in, 1, 1, -1.0d+0)
00294           call assign_reactantproduct(amns%process(ireac)%products(2), &
00295                'H|D|T', 1, d_mass_in_amu, 1, 0, 1.0d+0)
00296        case (4, 8, 10, 11, 12) ! BR/prb, LR/plt, ZE/zcd, ZE2/ycd, EIP/ecd
00297           call allocate_process(amns%process(ireac), 1, 1)
00298           call assign_reactantproduct(amns%process(ireac)%reactants(1), &
00299                trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00300           call assign_reactantproduct(amns%process(ireac)%products(1), &
00301                trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00302        end select
00303
00304     end do
00305     write(*,*) 'Number of different coordinates = ', ncoord
00306
00307     return
00308
00309   contains
00310
00311     integer function handle_coordinates(ncoord, dtev, ddens)
00312
00313 ! amns%process%id(1, jproc) gives the dimension of the data table,
00314 ! i.e. the number of dependent variables (in the example shown here
00315 ! the variables are always density and temperature and the table is
00316 ! therefore 2d).
00317 ! amns%process%id(2, jproc) gives the position in the table, i.e. the last
00318 ! argument in the table. For instance for a 2D table, and process jproc the
00319 ! data are given by amns%process%tables_2d%table(:,;,jproc, ipos) where
00320 ! ipos =  amns%process%id(2, jproc).
00321 ! Dependent variables (electron density and temperature in this case)
00322
00323        implicit none
00324
00325        real (ids_real), dimension(:):: ddens
00326        real (ids_real), dimension(:):: dtev
00327        integer ncoord, icoord, itmax, idmax
00328
00329        itmax=size(dtev)
00330        idmax=size(ddens)
00331 ! check whether we already have the coordinate
```

```
00332       do icoord=1, ncoord
00333          if(itmax.ne.size(amns%coordinate_system(icoord)%coordinate(1)%values)) cycle
00334          if(idmax.ne.size(amns%coordinate_system(icoord)%coordinate(2)%values)) cycle
00335          write(*,*) maxval(abs(dtev-amns%coordinate_system(icoord)%coordinate(1)%values))
00336          if(maxval(abs(dtev-amns%coordinate_system(icoord)%coordinate(1)%values)).gt.0.0) cycle
00337          write(*,*) maxval(abs(6.0 + ddens-amns%coordinate_system(icoord)%coordinate(2)%values))
00338          if(maxval(abs(6.0 + ddens-amns%coordinate_system(icoord)%coordinate(2)%values)).gt.0.0) cycle
00339 ! we have a match
00340          handle_coordinates=icoord
00341          return
00342       enddo
00343 ! we have no match
00344       ncoord=ncoord+1
00345       allocate(amns%coordinate_system(ncoord)%coordinate(2))
00346       do idim=1, 2
00347          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%extrapolation_type(2))
00348          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%label(1))
00349          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%units(1))
00350       enddo
00351       allocate(amns%coordinate_system(ncoord)%coordinate(1)%values(itmax))
00352       allocate(amns%coordinate_system(ncoord)%coordinate(2)%values(idmax))
00353
00354       amns%coordinate_system(ncoord)%coordinate(1)%label(1) = 'Electron Temperature'
00355       amns%coordinate_system(ncoord)%coordinate(1)%extrapolation_type = (/ 2, 2 /)
00356       amns%coordinate_system(ncoord)%coordinate(1)%interpolation_type = 1
00357       amns%coordinate_system(ncoord)%coordinate(1)%units = 'eV'
00358       amns%coordinate_system(ncoord)%coordinate(1)%transformation = 1
00359       amns%coordinate_system(ncoord)%coordinate(1)%spacing = 0
00360       amns%coordinate_system(ncoord)%coordinate(1)%values = dtev(1:itmax)
00361
00362       amns%coordinate_system(ncoord)%coordinate(2)%label(1) = 'Electron Density'
00363       amns%coordinate_system(ncoord)%coordinate(2)%extrapolation_type = (/ 2, 2 /)
00364       amns%coordinate_system(ncoord)%coordinate(2)%interpolation_type = 1
00365       amns%coordinate_system(ncoord)%coordinate(2)%units = 'm^{-3}'
00366       amns%coordinate_system(ncoord)%coordinate(2)%transformation = 1
00367       amns%coordinate_system(ncoord)%coordinate(2)%spacing = 0
00368       amns%coordinate_system(ncoord)%coordinate(2)%values = 6.0 + ddens(1:idmax)
00369
00370       handle_coordinates=ncoord
00371       return
00372
00373     end function handle_coordinates
00374
00375     function upcase(string) result(upper)
00376       character(len=*), intent(in) :: string
00377       character(len=len(string)) :: upper
00378       integer :: j
00379       do j = 1,len(string)
00380          if(string(j:j) >= "a" .and. string(j:j) <= "z") then
00381             upper(j:j) = achar(iachar(string(j:j)) - 32)
00382          else
00383             upper(j:j) = string(j:j)
00384          end if
00385       end do
00386
00387     end function upcase
00388
00389 ! 4.10b extension, not in IMAS yet
00390     subroutine allocate_process(process, nr, np)
00391       use ids_schemas   ! IGNORE
00392       implicit none
00393       type (ids_amns_data_process) :: process
00394       integer :: nr, np
00395       integer :: irp
00396
00397       allocate(process%reactants(nr), process%products(np))
00398       do irp = 1, size(process%reactants)
00399          allocate(process%reactants(irp)%label(1))
00400          allocate(process%reactants(irp)%element(1))
00401 !         allocate(process%reactants(irp)%element(1)%label(1))
00402       enddo
00403       do irp = 1, size(process%products)
00404          allocate(process%products(irp)%label(1))
00405          allocate(process%products(irp)%element(1))
00406 !         allocate(process%products(irp)%element(1)%label(1))
00407       enddo
00408
00409     end subroutine allocate_process
00410
00411     subroutine assign_reactantproduct(reactantproduct, label, zn, amn, multiplicity, relative, za)
00412       use ids_schemas   ! IGNORE
00413       implicit none
00414       type (ids_amns_data_process_reactant) :: reactantproduct
00415       character(len=*) :: label
00416       integer :: zn, multiplicity, relative
00417       real (ids_real) :: amn, za
00418
```

```
00419       reactantproduct%label(1)=label
00420       reactantproduct%multiplicity=multiplicity
00421       reactantproduct%element(1)%atoms_n=multiplicity
00422 !      reactantproduct%element(1)%label(1)=label
00423       reactantproduct%element(1)%z_n=zn
00424       reactantproduct%element(1)%a=nint(amn)
00425       reactantproduct%mass=amn
00426       reactantproduct%relative_charge=relative
00427       reactantproduct%charge=za
00428
00429     end subroutine assign_reactantproduct
00430
```

Here is the call graph for this function:



14.4 amns_bms Module Reference

Functions/Subroutines

- subroutine bms_amns (amns, zn, am)
- subroutine allocate_process (process, nr, np)
- subroutine assign_reactantproduct (reactantproduct, label, zn, amn, multiplicity, relative, za)

Variables

- integer, parameter r8 = SELECTED_REAL_KIND (15, 300)

14.4.1 Function/Subroutine Documentation

14.4.1.1 allocate_process()

```
subroutine amns_bms::allocate_process (
             type (ids_amns_data_process) process,
             integer nr,
             integer np )
```

Definition at line 102 of file amns_bms.f90.

```
00103     use ids_schemas   ! IGNORE
00104     implicit none
00105     type (ids_amns_data_process) :: process
00106     integer :: nr, np
00107     integer :: irp
00108
00109     allocate(process%reactants(nr), process%products(np))
00110     do irp = 1, size(process%reactants)
00111        allocate(process%reactants(irp)%label(1))
```

```
00112        allocate(process%reactants(irp)%element(1))
00113        !         allocate(process%reactants(irp)%element(1)%label(1))
00114     enddo
00115     do irp = 1, size(process%products)
00116        allocate(process%products(irp)%label(1))
00117        allocate(process%products(irp)%element(1))
00118        !         allocate(process%products(irp)%element(1)%label(1))
00119     enddo
00120
```

14.4.1.2 assign_reactantproduct()

```
subroutine amns_bms::assign_reactantproduct (
             type (ids_amns_data_process_reactant) reactantproduct,
             character(len=*) label,
             integer zn,
             real (ids_real) amn,
             integer multiplicity,
             integer relative,
             real (ids_real) za )
```

Definition at line 123 of file amns_bms.f90.

```
00124     use ids_schemas   ! IGNORE
00125     implicit none
00126     type (ids_amns_data_process_reactant) :: reactantproduct
00127     character(len=*) :: label
00128     integer :: zn, multiplicity, relative
00129     real (ids_real) :: amn, za
00130
00131     reactantproduct%label(1)=label
00132     reactantproduct%multiplicity=multiplicity
00133     reactantproduct%element(1)%atoms_n=multiplicity
00134     !      reactantproduct%element(1)%label(1)=label
00135     reactantproduct%element(1)%z_n=zn
00136     reactantproduct%element(1)%a=nint(amn)
00137     reactantproduct%mass=amn
00138     reactantproduct%relative_charge=relative
00139     reactantproduct%charge=za
00140
```

14.4.1.3 bms_amns()

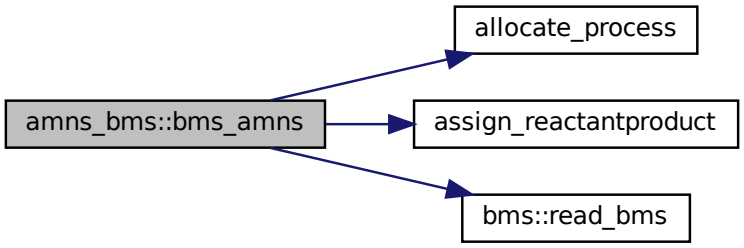
```
subroutine amns_bms::bms_amns (
             type (ids_amns_data) amns,
             integer zn,
             integer am )
```

Definition at line 7 of file amns_bms.f90.

```
00008     use ids_schemas   ! IGNORE
00009     use codata, only: h_mass_in_amu => proton_mass_in_u
00010     use bms
00011
00012     implicit none
00013
00014     type (ids_amns_data) :: amns
00015     integer :: zn, am
00016     integer :: nprocs, iproc
00017     integer :: ncoords, icoord
00018     integer :: npoints(4)
00019     real, allocatable :: y(:,:,:,:), x1(:), x2(:), x3(:), x4(:)
00020     integer :: i
00021
00022     nprocs = 1
00023     ncoords = 1
00024     allocate(amns%coordinate_system(ncoords))
00025     allocate(amns%process(nprocs))
00026     iproc=1
00027     allocate(amns%process(iproc)%label(1))
00028     amns%process(iproc)%label = 'BMS'
00029     call allocate_process(amns%process(iproc), 2, 2)
00030     call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00031          'H', 1, h_mass_in_amu, 1,  0, 0.0_r8)
00032     call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00033          'H', 1, h_mass_in_amu, 1,  1, 0.0_r8)
00034     call assign_reactantproduct(amns%process(iproc)%products(1), &
00035          'H', 1, h_mass_in_amu, 1,  1, 0.0_r8)
00036     call assign_reactantproduct(amns%process(iproc)%products(2), &
```

```
00037          'H', 1, h_mass_in_amu, 1,  0, 0.0_r8)
00038     allocate(amns%process(iproc)%result_label(1))
00039     amns%process(iproc)%result_label = 'BMS'
00040     allocate(amns%process(iproc)%result_units(1))
00041     amns%process(iproc)%result_units = '??'
00042     allocate(amns%process(iproc)%charge_state(1))
00043
00044     allocate(amns%process(iproc)%source(1))
00045     amns%process(iproc)%source = 'amns_driver/bms.f90'
00046     allocate(amns%process(iproc)%provider(1))
00047     amns%process(iproc)%provider = 'David.Coster@ipp.mpg.de on behalf of AMNS Team'
00048     allocate(amns%process(iproc)%citation(1))
00049     amns%process(iproc)%citation = 'ToDo'
00050     amns%process(iproc)%label(1)='BMS'
00051
00052     call read_bms('../../test-data/bms#h_h1_value.dat', y, x1, x2, x3, x4)
00053     write(*,*) 'x1: ', x1
00054     write(*,*) 'x2: ', x2
00055     write(*,*) 'x3: ', x3
00056     write(*,*) 'x4: ', x4
00057     write(*,*) minval(y), maxval(y)
00058     npoints=(/ size(x1), size(x2), size(x3), size(x4) /)
00059
00060     allocate(amns%process(iproc)%charge_state(1)%table_4d(size(x1), size(x2), size(x3), size(x4)))
00061     amns%process(iproc)%charge_state(1)%table_4d = y
00062     amns%process(iproc)%table_dimension = 4
00063     amns%process(iproc)%result_transformation = 0
00064
00065     icoord = 1
00066     amns%process(iproc)%coordinate_index = icoord
00067     allocate(amns%coordinate_system(icoord)%coordinate(4))
00068     do i=1,size(amns%coordinate_system(icoord)%coordinate)
00069        allocate(amns%coordinate_system(icoord)%coordinate(i)%values(npoints(i)),&
00070             &   amns%coordinate_system(icoord)%coordinate(i)%label(1),&
00071             &   amns%coordinate_system(icoord)%coordinate(i)%units(1),&
00072             &   amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(2))
00073        amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(1:2)=1 ! 0= No extrapolation
00074        amns%coordinate_system(icoord)%coordinate(i)%interpolation_type=1      ! 1= Linear
      interpolation
00075        amns%coordinate_system(icoord)%coordinate(i)%transformation=0
00076        amns%coordinate_system(icoord)%coordinate(i)%spacing=0
00077     end do
00078     amns%coordinate_system(icoord)%coordinate(1)%values = x1
00079     amns%coordinate_system(icoord)%coordinate(1)%label = 'NENG'
00080     amns%coordinate_system(icoord)%coordinate(1)%units = 'eV'
00081     amns%coordinate_system(icoord)%coordinate(2)%values = x2
00082     amns%coordinate_system(icoord)%coordinate(2)%label = 'NDENS'
00083     amns%coordinate_system(icoord)%coordinate(2)%units = 'm^{-3}'
00084     amns%coordinate_system(icoord)%coordinate(3)%values = x3
00085     amns%coordinate_system(icoord)%coordinate(3)%label = 'NTION'
00086     amns%coordinate_system(icoord)%coordinate(3)%units = 'eV'
00087     amns%coordinate_system(icoord)%coordinate(4)%values = x4
00088     amns%coordinate_system(icoord)%coordinate(4)%label = 'NTE'
00089     amns%coordinate_system(icoord)%coordinate(4)%units = 'eV'
00090
00091 !! test DPC   try interpolating the log10 of the data with log10 of the coordinates ...
00092     amns%process(iproc)%result_transformation = 1    ! interpolating using log10
00093     amns%process(iproc)%charge_state(1)%table_4d = log10(amns%process(iproc)%charge_state(1)%table_4d)
00094     do i=1,size(amns%coordinate_system(icoord)%coordinate)
00095        amns%coordinate_system(icoord)%coordinate(i)%transformation=1          ! interpolating using
      log10
00096        amns%coordinate_system(icoord)%coordinate(i)%values =
      log10(amns%coordinate_system(icoord)%coordinate(i)%values)
00097     enddo
00098 !! end of test
00099
```

Here is the call graph for this function:



14.4.2 Variable Documentation

14.4.2.1 r8

```
integer, parameter amns_bms::r8 = SELECTED_REAL_KIND (15, 300)
```

Definition at line 3 of file amns_bms.f90.

```
00003   INTEGER,  PARAMETER :: R8 = selected_real_kind (15, 300) ! Real*8
```

14.5 amns_dump_index Namespace Reference

Functions

- def str2bool (v)

Variables

- parser
- type
- str
- help
- default
- str2bool
- nargs
- const
- True
- False
- args = parser.parse_args()
- int shot = 0

  Define the Database entry.
- int run = 1
- pulse_index = imas.ids(shot, run)

  Open the database.
- AMNS_index = pulse_index.amns_data

  Get the data from the index block.
- pulse = imas.ids(e.shot, e.run)

  Print information about the number of available releases.
- AMNS = pulse.amns_data

  Acquiring the amns_data from this new shot/run.

14.5.1 Detailed Description

```
This python program dumps the index block for the AMNS system stored
in the amns_data IDS under the user specified by "--user" (defaulting
to the user running the program) in shot 1, run 0 stored in the "amns"
device.

The program reads the shot 1, run 0 data from the "amns" device and
then loops over the "release_entry" entries, printing out a subset of
the data stored "release_entry", before opening and extracting a
subset of data stored in the "amns" IDS pointed to by each entry
stored in "data_entry" under "release_entry".
```

14.5.2 Function Documentation

14.5.2.1 str2bool()

```
def amns_dump_index.str2bool (
              v )
```

Definition at line 18 of file amns_dump_index.py.

```
00018 def str2bool(v):
00019     if isinstance(v, bool):
00020        return v
00021     if v.lower() in ('yes', 'true', 't', 'y', '1'):
00022         return True
00023     elif v.lower() in ('no', 'false', 'f', 'n', '0'):
00024         return False
00025     else:
00026         raise argparse.ArgumentTypeError('Boolean value expected.')
00027
```

14.5.3 Variable Documentation

14.5.3.1 AMNS

```
amns_dump_index.AMNS = pulse.amns_data
```

Acquiring the amns_data from this new shot/run.
Definition at line 61 of file amns_dump_index.py.

14.5.3.2 AMNS_index

```
amns_dump_index.AMNS_index = pulse_index.amns_data
```

Get the data from the index block.
Definition at line 44 of file amns_dump_index.py.

14.5.3.3 args

```
amns_dump_index.args = parser.parse_args()
```

Definition at line 33 of file amns_dump_index.py.

14.5.3.4 const

```
amns_dump_index.const
```

Definition at line 32 of file amns_dump_index.py.

14.5.3.5 default

amns_dump_index.default
Definition at line 31 of file amns_dump_index.py.

14.5.3.6 False

amns_dump_index.False
Definition at line 32 of file amns_dump_index.py.

14.5.3.7 help

amns_dump_index.help
Definition at line 31 of file amns_dump_index.py.

14.5.3.8 nargs

amns_dump_index.nargs
Definition at line 32 of file amns_dump_index.py.

14.5.3.9 parser

amns_dump_index.parser
Initial value:
```
00001 =  argparse.ArgumentParser(description="Explore AMNS data", epilog
00002 =, formatter_class=argparse.RawTextHelpFormatter)
```
Definition at line 28 of file amns_dump_index.py.

14.5.3.10 pulse

amns_dump_index.pulse = imas.ids(e.shot, e.run)
Print information about the number of available releases.
Loop over the releases
Printing high level about this release
Before looping over each entry
Printing high level information from the entry (shot and run)
And then opening the indicated shot/run
Definition at line 58 of file amns_dump_index.py.

14.5.3.11 pulse_index

amns_dump_index.pulse_index = imas.ids(shot, run)
Open the database.
Definition at line 41 of file amns_dump_index.py.

14.5.3.12 run

int amns_dump_index.run = 1
Definition at line 39 of file amns_dump_index.py.

14.5.3.13 shot

```
int amns_dump_index.shot = 0
```
Define the Database entry.
Definition at line 38 of file amns_dump_index.py.

14.5.3.14 str

```
amns_dump_index.str
```
Definition at line 31 of file amns_dump_index.py.

14.5.3.15 str2bool

```
amns_dump_index.str2bool
```
Definition at line 32 of file amns_dump_index.py.

14.5.3.16 True

```
amns_dump_index.True
```
Definition at line 32 of file amns_dump_index.py.

14.5.3.17 type

```
amns_dump_index.type
```
Definition at line 31 of file amns_dump_index.py.

14.6 amns_external_functions Module Reference

Data Types

- type fun_err_t

Functions/Subroutines

- subroutine nuclear_data_1001 (function_parameters, x, f, with_warnings, fun_err)

  AMNS External function ...
- subroutine nuclear_data_1002 (function_parameters, Tin, f, with_warnings, fun_err)

  AMNS External function ...
- subroutine rct_data_1003 (function_parameters, Tin, f, with_warnings, fun_err)

  AMNS External function ...
- subroutine sputter_data_1004 (function_parameters, energy_arr, angle_arr, yield_arr, with_warnings, fun_↩ err)

  AMNS External function ...
- subroutine reflect_data_1005 (function_parameters, energy_arr, angle_arr, refl_arr, with_warnings, fun_err)

  AMNS External function ...
- subroutine nuclear_data_1006 (function_parameters, x, f, with_warnings, fun_err)

  AMNS External function ...

14.7 amns_module Module Reference

Data Types

- interface imas_amns_rx

  get the rates associated with the input args for a particular reaction (generic interface for 1d, 2d & 3d arrays)

Functions/Subroutines

- subroutine errorstop (error_message)
- subroutine imas_amns_setup (handle, version, error_status)

  initialization call for the AMNS package
- subroutine imas_amns_setup_table (handle, reaction_type, reactants, handle_rx, error_status)

  initialization call for a particular reaction
- subroutine imas_amns_finish (handle, error_status)

  finalization call for the AMNS package
- subroutine imas_amns_finish_table (handle_rx, error_status)

  finalization call for a particular reaction
- subroutine imas_amns_query (handle, query, answer, error_status)

  query routine for the AMNS package
- subroutine imas_amns_query_table (handle_rx, query, answer, error_status)

  query routine for a particular reaction
- subroutine imas_amns_set (handle, set, error_status)

  set a parameter for the AMNS package
- subroutine imas_amns_set_table (handle_rx, set, error_status)

  set a parameter for a particular reaction
- subroutine imas_amns_rx_0 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (0d) args for a particular reaction
- subroutine imas_amns_rx_1 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (1d) args for a particular reaction
- subroutine imas_amns_rx_2 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (2d) args for a particular reaction
- subroutine imas_amns_rx_3 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (3d) args for a particular reaction

Variables

- integer, save version_no
- character(len=256), save user
- character(len=32), save ds_version ='3'
- character(len=32), save backend ='mdsplus'
- type(ids_amns_data), save amns00

14.7.1 Function/Subroutine Documentation

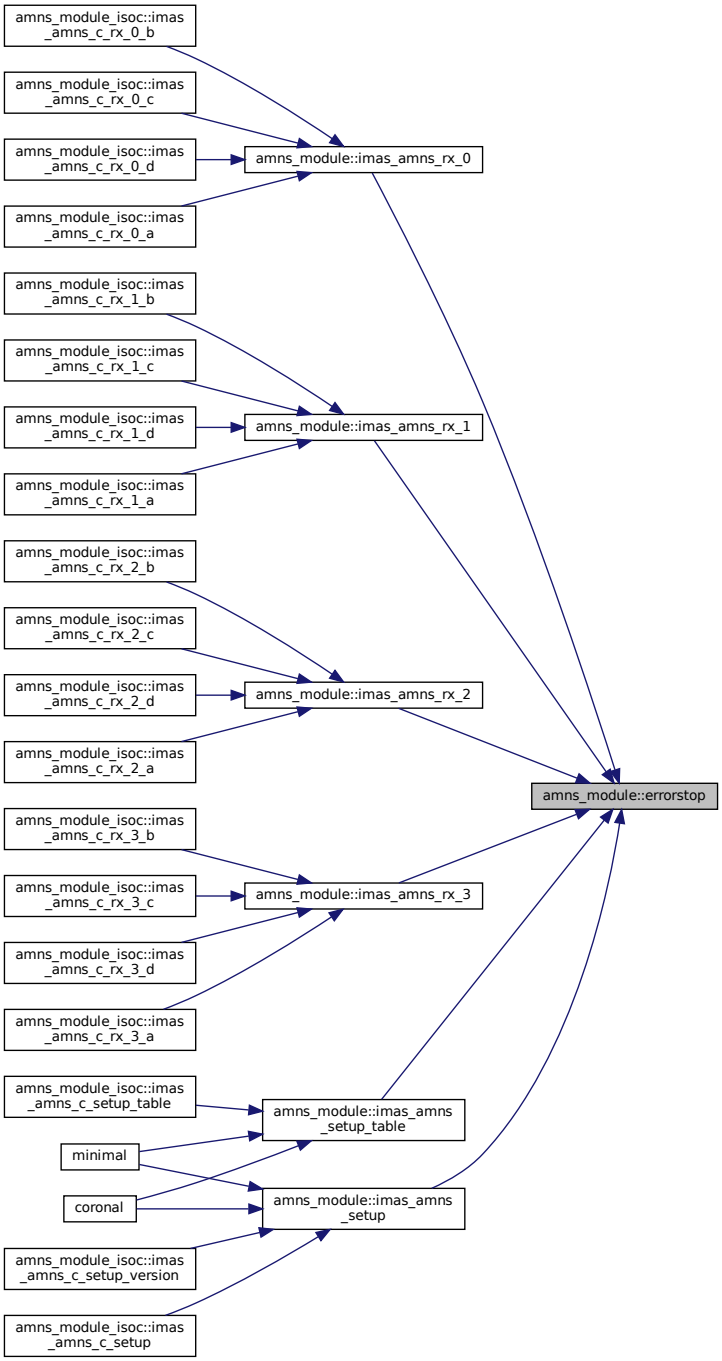
14.7.1.1 errorstop()

```
subroutine amns_module::errorstop (
             character (len=*) error_message )
```

Definition at line 41 of file amns_module.f90.

```
00042     character (len=*) :: error_message
00043     write(0,*) 'Error - ' // error_message
00044     stop 1
```

Here is the caller graph for this function:



14.7.1.2 imas_amns_rx_0()

```
subroutine amns_module::imas_amns_rx_0 (
            type(amns_handle_rx_type), intent(inout) handle_rx,
            real (kind=ids_real), intent(out) out,
            real (kind=ids_real), intent(in) arg1,
            real (kind=ids_real), intent(in), optional arg2,
            real (kind=ids_real), intent(in), optional arg3,
            real (kind=ids_real), intent(in), optional arg4,
            type(amns_error_type), intent(out), optional error_status )
```

get the rates associated with the input (0d) args for a particular reaction

Parameters

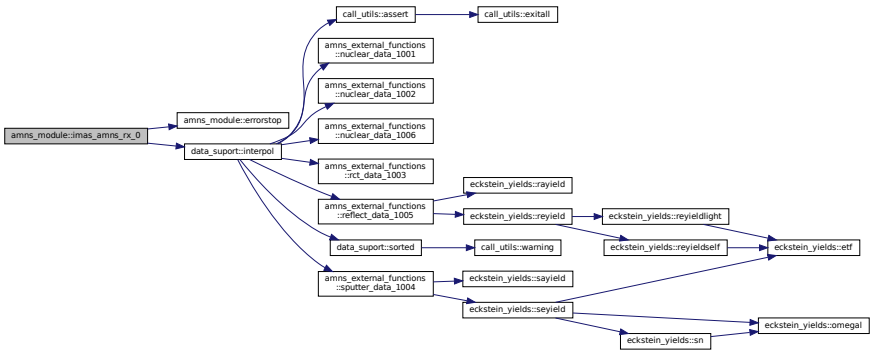
in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system (scalar)
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; (scalar) of the same size as out
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; (scalar) of the same size as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; (scalar) of the same size as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; (scalar) of the same size as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

Definition at line 785 of file amns_module.f90.

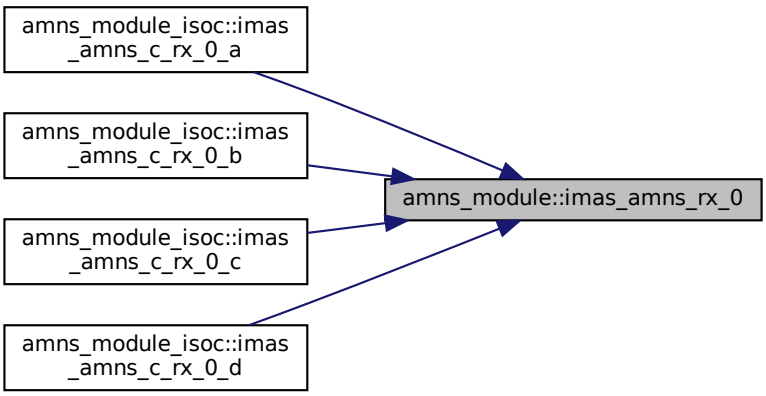
```
00786     use ids_types  ! IGNORE
00787     use amns_types
00788     use data_suport
00789     implicit none
00790     optional arg2,arg3,arg4,error_status
00791     type(amns_handle_rx_type), intent(inout) :: handle_rx
00792     real (kind=ids_real), intent(out) :: out
00793     real (kind=ids_real), intent(in) :: arg1,arg2,arg3,arg4
00794     type(amns_error_type), intent(out) :: error_status
00795
00796     type(data_error_t) :: data_error
00797     real (kind=ids_real) :: out_d(1),arg1_d(1),arg2_d(1),arg3_d(1),arg4_d(1)
00798
00799     if(present(error_status)) then
00800         error_status%flag=.false.
00801         error_status%string="No error"
00802     end if
00803
00804     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_0 start'
00805     if(present(arg4)) then
00806        if(present(arg3).and.present(arg2)) then
00807           if(handle_rx%filled) then
00808              arg1_d(1)=arg1
00809              arg2_d(1)=arg2
00810              arg3_d(1)=arg3
00811              arg4_d(1)=arg4
00812              call interpol( &
00813                   reshape(arg1_d,(/size(arg1_d)/)), &
00814                   reshape(arg2_d,(/size(arg2_d)/)), &
00815                   reshape(arg3_d,(/size(arg3_d)/)), &
00816                   reshape(arg4_d,(/size(arg4_d)/)), &
00817                   fd1=out_d, grid=handle_rx%grid, &
00818                   data_error=data_error)
00819              if(present(error_status)) then
00820                 error_status%flag = data_error%ierr .ne. 0
00821                 error_status%string = data_error%cerr
00822              else
00823                 if(data_error%ierr.ne.0) then
00824                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
00825                    call errorstop('interpol returned an error')
00826                 endif
00827              endif
00828              out=out_d(1)
00829           else
00830              write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00831           endif
```

```
00832        else
00833           write(*,*) 'IMAS_AMNS_RX_0: arg4 present but not arg2/arg3!'
00834        endif
00835     else
00836        if(present(arg3)) then
00837           if(present(arg2)) then
00838              if(handle_rx%filled) then
00839                 arg1_d(1)=arg1
00840                 arg2_d(1)=arg2
00841                 arg3_d(1)=arg3
00842                 call interpol( &
00843                      reshape(arg1_d,(/size(arg1_d)/)), &
00844                      reshape(arg2_d,(/size(arg2_d)/)), &
00845                      reshape(arg3_d,(/size(arg3_d)/)), &
00846                      fd1=out_d, grid=handle_rx%grid, &
00847                      data_error=data_error)
00848                 if(present(error_status)) then
00849                    error_status%flag = data_error%ierr .ne. 0
00850                    error_status%string = data_error%cerr
00851                 else
00852                    if(data_error%ierr.ne.0) then
00853                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
00854                       call errorstop('interpol returned an error')
00855                    endif
00856                 endif
00857                 out=out_d(1)
00858              else
00859                 write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00860              endif
00861           else
00862              write(*,*) 'IMAS_AMNS_RX_0: arg3 present but not arg2!'
00863           endif
00864        else
00865           if(present(arg2)) then
00866              if(handle_rx%filled) then
00867                 arg1_d(1)=arg1
00868                 arg2_d(1)=arg2
00869                 call interpol( &
00870                      reshape(arg1_d,(/size(arg1_d)/)), &
00871                      reshape(arg2_d,(/size(arg2_d)/)), &
00872                      fd1=out_d, grid=handle_rx%grid, &
00873                      data_error=data_error)
00874                 if(present(error_status)) then
00875                    error_status%flag = data_error%ierr .ne. 0
00876                    error_status%string = data_error%cerr
00877                 else
00878                    if(data_error%ierr.ne.0) then
00879                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
00880                       call errorstop('interpol returned an error')
00881                    endif
00882                 endif
00883                 out=out_d(1)
00884              else
00885                 write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00886              endif
00887           else
00888              if(handle_rx%filled) then
00889                 arg1_d(1)=arg1
00890                 call interpol( &
00891                      reshape(arg1_d,(/size(arg1_d)/)), &
00892                      fd1=out_d, grid=handle_rx%grid, &
00893                      data_error=data_error)
00894                 if(present(error_status)) then
00895                    error_status%flag = data_error%ierr .ne. 0
00896                    error_status%string = data_error%cerr
00897                 else
00898                    if(data_error%ierr.ne.0) then
00899                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
00900                       call errorstop('interpol returned an error')
00901                    endif
00902                 endif
00903                 out=out_d(1)
00904              else
00905                 write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00906              endif
00907           endif
00908        endif
00909     endif
00910     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_0 end'
00911
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.7.1.3 imas_amns_rx_1()

```
subroutine amns_module::imas_amns_rx_1 (
            type(amns_handle_rx_type), intent(inout) handle_rx,
            real (kind=ids_real), dimension(:), intent(out) out,
            real (kind=ids_real), dimension(:), intent(in) arg1,
            real (kind=ids_real), dimension(:), intent(in), optional arg2,
            real (kind=ids_real), dimension(:), intent(in), optional arg3,
            real (kind=ids_real), dimension(:), intent(in), optional arg4,
            type(amns_error_type), intent(out), optional error_status )
```

get the rates associated with the input (1d) args for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system (1d array)
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; (1d array) of the same size as out

Parameters

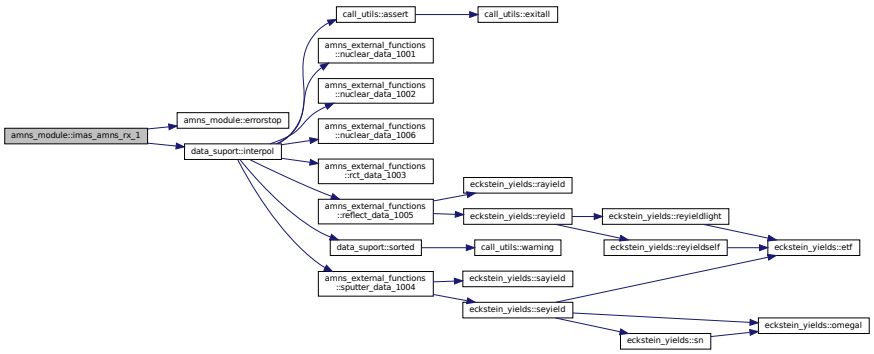
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; (1d array) of the same size as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; (1d array) of the same size as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; (1d array) of the same size as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

Definition at line 923 of file amns_module.f90.

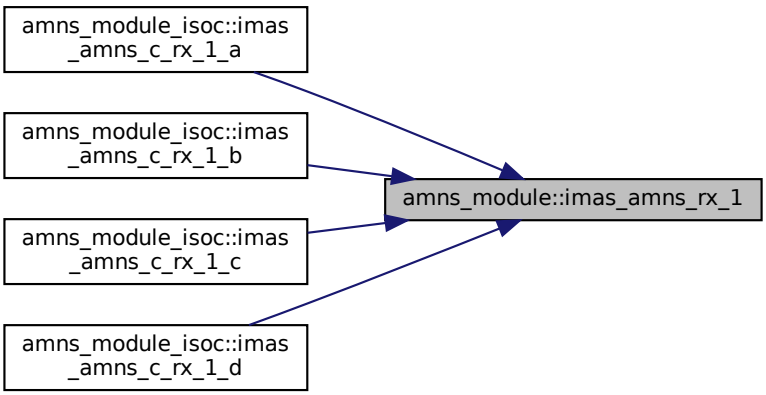
```
00924     use ids_types  ! IGNORE
00925     use amns_types
00926     use data_suport
00927     implicit none
00928     optional arg2,arg3,arg4,error_status
00929     type(amns_handle_rx_type), intent(inout) :: handle_rx
00930     real (kind=ids_real), intent(out) :: out(:)
00931     real (kind=ids_real), intent(in) :: arg1(:),arg2(:),arg3(:),arg4(:)
00932     type(amns_error_type), intent(out) :: error_status
00933
00934     type(data_error_t) :: data_error
00935
00936     if(present(error_status)) then
00937         error_status%flag=.false.
00938         error_status%string="No error"
00939     end if
00940
00941     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1 start'
00942     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(out) = ',lbound(out),ubound(out)
00943     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg1) = ',lbound(arg1),ubound(arg1)
00944     if(present(arg4)) then
00945        if(present(arg3).and.present(arg2)) then
00946           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) = ',lbound(arg2),ubound(arg2)
00947           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg3) = ',lbound(arg3),ubound(arg3)
00948           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg4) = ',lbound(arg4),ubound(arg4)
00949           if(handle_rx%filled) then
00950              call interpol( &
00951                   reshape(arg1,(/size(arg1)/)), &
00952                   reshape(arg2,(/size(arg2)/)), &
00953                   reshape(arg3,(/size(arg3)/)), &
00954                   reshape(arg4,(/size(arg4)/)), &
00955                   fd1=out, grid=handle_rx%grid, &
00956                   data_error=data_error)
00957              if(present(error_status)) then
00958                 error_status%flag = data_error%ierr .ne. 0
00959                 error_status%string = data_error%cerr
00960              else
00961                 if(data_error%ierr.ne.0) then
00962                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
00963                    call errorstop('interpol returned an error')
00964                 endif
00965              endif
00966           else
00967              write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
00968           endif
00969        else
00970           write(*,*) 'IMAS_AMNS_RX_1: arg4 present but not arg2/arg3!'
00971        endif
00972     else
00973        if(present(arg3)) then
00974           if(present(arg2)) then
00975              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
00976              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg3) =
      ',lbound(arg3),ubound(arg3)
00977              if(handle_rx%filled) then
00978                 call interpol( &
00979                      reshape(arg1,(/size(arg1)/)), &
00980                      reshape(arg2,(/size(arg2)/)), &
00981                      reshape(arg3,(/size(arg3)/)), &
00982                      fd1=out, grid=handle_rx%grid, &
00983                      data_error=data_error)
00984                 if(present(error_status)) then
00985                    error_status%flag = data_error%ierr .ne. 0
00986                    error_status%string = data_error%cerr
00987                 else
00988                    if(data_error%ierr.ne.0) then
00989                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
00990                       call errorstop('interpol returned an error')
```

```
00991                  endif
00992               endif
00993            else
00994               write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
00995            endif
00996         else
00997            write(*,*) 'IMAS_AMNS_RX_1: arg3 present but not arg2!'
00998         endif
00999      else
01000         if(present(arg2)) then
01001            if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01002            if(handle_rx%filled) then
01003               call interpol( &
01004                    reshape(arg1,(/size(arg1)/)), &
01005                    reshape(arg2,(/size(arg2)/)), &
01006                    fd1=out, grid=handle_rx%grid, &
01007                    data_error=data_error)
01008               if(present(error_status)) then
01009                  error_status%flag = data_error%ierr .ne. 0
01010                  error_status%string = data_error%cerr
01011               else
01012                  if(data_error%ierr.ne.0) then
01013                     write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01014                     call errorstop('interpol returned an error')
01015                  endif
01016               endif
01017            else
01018               write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
01019            endif
01020         else
01021            if(handle_rx%filled) then
01022               call interpol( &
01023                    reshape(arg1,(/size(arg1)/)), &
01024                    fd1=out, grid=handle_rx%grid, &
01025                    data_error=data_error)
01026               if(present(error_status)) then
01027                  error_status%flag = data_error%ierr .ne. 0
01028                  error_status%string = data_error%cerr
01029               else
01030                  if(data_error%ierr.ne.0) then
01031                     write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01032                     call errorstop('interpol returned an error')
01033                  endif
01034               endif
01035            else
01036               write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
01037            endif
01038         endif
01039      endif
01040   endif
01041   if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1 end'
01042
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.7.1.4 imas_amns_rx_2()

```
subroutine amns_module::imas_amns_rx_2 (
             type(amns_handle_rx_type), intent(inout) handle_rx,
             real (kind=ids_real), dimension(:,:), intent(out) out,
             real (kind=ids_real), dimension(:,:), intent(in) arg1,
             real (kind=ids_real), dimension(:,:), intent(in), optional arg2,
             real (kind=ids_real), dimension(:,:), intent(in), optional arg3,
             real (kind=ids_real), dimension(:,:), intent(in), optional arg4,
             type(amns_error_type), intent(out), optional error_status )
```

get the rates associated with the input (2d) args for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system (2d array)
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; (2d array) of the same size as out
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; (2d array) of the same size as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; (2d array) of the same size as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; (2d array) of the same size as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

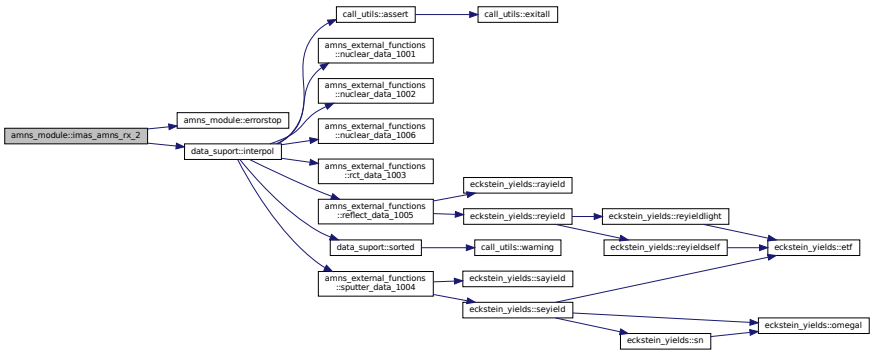
Definition at line 1054 of file amns_module.f90.

```
01055     use ids_types  ! IGNORE
01056     use amns_types
01057     use data_suport
01058     implicit none
01059     optional arg2,arg3,arg4,error_status
01060     type(amns_handle_rx_type), intent(inout) :: handle_rx
01061     real (kind=ids_real), intent(out) :: out(:,:)
01062     real (kind=ids_real), intent(in) :: arg1(:,:),arg2(:,:),arg3(:,:),arg4(:,:)
01063     type(amns_error_type), intent(out) :: error_status
01064
```

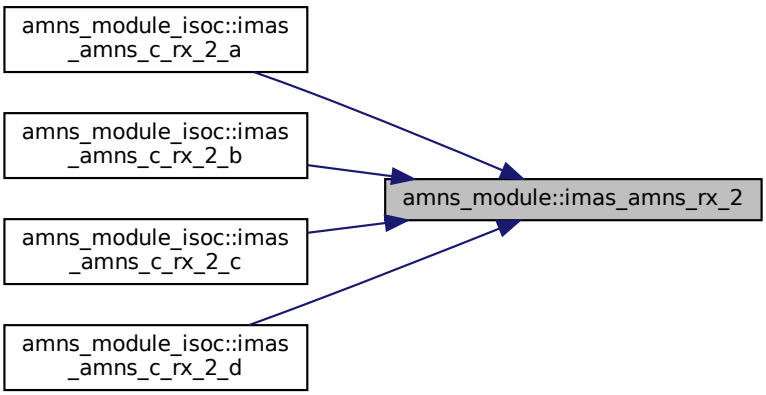
```
01065     real (kind=ids_real) :: tmp_out(size(out))
01066
01067     type(data_error_t) :: data_error
01068
01069     if(present(error_status)) then
01070         error_status%flag=.false.
01071         error_status%string="No error"
01072     end if
01073
01074     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2 start'
01075     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(out) = ',lbound(out),ubound(out)
01076     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg1) = ',lbound(arg1),ubound(arg1)
01077     if(present(arg4)) then
01078        if(present(arg3).and.present(arg2)) then
01079           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) = ',lbound(arg2),ubound(arg2)
01080           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg3) = ',lbound(arg3),ubound(arg3)
01081           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg4) = ',lbound(arg4),ubound(arg4)
01082           if(handle_rx%filled) then
01083              call interpol( &
01084                   reshape(arg1,(/size(arg1)/)), &
01085                   reshape(arg2,(/size(arg2)/)), &
01086                   reshape(arg3,(/size(arg3)/)), &
01087                   reshape(arg4,(/size(arg4)/)), &
01088                   fd1=tmp_out, grid=handle_rx%grid, &
01089                   data_error=data_error)
01090              if(present(error_status)) then
01091                 error_status%flag = data_error%ierr .ne. 0
01092                 error_status%string = data_error%cerr
01093              else
01094                 if(data_error%ierr.ne.0) then
01095                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
01096                    call errorstop('interpol returned an error')
01097                 endif
01098              endif
01099              out=reshape(tmp_out,shape(out))
01100           else
01101              write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01102           endif
01103        else
01104           write(*,*) 'IMAS_AMNS_RX_2: arg4 present but not arg2/arg3!'
01105        endif
01106     else
01107        if(present(arg3)) then
01108           if(present(arg2)) then
01109              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) =
       ',lbound(arg2),ubound(arg2)
01110              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg3) =
       ',lbound(arg3),ubound(arg3)
01111              if(handle_rx%filled) then
01112                 call interpol( &
01113                      reshape(arg1,(/size(arg1)/)), &
01114                      reshape(arg2,(/size(arg2)/)), &
01115                      reshape(arg3,(/size(arg3)/)), &
01116                      fd1=tmp_out, grid=handle_rx%grid, &
01117                      data_error=data_error)
01118                 if(present(error_status)) then
01119                    error_status%flag = data_error%ierr .ne. 0
01120                    error_status%string = data_error%cerr
01121                 else
01122                    if(data_error%ierr.ne.0) then
01123                       write(*,*) 'interpol returned an error: ', data_error%ierr,
       trim(data_error%cerr)
01124                       call errorstop('interpol returned an error')
01125                    endif
01126                 endif
01127                 out=reshape(tmp_out,shape(out))
01128              else
01129                 write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01130              endif
01131           else
01132              write(*,*) 'IMAS_AMNS_RX_2: arg3 present but not arg2!'
01133           endif
01134        else
01135           if(present(arg2)) then
01136              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) =
       ',lbound(arg2),ubound(arg2)
01137              if(handle_rx%filled) then
01138                 call interpol( &
01139                      reshape(arg1,(/size(arg1)/)), &
01140                      reshape(arg2,(/size(arg2)/)), &
01141                      fd1=tmp_out, grid=handle_rx%grid, &
01142                      data_error=data_error)
01143                 if(present(error_status)) then
01144                    error_status%flag = data_error%ierr .ne. 0
01145                    error_status%string = data_error%cerr
01146                 else
01147                    if(data_error%ierr.ne.0) then
```

```
01148                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01149                      call errorstop('interpol returned an error')
01150                   endif
01151                endif
01152                out=reshape(tmp_out,shape(out))
01153             else
01154                write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01155             endif
01156          else
01157             if(handle_rx%filled) then
01158                call interpol( &
01159                     reshape(arg1,(/size(arg1)/)), &
01160                     fd1=tmp_out, grid=handle_rx%grid, &
01161                     data_error=data_error)
01162                if(present(error_status)) then
01163                   error_status%flag = data_error%ierr .ne. 0
01164                   error_status%string = data_error%cerr
01165                else
01166                   if(data_error%ierr.ne.0) then
01167                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01168                      call errorstop('interpol returned an error')
01169                   endif
01170                endif
01171                out=reshape(tmp_out,shape(out))
01172             else
01173                write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01174             endif
01175          endif
01176       endif
01177    endif
01178    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2 end'
01179
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.7.1.5 imas_amns_rx_3()

```
subroutine amns_module::imas_amns_rx_3 (
             type(amns_handle_rx_type), intent(inout) handle_rx,
             real (kind=ids_real), dimension(:,:,:), intent(out) out,
             real (kind=ids_real), dimension(:,:,:), intent(in) arg1,
             real (kind=ids_real), dimension(:,:,:), intent(in), optional arg2,
             real (kind=ids_real), dimension(:,:,:), intent(in), optional arg3,
             real (kind=ids_real), dimension(:,:,:), intent(in), optional arg4,
             type(amns_error_type), intent(out), optional error_status )
```

get the rates associated with the input (3d) args for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system (3d array)
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; (3d array) of the same size as out
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; (3d array) of the same size as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; (3d array) of the same size as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; (3d array) of the same size as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

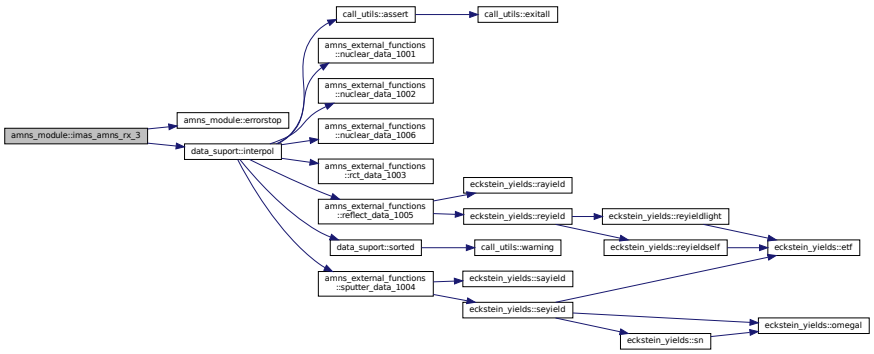
Definition at line 1191 of file amns_module.f90.

```
01192     use ids_types  ! IGNORE
01193     use amns_types
01194     use data_suport
01195     implicit none
01196     optional arg2,arg3,arg4,error_status
01197     type(amns_handle_rx_type), intent(inout) :: handle_rx
01198     real (kind=ids_real), intent(out) :: out(:,:,:)
01199     real (kind=ids_real), intent(in) :: arg1(:,:,:),arg2(:,:,:),arg3(:,:,:),arg4(:,:,:)
01200     type(amns_error_type), intent(out) :: error_status
01201
```

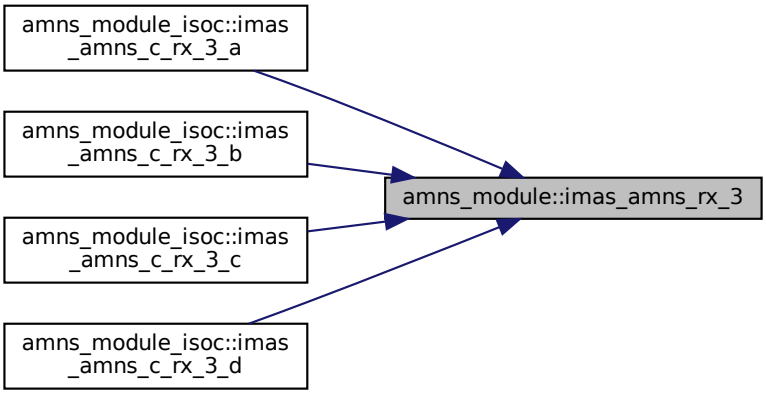
```
01202     real (kind=ids_real) :: tmp_out(size(out))
01203
01204     type(data_error_t) :: data_error
01205
01206     if(present(error_status)) then
01207         error_status%flag=.false.
01208         error_status%string="No error"
01209     end if
01210
01211     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3 start'
01212     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(out) = ',lbound(out),ubound(out)
01213     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg1) = ',lbound(arg1),ubound(arg1)
01214     if(present(arg4)) then
01215        if(present(arg3).and.present(arg2)) then
01216           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) = ',lbound(arg2),ubound(arg2)
01217           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg3) = ',lbound(arg3),ubound(arg3)
01218           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg4) = ',lbound(arg4),ubound(arg4)
01219           if(handle_rx%filled) then
01220              call interpol( &
01221                   reshape(arg1,(/size(arg1)/)), &
01222                   reshape(arg2,(/size(arg2)/)), &
01223                   reshape(arg3,(/size(arg3)/)), &
01224                   reshape(arg4,(/size(arg4)/)), &
01225                   fd1=tmp_out, grid=handle_rx%grid, &
01226                   data_error=data_error)
01227              if(present(error_status)) then
01228                 error_status%flag = data_error%ierr .ne. 0
01229                 error_status%string = data_error%cerr
01230              else
01231                 if(data_error%ierr.ne.0) then
01232                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
01233                    call errorstop('interpol returned an error')
01234                 endif
01235              endif
01236              out=reshape(tmp_out,shape(out))
01237           else
01238              write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01239           endif
01240        else
01241           write(*,*) 'IMAS_AMNS_RX_3: arg4 present but not arg2/arg3!'
01242        endif
01243     else
01244        if(present(arg3)) then
01245           if(present(arg2)) then
01246              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01247              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg3) =
      ',lbound(arg3),ubound(arg3)
01248              if(handle_rx%filled) then
01249                 call interpol( &
01250                      reshape(arg1,(/size(arg1)/)), &
01251                      reshape(arg2,(/size(arg2)/)), &
01252                      reshape(arg3,(/size(arg3)/)), &
01253                      fd1=tmp_out, grid=handle_rx%grid, &
01254                      data_error=data_error)
01255                 if(present(error_status)) then
01256                    error_status%flag = data_error%ierr .ne. 0
01257                    error_status%string = data_error%cerr
01258                 else
01259                    if(data_error%ierr.ne.0) then
01260                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01261                       call errorstop('interpol returned an error')
01262                    endif
01263                 endif
01264                 out=reshape(tmp_out,shape(out))
01265              else
01266                 write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01267              endif
01268           else
01269              write(*,*) 'IMAS_AMNS_RX_3: arg3 present but not arg2!'
01270           endif
01271        else
01272           if(present(arg2)) then
01273              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01274              if(handle_rx%filled) then
01275                 call interpol( &
01276                      reshape(arg1,(/size(arg1)/)), &
01277                      reshape(arg2,(/size(arg2)/)), &
01278                      fd1=tmp_out, grid=handle_rx%grid, &
01279                      data_error=data_error)
01280                 if(present(error_status)) then
01281                    error_status%flag = data_error%ierr .ne. 0
01282                    error_status%string = data_error%cerr
01283                 else
01284                    if(data_error%ierr.ne.0) then
```

```
01285                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01286                      call errorstop('interpol returned an error')
01287                   endif
01288                endif
01289                out=reshape(tmp_out,shape(out))
01290             else
01291                write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01292             endif
01293          else
01294             if(handle_rx%filled) then
01295                call interpol( &
01296                     reshape(arg1,(/size(arg1)/)), &
01297                     fd1=tmp_out, grid=handle_rx%grid, &
01298                     data_error=data_error)
01299                if(present(error_status)) then
01300                   error_status%flag = data_error%ierr .ne. 0
01301                   error_status%string = data_error%cerr
01302                else
01303                   if(data_error%ierr.ne.0) then
01304                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01305                      call errorstop('interpol returned an error')
01306                   endif
01307                endif
01308                out=reshape(tmp_out,shape(out))
01309             else
01310                write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01311             endif
01312          endif
01313       endif
01314    endif
01315    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3 end'
01316
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.7.2 Variable Documentation

14.7.2.1 amns00

```
type (ids_amns_data), save amns_module::amns00
```

Definition at line 37 of file amns_module.f90.

```
00037   type (ids_amns_data), save      :: amns00
```

14.7.2.2 backend

```
character (len=32), save amns_module::backend ='mdsplus'
```

Definition at line 36 of file amns_module.f90.

14.7.2.3 ds_version

```
character (len=32), save amns_module::ds_version ='3'
```

Definition at line 36 of file amns_module.f90.

```
00036   character (len=32), save        :: ds_version='3', backend='mdsplus'
```

14.7.2.4 user

```
character (len=256), save amns_module::user
```

Definition at line 35 of file amns_module.f90.

```
00035   character (len=256), save       :: USER
```

14.7.2.5 version_no

```
integer, save amns_module::version_no
```

Definition at line 34 of file amns_module.f90.

```
00034   integer, save                   :: version_no
```

14.8 amns_module_isoc Module Reference

Data Types

- interface copy

Functions/Subroutines

- pure character(size(a)) function copy_a2s (a)
- pure character function, dimension(len(s)) copy_s2a (s)
- subroutine imas_amns_c_setup (handle, error_status_fc)

  provides IMAS_AMNS_C_SETUP by calling IMAS_AMNS_SETUP
- subroutine imas_amns_c_setup_version (handle, version_fc, error_status_fc)

  provides IMAS_AMNS_C_SETUP by calling IMAS_AMNS_SETUP with version argument
- subroutine imas_amns_c_finish (handle, error_status_fc)

  provides IMAS_AMNS_C_FINISH by calling IMAS_AMNS_FINISH
- subroutine imas_amns_c_finish_table (handle_rx, error_status_fc)

  provides IMAS_AMNS_C_FINISH_TABLE by calling IMAS_AMNS_FINISH_TABLE
- subroutine imas_amns_c_set (handle, set_fc, error_status_fc)

  provides IMAS_AMNS_C_SET by calling IMAS_AMNS_SET
- subroutine imas_amns_c_query (handle, query_fc, answer_fc, error_status_fc)

  provides IMAS_AMNS_C_QUERY by calling IMAS_AMNS_QUERY
- subroutine imas_amns_c_setup_table (handle, reaction_type_fc, reactant, handle_rx, error_status_fc)

  provides IMAS_AMNS_C_SETUP_TABLE by calling IMAS_AMNS_SETUP_TABLE
- subroutine imas_amns_c_query_table (handle_rx, query_fc, answer_fc, error_status_fc)

  provides IMAS_AMNS_C_QUERY_TABLE by calling IMAS_AMNS_QUERY_TABLE
- subroutine imas_amns_c_set_table (handle_rx, set_fc, error_status_fc)

  provides IMAS_AMNS_C_SET_TABLE by calling IMAS_AMNS_SET_TABLE
- subroutine imas_amns_c_rx_0_a (handle_rx, out, arg1, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine imas_amns_c_rx_0_b (handle_rx, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine imas_amns_c_rx_0_c (handle_rx, out, arg1, arg2, arg3, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine imas_amns_c_rx_0_d (handle_rx, out, arg1, arg2, arg3, arg4, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine imas_amns_c_rx_1_a (handle_rx, nx, out, arg1, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine imas_amns_c_rx_1_b (handle_rx, nx, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine imas_amns_c_rx_1_c (handle_rx, nx, out, arg1, arg2, arg3, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine imas_amns_c_rx_1_d (handle_rx, nx, out, arg1, arg2, arg3, arg4, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine imas_amns_c_rx_2_a (handle_rx, nx, ny, out, arg1, error_status_fc)

get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine imas_amns_c_rx_2_b (handle_rx, nx, ny, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine imas_amns_c_rx_2_c (handle_rx, nx, ny, out, arg1, arg2, arg3, error_status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine imas_amns_c_rx_2_d (handle_rx, nx, ny, out, arg1, arg2, arg3, arg4, error_status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine imas_amns_c_rx_3_a (handle_rx, nx, ny, nz, out, arg1, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine imas_amns_c_rx_3_b (handle_rx, nx, ny, nz, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into three separate subroutine for 1,2 or 3 arguments.

- subroutine imas_amns_c_rx_3_c (handle_rx, nx, ny, nz, out, arg1, arg2, arg3, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into three separate subroutine for 1,2 or 3 arguments.

- subroutine imas_amns_c_rx_3_d (handle_rx, nx, ny, nz, out, arg1, arg2, arg3, arg4, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine imas_amns_c_setup_reactants (reactants_handle, string, index, n_reactants)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

- subroutine imas_amns_c_set_reactant (reactants_handle, reactant_index, reactant)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

- subroutine imas_amns_c_get_reactant (reactants_handle, reactant_index, reactant)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

- subroutine imas_amns_c_finish_reactants (reactants_handle)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

14.8.1 Function/Subroutine Documentation

14.8.1.1 copy_a2s()

```
pure character(size(a)) function amns_module_isoc::copy_a2s (
            character, dimension(:), intent(in) a )
```

Definition at line 25 of file amns_module_isoc.f90.

```
00026     character,intent(in) :: a(:)
00027     character(size(a)) :: s
00028     integer :: i
00029     do i = 1,size(a)
00030        s(i:i) = a(i)
00031     end do
```

14.8.1.2 copy_s2a()

```
pure character function, dimension(len(s)) amns_module_isoc::copy_s2a (
            character(*), intent(in) s )
```

Definition at line 35 of file amns_module_isoc.f90.

```
00036     character(*),intent(in) :: s
00037     character :: a(len(s))
00038     integer :: i
00039     do i = 1,len(s)
00040        a(i) = s(i:i)
00041     end do
```

14.9 amns_nuclear Module Reference

Functions/Subroutines

- subroutine nuclear_amns (amns, zn, am)
- subroutine allocate_process (process, nr, np)
- subroutine assign_reactantproduct (reactantproduct, label, zn, amn, multiplicity, relative, za)
- def nuclear_HB_tt (r1, r2, p1, p2, reac)
- def Energy (R)

Variables

- integer, parameter r8 = SELECTED_REAL_KIND (15, 300)
- dictionary masses
- amnsdb = amns.Amns()
- def D_D_p_T = nuclear_HB_tt((1,0,2), (1,0,2), (1,0,1), (1,0,3), b'NUC_TT')
- def D_D_n_He = nuclear_HB_tt((1,0,2), (1,0,2), (0,0,1), (2,0,3), b'NUC_TT')
- def D_T_n_He = nuclear_HB_tt((1,0,2), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
- def D_He_p_He = nuclear_HB_tt((1,0,2), (2,0,3), (1,0,1), (2,0,4), b'NUC_TT')
- def T_T_n_He = nuclear_HB_tt((1,0,3), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
- E = np.array((10**(np.arange(101)/((101-1)/5.0))*10),order='F')
- figsize
- def Rxn = Energy(Rxn['R']) ; plt.loglog(E, Rxn['RX'].data(E), label="%s\n$E_T=%5.2f,\;E_1=%5.2f,\;E_↩
  2=%5.2f$ MeV" % (Rxn['RX'].result_label, E_T/1e6, E_1/1e6, E_2/1e6))
- loc
- fontsize
- label

14.9.1 Function/Subroutine Documentation

14.9.1.1 allocate_process()

```
subroutine amns_nuclear::allocate_process (
            type (ids_amns_data_process) process,
            integer nr,
            integer np )
```

Definition at line 545 of file amns_nuclear.f90.

```
00546     use ids_schemas   ! IGNORE
00547     implicit none
00548     type (ids_amns_data_process) :: process
00549     integer :: nr, np
00550     integer :: irp
00551
00552     allocate(process%reactants(nr), process%products(np))
00553     do irp = 1, size(process%reactants)
00554        allocate(process%reactants(irp)%label(1))
00555        allocate(process%reactants(irp)%element(1))
00556 !       allocate(process%reactants(irp)%element(1)%label(1))
00557     enddo
```

```
00558     do irp = 1, size(process%products)
00559        allocate(process%products(irp)%label(1))
00560        allocate(process%products(irp)%element(1))
00561 !       allocate(process%products(irp)%element(1)%label(1))
00562     enddo
00563
```

14.9.1.2 assign_reactantproduct()

```
subroutine amns_nuclear::assign_reactantproduct (
             type (ids_amns_data_process_reactant) reactantproduct,
             character(len=* ) label,
             integer zn,
             real (kind=r8) amn,
             integer multiplicity,
             integer relative,
             real (kind=r8) za )
```

Definition at line 566 of file amns_nuclear.f90.

```
00567     use ids_schemas   ! IGNORE
00568 !    use imas_types       ! IGNORE
00569     implicit none
00570     type (ids_amns_data_process_reactant) :: reactantproduct
00571     character(len=* ):: label
00572     integer :: zn, multiplicity, relative
00573     real (kind=r8) :: amn, za
00574
00575     reactantproduct%label(1)=label
00576 !    reactantproduct%element(1)%label(1)=label
00577     reactantproduct%element(1)%z_n=zn
00578     reactantproduct%element(1)%a=nint(amn)
00579     reactantproduct%element(1)%multiplicity=multiplicity
00580     reactantproduct%mass=amn
00581     reactantproduct%relative_charge=relative
00582     reactantproduct%charge=za
00583
```

14.9.1.3 Energy()

```
def amns_nuclear.Energy (
              R )
```

Definition at line 28 of file amns_nuclear.py.

```
00028 def Energy(R):
00029     mass = []
00030     mass_excess = 0.0; r = 0; p = 0;
00031     for S in R.value():
00032         zn = int(round(S["ZN"]))
00033         mi = int(round(S["MI"]))
00034         try:
00035             m = masses[zn][mi]
00036             mass.append(m)
00037         except KeyError as exc:
00038             raise NotImplementedError("unknown mass for particle with atomic mass number %s and %s
      protons" %(mi, zn)) from None
00039         if S["LR"] == 0:
00040             r = r + 1
00041             mass_excess = mass_excess + m
00042         else:
00043             p = p + 1
00044             mass_excess = mass_excess - m
00045     assert r == 2, 'coded for two reactants and two products'
00046     assert p == 2, 'coded for two reactants and two products'
00047     E_t =  mass_excess * const.c**2 / const.eV
00048     E_1 = E_t / (mass[2]/mass[3] + 1)
00049     E_2 = E_t / (mass[3]/mass[2] + 1)
00050     return E_t, E_1, E_2
00051
```

14.9.1.4 nuclear_amns()

```
subroutine amns_nuclear::nuclear_amns (
             type (ids_amns_data) amns,
```

```
            integer zn,
            integer am )
```

Definition at line 7 of file amns_nuclear.f90.

```
00008     use ids_schemas   ! IGNORE
00009     use codata, only: &
00010          & e_mass_in_amu => electron_mass_in_u, &
00011          & n_mass_in_amu => neutron_mass_in_u,  &
00012          & h_mass_in_amu => proton_mass_in_u,   &
00013          & d_mass_in_amu => deuteron_mass_in_u, &
00014          & t_mass_in_amu => triton_mass_in_u,   &
00015          & he3_mass_in_amu => helion_mass_in_u, &
00016          & he4_mass_in_amu => alpha_particle_mass_in_u
00017
00018     use beamtargetreactions , only : &
00019          & btr_reaction_dtn4he,  &
00020          & btr_reaction_tdn4he,  &
00021          & btr_reaction_ddpt,    &
00022          & btr_reaction_ddn3he,  &
00023          & btr_reaction_d3hep4he,&
00024          & btr_reaction_3hedp4he
00025
00026     implicit none
00027
00028     type (ids_amns_data) :: amns
00029     integer :: zn, am
00030     integer :: nprocs, iproc
00031     integer :: ncoords,icoord
00032     integer :: npoints(2)
00033     integer :: i,iene,itemp,btreaction,err
00034     real(kind=r8) :: emin,emax,tmin,tmax
00035     integer :: lastWrittenCoordinate !This is used to keep track of when the coords have been already
      written
00036     ! when calling add_beam_target multiple times
00037
00038     lastwrittencoordinate=0
00039
00040     nprocs=0
00041     if(zn.eq.1) then           ! H, D or T
00042        if(am.eq.2) then        ! D (shot = 2001)
00043           nprocs=6
00044           ncoords=1
00045           allocate(amns%coordinate_system(ncoords))
00046           allocate(amns%process(nprocs))
00047           do iproc=1, nprocs
00048              allocate(amns%process(iproc)%label(1))
00049              allocate(amns%process(iproc)%result_label(1))
00050              allocate(amns%process(iproc)%result_units(1))
00051              allocate(amns%process(iproc)%charge_state(1))
00052              allocate(amns%process(iproc)%source(1))
00053              amns%process(iproc)%source = 'amns_driver/amns_nuclear.f90'
00054              allocate(amns%process(iproc)%provider(1))
00055              amns%process(iproc)%provider = 'David.Coster@ipp.mpg.de on behalf of AMNS Team'
00056              allocate(amns%process(iproc)%citation(1))
00057              amns%process(iproc)%citation = 'ToDo'
00058           enddo
00059
00060           iproc=1
00061           amns%process(iproc)%label(1)='NUC_BB'                ! D(D,p)T
00062           call allocate_process(amns%process(iproc), 2, 2)
00063           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00064                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00065           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00066                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00067           call assign_reactantproduct(amns%process(iproc)%products(1), &
00068                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00069           call assign_reactantproduct(amns%process(iproc)%products(2), &
00070                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00071           amns%process(iproc)%result_label = 'cross section for D(D,p)T'
00072           amns%process(iproc)%result_units = 'm^{2}'
00073           amns%process(iproc)%result_transformation = 1001
00074           amns%process(iproc)%table_dimension = 1
00075           amns%process(iproc)%coordinate_index = -1
00076           allocate(amns%process(iproc)%charge_state(1)%table_2d(12,1))
00077           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00078                (/ 31.3970_r8,  &
00079                   5.5576e4_r8, 2.1054e2_r8, -3.2638e-2_r8, 1.4987e-6_r8, 1.8181e-10_r8, &
00080                   0.0_r8, 0.0_r8, 0.0_r8, 0.0_r8, &
00081                   0.5_r8, 5000.0_r8 /), &
00082                (/ 12, 1 /) )
00083
00084           iproc=2
00085           amns%process(iproc)%label(1)='NUC_BB'                ! D(D,n)^3He
00086           call allocate_process(amns%process(iproc), 2, 2)
00087           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00088                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00089           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
```

```
00090             'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00091          call assign_reactantproduct(amns%process(iproc)%products(1), &
00092             'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00093          call assign_reactantproduct(amns%process(iproc)%products(2), &
00094             'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00095          amns%process(iproc)%result_label = 'cross section for D(D,n)^3He'
00096          amns%process(iproc)%result_units = 'm^{2}'
00097          amns%process(iproc)%result_transformation = 1001
00098          amns%process(iproc)%table_dimension = 1
00099          amns%process(iproc)%coordinate_index = -1
00100          allocate(amns%process(iproc)%charge_state(1)%table_2d(12,1))
00101          amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00102               (/ 31.3970_r8, &
00103                  5.3701e4_r8, 3.3027e2_r8, -1.2706e-1_r8, 2.9327e-5_r8, -2.5151e-9_r8, &
00104                  0.0_r8, 0.0_r8, 0.0_r8, 0.0_r8, &
00105                  0.5_r8, 4900.0_r8 /), &
00106               (/ 12, 1 /) )
00107
00108          iproc=3
00109          amns%process(iproc)%label(1)='NUC_TT'                ! tt D(D,p)T
00110          call allocate_process(amns%process(iproc), 2, 2)
00111          call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00112               'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00113          call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00114               'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00115          call assign_reactantproduct(amns%process(iproc)%products(1), &
00116               'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00117          call assign_reactantproduct(amns%process(iproc)%products(2), &
00118               'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00119          amns%process(iproc)%result_label = 'reactivity for tt D(D,p)T'
00120          amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00121          amns%process(iproc)%result_transformation = 1002
00122          amns%process(iproc)%table_dimension = 1
00123          amns%process(iproc)%coordinate_index = -1
00124          allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00125          amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00126               (/ 31.3970_r8,     937814.0_r8,&
00127               & 5.65718e-12_r8,   3.41267e-3_r8,   1.99167e-3_r8,  0.0_r8, &
00128               & 1.05060e-5_r8,  0.0_r8,          0.0_r8,  &
00129                  0.2_r8, 100.0_r8 /), &
00130               (/ 11, 1 /) )
00131
00132          iproc=4
00133          amns%process(iproc)%label(1)='NUC_TT'                ! tt D(D,n)^3He
00134          call allocate_process(amns%process(iproc), 2, 2)
00135          call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00136               'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00137          call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00138               'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00139          call assign_reactantproduct(amns%process(iproc)%products(1), &
00140               'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00141          call assign_reactantproduct(amns%process(iproc)%products(2), &
00142               'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00143          amns%process(iproc)%result_label = 'reactivity for tt D(D,n)^3He'
00144          amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00145          amns%process(iproc)%result_transformation = 1002
00146          amns%process(iproc)%table_dimension = 1
00147          amns%process(iproc)%coordinate_index = -1
00148          allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00149          amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00150               (/ 31.3970_r8,       937814.0_r8,&
00151               &  5.43360e-12_r8,    5.85778e-3_r8,   7.68222e-3_r8,  0.0_r8, &
00152               &  -2.96400e-6_r8,  0.0_r8,         0.0_r8,  &
00153               &  0.2_r8, 100.0_r8 /), &
00154               (/ 11, 1 /) )
00155
00156          iproc=5
00157          icoord=1
00158          amns%process(iproc)%label(1)='bt D(D,p)T'
00159          call allocate_process(amns%process(iproc), 2, 2)
00160          call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00161               'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00162          call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00163               'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00164          call assign_reactantproduct(amns%process(iproc)%products(1), &
00165               'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00166          call assign_reactantproduct(amns%process(iproc)%products(2), &
00167               'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00168          btreaction= btr_reaction_ddpt ! these are defined in beamTargetReactions.F90
00169          call add_beam_target(iproc,icoord,btreaction)
00170          amns%process(iproc)%label(1)='NUC_BT'                ! bt D(D,p)T
00171
00172          iproc=6
00173          icoord=1
00174          amns%process(6)%label(1)='bt D(D,n)^3He'
00175          call allocate_process(amns%process(6), 2, 2)
00176          call assign_reactantproduct(amns%process(6)%reactants(1), &
```

```
00177              'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00178           call assign_reactantproduct(amns%process(6)%reactants(2), &
00179              'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00180           call assign_reactantproduct(amns%process(6)%products(1), &
00181              'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00182           call assign_reactantproduct(amns%process(6)%products(2), &
00183              'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00184           btreaction= btr_reaction_ddn3he ! these are defined in beamTargetReactions.F90
00185           call add_beam_target(iproc,icoord,btreaction)
00186           amns%process(6)%label(1)='NUC_BT'                ! bt D(D,n)^3He
00187
00188        elseif(am.eq.3) then    ! T (shot = 3001)
00189           nprocs=6
00190           ncoords=1
00191           allocate(amns%coordinate_system(ncoords))
00192           allocate(amns%process(nprocs))
00193           do iproc=1, nprocs
00194              allocate(amns%process(iproc)%label(1))
00195              allocate(amns%process(iproc)%result_label(1))
00196              allocate(amns%process(iproc)%result_units(1))
00197              allocate(amns%process(iproc)%charge_state(1))
00198           enddo
00199
00200           iproc=1
00201           amns%process(iproc)%label(1)='NUC_BB'                ! D(T,n)^4He
00202           call allocate_process(amns%process(iproc), 2, 2)
00203           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00204              'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00205           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00206              'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00207           call assign_reactantproduct(amns%process(iproc)%products(1), &
00208              'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00209           call assign_reactantproduct(amns%process(iproc)%products(2), &
00210              'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00211           amns%process(iproc)%result_label = 'cross section for D(T,n)^4He'
00212           amns%process(iproc)%result_units = 'm^{2}'
00213           amns%process(iproc)%result_transformation = 1001
00214           amns%process(iproc)%table_dimension = 1
00215           amns%process(iproc)%coordinate_index = -1
00216           allocate(amns%process(iproc)%charge_state(1)%table_2d(12,2))
00217           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00218                (/ 34.3827_r8,  &
00219                   6.927e4_r8, 7.454e8_r8, 2.050e6_r8, 5.2002e4_r8, 0.0_r8, &
00220                   6.38e1_r8, -9.95e-1_r8, 6.981e-5_r8, 1.728e-4_r8, &
00221                   0.5_r8, 530.0_r8, &
00222                   34.3827_r8, &
00223                   -1.4714e6_r8, 0.0_r8, 0.0_r8, 0.0_r8, 0.0_r8, &
00224                   -8.4127e-3_r8, 4.7983e-6_r8, -1.0748e-9_r8, 8.5184e-14_r8, &
00225                   530.0_r8, 4700.0_r8 /), &
00226                (/ 12, 2 /) )
00227
00228           iproc=2
00229           amns%process(iproc)%label(1)='NUC_TT'                ! tt D(T,n)^4He
00230           call allocate_process(amns%process(iproc), 2, 2)
00231           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00232                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00233           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00234                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00235           call assign_reactantproduct(amns%process(iproc)%products(1), &
00236                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00237           call assign_reactantproduct(amns%process(iproc)%products(2), &
00238                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00239           amns%process(iproc)%result_label = 'reactivity for tt D(T,n)^4He'
00240           amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00241           amns%process(iproc)%result_transformation = 1002
00242           amns%process(iproc)%table_dimension = 1
00243           amns%process(iproc)%coordinate_index = -1
00244           allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00245           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00246                (/ 34.3827_r8,  1124656.0_r8,&
00247                &  1.17302e-9_r8,  1.51361e-2_r8, 7.51886e-2_r8, 4.60643e-3_r8, &
00248                   1.35000e-2_r8, -1.06750e-4_r8, 1.36600e-5_r8, &
00249                   0.2_r8, 100.0_r8 /), &
00250                (/ 11, 1 /) )
00251
00252           iproc=3
00253           icoord=1
00254           amns%process(iproc)%label(1)='bt D(T,n)^4He'
00255           call allocate_process(amns%process(iproc), 2, 2)
00256           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00257                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00258           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00259                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00260           call assign_reactantproduct(amns%process(iproc)%products(1), &
00261                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00262           call assign_reactantproduct(amns%process(iproc)%products(2), &
00263                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
```

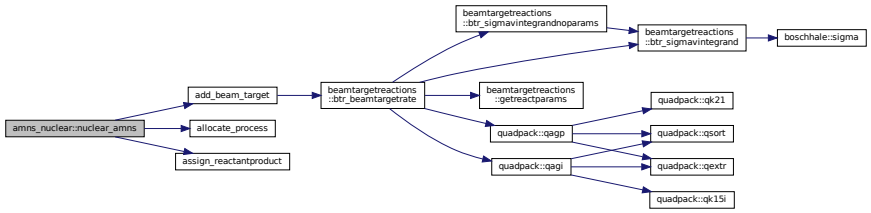
```
00264           btreaction= btr_reaction_dtn4he ! these are defined in beamTargetReactions.F90
00265           call add_beam_target(iproc,icoord,btreaction)
00266           amns%process(iproc)%label(1)='NUC_BT'                 ! bt D(T,n)^4He
00267
00268           iproc=4
00269           icoord=1
00270           amns%process(iproc)%label(1)='bt T(D,n)^4He'
00271           call allocate_process(amns%process(iproc), 2, 2)
00272           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00273                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00274           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00275                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00276           call assign_reactantproduct(amns%process(iproc)%products(1), &
00277                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00278           call assign_reactantproduct(amns%process(iproc)%products(2), &
00279                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00280           btreaction= btr_reaction_tdn4he ! these are defined in beamTargetReactions.F90
00281           call add_beam_target(iproc,icoord,btreaction)
00282           amns%process(iproc)%label(1)='NUC_BT'                 ! bt T(D,n)^4He
00283
00284           iproc=5
00285           amns%process(iproc)%label(1)='NUC_BB'                 ! T(T,2n)^4He
00286           call allocate_process(amns%process(iproc), 2, 2)
00287           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00288                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00289           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00290                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00291           call assign_reactantproduct(amns%process(iproc)%products(1), &
00292                'n', 0, n_mass_in_amu, 2, -1, 0.0_r8)
00293           call assign_reactantproduct(amns%process(iproc)%products(2), &
00294                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00295           amns%process(iproc)%result_label = 'cross section for T(T,2n)^4He'
00296           amns%process(iproc)%result_units = 'm^{2}'
00297           amns%process(iproc)%result_transformation = 1006
00298           amns%process(iproc)%table_dimension = 1
00299           amns%process(iproc)%coordinate_index = -1
00300           allocate(amns%process(iproc)%charge_state(1)%table_2d(14,2))
00301 !
00302 ! Data taken from
00303 !
00304 !  https://gforge6.eufus.eu/svn/amnsproto/curation/nuclear/work_T(t,2n)4He.dat
00305 !
00306 ! revision 640
00307 !
00308           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00309                (/ 38.4224306526_r8, &
00310                   1.83514616e+05_r8, -3.17779782e+02_r8, 3.76709752e+00_r8, &
00311                   -2.60896847e-03_r8, 1.21563195e-06_r8, 6.78016367e-11_r8, &
00312                   3.36105299e-05_r8, 1.86611385e-05_r8, -2.88648569e-08_r8, &
00313                   1.45561737e-11_r8, -5.27692243e-16_r8, &
00314                   0.25_r8, 10.0e03_r8 /), &
00315                (/ 14, 1 /) )
00316
00317           iproc=6
00318           amns%process(iproc)%label(1)='NUC_TT'                 ! tt T(T,2n)^4He
00319           call allocate_process(amns%process(iproc), 2, 2)
00320           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00321                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00322           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00323                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00324           call assign_reactantproduct(amns%process(iproc)%products(1), &
00325                'n', 0, n_mass_in_amu, 2, -1, 0.0_r8)
00326           call assign_reactantproduct(amns%process(iproc)%products(2), &
00327                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00328           amns%process(iproc)%result_label = 'reactivity for tt T(T,2n)^4He'
00329           amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00330           amns%process(iproc)%result_transformation = 1002
00331           amns%process(iproc)%table_dimension = 1
00332           amns%process(iproc)%coordinate_index = -1
00333           allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00334 !
00335 ! Data taken from
00336 !
00337 ! https://gforge6.eufus.eu/svn/amnsproto/curation/nuclear/work_thermonuclear_T(t,2n)4He.dat
00338 !
00339 ! Revision: 641
00340 !
00341           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00342                (/ 38.4224306526_r8, 1404460.50227_r8, &
00343                   2.86867049e-11_r8, -1.26581396e-02_r8, -6.68347948e-01_r8, 6.31775307e-03_r8, &
00344                   1.44900887e-01_r8, -3.93623787e-03_r8, 1.74153264e-02_r8, &
00345                   0.2_r8, 100.0_r8 /), &
00346                (/ 11, 1 /) )
00347
00348        endif
00349
00350
```

```
00351     elseif(zn.eq.2) then       ! 3-He, 4-He
00352        if(am.eq.3) then        ! 3-He (shot = 3002)
00353           nprocs=4
00354           ncoords=1
00355           allocate(amns%coordinate_system(ncoords))
00356           allocate(amns%process(nprocs))
00357           do iproc=1, nprocs
00358              allocate(amns%process(iproc)%label(1))
00359              allocate(amns%process(iproc)%result_label(1))
00360              allocate(amns%process(iproc)%result_units(1))
00361              allocate(amns%process(iproc)%charge_state(1))
00362           enddo
00363
00364           iproc=1
00365           amns%process(iproc)%label(1)='NUC_BB'                 ! D(^3He,p)^4He
00366           call allocate_process(amns%process(iproc), 2, 2)
00367           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00368                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00369           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00370                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00371           call assign_reactantproduct(amns%process(iproc)%products(1), &
00372                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00373           call assign_reactantproduct(amns%process(iproc)%products(2), &
00374                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00375           amns%process(iproc)%result_label = 'cross section for D(^3He,p)^4He'
00376           amns%process(iproc)%result_units = 'm^{2}'
00377           amns%process(iproc)%result_transformation = 1001
00378           amns%process(iproc)%table_dimension = 1
00379           amns%process(iproc)%coordinate_index = -1
00380           allocate(amns%process(iproc)%charge_state(1)%table_2d(12,2))
00381           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00382                (/ 68.7508_r8,  &
00383                   5.7501e6_r8, 2.5226e3_r8, 4.5566e1_r8, 0.0_r8, 0.0_r8, &
00384                   -3.1995e-3_r8, -8.5530e-6_r8, 5.9014e-8_r8, 0.0_r8, &
00385                   0.3_r8, 900.0_r8, &
00386                   68.7508_r8, &
00387                   -8.3993e5_r8, 0.0_r8, 0.0_r8, 0.0_r8, 0.0_r8, &
00388                   -2.6830e-3_r8, 1.1633e-6_r8, -2.1332e-10_r8, 1.4250e-14_r8, &
00389                   900.0_r8, 4800.0_r8 /), &
00390                (/ 12, 2 /) )
00391
00392           iproc=2
00393           amns%process(iproc)%label(1)='NUC_TT'                 ! tt D(^3He,p)^4He
00394           call allocate_process(amns%process(iproc), 2, 2)
00395           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00396                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00397           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00398                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00399           call assign_reactantproduct(amns%process(iproc)%products(1), &
00400                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00401           call assign_reactantproduct(amns%process(iproc)%products(2), &
00402                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00403           amns%process(iproc)%result_label = 'reactivity for tt D(^3He,p)^4He'
00404           amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00405           amns%process(iproc)%result_transformation = 1002
00406           amns%process(iproc)%table_dimension = 1
00407           amns%process(iproc)%coordinate_index = -1
00408           allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00409           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00410                (/ 68.7508_r8, 1124572.0_r8, &
00411                &  5.51036e-10_r8, 6.41918e-3_r8, -2.02896e-3_r8, -1.91080e-5_r8, &
00412                   1.35776e-4_r8,  0.0_r8,         0.0_r8, &
00413                   0.5_r8, 190.0_r8 /), &
00414                (/ 11, 1 /) )
00415
00416           iproc=3
00417           icoord=1
00418           amns%process(iproc)%label(1)='bt ^3He(D,p)^4He'
00419           call allocate_process(amns%process(iproc), 2, 2)
00420           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00421                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00422           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00423                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00424           call assign_reactantproduct(amns%process(iproc)%products(1), &
00425                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00426           call assign_reactantproduct(amns%process(iproc)%products(2), &
00427                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00428           btreaction= btr_reaction_3hedp4he ! these are defined in beamTargetReactions.F90
00429           call add_beam_target(iproc,icoord,btreaction)
00430           amns%process(iproc)%label(1)='NUC_BT'                 ! bt ^3He(D,p)^4He
00431
00432           iproc=4
00433           icoord=1
00434           amns%process(iproc)%label(1)='bt D(^3He,p)^4He'
00435           call allocate_process(amns%process(iproc), 2, 2)
00436           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00437                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
```

```
00438           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00439                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00440           call assign_reactantproduct(amns%process(iproc)%products(1), &
00441                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00442           call assign_reactantproduct(amns%process(iproc)%products(2), &
00443                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00444           btreaction= btr_reaction_d3hep4he ! these are defined in beamTargetReactions.F90
00445           call add_beam_target(iproc,icoord,btreaction)
00446           amns%process(iproc)%label(1)='NUC_BT'                 ! bt D(^3He,p)^4He
00447
00448        endif
00449
00450     endif
00451     if(nprocs.gt.0) then
00452 !       allocate(amns%version(1), amns%source(1))
00453 !       amns%version = 'v0'
00454 !       amns%source  = 'NUCLEAR'
00455        amns%z_n     = zn
00456        amns%a       = am
00457     endif
00458
00459   contains
00460     subroutine add_beam_target(iproc,icoord,btreaction)
00461       use beamtargetreactions , only : btr_beamtargetrate
00462       implicit none
00463       integer, intent(in) :: iproc,icoord,btreaction
00464
00465       allocate(amns%process(iproc)%result_label(1))
00466       allocate(amns%process(iproc)%result_units(1))
00467
00468       amns%process(iproc)%result_label = 'Reaction rate for ' // trim(amns%process(iproc)%label(1))
00469       amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00470       amns%process(iproc)%result_transformation = 1 !0=no transformation, 1=10^(), 2=exp()
00471       amns%process(iproc)%table_dimension = 2
00472
00473       ! Set up the coordinates for the table
00474       amns%process(iproc)%coordinate_index = icoord
00475       npoints=(/79,199/)
00476
00477
00478       tmin=2.0_r8 !10^() eV   ! was 2.0_R8 (DPC: 2012-06-28)
00479       tmax=9.0_r8 !10^() eV
00480       emin=1.0_r8 !10^() eV
00481       emax=9.0_r8 !10^() eV   ! was 2.0_R8 (DPC: 2012-06-28)
00482
00483       if(lastwrittencoordinate<icoord) then
00484          allocate(amns%coordinate_system(icoord)%coordinate(2))
00485          do i=1,size(amns%coordinate_system(icoord)%coordinate)
00486             allocate(amns%coordinate_system(icoord)%coordinate(i)%values(npoints(i)),&
00487                  &   amns%coordinate_system(icoord)%coordinate(i)%label(1),&
00488                  &   amns%coordinate_system(icoord)%coordinate(i)%units(1),&
00489                  &   amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(2))
00490             amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(1:2)=1 ! 0= No
    extrapolation
00491             amns%coordinate_system(icoord)%coordinate(i)%interpolation_type=1      ! 1= Linear
    interpolation
00492          end do
00493
00494          amns%coordinate_system(icoord)%coordinate(1)%label(1)='Temperature x kB'
00495          amns%coordinate_system(icoord)%coordinate(2)%label(1)='Particle energy'
00496          amns%coordinate_system(icoord)%coordinate(1)%units(1)='eV'
00497          amns%coordinate_system(icoord)%coordinate(2)%units(1)='eV'
00498          amns%coordinate_system(icoord)%coordinate(1)%transformation = 1
00499          amns%coordinate_system(icoord)%coordinate(2)%transformation = 1
00500          amns%coordinate_system(icoord)%coordinate(1)%spacing = 0
00501          amns%coordinate_system(icoord)%coordinate(2)%spacing = 0
00502
00503          do i=1,npoints(1)
00504             amns%coordinate_system(icoord)%coordinate(1)%values(i) = &
00505                  & tmin + (tmax-tmin)/real(npoints(1)-1,kind=r8)*real(i-1,kind=r8)
00506          end do
00507          do i=1,npoints(2)
00508             amns%coordinate_system(icoord)%coordinate(2)%values(i) = &
00509                  & emin + (emax-emin)/real(npoints(2)-1,kind=r8)*real(i-1,kind=r8)
00510          end do
00511       end if
00512
00513       lastwrittencoordinate=icoord
00514
00515       ! Fill the table itself
00516       allocate(amns%process(iproc)%charge_state(1)%table_2d(npoints(1),npoints(2)))
00517
00518       write(*,*) 'Integrating beam-target fusion reaction "'//trim(amns%process(iproc)%label(1))//'".'
00519       do itemp=1,npoints(1)
00520          do iene=1,npoints(2)
00521             call btr_beamtargetrate(btreaction,&
00522                  & 10.0_r8 ** amns%coordinate_system(icoord)%coordinate(2)%values(iene ),&
```

```
00523                 & 10.0_r8 ** amns%coordinate_system(icoord)%coordinate(1)%values(itemp),&
00524                 & amns%process(iproc)%charge_state(1)%table_2d(itemp,iene),err)
00525            if(err /= 0) then
00526               write(*,*) 'error: reaction',btreaction,&
00527                   &'Temperature',10.0_r8 **
      amns%coordinate_system(icoord)%coordinate(1)%values(itemp),&
00528                   &'Energy',     10.0_r8 ** amns%coordinate_system(icoord)%coordinate(2)%values(iene
      ),&
00529                   &'err',err
00530               write(*,*) 'FILE: "'//__file__//'" line:',__line__
00531               stop 1
00532            end if
00533            if(amns%process(iproc)%charge_state(1)%table_2d(itemp,iene).ge.1e-300_r8) then
00534
      amns%process(iproc)%charge_state(1)%table_2d(itemp,iene)=log10(amns%process(iproc)%charge_state(1)%table_2d(itemp,iene
00535            else
00536               amns%process(iproc)%charge_state(1)%table_2d(itemp,iene)=-300.0_r8
00537            endif
00538         end do
00539      end do
00540   end subroutine add_beam_target
00541
00542
```

Here is the call graph for this function:



14.9.1.5 nuclear_HB_tt()

```
def amns_nuclear.nuclear_HB_tt (
              r1,
              r2,
              p1,
              p2,
              reac )
```

Definition at line 19 of file amns_nuclear.py.

```
00019 def nuclear_HB_tt(r1, r2, p1, p2, reac):
00020
00021     r = amns.Reactants()
00022     r.add(r1[0],r1[1],r1[2])
00023     r.add(r2[0],r2[1],r2[2])
00024     r.add(p1[0],p1[1],p1[2],lr=1)
00025     r.add(p2[0],p2[1],p2[2],lr=1)
00026     return dict(RX=amnsdb.get_table(reac, r, isotope_resolved=1), R=r)
00027
```

14.9.2 Variable Documentation

14.9.2.1 amnsdb

```
amns_nuclear.amnsdb = amns.Amns()
```

Definition at line 52 of file amns_nuclear.py.

14.9.2.2 D_D_n_He

```
def amns_nuclear.D_D_n_He = nuclear_HB_tt( (1,0,2), (1,0,2), (0,0,1), (2,0,3), b'NUC_TT')
```

Definition at line 56 of file amns_nuclear.py.

14.9.2.3 D_D_p_T

```
def amns_nuclear.D_D_p_T = nuclear_HB_tt( (1,0,2), (1,0,2), (1,0,1), (1,0,3), b'NUC_TT')
```
Definition at line 55 of file amns_nuclear.py.

14.9.2.4 D_He_p_He

```
def amns_nuclear.D_He_p_He = nuclear_HB_tt( (1,0,2), (2,0,3), (1,0,1), (2,0,4), b'NUC_TT')
```
Definition at line 58 of file amns_nuclear.py.

14.9.2.5 D_T_n_He

```
def amns_nuclear.D_T_n_He = nuclear_HB_tt( (1,0,2), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
```
Definition at line 57 of file amns_nuclear.py.

14.9.2.6 E

```
amns_nuclear.E = np.array((10**(np.arange(101)/((101-1)/5.0))*10),order='F')
```
Definition at line 61 of file amns_nuclear.py.

14.9.2.7 figsize

```
amns_nuclear.figsize
```
Definition at line 63 of file amns_nuclear.py.

14.9.2.8 fontsize

```
amns_nuclear.fontsize
```
Definition at line 78 of file amns_nuclear.py.

14.9.2.9 label

```
amns_nuclear.label
```
Definition at line 84 of file amns_nuclear.py.

14.9.2.10 loc

```
amns_nuclear.loc
```
Definition at line 78 of file amns_nuclear.py.

14.9.2.11 masses

```
dictionary amns_nuclear.masses
```
Initial value:
```
00001 =  { 0:{ 1:const.m_n },
00002            1:{ 1:const.m_p,
00003                2:const.value('deuteron mass'),
00004                3:const.value('triton mass'),
00005              },
00006            2:{ 3:const.value('helion mass'),
00007                4:const.value('alpha particle mass')
```

```
00008             },
00009         }
```

Definition at line 9 of file amns_nuclear.py.

14.9.2.12 r8

```
integer, parameter amns_nuclear::r8 = SELECTED_REAL_KIND (15, 300)
```

Definition at line 3 of file amns_nuclear.f90.

```
00003   INTEGER,  PARAMETER :: R8 = selected_real_kind (15, 300) ! Real*8
```

14.9.2.13 Rxn

```
def amns_nuclear.Rxn = Energy(Rxn['R']) ; plt.loglog(E, Rxn['RX'].data(E), label="%s\n$E_↩
T=%5.2f,\;E_1=%5.2f,\;E_2=%5.2f$ MeV" % (Rxn['RX'].result_label, E_T/1e6, E_1/1e6, E_2/1e6) )
```

Definition at line 67 of file amns_nuclear.py.

14.9.2.14 T_T_n_He

```
def amns_nuclear.T_T_n_He = nuclear_HB_tt( (1,0,3), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
```

Definition at line 59 of file amns_nuclear.py.

14.10 amns_nuclear_densities Namespace Reference

Functions

- def nuclear_HB_tt (r1, r2, p1, p2, reac)
- def advance_densities (N, M, dt, Nt)

Variables

- Species = np.array([[1,0,1], [1,0,2], [1,0,3], [2,0,3], [2,0,4], [0,0,1]])
- amnsdb = amns.Amns()
- def D_D_p_T = nuclear_HB_tt((1,0,2), (1,0,2), (1,0,1), (1,0,3), b'NUC_TT')
- def D_D_n_He = nuclear_HB_tt((1,0,2), (1,0,2), (0,0,1), (2,0,3), b'NUC_TT')
- def D_T_n_He = nuclear_HB_tt((1,0,2), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
- def D_He_p_He = nuclear_HB_tt((1,0,2), (2,0,3), (1,0,1), (2,0,4), b'NUC_TT')
- def T_T_n_He = nuclear_HB_tt((1,0,3), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
- E = np.array((10**(np.arange(101)/((101-1)/5.0))*10),order='F')
- M = np.zeros([6,6,6,len(E)])
- N = np.outer(np.array([0.0, 5e19, 5e19, 0.0, 0.0, 0.0]), np.ones(len(E)))
- Times
- N_Times
- loc
- Ie
- d = pdf.infodict()

14.10.1 Detailed Description

```
Calculate the 0D densities of H, D, T, He-3, He-4, n produced and/or consumed in thermonuclear reactions.
The results are plotted at the end-time as a function of temperature, and at around 20 keV as a function of ti
Three starting scenarios are explored: 50:50 D:T, 100% D and 100% T.
```

14.10.2 Function Documentation

14.10.2.1 advance_densities()

```
def amns_nuclear_densities.advance_densities (
                N,
                M,
                dt,
                Nt )
```

Given the transition matrix M, advance the densities N by Nt steps of dt

Definition at line 73 of file amns_nuclear_densities.py.

```
00073 def advance_densities(N, M, dt, Nt):
00074     """
00075     Given the transition matrix M, advance the densities N by Nt steps of dt
00076     """
00077     assert N.shape[0] == M.shape[0]
00078     assert M.shape[0] == M.shape[1]
00079     assert M.shape[0] == M.shape[2]
00080     Times = np.arange(Nt+1)*dt
00081     N_Times = np.zeros([len(Times),N.shape[0],N.shape[1]])
00082     N_Times[0] = N
00083     for t,T in enumerate(Times[1:]):
00084         N = N + np.einsum('ijkl,il,jl->kl', M, N, N) * dt
00085         N_Times[t+1] = N
00086     return Times, N_Times
00087
```

14.10.2.2 nuclear_HB_tt()

```
def amns_nuclear_densities.nuclear_HB_tt (
                r1,
                r2,
                p1,
                p2,
                reac )
```

Return a dictionary containing the rate coefficient table, reactants/products, transition matrix, reactant1, r

Definition at line 19 of file amns_nuclear_densities.py.

```
00019 def nuclear_HB_tt(r1, r2, p1, p2, reac):
00020     """
00021     Return a dictionary containing the rate coefficient table, reactants/products, transition matrix,
       reactant1, reactant2, scaling factor for identical reactants
00022     """
00023     i1 = np.array([(r == np.array(r1)).all() for r in Species]).argmax()
00024     i2 = np.array([(r == np.array(r2)).all() for r in Species]).argmax()
00025     i3 = np.array([(r == np.array(p1)).all() for r in Species]).argmax()
00026     i4 = np.array([(r == np.array(p2)).all() for r in Species]).argmax()
00027
00028     V=np.zeros([6])
00029
00030     r = amns.Reactants()
00031     r.add(r1[0],r1[1],r1[2])
00032     r.add(r2[0],r2[1],r2[2])
00033     r.add(p1[0],p1[1],p1[2],lr=1)
00034     r.add(p2[0],p2[1],p2[2],lr=1)
00035
00036     V[i1] = V[i1] - 1
00037     V[i2] = V[i2] - 1
00038     V[i3] = V[i3] + 1
00039     V[i4] = V[i4] + 1
00040
00041     if i1 == i2:
00042         f = 0.5
00043     else:
00044         f = 1.0
00045
00046     # check mass and charge
00047     M = 0; Z = 0;
00048     for r_iter in r.value():
00049         if r_iter['LR'] == 0:
00050             M+=r_iter['MI']
00051             Z+=r_iter['ZN']
00052         else:
00053             M+=-r_iter['MI']
00054             Z+=-r_iter['ZN']
00055     print('Before correction: M = %s, Z = %s' % (M, Z))
```

```
00056     # try to see if doubling one of the products solves the mass or charge discrepancy
00057     # we assume two products are specified, one of which might have multiplocity two
00058     if (M != 0) or (Z != 0):
00059         for i, r_iter in enumerate(r.value()):
00060             if r_iter['LR'] == 1:
00061                 if ((M - r_iter['MI']) == 0) and ((Z - r_iter['ZN']) == 0):
00062                     if i+1 == 3:
00063                         V[i3] += 1
00064                     else:
00065                         V[i4] += 1
00066                     M+=-r_iter['MI']
00067                     Z+=-r_iter['ZN']
00068                     break
00069     print('After correction: M = %s, Z = %s' % (M, Z))
00070 
00071     return dict(RX=amnsdb.get_table(reac, r, isotope_resolved=1), R=r, V=V, I1=i1, I2=i2, F=f)
00072 
```

14.10.3 Variable Documentation

14.10.3.1 amnsdb

```
amns_nuclear_densities.amnsdb = amns.Amns()
```

Definition at line 88 of file amns_nuclear_densities.py.

14.10.3.2 d

```
amns_nuclear_densities.d = pdf.infodict()
```

Definition at line 238 of file amns_nuclear_densities.py.

14.10.3.3 D_D_n_He

```
def amns_nuclear_densities.D_D_n_He = nuclear_HB_tt( (1,0,2), (1,0,2), (0,0,1), (2,0,3), b'NUC_TT')
```

Definition at line 93 of file amns_nuclear_densities.py.

14.10.3.4 D_D_p_T

```
def amns_nuclear_densities.D_D_p_T = nuclear_HB_tt( (1,0,2), (1,0,2), (1,0,1), (1,0,3), b'NUC_TT')
```

Definition at line 92 of file amns_nuclear_densities.py.

14.10.3.5 D_He_p_He

```
def amns_nuclear_densities.D_He_p_He = nuclear_HB_tt( (1,0,2), (2,0,3), (1,0,1), (2,0,4), b'NUC_TT')
```

Definition at line 95 of file amns_nuclear_densities.py.

14.10.3.6 D_T_n_He

```
def amns_nuclear_densities.D_T_n_He = nuclear_HB_tt( (1,0,2), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
```

Definition at line 94 of file amns_nuclear_densities.py.

14.10.3.7 E

```
amns_nuclear_densities.E = np.array((10**(np.arange(101)/((101-1)/5.0))*10),order='F')
```
Definition at line 101 of file amns_nuclear_densities.py.

14.10.3.8 Ie

```
amns_nuclear_densities.Ie
```
Definition at line 132 of file amns_nuclear_densities.py.

14.10.3.9 loc

```
amns_nuclear_densities.loc
```
Definition at line 126 of file amns_nuclear_densities.py.

14.10.3.10 M

```
amns_nuclear_densities.M = np.zeros([6,6,6,len(E)])
```
Definition at line 107 of file amns_nuclear_densities.py.

14.10.3.11 N

```
amns_nuclear_densities.N = np.outer(np.array([0.0, 5e19, 5e19, 0.0, 0.0, 0.0]), np.ones(len(E)))
```
Definition at line 119 of file amns_nuclear_densities.py.

14.10.3.12 N_Times

```
amns_nuclear_densities.N_Times
```
Definition at line 121 of file amns_nuclear_densities.py.

14.10.3.13 Species

```
amns_nuclear_densities.Species = np.array([[1,0,1], [1,0,2], [1,0,3], [2,0,3], [2,0,4], [0,0,1]])
```
Definition at line 17 of file amns_nuclear_densities.py.

14.10.3.14 T_T_n_He

```
def amns_nuclear_densities.T_T_n_He = nuclear_HB_tt( (1,0,3), (1,0,3), (0,0,1), (2,0,4), b'N↩
UC_TT')
```
Definition at line 96 of file amns_nuclear_densities.py.

14.10.3.15 Times

```
amns_nuclear_densities.Times
```
Definition at line 121 of file amns_nuclear_densities.py.

14.11 amns_provider_types Module Reference

Data types used for the ITM AMNS provider routines.

Data Types

- type amns_ids_list

  type for linked list of amns cpos

14.11.1 Detailed Description

Data types used for the ITM AMNS provider routines.

Author

David Coster

14.12 amns_scan Namespace Reference

Functions

- def summarize_data (shot, run, USER, IDS, DATAVERSION, file=None)
- def summarize (USER, IDS, DATAVERSION, file=None)

Variables

- parser
- type
- str
- help
- default
- args = parser.parse_args()
- def f = open('amns_scan_%s_%s.tex' % (os.getenv("IMAS_VERSION"),args.user), 'w') ; summarize(args.↩ user, 'amns', os.getenv("IMAS_VERSION"), f) ; f.close()

14.12.1 Detailed Description

```
This python program scans through the amns IDS's and then outputs a
LaTeX file containing information about the available reactions.

pdflatex can then be used to produce a PDF document..
```

14.12.2 Function Documentation

14.12.2.1 summarize()

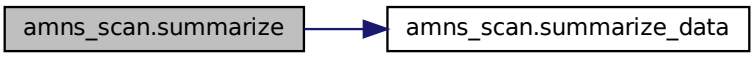
```
def amns_scan.summarize (
               USER,
               IDS,
               DATAVERSION,
               file = None )

Produce LaTeX describing the AMNS data stored by a particular user
```

Definition at line 129 of file amns_scan.py.

```
00129 def summarize (USER, IDS, DATAVERSION, file=None):
00130     """
00131     Produce LaTeX describing the AMNS data stored by a particular user
00132     """
00133     periodic = re.compile("[A-Z][a-z]*").findall
      ("""HHeLiBeBCNOFNeNaMgAlSiPSClArKCaScTiVCrMnFeCoNiCuZnGaGeAsSeBrKrRbSrYZrNbMoTcRuRhPdAgCdInSnSbTeIXeCsBaLaCePrNdPmSmEu
00134
00135     amns_index = imas.ids(0,1)
00136     amns_index.open_env(USER, IDS, DATAVERSION)
00137     amns_index.amns_data.get()
00138     AMNS = amns_index.amns_data
```

```
00139
00140     nversions = len(AMNS.release)
00141
00142     print("'\\documentclass[10pt]{article}
00143
00144 \\usepackage[a4paper,margin=1in,landscape]{geometry}
00145
00146 \\setlength{\\pdfpagewidth}{\\paperwidth}
00147 \\setlength{\\pdfpageheight}{\\paperheight}
00148
00149 \\usepackage{longtable,booktabs,paralist,parskip,spverbatim,hyperref}
00150
00151 \\setdefaultleftmargin{0.5em}{0.5em}{0.5em}{0.5em}{0.5em}{0.5em}
00152
00153 \\begin{document}
00154
00155 \\tableofcontents"', file=file)
00156     print('', file=file)
00157     print('\\newpage\\section{AMNS reactions %s (user %s)}' % (DATAVERSION, USER), file=file)
00158     print('', file=file)
00159     print('Based on data from USER "%s", using the IDS "%s" and DATAVERSION "%s".' % (USER, IDS,
      DATAVERSION), file=file)
00160     print('', file=file)
00161     print('Prepared at %s' % (time.strftime("%Y-%m-%d %H:%M:%S UTC",time.gmtime(time.time())),),
      file=file)
00162     print('', file=file)
00163
00164     for i in range(nversions) :
00165         default=''
00166         if i+1 == nversions:
00167             default = '[DEFAULT]'
00168         release=AMNS.release[i]
00169         print('\\newpage\\subsection{Release %s (%s) %s}' % (i+1, release.date, default), file=file)
00170         print('', file=file)
00171         print('Description: \\spverb|%s|' % (release.description,), file=file)
00172         print('', file=file)
00173         print('Date: \\spverb|%s|' % (release.date,), file=file)
00174         print('', file=file)
00175
00176         nrelease =  len(release.data_entry)
00177         for j in range(nrelease) :
00178             data=release.data_entry[j]
00179             if data.shot >= 1000 :
00180                 print('\\newpage\\subsubsection{Data for %s-%s (Release %s on %s)}' %
      (int(data.shot/1000), periodic[data.shot%1000-1], i+1, release.date), file=file)
00181             else :
00182                 print('\\newpage\\subsubsection{Data for %s (Release %s on %s)}' %
      (periodic[data.shot-1], i+1, release.date), file=file)
00183             print('', file=file)
00184             print('The data is stored in SHOT=%s RUN=%s' % (data.shot, data.run), file=file)
00185             print('', file=file)
00186             print('Description: \\spverb|%s|' % (data.description,), file=file)
00187             print('', file=file)
00188             summarize_data(data.shot, data.run, USER, IDS, DATAVERSION, file)
00189             print('', file=file)
00190
00191     print('\end{document}', file=file)
00192
00193     return
00194
```

Here is the call graph for this function:



14.12.2.2 summarize_data()

```
def amns_scan.summarize_data (
              shot,
              run,
```

USER,
IDS,
DATAVERSION,
file = None)

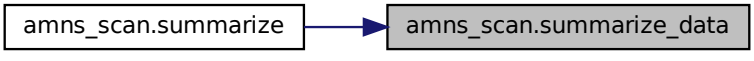
Produce LaTeX summarizing the data contained withi a particular shot/run

Definition at line 15 of file amns_scan.py.

```
00015 def summarize_data(shot, run, USER, IDS, DATAVERSION, file=None):
00016     """
00017     Produce LaTeX summarizing the data contained withi a particular shot/run
00018     """
00019     amns = imas.ids(shot,run)
00020     amns.open_env(USER, IDS, DATAVERSION)
00021     amns.amns_data.get()
00022     AMNS = amns.amns_data
00023 
00024     print("ids\_properties/comment: \\spverb|%s|" % (AMNS.ids_properties.comment,), file=file)
00025     print(", file=file)
00026     print("ids\_properties/source: \\spverb|%s|" % (AMNS.ids_properties.source,), file=file)
00027     print(", file=file)
00028     print("ids\_properties/provider: \\spverb|%s|" % (AMNS.ids_properties.provider,), file=file)
00029     print(", file=file)
00030     print("ids\_properties/creation\_date: \\spverb|%s|" % (AMNS.ids_properties.creation_date,),
      file=file)
00031     print(", file=file)
00032     print("code/name: \\spverb|%s|" % (AMNS.code.name,), file=file)
00033     print(", file=file)
00034     print("code/commit: \\spverb|%s|" % (AMNS.code.commit,), file=file)
00035     print(", file=file)
00036     print("code/version: \\spverb|%s|" % (AMNS.code.version,), file=file)
00037     print(", file=file)
00038     print("code/repository: \\spverb|%s|" % (AMNS.code.repository,), file=file)
00039     print(", file=file)
00040     print(", file=file)
00041     print('Charge and mass: ZN=%s, AMN=%s' % (AMNS.z_n, AMNS.a), file=file)
00042     print(", file=file)
00043 
00044     columns=['0.07', '0.10', '0.07', '0.06', '0.20', '0.20', '0.15']
00045     print('\\begin{longtable}[]{@{}|l|l|l|l|l|l|l|l|l|l|@{}}', file=file)
00046     print('  \\toprule', file=file)
00047     print('  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item IND \\item
      PROC \\end{compactitem} \\end{minipage} &' % (columns[0],), file=file)
00048     print('  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright LABEL\\strut \\end{minipage} &' %
      (columns[1],), file=file)
00049     print('  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item NDIM
      \\item UNITS \\item COORD \\end{compactitem} \\end{minipage} &' % (columns[2],), file=file)
00050     print('  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright INTERP FUN\\strut \\end{minipage} &' %
      (columns[3],), file=file)
00051     print('  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item SOURCE
      \\item PROVIDER \\item CITATION \\end{compactitem} \\end{minipage} &' % (columns[4],), file=file)
00052     print('  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright COORD LABELS\\strut \\end{minipage} &'
      % (columns[5],), file=file)
00053     print('  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright REACTION\\strut
      \\end{minipage}\\tabularnewline' % (columns[6],), file=file)
00054     print('  \\midrule', file=file)
00055     print('  \\endhead', file=file)
00056 
00057     lines=[]
00058     nproc = len(AMNS.process)
00059     for k in range(nproc) :
00060         process=AMNS.process[k]
00061         if k>0:
00062             print('  \midrule', file=file)
00063         line = []
00064         line.append(k+1)
00065         line.append(process.label)
00066         print('  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item %s
      \\item \\spverb|%s| \\end{compactitem} \\end{minipage} &' % (columns[0], k+1, process.label),
      file=file)
00067         line.append(process.result_label)
00068         print('  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright \\spverb|%s|\\strut
      \\end{minipage} &' % (columns[1], process.result_label), file=file)
00069         line.append(process.table_dimension)
00070         line.append(process.result_units)
00071         line.append(process.coordinate_index)
00072         print('  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item %s
      \\item $%s$ \\item %s \\end{compactitem} \\end{minipage} &' % (columns[2], process.table_dimension,
      process.result_units, process.coordinate_index), file=file)
00073         line.append(process.result_transformation)
00074         print('  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright %s\\strut \\end{minipage} &' %
      (columns[3], process.result_transformation), file=file)
00075         line.append('%s\n%s\n%s' % (process.source.replace('/','/ '), process.provider,
      process.citation))
```

```
00076         print('  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item
      \\spverb|%s| \\item \\spverb|%s| \\item \\spverb|%s| \\end{compactitem} \\end{minipage} &' %
      (columns[4], process.source.replace('/','/ '), process.provider, process.citation), file=file)
00077         print('  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright ' % (columns[5],) , end='',
      file=file)
00078         entry=""
00079         l=process.coordinate_index
00080         if l > 0 :
00081             for m in range(len(AMNS.coordinate_system[l-1].coordinate)) :
00082                 if m>0:
00083                     print('\\\\', end='', file=file)
00084                 print('%s: %s ($%s$)' % (m+1, AMNS.coordinate_system[l-1].coordinate[m].label,
      AMNS.coordinate_system[l-1].coordinate[m].units), end='', file=file)
00085                 entry=entry+' %s: %s ($%s$)\n' % (m+1,
      AMNS.coordinate_system[l-1].coordinate[m].label, AMNS.coordinate_system[l-1].coordinate[m].units)
00086         line.append(entry)
00087         print('\\strut \\end{minipage} &', file=file)
00088         print('  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright ' % (columns[6],), end='',
      file=file)
00089         if len(process.reactants)+len(process.products) > 0:
00090             print('$', end=' ', file=file)
00091             for m in range(len(process.reactants)) :
00092                 if m != 0:
00093                     print(' + ', end=' ', file=file)
00094                 if process.reactants[m].element[0].multiplicity > 1:
00095                     print('%i ' % (process.reactants[m].element[0].multiplicity), end=' ', file=file)
00096                 print('\\textrm{%s}^{' % (process.reactants[m].label.replace('|','{\\textbar}')),
      end=' ', file=file)
00097                 if process.reactants[m].relative_charge == 1:
00098                     print('z', end=' ', file=file)
00099                 if process.reactants[m].relative_charge == -1:
00100                     print('}', end=' ', file=file)
00101                 else:
00102                     print('%+i}' % (int(process.reactants[m].charge)), end=' ', file=file)
00103             print(' $ \\\\ \\center $ \\rightarrow $ \\\\ \\raggedleft $', end=' ', file=file)
00104             for m in range(len(process.products)) :
00105                 if m != 0:
00106                     print(' + ', end=' ', file=file)
00107                 if process.products[m].element[0].multiplicity > 1:
00108                     print('%i ' % (process.products[m].element[0].multiplicity), end=' ', file=file)
00109                 print('\\textrm{%s}^{' % (process.products[m].label.replace('|','{\\textbar}')), end='
      ', file=file)
00110                 if process.products[m].relative_charge == 1:
00111                     print('z', end=' ', file=file)
00112                 if process.products[m].relative_charge == -1:
00113                     print('}', end=' ', file=file)
00114                 else:
00115                     print('%+i}' % (int(process.products[m].charge)), end=' ', file=file)
00116             print('$', end='', file=file)
00117         print('\\strut \\end{minipage}\\tabularnewline', file=file)
00118         line.append('')
00119         lines.append(line)
00120
00121     print('  \\bottomrule', file=file)
00122     print('\\end{longtable}', file=file)
00123     output = open('amns_scan.pckl', 'wb')
00124     pickle.dump(lines,output)
00125
00126     return
00127
00128
```

Here is the caller graph for this function:



14.12.3 Variable Documentation

14.12.3.1 args

```
amns_scan.args = parser.parse_args()
```
Definition at line 199 of file amns_scan.py.

14.12.3.2 default

```
amns_scan.default
```
Definition at line 198 of file amns_scan.py.

14.12.3.3 f

```
def amns_scan.f = open('amns_scan_%s_%s.tex' % (os.getenv("IMAS_VERSION"),args.user), 'w') ;
summarize(args.user, 'amns', os.getenv("IMAS_VERSION"), f) ; f.close()
```
Definition at line 203 of file amns_scan.py.

14.12.3.4 help

```
amns_scan.help
```
Definition at line 198 of file amns_scan.py.

14.12.3.5 parser

```
amns_scan.parser
```
Initial value:
```
00001 =  argparse.ArgumentParser(description="Catalogue the AMNS data available on the current system",
      epilog
00002 =, formatter_class=argparse.RawTextHelpFormatter)
```
Definition at line 195 of file amns_scan.py.

14.12.3.6 str

```
amns_scan.str
```
Definition at line 198 of file amns_scan.py.

14.12.3.7 type

```
amns_scan.type
```
Definition at line 198 of file amns_scan.py.

14.13 amns_test Namespace Reference

Variables

- list nx = [101, 101]
- te = np.empty(nx, dtype=np.float64, order='F')
- ne = np.empty(nx, dtype=np.float64, order='F')
- amnsdb = amns.Amns()
- r = amns.Reactants()
- lr
- table = amnsdb.get_table(b"CX", r)
- res = table.data(te, ne)
- f = open('amns_test_cx_te_c.out', 'w')

- iy = int(nx[1]/2)
- int height = 210/25.4
- int width = 297/25.4
- figsize
- res_max = m.ceil(m.log10(res.max()))
- res_min = m.floor(m.log10(res.min()))
- float v = 10.0**np.arange(res_min,res_max+1)
- norm
- ticks
- format

14.13.1 Variable Documentation

14.13.1.1 amnsdb

```
amns_test.amnsdb = amns.Amns()
```

Definition at line 19 of file amns_test.py.

14.13.1.2 f

```
amns_test.f = open('amns_test_cx_te_c.out', 'w')
```

Definition at line 36 of file amns_test.py.

14.13.1.3 figsize

```
amns_test.figsize
```

Definition at line 57 of file amns_test.py.

14.13.1.4 format

```
amns_test.format
```

Definition at line 71 of file amns_test.py.

14.13.1.5 height

```
int amns_test.height = 210/25.4
```

Definition at line 55 of file amns_test.py.

14.13.1.6 iy

```
amns_test.iy = int(nx[1]/2)
```

Definition at line 39 of file amns_test.py.

14.13.1.7 lr

```
amns_test.lr
```

Definition at line 24 of file amns_test.py.

14.13.1.8 ne

```
amns_test.ne = np.empty(nx, dtype=np.float64, order='F')
```
Definition at line 11 of file amns_test.py.

14.13.1.9 norm

```
amns_test.norm
```
Definition at line 65 of file amns_test.py.

14.13.1.10 nx

```
list amns_test.nx = [101, 101]
```
Definition at line 9 of file amns_test.py.

14.13.1.11 r

```
amns_test.r = amns.Reactants()
```
Definition at line 21 of file amns_test.py.

14.13.1.12 res

```
amns_test.res = table.data(te, ne)
```
Definition at line 32 of file amns_test.py.

14.13.1.13 res_max

```
amns_test.res_max = m.ceil(m.log10(res.max()))
```
Definition at line 61 of file amns_test.py.

14.13.1.14 res_min

```
amns_test.res_min = m.floor(m.log10(res.min()))
```
Definition at line 62 of file amns_test.py.

14.13.1.15 table

```
amns_test.table = amnsdb.get_table(b"CX", r)
```
Definition at line 28 of file amns_test.py.

14.13.1.16 te

```
amns_test.te = np.empty(nx, dtype=np.float64, order='F')
```
Definition at line 10 of file amns_test.py.

14.13.1.17 ticks

```
amns_test.ticks
```
Definition at line 71 of file amns_test.py.

14.13.1.18 v

```
amns_test.v = 10.0**np.arange(res_min,res_max+1)
```
Definition at line 64 of file amns_test.py.

14.13.1.19 width

```
int amns_test.width = 297/25.4
```
Definition at line 56 of file amns_test.py.

14.14 amns_test_adf11_versions Namespace Reference

Variables

- parser
- type
- str
- help
- default
- int
- args = parser.parse_args()
- periodic = re.compile("[A-Z][a-z]*").findall ()
- te = np.logspace(-1,5,121)
- int ne = te*0+5e19
- amnsdb_df = amns.Amns(version_user = args.user)
- list amnsdb_v = [ ]
- r = amns.Reactants()
- lr
- T = A.get_table("LR", r)
- label
- loc
- fontsize
- framealpha

14.14.1 Variable Documentation

14.14.1.1 amnsdb_df

```
amns_test_adf11_versions.amnsdb_df = amns.Amns(version_user = args.user)
```
Definition at line 23 of file amns_test_adf11_versions.py.

14.14.1.2 amnsdb_v

```
list amns_test_adf11_versions.amnsdb_v = [ ]
```
Definition at line 26 of file amns_test_adf11_versions.py.

14.14.1.3 args

```
amns_test_adf11_versions.args = parser.parse_args()
```
Definition at line 16 of file amns_test_adf11_versions.py.

14.14.1.4 default

amns_test_adf11_versions.default
Definition at line 14 of file amns_test_adf11_versions.py.

14.14.1.5 fontsize

amns_test_adf11_versions.fontsize
Definition at line 44 of file amns_test_adf11_versions.py.

14.14.1.6 framealpha

amns_test_adf11_versions.framealpha
Definition at line 44 of file amns_test_adf11_versions.py.

14.14.1.7 help

amns_test_adf11_versions.help
Definition at line 14 of file amns_test_adf11_versions.py.

14.14.1.8 int

amns_test_adf11_versions.int
Definition at line 15 of file amns_test_adf11_versions.py.

14.14.1.9 label

amns_test_adf11_versions.label
Definition at line 39 of file amns_test_adf11_versions.py.

14.14.1.10 loc

amns_test_adf11_versions.loc
Definition at line 44 of file amns_test_adf11_versions.py.

14.14.1.11 lr

amns_test_adf11_versions.lr
Definition at line 35 of file amns_test_adf11_versions.py.

14.14.1.12 ne

int amns_test_adf11_versions.ne = te*0+5e19
Definition at line 21 of file amns_test_adf11_versions.py.

14.14.1.13 parser

amns_test_adf11_versions.parser
Initial value:
```
00001 =  argparse.ArgumentParser(description="Compare different versions of the AMNS adf11 data", epilog
00002 =, formatter_class=argparse.RawTextHelpFormatter)
```
Definition at line 11 of file amns_test_adf11_versions.py.

14.14.1.14 periodic

```
amns_test_adf11_versions.periodic = re.compile("[A-Z][a-z]*").findall ()
```
Definition at line 18 of file amns_test_adf11_versions.py.

14.14.1.15 r

```
amns_test_adf11_versions.r = amns.Reactants()
```
Definition at line 33 of file amns_test_adf11_versions.py.

14.14.1.16 str

```
amns_test_adf11_versions.str
```
Definition at line 14 of file amns_test_adf11_versions.py.

14.14.1.17 T

```
amns_test_adf11_versions.T = A.get_table("LR", r)
```
Definition at line 38 of file amns_test_adf11_versions.py.

14.14.1.18 te

```
amns_test_adf11_versions.te = np.logspace(-1,5,121)
```
Definition at line 20 of file amns_test_adf11_versions.py.

14.14.1.19 type

```
amns_test_adf11_versions.type
```
Definition at line 14 of file amns_test_adf11_versions.py.

14.15 amns_test_bms Namespace Reference

Functions

- def bms_calc (n)
- def plot (eng, dens, tion, te, res, coordinates, results, reactants)

14.15.1 Function Documentation

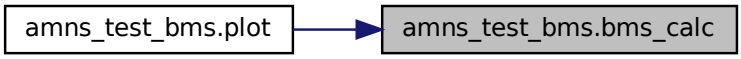
14.15.1.1 bms_calc()

```
def amns_test_bms.bms_calc (
              n )
```
Definition at line 11 of file amns_test_bms.py.

```
00011 def bms_calc(n):
00012     assert n>1, "n should be an integer greater than 1"
00013     nx = [n, n, n, n]
00014
00015     ENG  = np.logspace(np.log10(2.000E+04), np.log10(1.500E+05), nx[0])
00016     DENS = np.logspace(np.log10(1.000E+17), np.log10(1.000E+21), nx[1])
00017     TION = np.logspace(np.log10(1.000E+02), np.log10(2.000E+04), nx[2])
00018     TE   = np.logspace(np.log10(1.000E+02), np.log10(1.000E+04), nx[3])
00019
```

```
00020     start = time.time()
00021 #     eng, dens, tion, te = np.array([[w,x,y,z] for z in TE for y in TION for x in DENS for w in
      ENG]).T
00022     te, tion, dens, eng = np.array(list(itertools.product(TE,TION,DENS,ENG))).T
00023     end = time.time()
00024     print('\nSetting up the coordinates (%s elements) took %0.3f seconds' % (np.product(nx),
      end-start))
00025
00026     eng=eng.reshape(nx[::-1])
00027     dens=dens.reshape(nx[::-1])
00028     tion=tion.reshape(nx[::-1])
00029     te=te.reshape(nx[::-1])
00030
00031     amnsdb = amns.Amns()
00032
00033     r = amns.Reactants()
00034     r.add(1,0,1)
00035     r.add(1,1,1)
00036     r.add(1,1,1,lr=1)
00037     r.add(1,0,1,lr=1)
00038     print("Reactants:", r)
00039
00040     start = time.time()
00041     table = amnsdb.get_table(b"BMS", r)
00042     end = time.time()
00043     print('Setting up the bms table took %0.3f seconds' % (end-start))
00044
00045     print("table.no_of_reactants")
00046     print("table.no_of_reactants", table.no_of_reactants)
00047
00048     start = time.time()
00049     res = table.data(eng.ravel(), dens.ravel(), tion.ravel(), te.ravel()).reshape(nx[::-1])
00050     end = time.time()
00051     print('Calculating the values for %s elements took %0.3f seconds' % (res.size, end-start))
00052
00053     coordinates=table.coordinates.split(" ")
00054     results=table.result_label+"/"+table.result_unit
00055     reactants=table.reactants
00056
00057     start = time.time()
00058     table.finalize()
00059     amnsdb.finalize()
00060     end = time.time()
00061     print('Finishing took %0.3f seconds\n' % (end-start))
00062
00063     return eng, dens, tion, te, res, coordinates, results, reactants
00064
```

Here is the caller graph for this function:



14.15.1.2 plot()

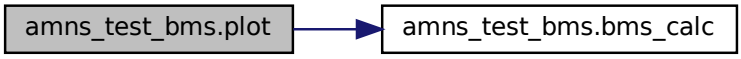
```
def amns_test_bms.plot (
                eng,
                dens,
                tion,
                te,
                res,
                coordinates,
                results,
                reactants )
```

Definition at line 65 of file amns_test_bms.py.

```
00065 def plot(eng, dens, tion, te, res, coordinates, results, reactants):
00066
```

```
00067     x1=np.array([20000.0000, 23100.0000, 26670.0000, 30800.0000, 35570.0000, 41070.0000, 47430.0000,
      54770.0000, 63250.0000, 73040.0000, 84350.0000, 97400.0000, 112500.000, 129900.000, 150000.000])
00068     x2=np.array([9.99999984E+16, 2.78299999E+17, 7.74300003E+17, 2.15400004E+18, 5.99499974E+18,
      1.66800004E+19, 4.64199999E+19, 1.29199996E+20, 3.59400022E+20, 1.00000002E+21])
00069     x3=np.array([100.000000, 180.199997, 324.600006, 584.799988, 1054.00000, 1898.00000, 3420.00000,
      6162.00000, 11100.0000, 20000.0000])
00070     x4=np.array([100.000000, 166.800003, 278.299988, 464.200012, 774.299988, 1292.00000, 2154.00000,
      3594.00000, 5995.00000, 10000.0000])
00071
00072     D1=np.array([1.34100006E-07, 1.30900006E-07, 1.28400004E-07, 1.25499994E-07, 1.20600006E-07,
      1.15600002E-07, 1.09900000E-07, 1.03799998E-07, 9.80599992E-08, 9.36700033E-08, 9.02900013E-08,
      8.77199966E-08, 8.49500026E-08, 8.25599997E-08, 8.06399996E-08])
00073     D2=np.array([1.34100006E-07, 1.34999993E-07, 1.36400004E-07, 1.38399997E-07, 1.41200005E-07,
      1.45300007E-07, 1.51099997E-07, 1.59999999E-07, 1.71099998E-07, 1.80300006E-07])
00074     D3=np.array([1.34100006E-07, 1.33900002E-07, 1.33599997E-07, 1.33200004E-07, 1.32500006E-07,
      1.31299998E-07, 1.29000000E-07, 1.25400007E-07, 1.20199999E-07, 1.13500001E-07])
00075     D4=np.array([1.34100006E-07, 1.34299995E-07, 1.32599993E-07, 1.29699998E-07, 1.26200007E-07,
      1.22599999E-07, 1.19100001E-07, 1.16099997E-07, 1.13399999E-07, 1.11100000E-07])
00076
00077     e1, v1 = np.loadtxt('../fortran/eng_cut.dat').T
00078     e2, v2 = np.loadtxt('../fortran/dens_cut.dat').T
00079     e3, v3 = np.loadtxt('../fortran/tion_cut.dat').T
00080     e4, v4 = np.loadtxt('../fortran/te_cut.dat').T
00081
00082     plt.figure(figsize=(12,8))
00083     plt.subplot(2,2,1)
00084     plt.semilogx(eng [0, 0, 0, :], res[0, 0, 0, :], label='python full')
00085     #plt.semilogx(eng [0, 0, 0, :], table.data(eng[0, 0, 0, :].copy(), dens[0, 0, 0, :].copy(),
      tion[0, 0, 0, :].copy(), te[0, 0, 0, :].copy()), label='slice')
00086     plt.semilogx(x1, D1, 'o', label='ADAS values');
00087     plt.semilogx(e1, v1, 'o', label='fortran full')
00088     plt.xlabel('$%s$' % coordinates[0])
00089     plt.ylabel('$%s$' % results)
00090     plt.legend(loc=0)
00091
00092     plt.subplot(2,2,2)
00093     plt.semilogx(dens[0, 0, :, 0], res[0, 0, :, 0], label='python full')
00094     #plt.semilogx(dens[0, 0, :, 0], table.data(eng[0, 0, :, 0].copy(), dens[0, 0, :, 0].copy(),
      tion[0, 0, :, 0].copy(), te[0, 0, :, 0].copy()), label='slice')
00095     plt.semilogx(x2, D2, 'o', label='ADAS values')
00096     plt.semilogx(e2, v2, 'o', label='fortran full')
00097     plt.xlabel('$%s$' % coordinates[1])
00098     plt.ylabel('$%s$' % results)
00099     plt.legend(loc=0)
00100
00101     plt.subplot(2,2,3)
00102     plt.semilogx(tion[0, :, 0, 0], res[0, :, 0, 0], label='python full')
00103     #plt.semilogx(tion[0, :, 0, 0], table.data(eng[0, :, 0, 0].copy(), dens[0, :, 0, 0].copy(),
      tion[0, :, 0, 0].copy(), te[0, :, 0, 0].copy()), label='slice')
00104     plt.semilogx(x3, D3, 'o', label='ADAS values')
00105     plt.semilogx(e3, v3, 'o', label='fortran full')
00106     plt.xlabel('$%s$' % coordinates[2])
00107     plt.ylabel('$%s$' % results)
00108     plt.legend(loc=0)
00109
00110     plt.subplot(2,2,4)
00111     plt.semilogx(te  [:, 0, 0, 0], res[:, 0, 0, 0], label='python full')
00112     #plt.semilogx(te  [:, 0, 0, 0], table.data(eng[:, 0, 0, 0].copy(), dens[:, 0, 0, 0].copy(),
      tion[:, 0, 0, 0].copy(), te[:, 0, 0, 0].copy()), label='slice')
00113     plt.semilogx(x4, D4, 'o', label='ADAS values')
00114     plt.semilogx(e4, v4, 'o', label='fortran full')
00115     plt.xlabel('$%s$' % coordinates[3])
00116     plt.ylabel('$%s$' % results)
00117     plt.legend(loc=0)
00118
00119     plt.suptitle(reactants)
00120
00121     plt.subplots_adjust(left=0.07, right=0.98, bottom=0.07, top=0.90, wspace=0.20, hspace=0.25)
00122
00123
00124 plt.ion()
00125 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(2)
00126 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00127 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(5)
00128 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00129 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(11)
00130 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00131 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(21)
00132 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00133 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(51)
00134 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00135 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(101)
00136 plot(eng, dens, tion, te, res, coordinates, results, reactants)
```

Here is the call graph for this function:



14.16 amns_test_bms_interpolation_options Namespace Reference

Variables

- list nx = [1001, 2, 2, 2]
- ENG = np.logspace(np.log10(2.000E+04), np.log10(1.500E+05), nx[0])
- DENS = np.logspace(np.log10(1.000E+17), np.log10(1.000E+21), nx[1])
- TION = np.logspace(np.log10(1.000E+02), np.log10(2.000E+04), nx[2])
- TE = np.logspace(np.log10(1.000E+02), np.log10(1.000E+04), nx[3])
- start = time.time()
- te = te.reshape(nx[::-1])
- tion = tion.reshape(nx[::-1])
- dens = dens.reshape(nx[::-1])
- eng = eng.reshape(nx[::-1])
- end = time.time()
- amnsdb = amns.Amns()
- r = amns.Reactants()
- lr
- table = amnsdb.get_table(b"BMS", r)
- res = table.data(eng.ravel(), dens.ravel(), tion.ravel(), te.ravel()).reshape(nx[::-1])
- coordinates = table.coordinates.split(" ")
- string results = table.result_label+"/"+table.result_unit
- reactants = table.reactants
- x1 = np.array([20000.0000, 23100.0000, 26670.0000, 30800.0000, 35570.0000, 41070.0000, 47430.0000, 54770.0000, 63250.0000, 73040.0000, 84350.0000, 97400.0000, 112500.000, 129900.000, 150000.000])
- x2 = np.array([9.99999984E+16, 2.78299999E+17, 7.74300003E+17, 2.15400004E+18, 5.99499974E+18, 1.66800004E+19, 4.64199999E+19, 1.29199996E+20, 3.59400022E+20, 1.00000002E+21])
- x3 = np.array([100.000000, 180.199997, 324.600006, 584.799988, 1054.00000, 1898.00000, 3420.00000, 6162.00000, 11100.0000, 20000.0000])
- x4 = np.array([100.000000, 166.800003, 278.299988, 464.200012, 774.299988, 1292.00000, 2154.00000, 3594.00000, 5995.00000, 10000.0000])
- D1 = np.array([1.34100006E-07, 1.30900006E-07, 1.28400004E-07, 1.25499994E-07, 1.20600006E-07, 1.15600002E-07, 1.09900000E-07, 1.03799998E-07, 9.80599992E-08, 9.36700033E-08, 9.02900013E-08, 8.77199966E-08, 8.49500026E-08, 8.25599997E-08, 8.06399996E-08])
- D2 = np.array([1.34100006E-07, 1.34999993E-07, 1.36400004E-07, 1.38399997E-07, 1.41200005E-07, 1.45300007E-07, 1.51099997E-07, 1.59999999E-07, 1.71099998E-07, 1.80300006E-07])
- D3 = np.array([1.34100006E-07, 1.33900002E-07, 1.33599997E-07, 1.33200004E-07, 1.32500006E-07, 1.31299998E-07, 1.29000000E-07, 1.25400007E-07, 1.20199999E-07, 1.13500001E-07])
- D4 = np.array([1.34100006E-07, 1.34299995E-07, 1.32599993E-07, 1.29699998E-07, 1.26200007E-07, 1.22599999E-07, 1.19100001E-07, 1.16099997E-07, 1.13399999E-07, 1.11100000E-07])
- e1
- v1
- e2
- v2

- e3
- v3
- e4
- v4
- label
- loc

14.16.1 Variable Documentation

14.16.1.1 amnsdb

```
amns_test_bms_interpolation_options.amnsdb = amns.Amns()
```
Definition at line 29 of file amns_test_bms_interpolation_options.py.

14.16.1.2 coordinates

```
amns_test_bms_interpolation_options.coordinates = table.coordinates.split(" ")
```
Definition at line 53 of file amns_test_bms_interpolation_options.py.

14.16.1.3 D1

```
amns_test_bms_interpolation_options.D1 = np.array([1.34100006E-07, 1.30900006E-07, 1.28400004E-07,
1.25499994E-07, 1.20600006E-07, 1.15600002E-07, 1.09900000E-07, 1.03799998E-07, 9.80599992E-08,
9.36700033E-08, 9.02900013E-08, 8.77199966E-08, 8.49500026E-08, 8.25599997E-08, 8.06399996E-08])
```
Definition at line 64 of file amns_test_bms_interpolation_options.py.

14.16.1.4 D2

```
amns_test_bms_interpolation_options.D2 = np.array([1.34100006E-07, 1.34999993E-07, 1.36400004E-07,
1.38399997E-07, 1.41200005E-07, 1.45300007E-07, 1.51099997E-07, 1.59999999E-07, 1.71099998E-07,
1.80300006E-07])
```
Definition at line 65 of file amns_test_bms_interpolation_options.py.

14.16.1.5 D3

```
amns_test_bms_interpolation_options.D3 = np.array([1.34100006E-07, 1.33900002E-07, 1.33599997E-07,
1.33200004E-07, 1.32500006E-07, 1.31299998E-07, 1.29000000E-07, 1.25400007E-07, 1.20199999E-07,
1.13500001E-07])
```
Definition at line 66 of file amns_test_bms_interpolation_options.py.

14.16.1.6 D4

```
amns_test_bms_interpolation_options.D4 = np.array([1.34100006E-07, 1.34299995E-07, 1.32599993E-07,
1.29699998E-07, 1.26200007E-07, 1.22599999E-07, 1.19100001E-07, 1.16099997E-07, 1.13399999E-07,
1.11100000E-07])
```
Definition at line 67 of file amns_test_bms_interpolation_options.py.

14.16.1.7 DENS

```
amns_test_bms_interpolation_options.DENS = np.logspace(np.log10(1.000E+17), np.log10(1.↩
000E+21), nx[1])
```
Definition at line 14 of file amns_test_bms_interpolation_options.py.

14.16.1.8 dens

```
amns_test_bms_interpolation_options.dens = dens.reshape(nx[::-1])
```
Definition at line 20 of file amns_test_bms_interpolation_options.py.

14.16.1.9 e1

```
amns_test_bms_interpolation_options.e1
```
Definition at line 69 of file amns_test_bms_interpolation_options.py.

14.16.1.10 e2

```
amns_test_bms_interpolation_options.e2
```
Definition at line 70 of file amns_test_bms_interpolation_options.py.

14.16.1.11 e3

```
amns_test_bms_interpolation_options.e3
```
Definition at line 71 of file amns_test_bms_interpolation_options.py.

14.16.1.12 e4

```
amns_test_bms_interpolation_options.e4
```
Definition at line 72 of file amns_test_bms_interpolation_options.py.

14.16.1.13 end

```
amns_test_bms_interpolation_options.end = time.time()
```
Definition at line 21 of file amns_test_bms_interpolation_options.py.

14.16.1.14 ENG

```
amns_test_bms_interpolation_options.ENG = np.logspace(np.log10(2.000E+04), np.log10(1.500E+05),
nx[0])
```
Definition at line 13 of file amns_test_bms_interpolation_options.py.

14.16.1.15 eng

```
amns_test_bms_interpolation_options.eng = eng.reshape(nx[::-1])
```
Definition at line 20 of file amns_test_bms_interpolation_options.py.

14.16.1.16 label

```
amns_test_bms_interpolation_options.label
```
Definition at line 75 of file amns_test_bms_interpolation_options.py.

14.16.1.17 loc

```
amns_test_bms_interpolation_options.loc
```
Definition at line 81 of file amns_test_bms_interpolation_options.py.

14.16.1.18 lr

```
amns_test_bms_interpolation_options.lr
```
Definition at line 36 of file amns_test_bms_interpolation_options.py.

14.16.1.19 nx

```
list amns_test_bms_interpolation_options.nx = [1001, 2, 2, 2]
```
Definition at line 11 of file amns_test_bms_interpolation_options.py.

14.16.1.20 r

```
amns_test_bms_interpolation_options.r = amns.Reactants()
```
Definition at line 33 of file amns_test_bms_interpolation_options.py.

14.16.1.21 reactants

```
amns_test_bms_interpolation_options.reactants = table.reactants
```
Definition at line 55 of file amns_test_bms_interpolation_options.py.

14.16.1.22 res

```
amns_test_bms_interpolation_options.res = table.data(eng.ravel(), dens.ravel(), tion.ravel(),
te.ravel()).reshape(nx[::-1])
```
Definition at line 49 of file amns_test_bms_interpolation_options.py.

14.16.1.23 results

```
string amns_test_bms_interpolation_options.results = table.result_label+"/"+table.result_unit
```
Definition at line 54 of file amns_test_bms_interpolation_options.py.

14.16.1.24 start

```
amns_test_bms_interpolation_options.start = time.time()
```
Definition at line 18 of file amns_test_bms_interpolation_options.py.

14.16.1.25 table

```
amns_test_bms_interpolation_options.table = amnsdb.get_table(b"BMS", r)
```
Definition at line 41 of file amns_test_bms_interpolation_options.py.

14.16.1.26 TE

```
amns_test_bms_interpolation_options.TE = np.logspace(np.log10(1.000E+02), np.log10(1.000E+04),
nx[3])
```
Definition at line 16 of file amns_test_bms_interpolation_options.py.

14.16.1.27 te

```
amns_test_bms_interpolation_options.te = te.reshape(nx[::-1])
```
Definition at line 20 of file amns_test_bms_interpolation_options.py.

14.16.1.28 TION

```
amns_test_bms_interpolation_options.TION = np.logspace(np.log10(1.000E+02), np.log10(2.↩
000E+04), nx[2])
```
Definition at line 15 of file amns_test_bms_interpolation_options.py.

14.16.1.29 tion

```
amns_test_bms_interpolation_options.tion = tion.reshape(nx[::-1])
```
Definition at line 20 of file amns_test_bms_interpolation_options.py.

14.16.1.30 v1

```
amns_test_bms_interpolation_options.v1
```
Definition at line 69 of file amns_test_bms_interpolation_options.py.

14.16.1.31 v2

```
amns_test_bms_interpolation_options.v2
```
Definition at line 70 of file amns_test_bms_interpolation_options.py.

14.16.1.32 v3

```
amns_test_bms_interpolation_options.v3
```
Definition at line 71 of file amns_test_bms_interpolation_options.py.

14.16.1.33 v4

```
amns_test_bms_interpolation_options.v4
```
Definition at line 72 of file amns_test_bms_interpolation_options.py.

14.16.1.34 x1

```
amns_test_bms_interpolation_options.x1 = np.array([20000.0000, 23100.0000, 26670.0000, 30800.↩
0000, 35570.0000, 41070.0000, 47430.0000, 54770.0000, 63250.0000, 73040.0000, 84350.0000,
97400.0000, 112500.000, 129900.000, 150000.000])
```
Definition at line 59 of file amns_test_bms_interpolation_options.py.

14.16.1.35 x2

```
amns_test_bms_interpolation_options.x2 = np.array([9.99999984E+16, 2.78299999E+17, 7.74300003E+17,
2.15400004E+18, 5.99499974E+18, 1.66800004E+19, 4.64199999E+19, 1.29199996E+20, 3.59400022E+20,
1.00000002E+21])
```
Definition at line 60 of file amns_test_bms_interpolation_options.py.

14.16.1.36 x3

```
amns_test_bms_interpolation_options.x3 = np.array([100.000000, 180.199997, 324.600006, 584.↩
799988, 1054.00000, 1898.00000, 3420.00000, 6162.00000, 11100.0000, 20000.0000])
```

Definition at line 61 of file amns_test_bms_interpolation_options.py.

14.16.1.37 x4

```
amns_test_bms_interpolation_options.x4 = np.array([100.000000, 166.800003, 278.299988, 464.↩
200012, 774.299988, 1292.00000, 2154.00000, 3594.00000, 5995.00000, 10000.0000])
```

Definition at line 62 of file amns_test_bms_interpolation_options.py.

14.17 amns_types Module Reference

The derived types defined here are meant to be interoperable with C. The ones for this is not the case are explicitly marked. Note that they are still required when using the AMNS interface from C, but only as opaque handle variables.

Data Types

- type amns_answer_type

  Type for answers from queries in the AMNS package (not interoperable)
- type amns_error_type

  Type for error returns from the AMNS interface (not interoperable)
- type amns_fc_answer_type

  Type for answers from queries in the AMNS package (interoperable with c)
- type amns_fc_error_type

  Type for error returns from the AMNS interface (interoperable with c)
- type amns_fc_query_type

  Type for querying parameters in the AMNS package (interoperable with c)
- type amns_fc_reaction_type

  Type used for specifying reactions when using the AMNS interface (interoperable with c)
- type amns_fc_set_type

  Type for setting parameters in the AMNS package (interoperable with c)
- type amns_fc_version_type

  Type for specifying the AMNS version (interoperable with c)
- type amns_handle_rx_type

  Type for the AMNS RX handle (opaque for user codes) NOT interoperable with C.
- type amns_handle_type

  type for the AMNS handle (opaque for user codes) NOT interoperable with C.
- type amns_ids_list

  Type for linked list of amns idss NOT interoperable with C.
- type amns_query_type

  Type for querying parameters in the AMNS package (not interoperable)
- type amns_reactant_type

  Type for indicating a single reactant or product when using the AMNS interface.
- type amns_reactants_type

  Type for indicating the reactants when using the AMNS interface NOT interoperable with C.
- type amns_reaction_type

  Type used for specifying reactions when using the AMNS interface (not interoperable)
- type amns_set_type

  Type for setting parameters in the AMNS package (not interoperable)
- type amns_version_type

  Type for specifying the AMNS version (not interoperable)

Variables

- integer, parameter version_length =32

  used to specify the maximum length of the string version number
- integer, parameter set_length =32

  maximum length of the string passed by the set calls
- integer, parameter reaction_length =16

  maximum length for specifying a reaction type
- integer, parameter query_length =16

  maximum length for a query argument
- integer, parameter answer_length =128

  maximum length of an answer

14.17.1 Detailed Description

The derived types defined here are meant to be interoperable with C. The ones for this is not the case are explicitly marked. Note that they are still required when using the AMNS interface from C, but only as opaque handle variables.

Warning

Changes made here might also have to be made in ../include/amns_interface.h

14.18 amns_utility Module Reference

Module implementing various utility functions for the AMNS interface.

Data Types

- interface string

Functions/Subroutines

- character ∗24 function int_to_string (int)

  convert an integer to a string

14.18.1 Detailed Description

Module implementing various utility functions for the AMNS interface.

Author

David Coster, updated by David Tskhakaya (updates: number of calls have been deleted)

14.19 amns_verify Namespace Reference

Functions

- def plot (x1, y1, x2, y2, xlabel, ylabel, title, file=None, line1='bo-', line2='r+-')
- def adas (zn, za, mi, reac, ref, file=None, texfile=None)
- def nuclear_HB (r1, r2, p1, p2, reac, ref, file=None, texfile=None)
- def nuclear_HB_tt (r1, r2, p1, p2, reac, ref, file=None, texfile=None)
- def nuclear_HB_bt (r1, r2, p1, p2, reac, ref, file=None, texfile=None)
- def total_cross_section_EL (zn, za, mi, reac, ref, file=None)
- def differential_cross_section_EL (zn, za, mi, reac, ref, file=None)
- def rct (zn, za, mi, reac, ref, file=None)
- def sputter (zn_w, za_w, mi_w, zn_p, za_p, mi_p, reac, ref, file=None)
- def reflect (zn_w, za_w, mi_w, zn_p, za_p, mi_p, reac, ref, file=None)

Variables

- amnsdb = amns.Amns()
- texfile = open('amns_verify.tex', 'w')
- file

14.19.1 Function Documentation

14.19.1.1 adas()

```
def amns_verify.adas (
                zn,
                za,
                mi,
                reac,
                ref,
                file = None,
                texfile = None )
```

Definition at line 48 of file amns_verify.py.

```
00048 def adas(zn, za, mi, reac, ref, file=None, texfile=None):
00049
00050     nx=[101,1]
00051
00052     x = np.loadtxt(ref)
00053     nxr=list(nx) ; nxr.reverse()
00054     te=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00055     ne=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00056
00057     r = amns.Reactants()
00058     if reac in ['RC']:
00059         r.add(zn,za,mi)
00060         r.add(0,-1,0)
00061         r.add(zn,za-1,mi,lr=1)
00062         r.add(0,-1,0,lr=1)
00063     elif reac in ['EI']:
00064         r.add(zn,za,mi)
00065         r.add(0,-1,0)
00066         r.add(zn,za+1,mi,lr=1)
00067         r.add(0,-1,0,lr=1)
00068     elif reac in ['CX']:
00069         r.add(zn,za,mi)
00070         r.add(1,0,2)
00071         r.add(zn,za-1,mi,lr=1)
00072         r.add(1,1,2,lr=1)
00073     elif reac in ['BR', 'LR', 'ZE', 'ZE2', 'EIP']:
00074         r.add(zn,za,mi)
00075         r.add(zn,za,mi,lr=1)
00076     else:
00077         raise ValueError('Invalid option %s' % (reac))
00078
00079     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00080     res = table.data(te, ne)
00081
00082     plot(te[:,0], res[:,0], x[:,0], x[:,2], 'Te', '%s [%s]' % (table.result_label, table.result_unit),
      '%s for ne=%s, ZN=%s, ZA=%s, MI=%s (%s)' % (reac, ne[0,0], zn, za, mi, table.state_label), file)
00083
00084     max_rel_err = np.max(np.abs((table.data(x[:,0].copy(), x[:,1].copy()) - x[:,2]) / x[:,2]))
00085     print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
00086
00087     if file:
00088         if texfile:
00089             print('', file=texfile)
00090             print('\\subsection{%s of %s}' %
      (table.result_label.replace('^','\\^').replace('_','\\_'),
      table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
00091             print('', file=texfile)
00092             print('Comparison of AMNS with %s' % (ref.replace('_','\\_'),) , file=texfile)
00093             print('', file=texfile)
00094             print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
      file=texfile)
00095             print('', file=texfile)
00096             print('Maximum relative error for %s = %10.03g' % (ref.replace('_','\\_'), max_rel_err),
      file=texfile)
00097             print('', file=texfile)
00098
00099
```

Here is the call graph for this function:

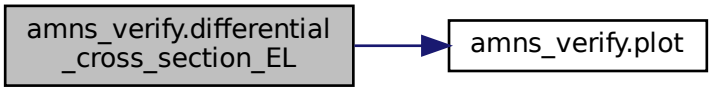
14.19.1.2 differential_cross_section_EL()

```
def amns_verify.differential_cross_section_EL (
                zn,
                za,
                mi,
                reac,
                ref,
                file = None )
```

Definition at line 222 of file amns_verify.py.

```
00222 def differential_cross_section_EL(zn,za,mi,reac,ref,file=None):
00223
00224     nx=[101,1]
00225
00226     x = np.loadtxt(ref)
00227     nxr=list(nx) ; nxr.reverse()
00228     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00229     angle=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00230
00231     r = amns.Reactants()
00232     r.add(zn,za,mi)
00233     r.add(zn,za,mi,lr=1)
00234     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00235     res = table.data(angle,E)
00236
00237     plot(E[:,0], res[:,0], x[:,0], x[:,2], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for angle=%s, ZN=%s, ZA=%s, MI=%s' % (reac, angle[0,0], zn, za, mi), file)
00238
00239
```

Here is the call graph for this function:



14.19.1.3 nuclear_HB()

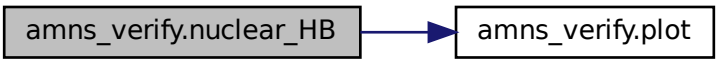
```
def amns_verify.nuclear_HB (
                r1,
                r2,
                p1,
```

```
                p2,
                reac,
                ref,
                file = None,
                texfile = None )
```

Definition at line 100 of file amns_verify.py.

```
00100 def nuclear_HB(r1, r2, p1, p2, reac,ref, file=None, texfile=None):
00101
00102     nx=[101,1]
00103
00104     x = np.loadtxt(ref)
00105     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
00106
00107     r = amns.Reactants()
00108     r.add(r1[0],r1[1],r1[2])
00109     r.add(r2[0],r2[1],r2[2])
00110     r.add(p1[0],p1[1],p1[2],lr=1)
00111     r.add(p2[0],p2[1],p2[2],lr=1)
00112     table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
00113     res = table.data(E)
00114
00115     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s' % (reac,), file)
00116
00117     max_rel_err = np.max(np.abs((table.data(x[:,0].copy()) - x[:,1]) / x[:,1]))
00118
00119     print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
00120
00121     if file:
00122         if texfile:
00123             print('', file=texfile)
00124             print('\\subsection{%s}' % (table.result_label.replace('^','\\^').replace('_','\\_'),),
      file=texfile)
00125             print('', file=texfile)
00126             print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
      file=texfile)
00127             print('', file=texfile)
00128             print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
      file=texfile)
00129             print('', file=texfile)
00130             print('Maximum relative error for %s = %10.03g' %
      (ref.replace('^','\\^').replace('_','\\_'), max_rel_err), file=texfile)
00131             print('', file=texfile)
00132
00133
```

Here is the call graph for this function:



14.19.1.4 nuclear_HB_bt()

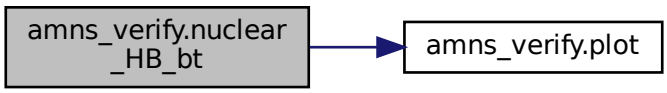
```
def amns_verify.nuclear_HB_bt (
                r1,
                r2,
                p1,
                p2,
                reac,
                ref,
                file = None,
                texfile = None )
```

Definition at line 172 of file amns_verify.py.

```
00172 def nuclear_HB_bt(r1, r2, p1, p2, reac, ref, file=None, texfile=None):
```

```
00173
00174     nx=[101,1]
00175
00176     x = np.loadtxt(ref)
00177     nxr=list(nx) ; nxr.reverse()
00178     Ti=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00179     E=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00180
00181     r = amns.Reactants()
00182     r.add(r1[0],r1[1],r1[2])
00183     r.add(r2[0],r2[1],r2[2])
00184     r.add(p1[0],p1[1],p1[2],lr=1)
00185     r.add(p2[0],p2[1],p2[2],lr=1)
00186     table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
00187     res = table.data(Ti, E)
00188
00189     plot(Ti[:,0], res[:,0], x[:,0], x[:,2], 'Ti', '%s [%s]' % (table.result_label, table.result_unit),
      '%s for E=%s' % (reac, E[0,0]), file)
00190
00191     max_rel_err = np.max(np.abs((table.data(x[:,0].copy(), x[:,1].copy()) - x[:,2]) / x[:,2]))
00192
00193     if file:
00194         if texfile:
00195             print('', file=texfile)
00196             print('\\subsection{%s of %s}' %
      (table.result_label.replace('^','\\^').replace('_','\\_'),
      table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
00197             print('', file=texfile)
00198             print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
      file=texfile)
00199             print('', file=texfile)
00200             print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
      file=texfile)
00201             print('', file=texfile)
00202             print('Maximum relative error for %s = %10.03g' %
      (ref.replace('^','\\^').replace('_','\\_'), max_rel_err), file=texfile)
00203             print('', file=texfile)
00204
00205
```

Here is the call graph for this function:



14.19.1.5 nuclear_HB_tt()

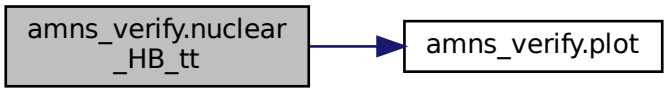
```
def amns_verify.nuclear_HB_tt (
                r1,
                r2,
                p1,
                p2,
                reac,
                ref,
                file = None,
                texfile = None )
```

Definition at line 134 of file amns_verify.py.

```
00134 def nuclear_HB_tt(r1, r2, p1, p2, reac, ref, file=None, texfile=None):
00135
00136     nx=[101,1]
00137
00138     x = np.loadtxt(ref)
00139     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10).repeat(nx[1]).reshape(nx),order='F')
00140
00141     height=210/25.4
```

```
00142     width=297/25.4
00143     plt.figure(1, figsize=(width,height))
00144 
00145     r = amns.Reactants()
00146     r.add(r1[0],r1[1],r1[2])
00147     r.add(r2[0],r2[1],r2[2])
00148     r.add(p1[0],p1[1],p1[2],lr=1)
00149     r.add(p2[0],p2[1],p2[2],lr=1)
00150     table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
00151     res = table.data(E)
00152 
00153     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Ti', '%s [%s]' % (table.result_label, table.result_unit),
      '%s' % (reac,), file)
00154 
00155     max_rel_err = np.max(np.abs((table.data(x[:,0].copy()) - x[:,1]) / x[:,1]))
00156 
00157     print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
00158 
00159     if file:
00160         if texfile:
00161             print('', file=texfile)
00162             print('\\subsection{%s of %s}' %
      (table.result_label.replace('^','\\^').replace('_','\\_'),
      table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
00163             print('', file=texfile)
00164             print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
      file=texfile)
00165             print('', file=texfile)
00166             print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
      file=texfile)
00167             print('', file=texfile)
00168             print('Maximum relative error for %s = %10.03g' %
      (ref.replace('^','\\^').replace('_','\\_'), max_rel_err), file=texfile)
00169             print('', file=texfile)
00170 
00171 
```

Here is the call graph for this function:



14.19.1.6 plot()

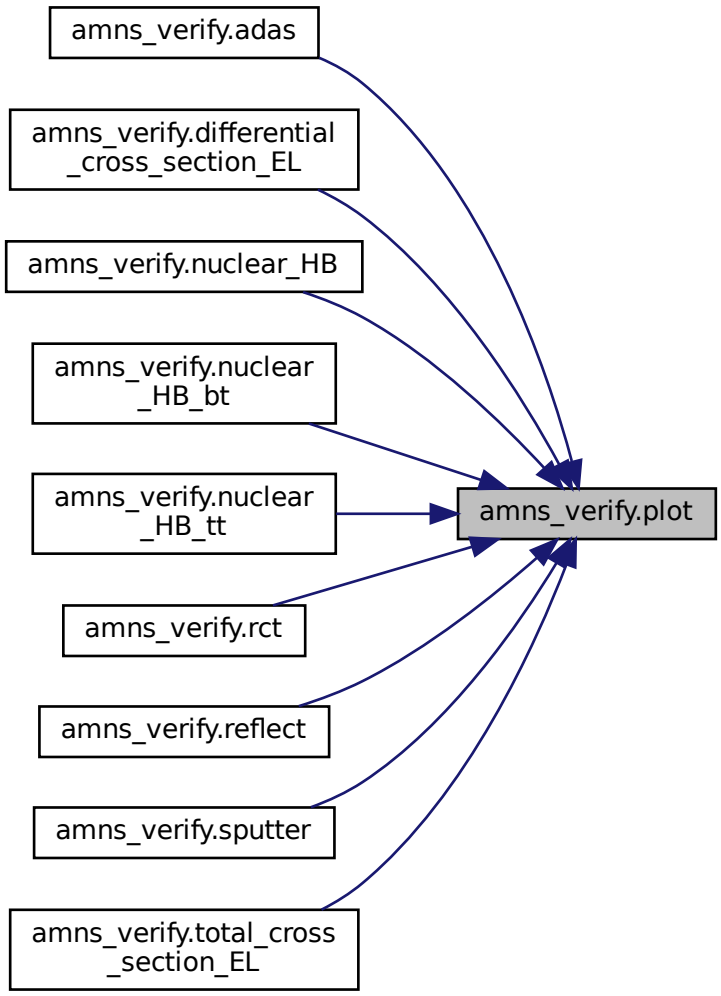
```
def amns_verify.plot (
                x1,
                y1,
                x2,
                y2,
                xlabel,
                ylabel,
                title,
                file = None,
                line1 = 'bo-',
                line2 = 'r+-' )
```

Definition at line 14 of file amns_verify.py.

```
00014 def plot(x1,y1,x2,y2,xlabel,ylabel,title,file=None,line1='bo-',line2='r+-'):
00015 
00016     height=210/25.4
00017     width=297/25.4
00018     plt.figure(1, figsize=(width,height))
00019 
00020     plt.clf()
00021     try:
```

```
00022         plt.loglog(x1, y1, line1, label='AMNS') # ValueError: Data has no positive values, and
      therefore can not be log-scaled.
00023         plt.loglog(x2, y2, line2, label='REF')
00024     except ValueError:
00025         print("Warning: Data has no positive values, not plotting log: %s, %s" % (title ,file))
00026         plt.plot(x1, y1, line1, label='AMNS')
00027         plt.plot(x2, y2, line2, label='REF')
00028     plt.xlabel(xlabel)
00029     plt.ylabel(ylabel)
00030     plt.title(title)
00031     y1_max=m.ceil(m.log10(np.nanmax(y1)))
00032     try:
00033         y1_min=m.floor(m.log10(np.nanmin(y1))) # ValueError: math domain error
00034     except ValueError:
00035         print("Warning: math domain error on (%e): %s, %s" % (np.nanmin(y1), title ,file))
00036         y1_min=m.floor(m.log10(np.nanmin(y1+1e-32)))
00037     y1_min=max(y1_max-20,y1_min)
00038     plt.ylim(10**y1_min,10**y1_max)
00039     plt.legend(loc=0)
00040     if (file is None):
00041         plt.show()
00042     else:
00043         try:
00044             plt.savefig(file, papertype='a4', orientation='landscape')
00045         except:
00046             plt.savefig(file)
00047
```

Here is the caller graph for this function:



14.19.1.7 rct()

```
def amns_verify.rct (
                zn,
                za,
                mi,
                reac,
                ref,
                file = None )
```

Definition at line 240 of file amns_verify.py.

```
00240 def rct(zn,za,mi,reac,ref,file=None):
00241
00242     nx=[101,1]
00243
00244     x = np.loadtxt(ref)
00245     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/6.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00246
00247     r = amns.Reactants()
```

```
00248     r.add(zn,za,mi)
00249     r.add(zn,0.0,mi)
00250     r.add(zn,za-1,mi,lr=1)
00251     r.add(zn,1.0,mi,lr=1)
00252     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00253     res = table.data(E)
00254
00255     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn, za, mi), file, line2='r*')
00256
00257
```

Here is the call graph for this function:

14.19.1.8 reflect()

```
def amns_verify.reflect (
                zn_w,
                za_w,
                mi_w,
                zn_p,
                za_p,
                mi_p,
                reac,
                ref,
                file = None )
```

Definition at line 277 of file amns_verify.py.

```
00277 def reflect(zn_w,za_w,mi_w,zn_p,za_p,mi_p,reac,ref,file=None):
00278
00279     nx=[101,1]
00280
00281     x = np.loadtxt(ref, skiprows=6)
00282     nxr=list(nx) ; nxr.reverse()
00283     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
00284     A=np.array(np.array([0.0]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00285
00286     r = amns.Reactants()
00287     r.add(zn_w,za_w,mi_w)
00288     r.add(zn_p,za_p,mi_p)
00289     r.add(zn_p,za_p,mi_p, lr=1)
00290     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00291     res = table.data(E,A)
00292
00293     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn_p, za_p, mi_p), file, line2='r*')
00294
```

Here is the call graph for this function:

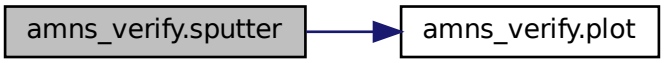
14.19.1.9 sputter()

```
def amns_verify.sputter (
                zn_w,
                za_w,
                mi_w,
                zn_p,
                za_p,
                mi_p,
                reac,
                ref,
                file = None )
```

Definition at line 258 of file amns_verify.py.

```
00258 def sputter(zn_w,za_w,mi_w,zn_p,za_p,mi_p,reac,ref,file=None):
00259
00260     nx=[101,1]
00261
00262     x = np.loadtxt(ref, skiprows=6)
00263     nxr=list(nx) ; nxr.reverse()
00264     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
00265     A=np.array(np.array([0.0]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00266
00267     r = amns.Reactants()
00268     r.add(zn_w,za_w,mi_w)
00269     r.add(zn_p,za_p,mi_p)
00270     r.add(zn_w,za_w,mi_w, lr=1)
00271     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00272     res = table.data(E,A)
00273
00274     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn_p, za_p, mi_p), file, line2='r*')
00275
00276
```

Here is the call graph for this function:



14.19.1.10 total_cross_section_EL()

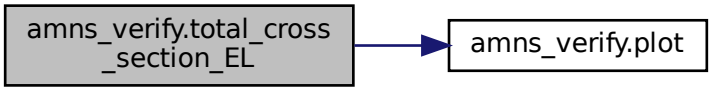
```
def amns_verify.total_cross_section_EL (
                zn,
                za,
                mi,
                reac,
                ref,
                file = None )
```

Definition at line 206 of file amns_verify.py.

```
00206 def total_cross_section_EL(zn,za,mi,reac,ref,file=None):
00207
00208     nx=[101,1]
00209
00210     x = np.loadtxt(ref)
00211     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00212
00213     r = amns.Reactants()
00214     r.add(zn,za,mi)
```

```
00215     r.add(zn,za,mi,lr=1)
00216     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00217     res = table.data(E)
00218
00219     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn, za, mi), file)
00220
00221
```

Here is the call graph for this function:



14.19.2 Variable Documentation

14.19.2.1 amnsdb

```
amns_verify.amnsdb = amns.Amns()
```

Definition at line 295 of file amns_verify.py.

14.19.2.2 file

```
amns_verify.file
```

Definition at line 312 of file amns_verify.py.

14.19.2.3 texfile

```
amns_verify.texfile = open('amns_verify.tex', 'w')
```

Definition at line 300 of file amns_verify.py.

14.20 Package amnsdemo

Classes

- class AmnsDemoCaseIdx

14.21 beamtargetreactions Module Reference

Functions/Subroutines

- subroutine, public btr_error (err)

  a routine for printing the error messages
- subroutine, public btr_beamtargetrate (reaction, E, T, rate, err)

  this subroutine calculates the fusion reaction rate in units of [m^3/s] for the beam particle having energy E [eV] colliding with Maxwellian target at temperature T [eV] The actual reaction rate is then obtained by mulitplying the result with the target particle density.
- integer function getreactparams (reaction, erange, mb, mt, sigmaMax, beamCorrection)

- real(kind=params_wp) function btr_sigmavintegrandnoparams (u)

  This function is the one called by the numerical integrator.
- real(kind=params_wp) function btr_sigmavintegrand (u, reaction, energyBeam, temperatureTarget)

  This function calculates the integrand for the beam-target fusion reaction rate.
- subroutine, public btr_test_beamtargetrate ()

  This routine test the calculation of the beam-target fusion rate.
- subroutine btr_test_integrand ()

  a test routine to print out the beam-target reaction rate integrand
- subroutine btr_test_integrandmatrix ()

  a test routine to print out the beam-target reaction rate integrand at some beam energy but at several temperatures

Variables

- real(kind=params_wp), parameter consts_pi = 3.14159265358979323846264338327950288419 71_↩ params_wp
- real(kind=params_wp), parameter consts_twopi = 2.0_params_wp ∗ consts_pi
- real(kind=params_wp), parameter consts_mtriton = 5.00735588e-27_params_wp
- real(kind=params_wp), parameter consts_amu = 1.660538782e-27_params_wp
- real(kind=params_wp), parameter consts_e = 1.602176487e-19_params_wp
- real(kind=params_wp), parameter consts_mdeuteron = 3.34358320e-27_params_wp
- real(kind=params_wp), parameter consts_mhe3 = 3.0160293_params_wp ∗ consts_amu
- integer, parameter btr_success = 0
- integer, parameter btr_unsupportedreaction = 1
- integer, parameter btr_reaction_dtn4he = 1

  These flags define the different reactions inside this module.
- integer, parameter btr_reaction_tdn4he = 2

  Tritium beam collides with deuterium plasma, yielding neutron and helium-4.
- integer, parameter btr_reaction_ddpt = 3

  Deuterium beam collides with deuterium plasma, yielding proton and tritium.
- integer, parameter btr_reaction_ddn3he = 4

  Deuterium beam collides with deuterium plasma, yielding neutron and helium-3.
- integer, parameter btr_reaction_d3hep4he = 5

  Deuterium beam collides with helium-3, yielding proton and helium-4.
- integer, parameter btr_reaction_3hedp4he = 6

  Helium-3 beam collides with deuterium, yielding proton and helium-4.

14.21.1 Function/Subroutine Documentation

14.21.1.1 btr_beamtargetrate()

```
subroutine, public beamtargetreactions::btr_beamtargetrate (
             integer, intent(in) reaction,
             real(kind=params_wp), intent(in) E,
             real(kind=params_wp), intent(in) T,
             real(kind=params_wp), intent(out) rate,
             integer, intent(out) err )
```

this subroutine calculates the fusion reaction rate in units of [m^3/s] for the beam particle having energy E [eV] colliding with Maxwellian target at temperature T [eV] The actual reaction rate is then obtained by mulitplying the result with the target particle density.

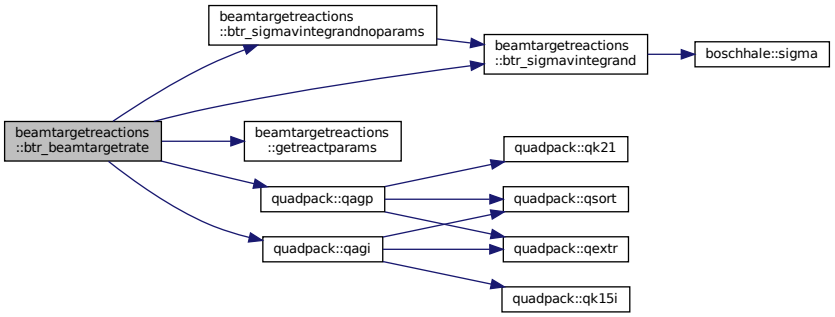
Definition at line 88 of file beamTargetReactions.f90.

```
00089     implicit none
00090     ! I/O variables
00091     integer, intent(in) :: reaction
00092     real(kind=params_wp), intent(in) :: e
```

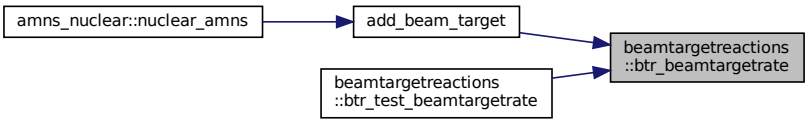
```
00093     real(kind=params_wp), intent(in) :: t
00094     real(kind=params_wp), intent(out) :: rate
00095     integer, intent(out) :: err
00096     ! internal variables
00097     integer, parameter :: nThermal = 15
00098     real(kind=params_wp) :: erange(2)
00099     real(kind=params_wp) :: urange(2)
00100     real(kind=params_wp) :: sigmamax
00101     real(kind=params_wp) :: epsrel
00102     real(kind=params_wp) :: epsabs
00103     real(kind=params_wp) :: abserr
00104     real(kind=params_wp) :: tmp
00105     real(kind=params_wp) :: upperbound,beamcorrection
00106     real(kind=params_wp) :: vthermal,vbeam
00107     real(kind=params_wp) :: mt,mb
00108     real(kind=params_wp) :: u( 2*(2*nthermal +1) +2 )
00109     integer :: nu,iu,inf
00110     integer :: neval
00111
00112
00113     err=btr_success
00114
00115
00116
00117     err = getreactparams(reaction,erange,mb,mt,sigmamax,beamcorrection)
00118     if(err/=0) return
00119
00120     urange    =  sqrt(2.0_params_wp * consts_e * erange  *(mb+mt)/(mb*mt))
00121     sigmamax =  sqrt(2.0_params_wp * consts_e * sigmamax *(mb+mt)/(mb*mt))
00122     vthermal =  sqrt(2.0_params_wp * consts_e * t       /mt) ! *(mb+mt)/(mb*mt))
00123     vbeam     =  sqrt(2.0_params_wp * consts_e * e       /mb) ! *(mb+mt)/(mb*mt))
00124
00125     ! set the parameters for the integrator
00126     epsabs=1e-60_params_wp
00127     epsrel=1e-20_params_wp
00128
00129     ! The list of "special points" in u:
00130     ! vbeam + n*vthermal (n=-5:5)
00131     ! equivalent velocity for erange min, max
00132     ! Maxima of the sigma
00133     ! the last bound...infty starts from max of the others +3*vthermal
00134
00135     nu=0
00136
00137     ! Beam velocity \pm Thermal velocity
00138     do iu=1,(nthermal *2 +1)
00139        nu=nu+1
00140        u(nu) = vbeam*beamcorrection + (iu-nthermal-1) * vthermal
00141        if(u(nu)<= 0.0_params_wp) then
00142           nu = nu - 1
00143        end if
00144     end do
00145
00146     if(.false.) then
00147        ! sigma peak velocity \pm Thermal velocity
00148        do iu=1,(nthermal *2 +1)
00149           nu=nu+1
00150           u(nu) = sigmamax + (iu-nthermal-1) * vthermal
00151           if(u(nu)<= 0.0_params_wp) then
00152              nu = nu - 1
00153           end if
00154        end do
00155     end if
00156
00157     u( (nu+1) : (nu+2) ) = urange
00158     nu=nu+2
00159
00160     if(minval(u(1:nu)) .le. 0.0_params_wp) then
00161        write(*,*) 'lt zero velocity'
00162        write(*,*) u(1:nu)
00163        stop 10
00164     end if
00165
00166     upperbound = 3.0_params_wp*vthermal + maxval(u(1:nu))
00167
00168
00169     ! initialize the integrand function
00170     tmp=btr_sigmavintegrand(upperbound,reaction,e,t)
00171
00172 #ifdef VERBOSE
00173     write(*,*) 'calculating the finite integral'
00174 #endif
00175 !    call qags(btr_sigmavIntegrandNoParams,u(iu-1),u(iu),epsabs,epsrel,tmp,abserr,neval,err)
00176     call qagp (btr_sigmavintegrandnoparams, 0.0_params_wp, upperbound, &
00177          &     nu, u(1:nu), &
00178          &     epsabs, epsrel,&
00179          &     tmp, abserr, &
```

```
00180          &     neval, err )
00181
00182 #ifdef VERBOSE
00183     write(*,*) '     err:',err,' neval:',neval
00184     write(*,*) '     rate',tmp,'+-',abserr
00185 #endif
00186     select case(err)
00187     case(0)
00188     case(2)
00189 #ifdef VERBOSE
00190        write(*,*) '    the occurrence of roundoff error is'
00191        write(*,*) '           detected, which prevents the requested'
00192        write(*,*) '           tolerance from being achieved.'
00193        write(*,*) '           the error may be under-estimated.'
00194 #endif
00195     case(3)
00196 #ifdef VERBOSE
00197        write(*,*) '    extremely bad integrand behavior occurs'
00198        write(*,*) '           at some points of the integration '
00199        write(*,*) '           interval.'
00200 #endif
00201     case(4)
00202 #ifdef VERBOSE
00203       write(*,*) 'the algorithm does not converge.  Roundoff error is detected in the'
00204       write(*,*) 'extrapolation table.  It is assumed that the requested tolerance'
00205       write(*,*) 'cannot be achieved, and that the returned result is the best which '
00206       write(*,*) 'can be obtained.'
00207 #endif
00208     case(5)
00209 #ifdef VERBOSE
00210        write(*,*) '  the integral is probably divergent, or slowly convergent.'
00211 #endif
00212     case default
00213        stop 1
00214     end select
00215
00216     rate = tmp
00217
00218 #ifdef VERBOSE
00219     write(*,*) 'calculating the infinite integral'
00220 #endif
00221     inf = 1
00222     call qagi ( btr_sigmavintegrandnoparams, upperbound, inf, epsabs, epsrel, tmp, abserr, neval, err
       )
00223
00224 #ifdef VERBOSE
00225     write(*,*) '     err:',err,' neval:',neval
00226     write(*,*) '     rate',tmp,'+-',abserr
00227     write(*,*)
00228 #endif
00229
00230     select case(err)
00231     case(0)
00232     case(2)
00233 #ifdef VERBOSE
00234        write(*,*) '    the occurrence of roundoff error is'
00235        write(*,*) '           detected, which prevents the requested'
00236        write(*,*) '           tolerance from being achieved.'
00237        write(*,*) '           the error may be under-estimated.'
00238 #endif
00239     case(3)
00240 #ifdef VERBOSE
00241        write(*,*) '    extremely bad integrand behavior occurs'
00242        write(*,*) '           at some points of the integration '
00243        write(*,*) '           interval.'
00244 #endif
00245     case(4)
00246 #ifdef VERBOSE
00247       write(*,*) 'the algorithm does not converge.  Roundoff error is detected in the'
00248       write(*,*) 'extrapolation table.  It is assumed that the requested tolerance'
00249       write(*,*) 'cannot be achieved, and that the returned result is the best which '
00250       write(*,*) 'can be obtained.'
00251 #endif
00252     case(5)
00253 #ifdef VERBOSE
00254        write(*,*) '  the integral is probably divergent, or slowly convergent.'
00255 #endif
00256     case default
00257        stop 1
00258     end select
00259
00260     rate = rate + tmp
00261     err = 0
00262
00263
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.21.1.2 btr_error()

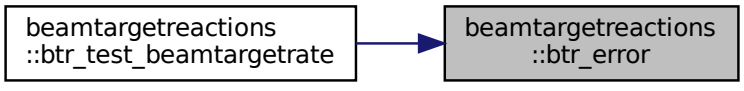
```
subroutine, public beamtargetreactions::btr_error (
            integer, intent(in) err )
```

a routine for printing the error messages

Definition at line 41 of file beamTargetReactions.f90.

```
00042     implicit none
00043     integer, intent(in) :: err
00044     select case(err)
00045     case(btr_success)
00046     case(btr_unsupportedreaction)
00047        write(*,*) '------------------------------'
00048        write(*,*) 'btr_error:',btr_unsupportedreaction
00049        write(*,*) 'unsupported reaction. Aborting'
00050        write(*,*) '------------------------------'
00051     case default
00052        write(*,*) '----------------------------------------'
00053        write(*,*) 'btr_error:'
00054        write(*,*) 'Unknown error identifier.'
00055        write(*,*) '----------------------------------------'
00056     end select
```

Here is the caller graph for this function:



14.21.1.3 btr_sigmavintegrand()

```
real(kind=params_wp) function beamtargetreactions::btr_sigmavintegrand (
             real(kind=params_wp), intent(in) u,
             integer, intent(in), optional reaction,
             real(kind=params_wp), intent(in), optional energyBeam,
             real(kind=params_wp), intent(in), optional temperatureTarget )
```

This function calculates the integrand for the beam-target fusion reaction rate.

Parameters

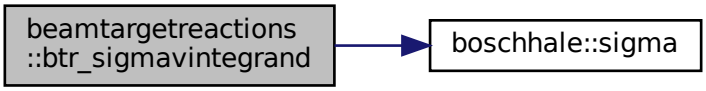
in	*u*	the collision velocity [m/s]
in	*reaction*	the integer flag for a certain fusion reaction
in	*energybeam*	energy of the beam [eV]
in	*temperaturetarget*	temperature of the target particles maxwellian distribution [eV]

Definition at line 331 of file beamTargetReactions.f90.

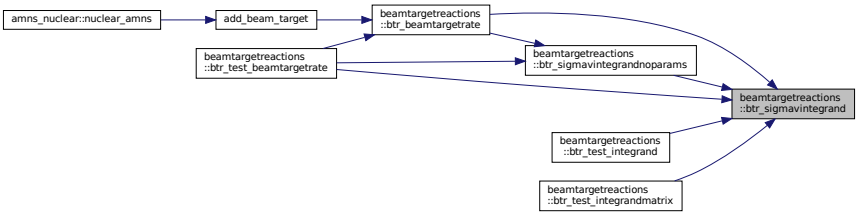
```
00332     implicit none
00333     ! I/O variables
00334     real(kind=params_wp), intent(in) :: u
00335     integer, intent(in), optional :: reaction
00336     real(kind=params_wp), intent(in), optional :: energybeam
00337     real(kind=params_wp), intent(in), optional :: temperaturetarget
00338     real(kind=params_wp) :: integrand
00339     ! Internal variables, some of them are saved to allow noParams version of the function for
      numerical integration
00340     real(kind=params_wp), save :: vb
00341     real(kind=params_wp), save :: tt
00342     real(kind=params_wp), save :: mt
00343     real(kind=params_wp), save :: mb
00344     integer, save :: reactionSave
00345     real(kind=params_wp) :: ecom
00346     real(kind=params_wp) :: crosssection
00347     real(kind=params_wp) :: firstexponent
00348     real(kind=params_wp) :: secondexponent
00349     real(kind=params_wp) :: hyperbolicsinarg
00350
00351     if(present(reaction)) then ! save the parameters
00352        select case(reaction)
00353        case(btr_reaction_dtn4he)
00354           mb=consts_mdeuteron
00355           mt=consts_mtriton
00356        case(btr_reaction_tdn4he)
00357           mb=consts_mtriton
00358           mt=consts_mdeuteron
00359        case(btr_reaction_ddpt)
00360           mb=consts_mdeuteron
00361           mt=consts_mdeuteron
00362        case(btr_reaction_ddn3he)
00363           mb=consts_mdeuteron
00364           mt=consts_mdeuteron
00365        case(btr_reaction_d3hep4he)
00366           mb=consts_mdeuteron
00367           mt=consts_mhe3
00368        case(btr_reaction_3hedp4he)
00369           mb=consts_mhe3
00370           mt=consts_mdeuteron
00371        case default
00372           write(*,*) 'error'
00373           stop 10
00374        end select
00375        vb=sqrt(2*consts_e*energybeam/mb) ! calculate the beam velocity [m/s] using classical model
00376        tt=temperaturetarget
00377        reactionsave=reaction
00378        integrand=0._params_wp
00379     else ! evaluate the integrand
00380        ecom=0.5_params_wp*mt*mb/(mt+mb)*u**2 ! center of mass frame energy for the reaction in Joules
00381        crosssection=sigma(ecom/(consts_e*1e3),reactionsave) ! the cross section needs the energy in
      keV
00382        ! We use different forms depending on u, vb and Tt to prevent numerical difficulties
00383        firstexponent=-mt*vb**2/(2*consts_e*tt)
00384        secondexponent=-mt*u**2/(2*consts_e*tt)
00385        hyperbolicsinarg=mt*u*vb/(consts_e*tt)
00386        if(hyperbolicsinarg<=10) then
00387           integrand=2*sqrt(mt/(consts_twopi*consts_e*tt))/vb*exp(firstexponent+secondexponent)&
```

```
00388                &*crosssection*u**2*sinh(hyperbolicsinarg)
00389         else ! for x>=10 the relative difference between sinh(x) and 1/2*exp(x) is less than 2e-9
00390
      integrand=sqrt(mt/(consts_twopi*consts_e*tt))/vb*exp(firstexponent+secondexponent+hyperbolicsinarg)*crosssection*u**2
00391         end if
00392      end if
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.21.1.4 btr_sigmavintegrandnoparams()

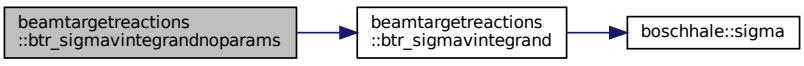
```
real(kind=params_wp) function beamtargetreactions::btr_sigmavintegrandnoparams (
            real(kind=params_wp), intent(in) u )
```

This function is the one called by the numerical integrator.

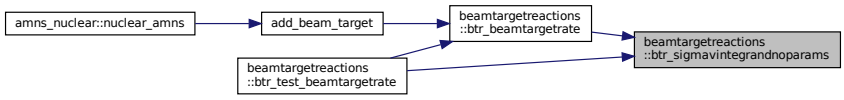
Definition at line 320 of file beamTargetReactions.f90.

```
00321     implicit none
00322     real(kind=params_wp), intent(in) :: u
00323     real(kind=params_wp) :: integrand
00324
00325     integrand=btr_sigmavintegrand(u)
00326
```

Here is the call graph for this function:



Here is the caller graph for this function:



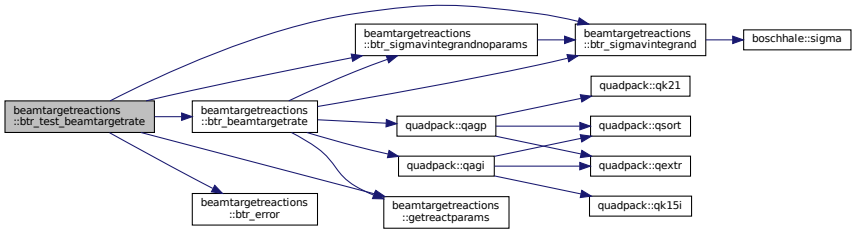
14.21.1.5 btr_test_beamtargetrate()

subroutine, public beamtargetreactions::btr_test_beamtargetrate
This routine test the calculation of the beam-target fusion rate.
Definition at line 397 of file beamTargetReactions.f90.

```
00398     implicit none
00399     real(kind=params_wp) :: e1,e2,u1
00400     real(kind=params_wp) :: t1,t2,u2
00401     real(kind=params_wp) :: erange(2),mb,mt,sigmamax,beamcorrection
00402     integer :: nE, nT, iE, iT, nu, iU
00403     real(kind=params_wp), allocatable :: rate(:,:), integrand(:,:,:)
00404     real(kind=params_wp), allocatable :: e(:)
00405     real(kind=params_wp), allocatable :: t(:)
00406     real(kind=params_wp), allocatable :: u(:)
00407     real(kind=params_wp), allocatable :: ulo(:,:),uhi(:,:)
00408     integer :: reaction
00409     integer :: err
00410     integer :: chn
00411
00412     err=btr_success
00413
00414     write(*,*) 'Give the reaction'
00415     write(*,*) '1 == D(T,n)^4He'
00416     write(*,*) '2 == T(D,n)^4He'
00417     write(*,*) '3 == D(D,p)T'
00418     write(*,*) '4 == D(D,n)^3He'
00419     write(*,*) '5 == D(^3He,p)^4He'
00420     write(*,*) '6 == ^3He(D,p)^4He'
00421     read(*,*) reaction
00422     if(reaction<1 .or. reaction>6) then
00423        write(*,*) 'Invalid reaction. Aborting...'
00424        return
00425     end if
00426
00427     write(*,*) 'Give the beam energy range (E1 E2) [keV], and the number of energies (min 2)'
00428     read(*,*) e1, e2, ne
00429     write(*,*) 'Give the plasma temperature range (T1 T2) [keV], and the nmbuer of temperatures (min
    2)'
00430     read(*,*) t1, t2, nt
00431     e1=e1*1e3
00432     e2=e2*1e3
00433     t1=t1*1e3
00434     t2=t2*1e3
00435     u1=1.0_params_wp ! m/s
00436     u2=3.0e8
00437     nu=200
00438
00439     if(ne<2 .or. nt<2) then
00440        write(*,*) 'Too few energies or temperatures'
00441        return
00442     end if
00443
00444     err=  getreactparams(reaction,erange,mb,mt,sigmamax,beamcorrection)
00445     if(err /= 0 ) return
00446
00447
00448     allocate(rate(ne,nt),e(ne),t(nt),u(nu),integrand(ne,nt,nu))
00449     allocate(ulo(ne,nt),uhi(ne,nt))
00450     e1=log10(e1)
00451     e2=log10(e2)
00452     t1=log10(t1)
00453     t2=log10(t2)
00454     u1=log10(u1)
00455     u2=log10(u2)
00456     e=e1+(/ ( (ie-1)*( (e2-e1)/(ne-1) ), ie=1,ne) /)
00457     t=t1+(/ ( (it-1)*( (t2-t1)/(nt-1) ), it=1,nt) /)
00458     u=u1+(/ ( (iu-1)*( (u2-u1)/(nu-1) ), iu=1,nu) /)
00459     e = 10.0_params_wp ** e
```

```
00460     t = 10.0_params_wp ** t
00461     u = 10.0_params_wp ** u
00462
00463
00464     chn=26
00465     open(unit=chn,file="btr_E.test")
00466     write(chn,*) e
00467     close(chn)
00468     write(*,*) 'wrote the E-vector into file "btr_E.test"'
00469     open(unit=chn,file="btr_T.test")
00470     write(chn,*) t
00471     close(chn)
00472     write(*,*) 'wrote the T-vector into file "btr_T.test"'
00473     open(unit=chn,file="btr_u.test")
00474     write(chn,*) u
00475     close(chn)
00476     write(*,*) 'wrote the u-vector into file "btr_u.test"'
00477
00478     do ie=1,ne
00479        do it=1,nt
00480           call btr_beamtargetrate(reaction,e(ie),t(it),rate(ie,it),err)
00481           if(err/=btr_success) then
00482              call btr_error(err)
00483              stop 2
00484           end if
00485           ulo(ie,it) =  sqrt(2.0_params_wp * consts_e * e(ie) /mb)*beamcorrection - &
00486                &   sqrt(2.0_params_wp * consts_e * t(it)      /mt)
00487           uhi(ie,it) =  sqrt(2.0_params_wp * consts_e * e(ie) /mb)*beamcorrection + &
00488                &   sqrt(2.0_params_wp * consts_e * t(it)      /mt)
00489           integrand(ie,it,1) = btr_sigmavintegrand(10.0_params_wp,reaction,e(ie),t(it))
00490           do iu=1,nu
00491              integrand(ie,it,iu)=btr_sigmavintegrandnoparams(u(iu))
00492           end do
00493        end do
00494     end do
00495
00496     open(unit=chn,file="btr_rate.test")
00497     do ie=1,ne
00498        write(chn,*) rate(ie,:)
00499     end do
00500     close(chn)
00501     write(*,*) 'wrote the resulting rates into file "btr_rate.test"'
00502
00503     open(unit=chn,file="btr_ulo.test")
00504     do ie=1,ne
00505        write(chn,*) ulo(ie,:)
00506     end do
00507     close(chn)
00508     write(*,*) 'wrote the resulting lowever limits into file "btr_ulo.test"'
00509     open(unit=chn,file="btr_uhi.test")
00510     do ie=1,ne
00511        write(chn,*) uhi(ie,:)
00512     end do
00513     close(chn)
00514     write(*,*) 'wrote the resulting lowever limits into file "btr_uhi.test"'
00515
00516     chn=25
00517     open(unit=chn,file="btr_integrand.test")
00518     do ie=1,ne
00519        do it=1,nt
00520           write(chn,*) integrand(ie,it,:)
00521        end do
00522     end do
00523     close(chn)
00524     write(*,*) 'wrote the resulting rates into file "btr_integrand.test"'
00525
00526     deallocate(rate,e,t,u,integrand,ulo,uhi)
00527
```

Here is the call graph for this function:



14.21.1.6 btr_test_integrand()

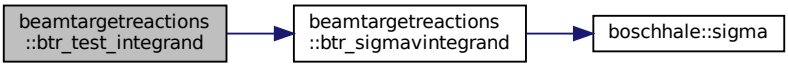
subroutine beamtargetreactions::btr_test_integrand

a test routine to print out the beam-target reaction rate integrand

Definition at line 533 of file beamTargetReactions.f90.

```
00534     implicit none
00535     real(kind=params_wp), allocatable :: u(:)
00536     real(kind=params_wp), allocatable :: integrand(:)
00537     integer :: reaction
00538     integer :: nu
00539     integer :: iu
00540     integer :: chn
00541     real(kind=params_wp) :: erange(2)
00542     real(kind=params_wp) :: urange(2)
00543     real(kind=params_wp) :: eb
00544     real(kind=params_wp) :: t
00545
00546     iu=0
00547     nu=10000
00548
00549     write(*,*) 'Give the reaction for which the integrand will be tabulated'
00550     write(*,*) '1 == D(T,n)^4He'
00551     write(*,*) '2 == T(D,n)^4He'
00552     write(*,*) '3 == D(D,p)T'
00553     write(*,*) '4 == D(D,n)^3He'
00554     write(*,*) '5 == D(^3He,p)^4He'
00555     write(*,*) '6 == ^3He(D,p)^4He'
00556     read(*,*) reaction
00557
00558     select case(reaction) ! the energy range is given in keV!!!
00559     case(btr_reaction_dtn4he,btr_reaction_tdn4he)
00560        erange = (/ 0.5_params_wp, 4700.0_params_wp /)
00561        urange =
       sqrt(2.0_params_wp*consts_e*1e3*erange*(consts_mdeuteron+consts_mtriton)/(consts_mdeuteron*consts_mtriton))
00562     case(btr_reaction_ddpt)
00563        erange = (/ 0.5_params_wp, 5000.0_params_wp /)
00564        urange = sqrt(4*consts_e*1e3*erange/consts_mdeuteron)
00565     case(btr_reaction_ddn3he)
00566        erange = (/ 0.5_params_wp, 4900.0_params_wp /)
00567        urange = sqrt(4*consts_e*1e3*erange/consts_mdeuteron)
00568     case(btr_reaction_d3hep4he,btr_reaction_3hedp4he)
00569        erange = (/ 0.3_params_wp, 4800.0_params_wp /)
00570        urange =
       sqrt(2*consts_e*1e3*erange*(consts_mdeuteron+consts_mhe3)/(consts_mdeuteron*consts_mhe3))
00571     case default
00572        write(*,*) 'Unsupported reaction. Try again'
00573        return
00574     end select
00575
00576     write(*,*) 'Give the beam energy [keV]'
00577     read(*,*) eb
00578     write(*,*) 'Give the plasma temperature [keV]'
00579     read(*,*) t
00580
00581     allocate(u(nu),integrand(nu))
00582     u=urange(1)+(/ ( (iu-1)*( (urange(2)-urange(1))/(nu-1) ), iu=1,nu) /) ! make the u vector
00583     integrand(1)=btr_sigmavintegrand(u(1),reaction,eb*1e3,t*1e3) ! initialize the integrand function
00584     do iu=1,nu
00585        integrand(iu)=btr_sigmavintegrand(u(iu))
00586     end do
00587
00588     chn=27
00589     open(unit=chn,file="btr_integrand.test")
00590     do iu=1,nu
00591        write(chn,*) u(iu),integrand(iu)
00592     end do
00593     close(chn)
00594     write(*,*) 'The integrand is now tabulated into file "btr_integrand.test"'
00595
00596     deallocate(u,integrand)
00597
```

Here is the call graph for this function:



14.21.1.7 btr_test_integrandmatrix()

`subroutine beamtargetreactions::btr_test_integrandmatrix`

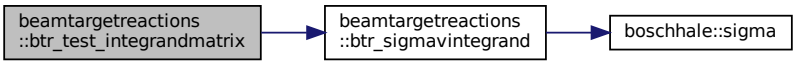
a test routine to print out the beam-target reaction rate integrand at some beam energy but at several temperatures

Definition at line 603 of file beamTargetReactions.f90.

```
00604     implicit none
00605     real(kind=params_wp), allocatable :: u(:)
00606     real(kind=params_wp), allocatable :: integrand(:,:)
00607     real(kind=params_wp), allocatable :: t(:)
00608     integer :: reaction
00609     integer :: nu
00610     integer :: iu
00611     integer :: chn
00612     real(kind=params_wp) :: erange(2)
00613     real(kind=params_wp) :: urange(2)
00614     real(kind=params_wp) :: eb
00615     real(kind=params_wp) :: t1,t2
00616     integer :: nT,iT
00617
00618     iu=0
00619     nu=10000
00620
00621     write(*,*) 'Give the reaction for which the integrand will be tabulated'
00622     write(*,*) '1 == D(T,n)^4He'
00623     write(*,*) '2 == T(D,n)^4He'
00624     write(*,*) '3 == D(D,p)T'
00625     write(*,*) '4 == D(D,n)^3He'
00626     write(*,*) '5 == D(^3He,p)^4He'
00627     write(*,*) '6 == ^3He(D,p)^4He'
00628     read(*,*) reaction
00629
00630     select case(reaction) ! the energy range is given in keV!!!
00631     case(btr_reaction_dtn4he,btr_reaction_tdn4he)
00632        erange = (/ 0.5_params_wp, 4700.0_params_wp /)
00633        urange =
      sqrt(2*consts_e*1e3*erange*(consts_mdeuteron+consts_mtriton)/(consts_mdeuteron*consts_mtriton))
00634     case(btr_reaction_ddpt)
00635        erange = (/ 0.5_params_wp, 5000.0_params_wp /)
00636        urange = sqrt(4*consts_e*1e3*erange/consts_mdeuteron)
00637     case(btr_reaction_ddn3he)
00638        erange = (/ 0.5_params_wp, 4900.0_params_wp /)
00639        urange = sqrt(4*consts_e*1e3*erange/consts_mdeuteron)
00640     case(btr_reaction_d3hep4he,btr_reaction_3hedp4he)
00641        erange = (/ 0.3_params_wp, 4800.0_params_wp /)
00642        urange =
      sqrt(2*consts_e*1e3*erange*(consts_mdeuteron+consts_mhe3)/(consts_mdeuteron*consts_mhe3))
00643     case default
00644        write(*,*) 'Unsupported reaction. Try again'
00645        return
00646     end select
00647
00648     write(*,*) 'Give the beam energy [keV]'
00649     read(*,*) eb
00650     write(*,*) 'Give the plasma temperature range (T1 T2) [keV], and the number of temperatures (min
      2)'
00651     read(*,*) t1, t2, nt
00652     eb=eb*1e3
00653     t1=t1*1e3
00654     t2=t2*1e3
00655
00656     allocate(u(nu),integrand(nu,nt),t(nt))
00657     u=urange(1)+(/ ( (iu-1)*( (urange(2)-urange(1))/(nu-1) ), iu=1,nu) /) ! make the u vector
00658     t=t1+(/ ( (it-1)*( (t2-t1)/(nt-1) ), it=1,nt) /)
00659     do it=1,nt
00660        integrand(1,it)=btr_sigmavintegrand(u(1),reaction,eb,t(it)) ! initialize the integrand function
00661        do iu=1,nu
00662           integrand(iu,it)=btr_sigmavintegrand(u(iu))
```

```
00663        end do
00664     end do
00665
00666     write(777,*) t
00667     write(888,*) u
00668     chn=29
00669     open(unit=chn,file="btr_integrandMatrix.test")
00670     do iu=1,nu
00671        write(chn,*) integrand(iu,:)
00672     end do
00673     close(chn)
00674     write(*,*) 'The integrand is now tabulated into file "btr_integrandMatrix.test"'
00675
00676     deallocate(u,integrand,t)
00677
```

Here is the call graph for this function:



14.21.1.8 getreactparams()

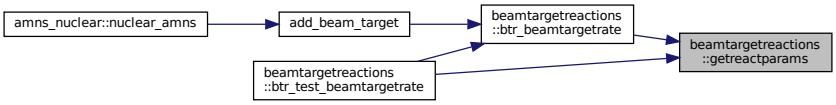
```
integer function beamtargetreactions::getreactparams (
              integer, intent(in) reaction,
              real(kind=params_wp), dimension(2), intent(out) erange,
              real(kind=params_wp), intent(out) mb,
              real(kind=params_wp), intent(out) mt,
              real(kind=params_wp), intent(out) sigmaMax,
              real(kind=params_wp), intent(out) beamCorrection )
```

Definition at line 267 of file beamTargetReactions.f90.

```
00268     implicit none
00269     integer ::  err
00270     integer, intent(in) :: reaction
00271     real(kind=params_wp),intent(out) :: erange(2)
00272     real(kind=params_wp),intent(out) :: sigmamax
00273     real(kind=params_wp),intent(out) :: beamcorrection
00274     real(kind=params_wp),intent(out) :: mt,mb
00275
00276     err = 0
00277
00278     select case(reaction)
00279     case(btr_reaction_dtn4he)
00280        erange = (/ 0.5_params_wp, 4700.0_params_wp /)*1e3_params_wp
00281        mb=consts_mdeuteron
00282        mt=consts_mtriton
00283        sigmamax = 65.0e3_params_wp
00284     case(btr_reaction_tdn4he)
00285        erange = (/ 0.5_params_wp, 4700.0_params_wp /)*1e3_params_wp
00286        mb=consts_mtriton
00287        mt=consts_mdeuteron
00288     case(btr_reaction_ddpt)
00289        erange = (/ 0.5_params_wp, 5000.0_params_wp /)*1e3_params_wp
00290        mb=consts_mdeuteron
00291        mt=consts_mdeuteron
00292        sigmamax = 1580.0e3_params_wp
00293     case(btr_reaction_ddn3he)
00294        erange = (/ 0.5_params_wp, 4900.0_params_wp /)*1e3_params_wp
00295        mb=consts_mdeuteron
00296        mt=consts_mdeuteron
00297        sigmamax = 1100.0e3_params_wp
00298     case(btr_reaction_d3hep4he)
00299        erange = (/ 0.3_params_wp, 4800.0_params_wp /)*1e3_params_wp
00300        mb=consts_mdeuteron
00301        mt=consts_mhe3
00302        sigmamax = 265.0e3_params_wp
00303     case(btr_reaction_3hedp4he)
00304        erange = (/ 0.3_params_wp, 4800.0_params_wp /)*1e3_params_wp
00305        mb=consts_mhe3
00306        mt=consts_mdeuteron
```

```
00307         sigmamax = 265.0e3_params_wp
00308      case default
00309         err=btr_unsupportedreaction
00310         return
00311      end select
00312
00313
00314      beamcorrection  = 1.0_params_wp !0.81_params_wp
00315
```

Here is the caller graph for this function:



14.21.2 Variable Documentation

14.21.2.1 btr_reaction_3hedp4he

```
integer, parameter beamtargetreactions::btr_reaction_3hedp4he = 6
```

Helium-3 beam collides with deuterium, yielding proton and helium-4.
Definition at line 30 of file beamTargetReactions.f90.

```
00030   integer, parameter :: btr_REACTION_3HEDp4HE = 6
```

14.21.2.2 btr_reaction_d3hep4he

```
integer, parameter beamtargetreactions::btr_reaction_d3hep4he = 5
```

Deuterium beam collides with helium-3, yielding proton and helium-4.
Definition at line 29 of file beamTargetReactions.f90.

```
00029   integer, parameter :: btr_REACTION_D3HEp4HE = 5
```

14.21.2.3 btr_reaction_ddn3he

```
integer, parameter beamtargetreactions::btr_reaction_ddn3he = 4
```

Deuterium beam collides with deuterium plasma, yielding neutron and helium-3.
Definition at line 28 of file beamTargetReactions.f90.

```
00028   integer, parameter :: btr_REACTION_DDn3HE = 4
```

14.21.2.4 btr_reaction_ddpt

```
integer, parameter beamtargetreactions::btr_reaction_ddpt = 3
```

Deuterium beam collides with deuterium plasma, yielding proton and tritium.
Definition at line 27 of file beamTargetReactions.f90.

```
00027   integer, parameter :: btr_REACTION_DDpT   = 3
```

14.21.2.5 btr_reaction_dtn4he

```
integer, parameter beamtargetreactions::btr_reaction_dtn4he = 1
```

These flags define the different reactions inside this module.
Deuterium beam collides with tritium plasma, yielding neutron and helium 4
Definition at line 25 of file beamTargetReactions.f90.

```
00025   integer, parameter :: btr_REACTION_DTn4HE = 1
```

14.21.2.6 btr_reaction_tdn4he

```
integer, parameter beamtargetreactions::btr_reaction_tdn4he = 2
```

Tritium beam collides with deuterium plasma, yielding neutron and helium-4.

Definition at line 26 of file beamTargetReactions.f90.

```
00026   integer, parameter :: btr_REACTION_TDn4HE = 2
```

14.21.2.7 btr_success

```
integer, parameter beamtargetreactions::btr_success = 0
```

Definition at line 20 of file beamTargetReactions.f90.

```
00020   integer, parameter :: btr_SUCCESS = 0
```

14.21.2.8 btr_unsupportedreaction

```
integer, parameter beamtargetreactions::btr_unsupportedreaction = 1
```

Definition at line 21 of file beamTargetReactions.f90.

```
00021   integer, parameter :: btr_UNSUPPORTEDREACTION = 1
```

14.21.2.9 consts_amu

```
real(kind=params_wp), parameter beamtargetreactions::consts_amu = 1.660538782e-27_params_wp
```

Definition at line 15 of file beamTargetReactions.f90.

```
00015   real(kind=params_wp), parameter :: consts_amu = 1.660538782e-27_params_wp ! kg
```

14.21.2.10 consts_e

```
real(kind=params_wp), parameter beamtargetreactions::consts_e = 1.602176487e-19_params_wp
```

Definition at line 16 of file beamTargetReactions.f90.

```
00016   real(kind=params_wp), parameter :: consts_e = 1.602176487e-19_params_wp   ! Coulomb
```

14.21.2.11 consts_mdeuteron

```
real(kind=params_wp), parameter beamtargetreactions::consts_mdeuteron = 3.34358320e-27_↩
params_wp
```

Definition at line 17 of file beamTargetReactions.f90.

```
00017   real(kind=params_wp), parameter :: consts_mdeuteron = 3.34358320e-27_params_wp
```

14.21.2.12 consts_mhe3

```
real(kind=params_wp), parameter beamtargetreactions::consts_mhe3 = 3.0160293_params_wp * consts_amu
```

Definition at line 18 of file beamTargetReactions.f90.

```
00018   real(kind=params_wp), parameter :: consts_mhe3      = 3.0160293_params_wp * consts_amu !wikipedia!!!
```

14.21.2.13 consts_mtriton

```
real(kind=params_wp), parameter beamtargetreactions::consts_mtriton = 5.00735588e-27_params_wp
```

Definition at line 14 of file beamTargetReactions.f90.

```
00014   real(kind=params_wp), parameter :: consts_mtriton   = 5.00735588e-27_params_wp
```

14.21.2.14 consts_pi

```
real(kind=params_wp), parameter beamtargetreactions::consts_pi = 3.14159265358979323846264338327950288419971↩
_params_wp
```

Definition at line 12 of file beamTargetReactions.f90.

```
00012   real(kind=params_wp), parameter :: consts_pi =
       3.14159265358979323846264338327950288419_params_wp!acos(-1.0_params_wp)
```

14.21.2.15 consts_twopi

```
real(kind=params_wp), parameter beamtargetreactions::consts_twopi = 2.0_params_wp * consts_pi
```

Definition at line 13 of file beamTargetReactions.f90.

```
00013   real(kind=params_wp), parameter :: consts_twopi = 2.0_params_wp * consts_pi
```

14.22 bms Module Reference

Functions/Subroutines

- subroutine read_bms (filename, y, x1, x2, x3, x4)

14.22.1 Function/Subroutine Documentation

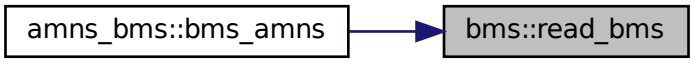
14.22.1.1 read_bms()

```
subroutine bms::read_bms (
            character (len=*) filename,
            real, dimension(:,:,:,:), allocatable y,
            real, dimension(:), allocatable x1,
            real, dimension(:), allocatable x2,
            real, dimension(:), allocatable x3,
            real, dimension(:), allocatable x4 )
```

Definition at line 9 of file bms.f90.

```
00010     real, allocatable :: y(:,:,:,:), x1(:), x2(:), x3(:), x4(:)
00011     integer n1, n2, n3, n4, i1, i2, i3, i4
00012     character (len=*) :: filename
00013
00014     open(unit=10, file=filename)
00015     read(10,*)  ! skip line 1
00016     read(10,*)  ! skip line 2
00017     read(10,*)  n1, n2, n3, n4
00018     read(10,*)  ! skip line 4
00019     allocate(x1(n1), x2(n2), x3(n3), x4(n4), y(n1, n2, n3, n4))
00020     read(10,*) (x1(i1), i1=1, n1)
00021     read(10,*) (x2(i2), i2=1, n2)
00022     read(10,*) (x3(i3), i3=1, n3)
00023     read(10,*) (x4(i4), i4=1, n4)
00024     read(10,*)  ! skip ?
00025     read(10,*) ((((y(i1,i2,i3,i4), i4=1, n4), i3=1, n3), i2=1, n2), i1=1, n1)
00026     x2 = x2 * 1e6
00027     return
```

Here is the caller graph for this function:



14.23 boschhale Module Reference

Functions/Subroutines

- real(kind=params_wp) function sigma (ENE, reaction)

14.23.1 Function/Subroutine Documentation

14.23.1.1 sigma()

```
real(kind=params_wp) function boschhale::sigma (
            real(kind=params_wp), intent(in) ENE,
            integer, intent(in), optional reaction )
```

CALCULATES THE FUSION CROSS-SECTION FOR A GIVEN kine (in CM-FRAME)

ACCORDING TO BOSCH & HALE 1992, NUCL. FUS. 32, PP. 611-631

@article{0029-5515-32-4-I07, author={H.-S. Bosch and G.M. Hale}, title={Improved formulas for fusion cross-sections and thermal reactivities}, journal={Nuclear Fusion}, volume={32}, number={4}, pages={611-31}, url={http://stacks.iop.org/0029-5515/32/611}, year={1992}, abstract={For interpreting fusion rate measurements in present fusion experiments and predicting the fusion performance of future devices or of d-t experiments in present devices, it is important to know the fusion cross-sections as precisely as possible. Usually, it is not measured data that are used, but parametrizations of the cross-section as a function of the ion energy and parametrizations of the Maxwellian reactivity as a function of the ion-temperature. Since the publication of the parametrizations now in use, new measurements have been made and evaluations of the measured data have been improved by applying R-matrix theory. The authors show that the old parametrizations no longer adequately represent the experimental data and present new parametrizations based on R-matrix calculations for fusion cross-sections and Maxwellian reactivities for the reactions D(d,n)3He, D(d,p)T, T(d,n)4He and 3He(d,p)4He}, doi={doi: 10.1088/0029-5515/32/4/I07}

}! ----------

Parameters

in	*ene*	kinetic energy (keV) in CM-FRAME

Returns

FUSION CROSS-SECTION (m^2)

Parameters

in	*reaction*	Variables for passing information to the internal function INTEG: react_reaction - REACTION FLAG: 1 = D(T,n)^4He 2 = T(D,n)^4He 3 = D(D,p)T 4 = D(D,n)^3He 5 = D(^3He,p)^4He 6 = ^3He(D,p)^4He

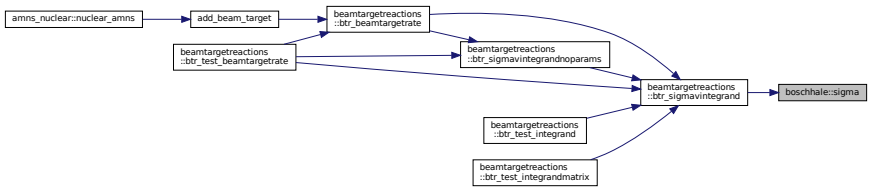
Definition at line 37 of file boschHale.f90.

```
00038
00039
00040     IMPLICIT NONE
00041
00042     !     ERANGE  - RANGE OF VALIDITY FOR THE MODEL
00043     !     S       - PADE POLYNOMIAL
00044     ! ------------------------------------------------------------------
00045
00046     REAL(KIND=params_wp), INTENT(IN) :: ene
00047     real(kind=params_wp) :: atene
00048     REAL(KIND=params_wp) :: a(5), b(4), bg
00049     REAL(KIND=params_wp), dimension(2) :: erange
00050     REAL(KIND=params_wp) :: s
00051     REAL(KIND=params_wp) :: sigma
```

```
00052     REAL(KIND=params_wp), parameter :: tom2 = 1.0e-31_params_wp !(originally mbarn)
00053
00054     INTEGER, SAVE :: ReactionSave
00055
00064     INTEGER, intent(in), optional :: Reaction
00065
00066 #ifdef IGNORE_LOW_E_LIM
00067     if(ene==0.0_params_wp) then
00068        !         write(*,*) 'ENE==0.0 => SIGMA=0.0 ENE=',ENE
00069        sigma = 0.0_params_wp
00070        RETURN
00071     end if
00072 #endif
00073
00074     if(present(reaction)) then
00075        reactionsave=reaction
00076        !         write(*,*) 'Saving reaction', reaction
00077     end if
00078
00079     ! Initialize vars to 0.
00080     bg=0.
00081     a=(/ 0., 0. ,0. ,0. ,0. /)
00082     b=(/ 0., 0. ,0. ,0. /)
00083     !
00084     !     SET THE CONSTANTS BG, A(), b(), AND THE VALIDITY RANGE ERANGE
00085
00086     SELECT CASE(reactionsave)
00087     CASE(1,2) ! DT
00088        IF(ene <= 530) THEN
00089           bg = 34.3827
00090           a = (/ 6.927e4, 7.454e8, 2.050e6, 5.2002e4, 0.0 /)
00091           b = (/ 6.38e1, -9.95e-1, 6.981e-5, 1.728e-4 /)
00092        ELSE
00093           bg = 34.3827
00094           a = (/ -1.4714e6, 0.0, 0.0, 0.0, 0.0 /)
00095           b = (/ -8.4127e-3, 4.7983e-6, -1.0748e-9, 8.5184e-14 /)
00096        ENDIF
00097        erange = (/ 0.5, 4700.0 /)
00098 #ifdef IGNORE_LOW_E_LIM
00099        erange(1) = 0.0
00100 #endif
00101     CASE(3) ! DD -> T
00102        bg = 31.3970
00103        a = (/ 5.5576e4, 2.1054e2, -3.2638e-2, 1.4987e-6, 1.8181e-10 /)
00104        b = (/ 0.0, 0.0, 0.0, 0.0 /)
00105        erange = (/ 0.5, 5000.0 /)
00106 #ifdef IGNORE_LOW_E_LIM
00107        erange(1) = 0.0
00108 #endif
00109     CASE(4) ! DD -> 3He
00110        bg = 31.3970
00111        a = (/ 5.3701e4, 3.3027e2, -1.2706e-1, 2.9327e-5, -2.5151e-9 /)
00112        b = (/ 0.0, 0.0, 0.0, 0.0 /)
00113        erange = (/ 0.5, 4900.0 /)
00114 #ifdef IGNORE_LOW_E_LIM
00115        erange(1) = 0.0
00116 #endif
00117     CASE(5,6) ! 3HeD
00118        IF(ene <= 900) THEN
00119           bg = 68.7508
00120           a = (/ 5.7501e6, 2.5226e3, 4.5566e1, 0.0, 0.0 /)
00121           b = (/ -3.1995e-3, -8.5530e-6, 5.9014e-8, 0.0 /)
00122        ELSE
00123           bg = 68.7508
00124           a = (/ -8.3993e5, 0.0, 0.0, 0.0, 0.0 /)
00125           b = (/ -2.6830e-3, 1.1633e-6, -2.1332e-10, 1.4250e-14 /)
00126        ENDIF
00127        erange = (/ 0.3, 4800.0 /)
00128 #ifdef IGNORE_LOW_E_LIM
00129        erange(1) = 0.0
00130 #endif
00131     CASE DEFAULT
00132        write(*,*) 'ASSERTION FAILED SVNROOT'
00133        write(*,*)  'reaction:',reactionsave,'Energy', ene
00134        write(*,*) 'FILE: "'//__file__//'" line:',__line__
00135        write(*,*) "Unknown fusion reaction!"
00136        stop 40
00137     END SELECT
00138
00139     !      IF(ENE < ERANGE(1) .OR. ENE > ERANGE(2)) THEN
00140     !         WRITE(*,*)"Energy", ENE, "outside the validity range (",  &
00141     !              & ERANGE(1),"...", ERANGE(2),") of the used fusion cross-section ", &
00142     !              "model encountered!"
00143     !         write(*,*) 'ASSERTION FAILED SVNROOT'
00144     !         write(*,*) 'FILE: "'//__FILE__//'" line:',__LINE__
00145     !         stop 45
00146     !      ENDIF
```

```
00147
00148     ! Clamp the energy for evaluating the Astrophysical S-factor to the maximum.
00149     atene=ene
00150     IF( ene > erange(2)) atene=erange(2)
00151
00152
00153     !     CALCULATE THE VALUE OF S FROM THE PADE POLYNOMIAL
00154     s = ( a(1) + atene*(a(2) + atene*(a(3) &
00155          & + atene*( a(4)+atene*a(5) ) ) ) ) &
00156          & /( 1 + atene*(b(1) + atene*(b(2) &
00157          & + atene*( b(3)+atene*b(4) ) ) ) )
00158
00159     ! There tends to happen numerical underflow, so let's catch it!
00160 #ifdef IGNORE_LOW_E_LIM
00161     if(bg/sqrt(atene)>710.0_params_wp )then
00162        !write(*,*) 'BG/SQRT(atENE)>710.0 => SIGMA=0.0 BG/SQRT(atENE)=',BG/SQRT(atENE)
00163        sigma = 0.0_params_wp
00164        RETURN
00165     end if
00166 #endif
00167
00168     !     CALCULATE SIGMA FROM S, energy, AND BG
00169     !     (using the actual energy, not the clamped one)
00170     !
00171     sigma = s / (ene * exp(bg/sqrt(ene))) * tom2
00172     !      OPEN(123,file='sigma.txt')
00173     !      WRITE(123,*) atENE, SIGMAX
00174     ! CLOSE(123)
00175
```

Here is the caller graph for this function:



14.24 call_utils Module Reference

utils module from Silvio Gori's grid package

Functions/Subroutines

- subroutine, public assert (lcond, message)
- subroutine, public warning (lcond, message)
- subroutine, public exiting (lcond, message)
- subroutine, public exitall ()
- subroutine, public sub_init (subin)
- subroutine, public sub_end ()

14.24.1 Detailed Description

utils module from Silvio Gori's grid package

Author

Silvio Gori

14.24.2 Function/Subroutine Documentation

14.24.2.1 assert()

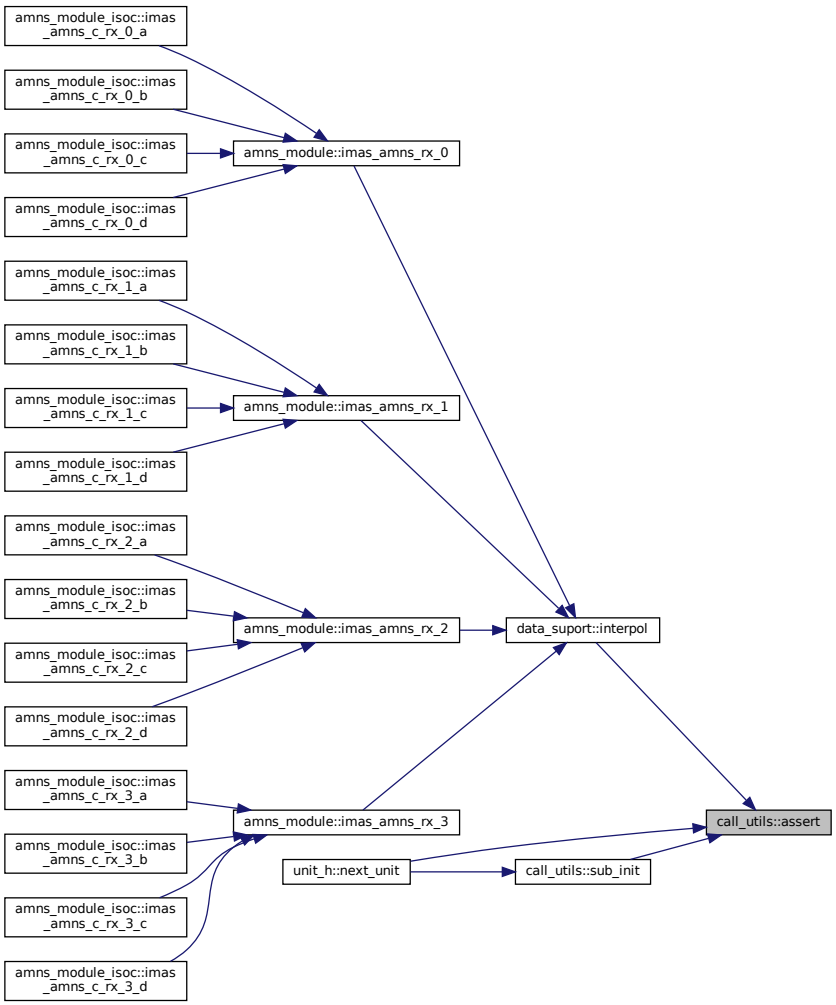
```
subroutine, public call_utils::assert (
              logical, intent(in) lcond,
              character(len=*), intent(in) message )
```

Definition at line 27 of file call_utils.f90.

```
00028
00029     logical, intent(in)          :: lcond
00030     character(len=*), intent(in) :: message
00031
00032     if(.not.lcond)then
00033        print*, message
00034        call exitall()
00035     endif
```

Here is the call graph for this function:

Here is the caller graph for this function:



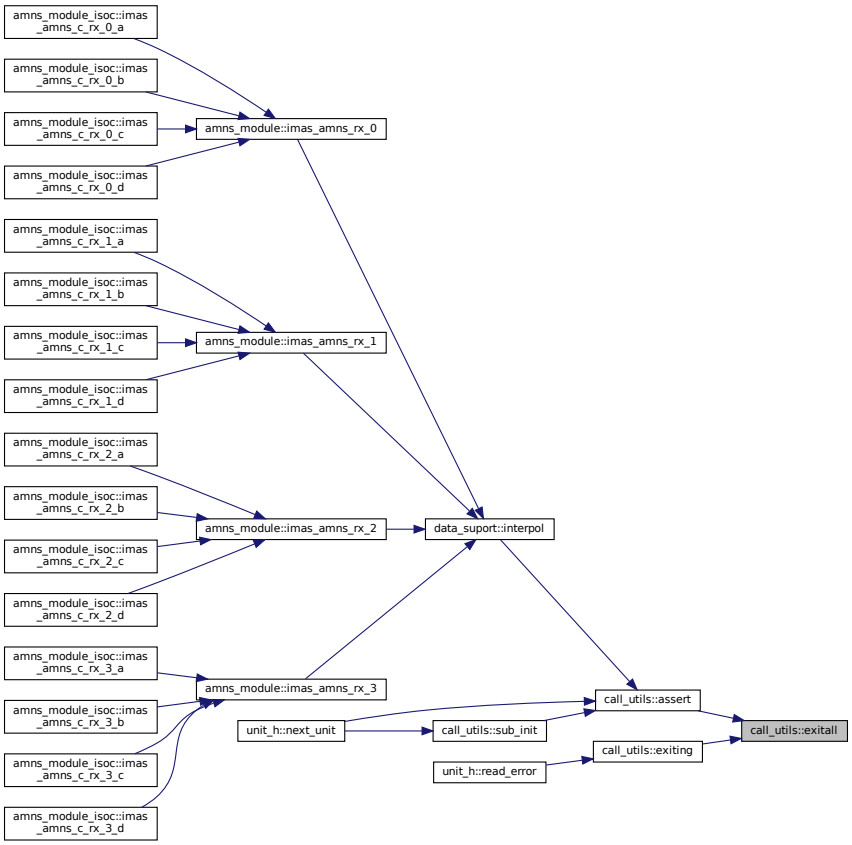
14.24.2.2 exitall()

```
subroutine, public call_utils::exitall
```

Definition at line 70 of file call_utils.f90.

```
00071
00072     integer :: i
00073
00074     do i=sub_called_pos,1,-1
00075        print*, trim(sub_called(sub_called_pos))
00076     enddo
00077     stop 'Error - exitall@utils: at eof'
00078
```

Here is the caller graph for this function:



14.24.2.3 exiting()

```
subroutine, public call_utils::exiting (
             logical, intent(in) lcond,
             character(len=*), intent(in) message )
```

Definition at line 54 of file call_utils.f90.

```
00055
00056     logical, intent(in)          :: lcond
00057     character(len=*), intent(in) :: message
00058
00059     if(.not.lcond)then
00060        print*,"Panik !: "// message
00061        call exitall()
00062     endif
```

Here is the call graph for this function:

Here is the caller graph for this function:

14.24.2.4 sub_end()

```
subroutine, public call_utils::sub_end
```

Definition at line 95 of file call_utils.f90.

```
00096
00097     ! print *,' ---> ',sub_called(sub_called_pos)
00098     sub_called_pos=sub_called_pos-1
00099
00100
```

Here is the caller graph for this function:

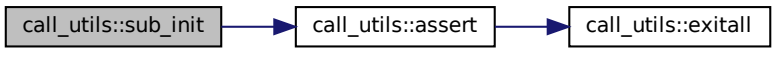
14.24.2.5 sub_init()

```
subroutine, public call_utils::sub_init (
             character(len=*), intent(in) subin )
```

Definition at line 84 of file call_utils.f90.

```
00085     character(len=*), intent(in) :: subin
00086     ! print *,' +++> ',subin    , sub_called_pos, max_call_deep
00087     call assert(sub_called_pos<max_call_deep,'sub_init error sub_called_pos>=max_call_deep')
00088     sub_called_pos=sub_called_pos+1
00089     sub_called(sub_called_pos)=trim(subin)
```

Here is the call graph for this function:



Here is the caller graph for this function:

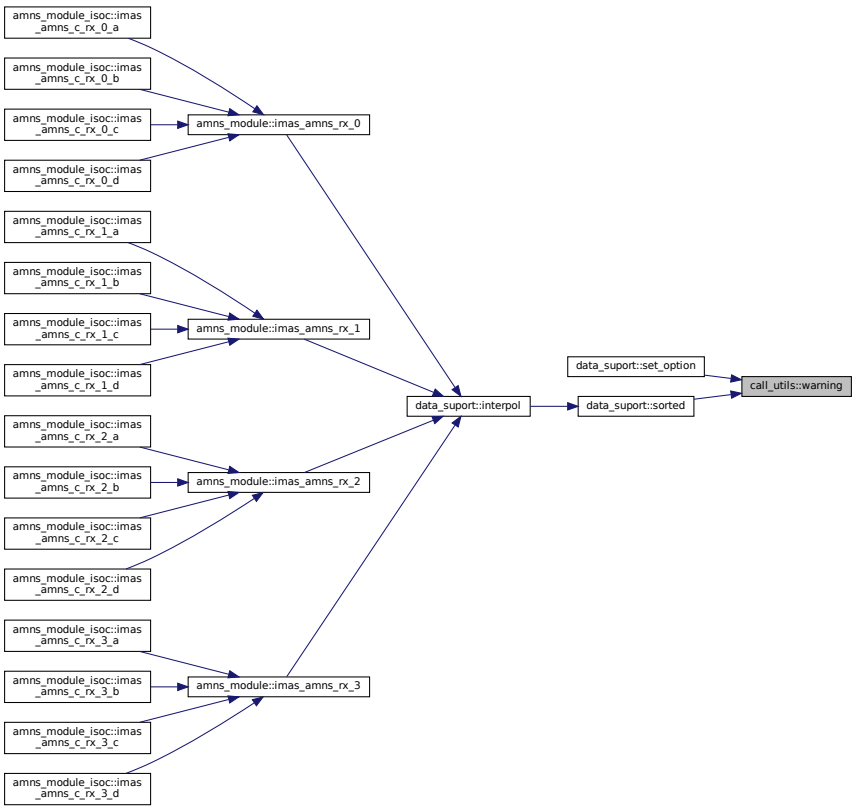
14.24.2.6 warning()

```
subroutine, public call_utils::warning (
             logical, intent(in) lcond,
             character(len=*), intent(in) message )
```

Definition at line 42 of file call_utils.f90.

```
00043
00044     logical, intent(in)          :: lcond
00045     character(len=*), intent(in) :: message
00046     if(.not.lcond)then
00047        print*,"warning!: "// message
00048     endif
```

Here is the caller graph for this function:



14.25 camns_interface Namespace Reference

14.26 coronal Namespace Reference

Functions

- def te_ne (te, ne)
- def TDMASolve (a, b, c, d)
- def point (ei, rc)
- def distribution (rates, te, ne)
- def rates (ZN)

14.26.1 Detailed Description

```
Tools to work with the coronal distribution
David.Coster@ipp.mpg.de
```

14.26.2 Function Documentation

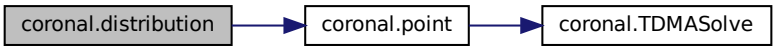
14.26.2.1 distribution()

```
def coronal.distribution (
              rates,
              te,
              ne )

inputs: rates dictionary, te and ne
outputs: a dictionary containing:
  the coronal fractional densities
  average charge
  line radiation efficiency
  recombination (including Bremsstrahlung) radiation efficiency
```

Definition at line 64 of file coronal.py.

```
00064 def distribution (rates, te, ne):
00065     '''
00066     inputs: rates dictionary, te and ne
00067     outputs: a dictionary containing:
00068       the coronal fractional densities
00069       average charge
00070       line radiation efficiency
00071       recombination (including Bremsstrahlung) radiation efficiency
00072     '''
00073
00074     n = len(rates['EI'])
00075     nte = te.shape[0]
00076     nne = ne.shape[1]
00077
00078     ei = np.zeros([n, nte, nne])
00079     rc = np.zeros([n, nte, nne])
00080     br = np.zeros([n, nte, nne])
00081     lr = np.zeros([n, nte, nne])
00082     na = np.zeros([n, nte, nne])
00083
00084     for i in range(n):
00085         ei[i,:,:] = rates['EI'][i].data(te,ne)
00086         rc[i,:,:] = rates['RC'][i].data(te,ne)
00087         br[i,:,:] = rates['BR'][i].data(te,ne)
00088         lr[i,:,:] = rates['LR'][i].data(te,ne)
00089
00090     for iit in range(nte):
00091         for iin in range(nne):
00092             na[:, iit, iin] = point(ei[:, iit, iin], rc[:, iit, iin])
00093
00094     ZA=(np.arange(n).repeat(nte).repeat(nne)).reshape([n,nte,nne])
00095
00096     Z=(na[:,:,:]*ZA).sum(axis=0)/na[:,:,:].sum(axis=0)
00097
00098     LR_rad = (na*lr).sum(axis=0)*ne
00099     BR_rad = (na*br).sum(axis=0)*ne
00100
00101     return dict(na=na, Z=Z, LR_rad=LR_rad, BR_rad=BR_rad)
00102
```

Here is the call graph for this function:



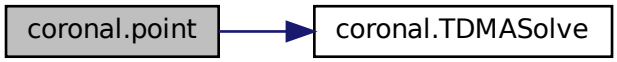
14.26.2.2 point()

```
def coronal.point (
              ei,
              rc )

 return the coronal fractional densities determined by the balance of ionization and recombination
```

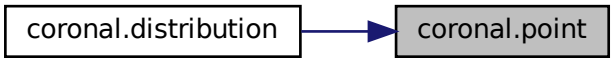
Definition at line 26 of file coronal.py.

```
00026 def point(ei, rc):
00027     "' return the coronal fractional densities determined by the balance of ionization and
       recombination "'
00028     n=len(ei)
00029
00030     l = np.zeros(n)
00031     d = np.zeros(n)
00032     u = np.zeros(n)
00033     na = np.zeros(n)
00034
00035     l[0] = 0.0
00036     d[0] = -ei[0]
00037     u[0] = rc[1]
00038     na[0] = 0.0
00039
00040     l[1:-1] = ei[0:-2]
00041     d[1:-1] = -ei[1:-1] - rc[1:-1]
00042     u[1:-1] = rc[2:]
00043     na[1:-1] = 0.0
00044
00045     l[-1] = ei[-2]
00046     d[-1] = -rc[-1]
00047     u[-1] = 0.0
00048     na[-1] = 0.0
00049
00050     try:
00051         isref = list(ei[:] < rc[:]).index(True)
00052     except:
00053         isref = len(ei[:])-1
00054     l[isref] = 0.0
00055     d[isref] = 1.0
00056     u[isref] = 0.0
00057     na[isref] = 1.0
00058
00059     na = TDMASolve(l,d,u,na)
00060     na = na / na.sum()
00061
00062     return na
00063
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.26.2.3 rates()

```
def coronal.rates (
              ZN )

 return the rates dictionary for the specified nuclear charge
```

Definition at line 103 of file coronal.py.

```
00103 def rates(ZN):
00104     "' return the rates dictionary for the specified nuclear charge "'
00105     reactantsRC=[]
00106     reactantsEI=[]
00107     reactantsRD=[]
00108     for i in range(ZN+1):
00109         reactantsRC.append(amns.Reactants())
00110         reactantsRC[i].add(ZN,i,0)
00111         reactantsRC[i].add(0,-1,0)
00112         reactantsRC[i].add(ZN,i-1,0,lr=1)
00113         reactantsRC[i].add(0,-1,0,lr=1)
00114         reactantsEI.append(amns.Reactants())
00115         reactantsEI[i].add(ZN,i,0)
00116         reactantsEI[i].add(0,-1,0)
00117         reactantsEI[i].add(ZN,i+1,0,lr=1)
00118         reactantsEI[i].add(0,-1,0,lr=1)
00119         reactantsRD.append(amns.Reactants())
00120         reactantsRD[i].add(ZN,i,0)
00121         reactantsRD[i].add(ZN,i,0,lr=1)
00122
00123     amnsdb = amns.Amns()
00124
00125     print('Setting up tables ...')
00126
00127     vals=dict(SP=[], EI=[], RC=[], BR=[], LR=[])
00128     for i in range(ZN+1):
00129         vals['EI'].append(amnsdb.get_table("EI", reactantsEI[i]))
00130         vals['RC'].append(amnsdb.get_table("RC", reactantsRC[i]))
00131         vals['BR'].append(amnsdb.get_table("BR", reactantsRD[i]))
00132         vals['LR'].append(amnsdb.get_table("LR", reactantsRD[i]))
00133         vals['SP'].append(vals['EI'][i].state_label)
00134
00135     return vals
```

14.26.2.4 TDMASolve()

```
def coronal.TDMASolve (
              a,
              b,
              c,
              d )

 Solve the tridiagonal system
```

Definition at line 14 of file coronal.py.

```
00014 def TDMASolve(a, b, c, d):
00015     "' Solve the tridiagonal system "'
00016     n = len(d) # n is the numbers of rows, a and c has length n-1
00017     bl = b.copy()
```

```
00018     dl = d.copy()
00019     for i in range(n-1):
00020         dl[i+1] -= dl[i] * a[i+1] / bl[i]
00021         bl[i+1] -= c[i] * a[i+1] / bl[i]
00022     for i in reversed(range(n-1)):
00023         dl[i] -= dl[i+1] * c[i] / bl[i+1]
00024     return dl / bl # return the solution
00025
```

Here is the caller graph for this function:

14.26.2.5 te_ne()

```
def coronal.te_ne (
              te,
              ne )

 Given 1d arrays of te and ne, return the appropriate outer product arrays of TE and NE
```

Definition at line 8 of file coronal.py.

```
00008 def te_ne (te, ne):
00009     "' Given 1d arrays of te and ne, return the appropriate outer product arrays of TE and NE "'
00010     TE=np.array(te.repeat(len(ne)).reshape([len(te), len(ne)]),order='F')
00011     NE=np.array(ne.repeat(len(te)).reshape([len(ne), len(te)]).transpose(),order='F')
00012     return TE, NE
00013
```

14.27 coronal_charge_state_edge Namespace Reference

Variables

- te
- ne
- rates = coronal.rates(Z)
- dist = coronal.distribution(rates, te, ne)
- label
- loc

14.27.1 Detailed Description

```
Calculate coronal charge state in edge plasma conditions
David.Coster@ipp.mpg.de
```

14.27.2 Variable Documentation

14.27.2.1 dist

```
coronal_charge_state_edge.dist = coronal.distribution(rates, te, ne)
```

Definition at line 17 of file coronal_charge_state_edge.py.

14.27.2.2 label

`coronal_charge_state_edge.label`
Definition at line 18 of file coronal_charge_state_edge.py.

14.27.2.3 loc

`coronal_charge_state_edge.loc`
Definition at line 21 of file coronal_charge_state_edge.py.

14.27.2.4 ne

`coronal_charge_state_edge.ne`
Definition at line 10 of file coronal_charge_state_edge.py.

14.27.2.5 rates

`coronal_charge_state_edge.rates = coronal.rates(Z)`
Definition at line 16 of file coronal_charge_state_edge.py.

14.27.2.6 te

`coronal_charge_state_edge.te`
Definition at line 10 of file coronal_charge_state_edge.py.

14.28 coronal_comparison_N+Ne Namespace Reference

Variables

- te
- ne
- rates_N = coronal.rates(7)
- rates_Ne = coronal.rates(10)
- dist_N = coronal.distribution(rates_N, te, ne)
- dist_Ne = coronal.distribution(rates_Ne, te, ne)
- NUM_COLORS = np.maximum(dist_N['na'].shape[0], dist_Ne['na'].shape[0])
- cm = plt.get_cmap('gist_rainbow')
- color
- label
- loc
- ncol

14.28.1 Variable Documentation

14.28.1.1 cm

`coronal_comparison_N+Ne.cm = plt.get_cmap('gist_rainbow')`
Definition at line 17 of file coronal_comparison_N+Ne.py.

14.28.1.2 color

coronal_comparison_N+Ne.color
Definition at line 19 of file coronal_comparison_N+Ne.py.

14.28.1.3 dist_N

```
coronal_comparison_N+Ne.dist_N = coronal.distribution(rates_N, te, ne)
```
Definition at line 11 of file coronal_comparison_N+Ne.py.

14.28.1.4 dist_Ne

```
coronal_comparison_N+Ne.dist_Ne = coronal.distribution(rates_Ne, te, ne)
```
Definition at line 12 of file coronal_comparison_N+Ne.py.

14.28.1.5 label

coronal_comparison_N+Ne.label
Definition at line 19 of file coronal_comparison_N+Ne.py.

14.28.1.6 loc

coronal_comparison_N+Ne.loc
Definition at line 25 of file coronal_comparison_N+Ne.py.

14.28.1.7 ncol

coronal_comparison_N+Ne.ncol
Definition at line 25 of file coronal_comparison_N+Ne.py.

14.28.1.8 ne

coronal_comparison_N+Ne.ne
Definition at line 6 of file coronal_comparison_N+Ne.py.

14.28.1.9 NUM_COLORS

```
coronal_comparison_N+Ne.NUM_COLORS = np.maximum(dist_N['na'].shape[0], dist_Ne['na'].shape[0])
```
Definition at line 16 of file coronal_comparison_N+Ne.py.

14.28.1.10 rates_N

```
coronal_comparison_N+Ne.rates_N = coronal.rates(7)
```
Definition at line 8 of file coronal_comparison_N+Ne.py.

14.28.1.11 rates_Ne

```
coronal_comparison_N+Ne.rates_Ne = coronal.rates(10)
```
Definition at line 9 of file coronal_comparison_N+Ne.py.

14.28.1.12 te

```
coronal_comparison_N+Ne.te
```
Definition at line 6 of file coronal_comparison_N+Ne.py.

14.29 coronal_info Namespace Reference

Variables

- parser
- type
- int
- help
- default
- args = parser.parse_args()
- rates = coronal.rates(args.nuclear_charge)
- te
- ne
- dist = coronal.distribution(rates, te, ne)
- header
- loc
- labels
- lw
- fontsize
- ncol

14.29.1 Detailed Description

```
Calculate coronal data
David.Coster@ipp.mpg.de
```

14.29.2 Variable Documentation

14.29.2.1 args

```
coronal_info.args = parser.parse_args()
```
Definition at line 16 of file coronal_info.py.

14.29.2.2 default

```
coronal_info.default
```
Definition at line 15 of file coronal_info.py.

14.29.2.3 dist

```
coronal_info.dist = coronal.distribution(rates, te, ne)
```
Definition at line 22 of file coronal_info.py.

14.29.2.4 fontsize

```
coronal_info.fontsize
```
Definition at line 56 of file coronal_info.py.

14.29.2.5 header

coronal_info.header
Definition at line 25 of file coronal_info.py.

14.29.2.6 help

coronal_info.help
Definition at line 15 of file coronal_info.py.

14.29.2.7 int

coronal_info.int
Definition at line 15 of file coronal_info.py.

14.29.2.8 labels

coronal_info.labels
Definition at line 34 of file coronal_info.py.

14.29.2.9 loc

coronal_info.loc
Definition at line 34 of file coronal_info.py.

14.29.2.10 lw

coronal_info.lw
Definition at line 40 of file coronal_info.py.

14.29.2.11 ncol

coronal_info.ncol
Definition at line 56 of file coronal_info.py.

14.29.2.12 ne

coronal_info.ne
Definition at line 20 of file coronal_info.py.

14.29.2.13 parser

coronal_info.parser
Initial value:
```
00001 =  argparse.ArgumentParser(description="Provide coronal information", epilog
00002 =, formatter_class=argparse.RawTextHelpFormatter)
```
Definition at line 12 of file coronal_info.py.

14.29.2.14 rates

```
coronal_info.rates = coronal.rates(args.nuclear_charge)
```
Definition at line 18 of file coronal_info.py.

14.29.2.15 te

```
coronal_info.te
```
Definition at line 20 of file coronal_info.py.

14.29.2.16 type

```
coronal_info.type
```
Definition at line 15 of file coronal_info.py.

14.30 coronal_radiation_efficiency Namespace Reference

Variables

- list L = [1,2,3,4,6,7,10,18,36,54,74]
- list rates = [ ]
- te
- ne
- list dist = [ ]
- label
- lw
- loc

14.30.1 Detailed Description

```
Calculate the coronal radiation efficiencies
David.Coster@ipp.mpg.de
```

14.30.2 Variable Documentation

14.30.2.1 dist

```
list coronal_radiation_efficiency.dist = [ ]
```
Definition at line 21 of file coronal_radiation_efficiency.py.

14.30.2.2 L

```
list coronal_radiation_efficiency.L = [1,2,3,4,6,7,10,18,36,54,74]
```
Definition at line 12 of file coronal_radiation_efficiency.py.

14.30.2.3 label

```
coronal_radiation_efficiency.label
```
Definition at line 31 of file coronal_radiation_efficiency.py.

14.30.2.4 loc

`coronal_radiation_efficiency.loc`
Definition at line 37 of file coronal_radiation_efficiency.py.

14.30.2.5 lw

`coronal_radiation_efficiency.lw`
Definition at line 31 of file coronal_radiation_efficiency.py.

14.30.2.6 ne

`coronal_radiation_efficiency.ne`
Definition at line 19 of file coronal_radiation_efficiency.py.

14.30.2.7 rates

`list coronal_radiation_efficiency.rates = []`
Definition at line 14 of file coronal_radiation_efficiency.py.

14.30.2.8 te

`coronal_radiation_efficiency.te`
Definition at line 19 of file coronal_radiation_efficiency.py.

14.31 coronal_version_comparison Namespace Reference

Variables

- parser
- type
- int
- help
- default
- str
- args = parser.parse_args()
- int nte = 141
- int nne = 121
- te
- ne
- ZN = args.nuclear_charge
- list reactantsRC = [ ]
- list reactantsEI = [ ]
- list reactantsRD = [ ]
- lr
- amnsdb_0 = amns.Amns(version_user=args.user, version_number=args.version_1)
- amnsdb_1 = amns.Amns(version_user=args.user, version_number=args.version_2)
- list SP = [ ]
- list EI_0 = [ ]
- list EI_1 = [ ]
- dist_0 = coronal.distribution(dict(EI=EI_0, RC=RC_0, BR=BR_0, LR=LR_0), te, ne)
- dist_1 = coronal.distribution(dict(EI=EI_1, RC=RC_1, BR=BR_1, LR=LR_1), te, ne)
- NUM_COLORS = dist_0['na'].shape[0]

- cm = plt.get_cmap('gist_rainbow')
- color
- label
- loc
- norm

14.31.1 Variable Documentation

14.31.1.1 amnsdb_0

```
coronal_version_comparison.amnsdb_0 = amns.Amns(version_user=args.user, version_number=args.↩
version_1)
```
Definition at line 46 of file coronal_version_comparison.py.

14.31.1.2 amnsdb_1

```
coronal_version_comparison.amnsdb_1 = amns.Amns(version_user=args.user, version_number=args.↩
version_2)
```
Definition at line 47 of file coronal_version_comparison.py.

14.31.1.3 args

```
coronal_version_comparison.args = parser.parse_args()
```
Definition at line 19 of file coronal_version_comparison.py.

14.31.1.4 cm

```
coronal_version_comparison.cm = plt.get_cmap('gist_rainbow')
```
Definition at line 70 of file coronal_version_comparison.py.

14.31.1.5 color

```
coronal_version_comparison.color
```
Definition at line 75 of file coronal_version_comparison.py.

14.31.1.6 default

```
coronal_version_comparison.default
```
Definition at line 15 of file coronal_version_comparison.py.

14.31.1.7 dist_0

```
coronal_version_comparison.dist_0 = coronal.distribution(dict(EI=EI_0, RC=RC_0, BR=BR_0, LR=L↩
R_0), te, ne)
```
Definition at line 66 of file coronal_version_comparison.py.

14.31.1.8 dist_1

```
coronal_version_comparison.dist_1 = coronal.distribution(dict(EI=EI_1, RC=RC_1, BR=BR_1, LR=L↩
R_1), te, ne)
```
Definition at line 67 of file coronal_version_comparison.py.

14.31.1.9 EI_0

```
list coronal_version_comparison.EI_0 = []
```
Definition at line 52 of file coronal_version_comparison.py.

14.31.1.10 EI_1

```
list coronal_version_comparison.EI_1 = []
```
Definition at line 53 of file coronal_version_comparison.py.

14.31.1.11 help

```
coronal_version_comparison.help
```
Definition at line 15 of file coronal_version_comparison.py.

14.31.1.12 int

```
coronal_version_comparison.int
```
Definition at line 15 of file coronal_version_comparison.py.

14.31.1.13 label

```
coronal_version_comparison.label
```
Definition at line 75 of file coronal_version_comparison.py.

14.31.1.14 loc

```
coronal_version_comparison.loc
```
Definition at line 80 of file coronal_version_comparison.py.

14.31.1.15 lr

```
coronal_version_comparison.lr
```
Definition at line 35 of file coronal_version_comparison.py.

14.31.1.16 ne

```
coronal_version_comparison.ne
```
Definition at line 23 of file coronal_version_comparison.py.

14.31.1.17 nne

```
int coronal_version_comparison.nne = 121
```
Definition at line 22 of file coronal_version_comparison.py.

14.31.1.18 norm

```
coronal_version_comparison.norm
```

Definition at line 150 of file coronal_version_comparison.py.

14.31.1.19 nte

```
int coronal_version_comparison.nte = 141
```

Definition at line 21 of file coronal_version_comparison.py.

14.31.1.20 NUM_COLORS

```
coronal_version_comparison.NUM_COLORS = dist_0['na'].shape[0]
```

Definition at line 69 of file coronal_version_comparison.py.

14.31.1.21 parser

```
coronal_version_comparison.parser
```

Initial value:

```
00001 =  argparse.ArgumentParser(description="Provide a comparison of the coronal results for two different
      versions of the AMNS data", epilog
00002 =, formatter_class=argparse.RawTextHelpFormatter)
```

Definition at line 12 of file coronal_version_comparison.py.

14.31.1.22 reactantsEI

```
list coronal_version_comparison.reactantsEI = []
```

Definition at line 29 of file coronal_version_comparison.py.

14.31.1.23 reactantsRC

```
list coronal_version_comparison.reactantsRC = []
```

Definition at line 28 of file coronal_version_comparison.py.

14.31.1.24 reactantsRD

```
list coronal_version_comparison.reactantsRD = []
```

Definition at line 30 of file coronal_version_comparison.py.

14.31.1.25 SP

```
list coronal_version_comparison.SP = []
```

Definition at line 51 of file coronal_version_comparison.py.

14.31.1.26 str

```
coronal_version_comparison.str
```

Definition at line 18 of file coronal_version_comparison.py.

14.31.1.27 te

coronal_version_comparison.te
Definition at line 23 of file coronal_version_comparison.py.

14.31.1.28 type

coronal_version_comparison.type
Definition at line 15 of file coronal_version_comparison.py.

14.31.1.29 ZN

coronal_version_comparison.ZN = args.nuclear_charge
Definition at line 25 of file coronal_version_comparison.py.

14.32 data_suport Module Reference

Functions/Subroutines

- subroutine, public delete (grid)

  deallocate a grid
- subroutine, public set_option (grid, warning)

  set the "with_warning" flag in grid to the value of "warning"
- subroutine, public interpol (w, x, y, z, fd1, fd2, fd3, fd4, grid, data_error)

  interpolate in the grid
- logical function sorted (x)

  return true if the passed array is sorted

14.32.1 Function/Subroutine Documentation

14.32.1.1 delete()

```
subroutine, public data_suport::delete (
             type(grid_t), intent(inout) grid )
```

deallocate a grid
Definition at line 289 of file data_suport.f90.

```
00290     implicit none
00291     type(grid_t),intent(inout):: grid
00292
00293     if(allocated(grid%f1d))then
00294        deallocate(grid%f1d)
00295        grid%exist_f1d=.false.
00296     endif
00297     if(allocated(grid%f2d))then
00298        deallocate(grid%f2d)
00299        grid%exist_f2d=.false.
00300     endif
00301     if(allocated(grid%f3d))then
00302        deallocate(grid%f3d)
00303        grid%exist_f3d=.false.
00304     endif
00305     if(allocated(grid%f4d))then
00306        deallocate(grid%f4d)
00307        grid%exist_f4d=.false.
00308     endif
00309     if(allocated(grid%axe)) then
00310        deallocate(grid%axe)
00311     endif
```

14.32.1.2 interpol()

```
subroutine, public data_suport::interpol (
             real (rkind), dimension(:), intent(in), target w,
             real (rkind), dimension(:), intent(in), optional, target x,
             real (rkind), dimension(:), intent(in), optional, target y,
             real (rkind), dimension(:), intent(in), optional, target z,
             real (rkind), dimension(:), intent(out), optional fd1,
             real (rkind), dimension(:,:), intent(out), optional fd2,
             real (rkind), dimension(:,:,:), intent(out), optional fd3,
             real (rkind), dimension(:,:,:,:), intent(out), optional fd4,
             type (grid_t), intent(inout) grid,
             type (data_error_t), intent(out) data_error )
```

interpolate in the grid
you can use this routine till the fourth dimension Returning eather a single dim. vect.(fd1), if it is present or a block of function values (fd2, fd3 or fd4). The interpolation routines are writen explicitly In principal it is posible to formulate the interpolation using only the 1D interpolation, if the grid were defined slightly different, (with only one support vector (4-D)) but this is not the case here. The routine sort first the position vectors (w,x,y,z) The it search for the left location of their position in the grid axes. Then the interpolation routines are called. table interpolation
nuclear data: function of 1 parameter
nuclear data: function of 1 parameter
resonant charge transfer data: function of 1 parameter
sputter yield fit formula: function of 2 parameters
reflection yield fit formula: function of 2 parameters
nuclear data: function of 1 parameter
Definition at line 341 of file data_suport.f90.

```
00342     use amns_external_functions
00343     implicit none
00344     real (rkind), target           ,intent(in) :: w(:)
00345     real (rkind), target, optional ,intent(in) :: x(:)
00346     real (rkind), target, optional ,intent(in) :: y(:)
00347     real (rkind), target, optional ,intent(in) :: z(:)
00348     real (rkind),         optional ,intent(out):: fd1(:)
00349     real (rkind),         optional ,intent(out):: fd2(:,:)
00350     real (rkind),         optional ,intent(out):: fd3(:,:,:)
00351     real (rkind),         optional ,intent(out):: fd4(:,:,:,:)
00352     type (grid_t)                  ,intent(inout):: grid
00353     type (data_error_t),              intent(out):: data_error
00354
00355     ! interpolation index
00356     !
00357     real (rkind) :: dw(size(w))
00358     integer      :: iw(size(w))
00359     integer      :: iw_sort(size(w))
00360     real (rkind), target :: w_s(size(w))
00361     real (rkind), pointer :: w_p(:)
00362
00363     real (rkind), allocatable         :: dx(:)  !size(x))
00364     integer , allocatable             :: ix(:)  !size(x))
00365     integer , allocatable             :: ix_sort(:)  !size(x))
00366     real (rkind), allocatable, target :: x_s(:)  !size(x))
00367     real (rkind), pointer             :: x_p(:)
00368
00369     real (rkind), allocatable         :: dy(:)  !size(yvec))
00370     integer , allocatable             :: iy(:)  !size(yvec))
00371     integer, allocatable              :: iy_sort(:)  !size(yvec))
00372     real (rkind), allocatable, target :: y_s(:)  !size(yvec))
00373     real (rkind), pointer             :: y_p(:)
00374
00375     real (rkind), allocatable         :: dz(:)  !size(zvec))
00376     integer  , allocatable            :: iz(:)  !size(zvec))
00377     integer , allocatable             :: iz_sort(:)  !size(zvec))
00378     real (rkind), allocatable, target :: z_s(:)  !size(zvec))
00379     real (rkind), pointer             :: z_p(:)
00380
00381     !local
00382     logical  :: with_x      ! =present(x)
00383     logical  :: with_y      ! =present(y)
00384     logical  :: with_z      ! =present(z)
00385     logical  :: one_d       ! =present(fd1)
00386     logical  :: two_d       ! =present(fd2)
00387     logical  :: three_d     ! =present(fd3)
00388     logical  :: four_d      ! =present(fd4)
00389     logical  :: is_sorted_w = .false. ! =present(fgrid3)
00390     logical  :: is_sorted_x = .false. ! =present(fgrid3)
```

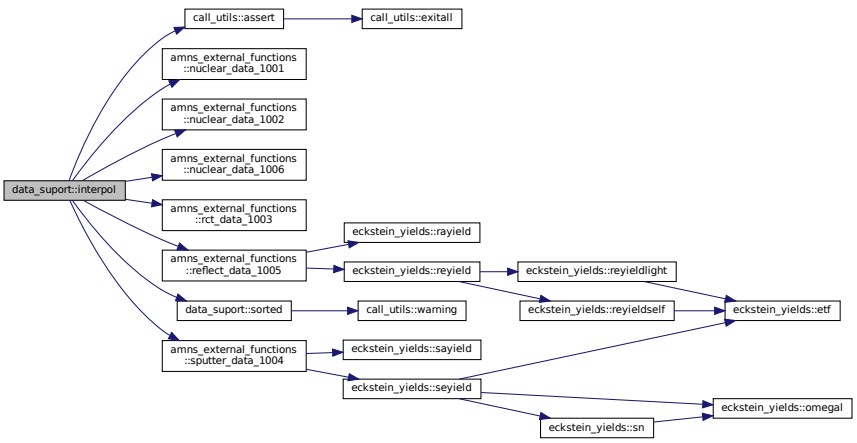
```
00391     logical  :: is_sorted_y = .false. ! =present(fgrid3)
00392     logical  :: is_sorted_z = .false. ! =present(fgrid3)
00393
00394     ! help swap
00395     integer :: iswap
00396     !loop index
00397     integer :: i, idpc
00398
00399     type (fun_err_t) :: fun_err
00400
00401     data_error%ierr = 0
00402     data_error%cerr = ”
00403     fun_err%ierr = 0
00404     fun_err%cerr = ”
00405
00406     with_x   =present(x)
00407     with_y   =present(y)
00408     with_z   =present(z)
00409     one_d    =present(fd1)
00410     two_d    =present(fd2)
00411     three_d  =present(fd3)
00412     four_d   =present(fd4)
00413
00414     !-irregular calls---
00415     call assert(grid%exist_f1d .or. grid%exist_f2d .or.grid%exist_f3d.or.grid%exist_f4d, &
00416          "Data not allocated for interpolation")
00417
00418     if(one_d) then
00419        call assert(one_d.and. .not.(two_d.or.three_d.or.four_d), &
00420             "Type mismatch, only one f vector type possible!!")
00421        if(with_x) call assert(size(x)==size(w), "Size w:"//size(w)// &
00422             "/= Size x:"//size(x))
00423        if(with_y) call assert(size(y)==size(w), "Size w:"//size(w)// &
00424             "/= Size y:"//size(y))
00425        if(with_z) call assert(size(z)==size(w), "Size w:"//size(x)// &
00426             "/= Size z:"//size(z))
00427     endif
00428
00429     if(two_d)then
00430        call assert(with_x.and..not.(with_y.or.with_z),"expect x present, y and z not present")
00431     endif
00432     if(three_d)then
00433        call assert(with_y .and. .not. with_z,"expect x and y present and z not present")
00434     endif
00435     if(four_d)then
00436        call assert(with_y .and. with_z,"expect x y and zvec present")
00437     endif
00438
00439     select case(grid%interpol_function)
00440
00441     case (0)
00443        if(with_x)then
00444           allocate(dx(size(x)),ix(size(x)),ix_sort(size(x)),x_s(size(x)))
00445        endif
00446        if(with_y)then
00447           allocate(dy(size(y)),iy(size(y)),iy_sort(size(y)),y_s(size(y)))
00448        endif
00449        if(with_z)then
00450           allocate(dz(size(z)),iz(size(z)),iz_sort(size(z)),z_s(size(z)))
00451        endif
00452
00453        is_sorted_w=sorted(w(:))
00454        if(with_x)is_sorted_x=sorted(x(:))
00455        if(with_y)is_sorted_y=sorted(y(:))
00456        if(with_z)is_sorted_z=sorted(z(:))
00457
00458     ! if the vectors are not sorted, we create a sorted vector, and save index map
00459     ! pointer x_p will show a sorted vector
00460
00461        if(.not.is_sorted_w) then
00462           iw_sort(:)= (/ (i,i=1,size(iw_sort(:))   ) /)
00463           call mrgrnk(w(:),iw_sort(:))
00464           w_s(:)=w(iw_sort(:))
00465           w_p => w_s
00466           if(grid%axe(1)%is_log) w_p = log10(w_p)
00467        else
00468           if(grid%axe(1)%is_log) then
00469              w_s = log10(w)
00470              w_p => w_s
00471           else
00472              w_p => w
00473           endif
00474        endif
00475
00476        if(with_x) then
00477           if(.not.is_sorted_x) then
00478              ix_sort(:)= (/ (i,i=1,size(ix_sort(:))   ) /)
```

```
00479             call mrgrnk(x(:),ix_sort(:))
00480             x_s(:)=x(ix_sort(:))
00481             x_p => x_s
00482             if(grid%axe(2)%is_log) x_p = log10(x_p)
00483          else
00484             if(grid%axe(2)%is_log) then
00485                x_s = log10(x)
00486                x_p => x_s
00487             else
00488                x_p => x
00489             endif
00490          endif
00491       endif
00492
00493       if(with_y) then
00494          if(.not.is_sorted_y)then
00495             iy_sort(:)= (/ (i,i=1,size(iy_sort(:))   ) /)
00496             call mrgrnk(y(:),iy_sort(:))
00497             y_s(:)=y(iy_sort(:))
00498             y_p => y_s
00499             if(grid%axe(3)%is_log) y_p = log10(y_p)
00500          else
00501             if(grid%axe(3)%is_log) then
00502                y_s = log10(y)
00503                y_p => y_s
00504             else
00505                y_p => y
00506             endif
00507          endif
00508       endif
00509
00510       if(with_z) then
00511          if(.not.is_sorted_z)then
00512             iz_sort(:)= (/ (i,i=1,size(iz_sort(:))   ) /)
00513             call mrgrnk(z(:),iz_sort(:))
00514             z_s(:)=z(ix_sort(:))
00515             z_p => z_s
00516             if(grid%axe(4)%is_log) z_p = log10(z_p)
00517          else
00518             z_p => z
00519             if(grid%axe(4)%is_log) then
00520                z_s = log10(z)
00521                z_p => z_s
00522             else
00523                z_p => z
00524             endif
00525          endif
00526       endif
00527
00528
00529    !--get index of w in the grid
00530       call interpol_index(w_p(:),grid%axe(1)%x(:),iw(:),dw(:),grid)
00531    ! reset ix to original xvec positions
00532    ! next comand work, because internaly the f90 compiler works with copy comands
00533
00534       if(.not. is_sorted_w)then
00535          iw(iw_sort(:))=iw(:)
00536          dw(iw_sort(:))=dw(:)
00537       endif
00538
00539
00540       if(with_x) then
00541          idpc=size(x)/2
00542          call interpol_index(x_p(:),grid%axe(2)%x(:),ix(:),dx(:),grid)
00543          if(.not. is_sorted_x)then
00544             ix(ix_sort(:))=ix(:)
00545             dx(ix_sort(:))=dx(:)
00546          endif
00547       endif
00548
00549       if(with_y) then
00550          call interpol_index(y_p(:),grid%axe(3)%x(:),iy(:),dy(:),grid)
00551          if(.not. is_sorted_y)then
00552             iy(iy_sort(:))=iy(:)
00553             dy(iy_sort(:))=dy(:)
00554          endif
00555       endif
00556
00557       if(with_z) then
00558          call interpol_index(z_p(:),grid%axe(4)%x(:),iz(:),dz(:),grid)
00559          if(.not. is_sorted_z)then
00560             iz(iz_sort(:))=iz(:)
00561             dz(iz_sort(:))=dz(:)
00562          endif
00563       endif
00564    !
00565    !-- now interpolate
```

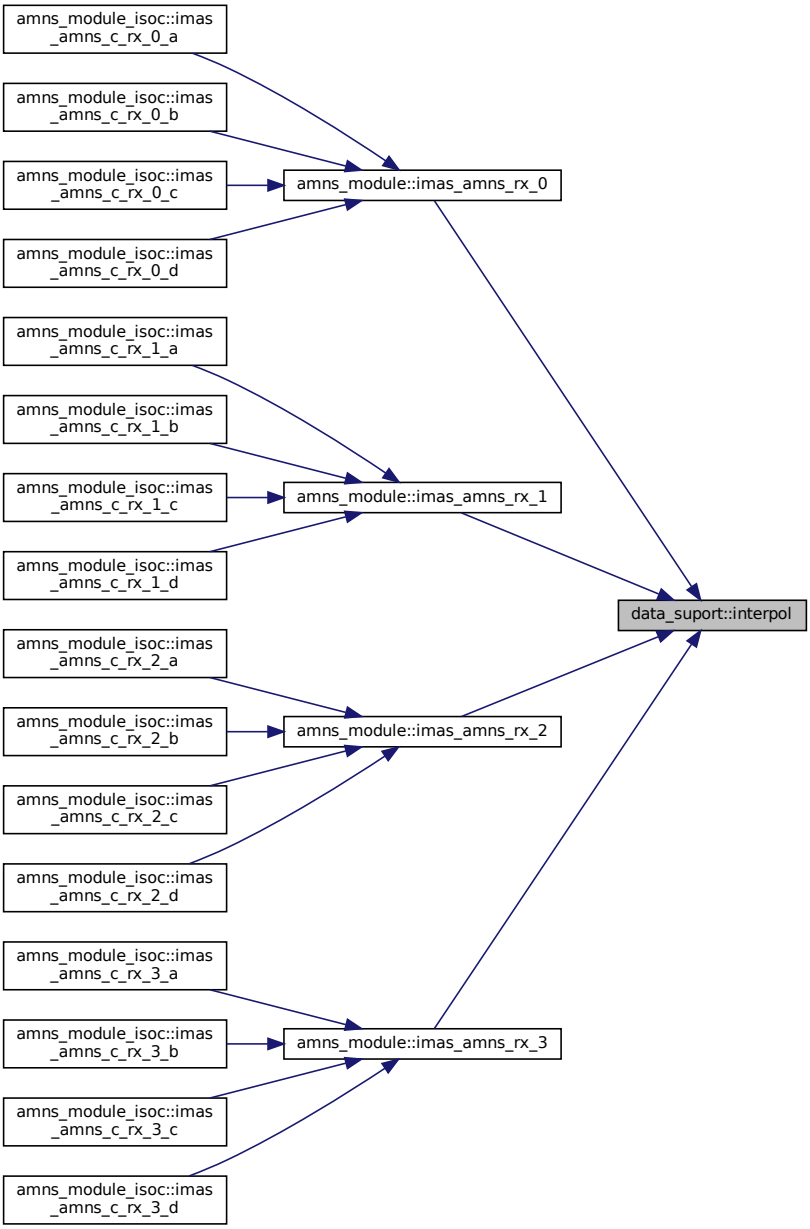
```
00566
00567       if(with_z)then
00568          if(one_d)then
00569             call interpo_4_1(iw(:),ix(:),iy(:),iz(:),dw(:),dx(:),dy(:),dz(:),fd1(:),grid)
00570          else
00571             call interpo_4_4(iw(:),ix(:),iy(:),iz(:),dw(:),dx(:),dy(:),dz(:),fd4(:,:,:,:),grid)
00572          endif
00573       elseif(with_y)then
00574          if(one_d)then
00575             call interpo_3_1(iw(:),ix(:),iy(:),dw(:),dx(:),dy(:),fd1(:),grid)
00576          else
00577             call interpo_3_3(iw(:),ix(:),iy(:),dw(:),dx(:),dy(:),fd3(:,:,:),grid)
00578          endif
00579       elseif(with_x)then
00580          if(one_d)then
00581             call interpo_2_1(iw(:),ix(:),dw(:),dx(:),fd1(:),grid)
00582          else
00583             call interpo_2_2(iw(:),ix(:),dw(:),dx(:),fd2(:,:),grid)
00584          endif
00585       else
00586          call interpo_1_1(iw(:),dw(:),fd1(:),grid)
00587       endif
00588
00589       if(with_x)then
00590          deallocate(dx,ix,ix_sort,x_s)
00591       endif
00592       if(with_y)then
00593          deallocate(dy,iy,iy_sort,y_s)
00594       endif
00595       if(with_z)then
00596          deallocate(dz,iz,iz_sort,z_s)
00597       endif
00598
00599       if(grid%is_log) then
00600          if(one_d)   fd1 = 10.0_rkind ** fd1
00601          if(two_d)   fd2 = 10.0_rkind ** fd2
00602          if(three_d) fd3 = 10.0_rkind ** fd3
00603          if(four_d)  fd4 = 10.0_rkind ** fd4
00604       endif
00605
00606    case (1001)    ! grid%interpol_function
00608       call assert(.not.(with_x.or.with_y.or.with_z), "nuclear data '1001' must be a function of 1
      parameter")
00609       call assert(grid%exist_f2d, "nuclear data '1001' requires 2d table of data")
00610       call assert(one_d, "nuclear data '1001' expected to be passed an effective 1d array")
00611       call nuclear_data_1001(grid%f2d, w, fd1, grid%with_warning, fun_err)
00612
00613    case (1002)    ! grid%interpol_function
00615       call assert(.not.(with_x.or.with_y.or.with_z), "nuclear data '1002' must be a function of 1
      parameter")
00616       call assert(grid%exist_f2d, "nuclear data '1002' requires 2d table of data")
00617       call assert(one_d, "nuclear data '1002' expected to be passed an effective 1d array")
00618       call nuclear_data_1002(grid%f2d, w, fd1, grid%with_warning, fun_err)
00619
00620    case (1003)    ! grid%interpol_function
00622       call assert(.not.(with_x.or.with_y.or.with_z), "resonant charge data '1003' must be a function
      of 1 parameter")
00623       call assert(grid%exist_f1d, "resonant charge transfer data '1003' requires 1d table of data")
00624       call assert(one_d, "resonant charge transfer '1003' expected to be passed an effective 1d
      array")
00625       call rct_data_1003(grid%f1d, w, fd1, grid%with_warning, fun_err)
00626
00627    case (1004)    ! grid%interpol_function
00629       call assert(.not.(with_y.or.with_z), "sputter data '1004' must be a function of 2 parameters")
00630       call assert(grid%exist_f2d, "sputter yield '1004' requires 2d table of data")
00631       call assert(one_d, "sputter yield '1004' expected to be passed an 2d array")
00632       call sputter_data_1004(grid%f2d, w, x, fd1, grid%with_warning, fun_err)
00633
00634    case (1005)    ! grid%interpol_function
00636       call assert(.not.(with_y.or.with_z), "reflection yield '1005' must be a function of 2
      parameters")
00637       call assert(grid%exist_f2d, "reflection yield '1005' requires 2d table of data")
00638       call assert(one_d, "reflection yield '1005' expected to be passed an 2d array")
00639       call reflect_data_1005(grid%f2d, w, x, fd1, grid%with_warning, fun_err)
00640
00641    case (1006)    ! grid%interpol_function
00643       call assert(.not.(with_x.or.with_y.or.with_z), "nuclear data '1006' must be a function of 1
      parameter")
00644       call assert(grid%exist_f2d, "nuclear data '1006' requires 2d table of data")
00645       call assert(one_d, "nuclear data '1006' expected to be passed an effective 1d array")
00646       call nuclear_data_1006(grid%f2d, w, fd1, grid%with_warning, fun_err)
00647
00648    case default
00649       write(*,*) 'Case for grid%interpol_function = ', grid%interpol_function, ' not yet coded'
00650
00651    end select
00652
```

```
00653     if(fun_err%ierr.ne.0) then
00654        data_error%ierr = fun_err%ierr
00655        data_error%cerr = fun_err%cerr
00656     endif
00657
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.32.1.3 set_option()

```
subroutine, public data_suport::set_option (
             type(grid_t) grid,
             logical, optional warning )
```

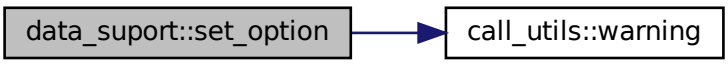
set the "with_warning" flag in grid to the value of "warning"

Definition at line 318 of file data_suport.f90.

```
00319     implicit none
00320     type(grid_t)          :: grid
00321     logical,optional      :: warning
00322
00323     if(present(warning)) grid%with_warning =warning
```

```
00324
```

Here is the call graph for this function:



14.32.1.4 sorted()

```
logical function data_suport::sorted (
             real(rkind), dimension(:), intent(in) x )
```

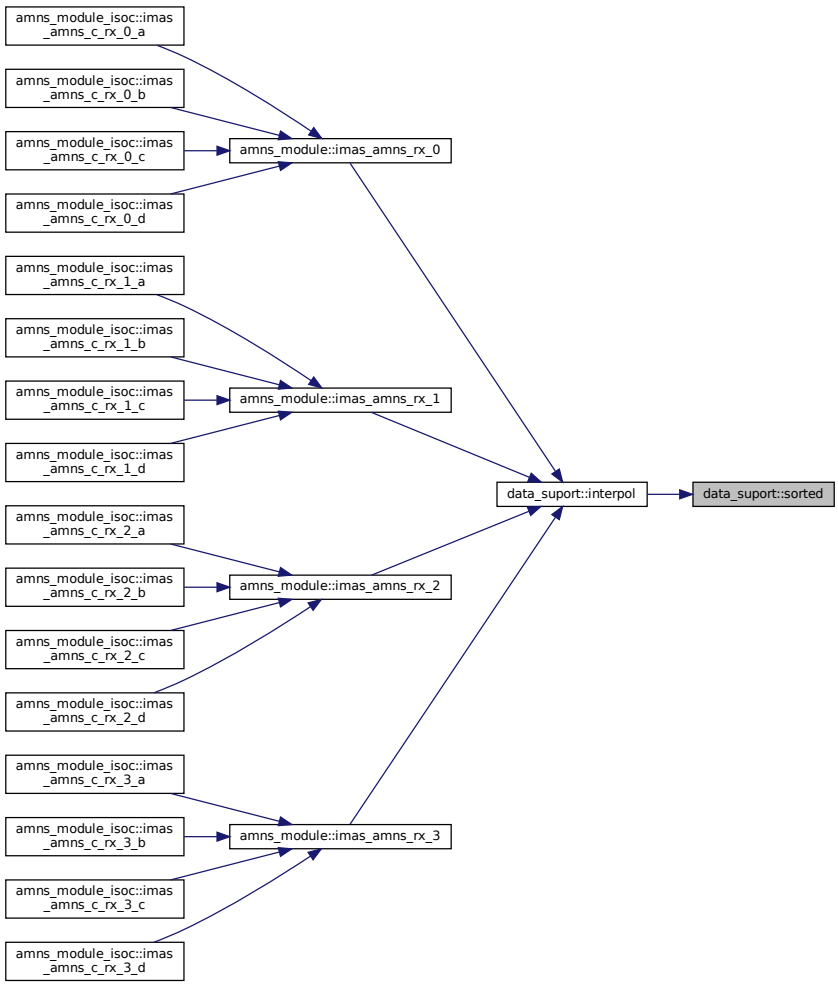
return true if the passed array is sorted

Definition at line 665 of file data_suport.f90.

```
00666     real(rkind), intent (in) :: x(:)
00667     logical ::result
00668
00669     !  logical:: order
00670     integer:: i
00671
00672     result=.true.
00673     do i=1,size(x)-1
00674        if(x(i)>x(i+1))then
00675           result=.false.
00676           exit
00677        endif
00678     enddo
```

Here is the call graph for this function:

Here is the caller graph for this function:



14.33 eckstein_yields Module Reference

Functions/Subroutines

- real(ids_real) function etf (M1, M2, Z1, Z2)

  AMNS External utility function ...
- real(ids_real) function omegal (epsl)

  AMNS External utility function ...
- real(ids_real) function sn (epsl)

  AMNS External utility function ...
- real(ids_real) function seyield (E0, M1, M2, Z1, Z2, q, lambda, u, ETh)

  AMNS External utility function ...
- real(ids_real) function sayield (E0, theta, f, b, c, Esp)

  AMNS External utility function ...
- real(ids_real) function reyieldlight (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...
- real(ids_real) function reyieldself (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...

- real(ids_real) function reyield (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...
- real(ids_real) function rayield (angledeg, C1, C2, C3, C4)

  AMNS External utility function ...

Variables

- real(ids_real), parameter mathpi = 3.14159265358979323846264338327 95
- real(ids_real), parameter degtorad = 0.01745329251994329576923690768489

14.33.1 Variable Documentation

14.33.1.1 degtorad

```
real(ids_real), parameter eckstein_yields::degtorad = 0.01745329251994329576923690768489
```

Definition at line 13 of file eckstein_yields.f90.

```
00013   real(ids_real), parameter :: degtorad = 0.01745329251994329576923690768489
```

14.33.1.2 mathpi

```
real(ids_real), parameter eckstein_yields::mathpi = 3.14159265358979323846264338327 95
```

Definition at line 12 of file eckstein_yields.f90.

```
00012   real(ids_real), parameter :: mathpi = 3.14159265358979323846264338327 95
```

14.34 f90_kind Module Reference

f90_kind module from Silvio Gori's grid package

Variables

- integer, parameter rkind = kind(1.0d0)
- integer, parameter ikind = kind(1)
- integer, parameter skind = 256

14.34.1 Detailed Description

f90_kind module from Silvio Gori's grid package

Author

Silvio Gori

14.34.2 Variable Documentation

14.34.2.1 ikind

```
integer, parameter f90_kind::ikind = kind(1)
```

Definition at line 14 of file f90_kind.f90.

```
00014   integer, parameter :: ikind  = kind(1)
```

14.34.2.2 rkind

```
integer, parameter f90_kind::rkind = kind(1.0d0)
```

Definition at line 13 of file f90_kind.f90.

```
00013   integer, parameter :: rkind  = kind(1.0d0)
```

14.34.2.3 skind

```
integer, parameter f90_kind::skind = 256
```

Definition at line 15 of file f90_kind.f90.

```
00015   integer, parameter :: skind  = 256
```

14.35 interface_to_amns Module Reference

Functions/Subroutines

- subroutine get_amns_data (reaction_type, reactants, grid, properties_comment, properties_source, properties_provider, properties_creation_date, code_name, code_commit, code_version, code_repository, source, provider, citation, shot, run, backend, user, ds_version, ierr, error_description, debug)

  transfer data from the IDS to the internal data structure
- character(len=answer_length) function assign_if_associated (ids_str, def_str)

  utility to assign if associated
- subroutine end_amns_data

  deallocate idss

Variables

- type(amns_ids_list), pointer first => null()

14.35.1 Variable Documentation

14.35.1.1 first

```
type (amns_ids_list), pointer interface_to_amns::first => null()
```

Definition at line 12 of file interface_to_amns.f90.

```
00012   type (amns_ids_list), pointer :: first => null()
```

14.36 m_mrgrnk Module Reference

Data Types

- interface mrgrnk

14.37 quadpack Module Reference

Functions/Subroutines

- subroutine aaaa
- subroutine qag (f, a, b, epsabs, epsrel, key, result, abserr, neval, ier)
- subroutine qage (f, a, b, epsabs, epsrel, key, limit, result, abserr, neval, ier, alist, blist, rlist, elist, iord, last)
- subroutine qagi (f, bound, inf, epsabs, epsrel, result, abserr, neval, ier)
- subroutine qagp (f, a, b, npts2, points, epsabs, epsrel, result, abserr, neval, ier)
- subroutine qags (f, a, b, epsabs, epsrel, result, abserr, neval, ier)
- subroutine qawc (f, a, b, c, epsabs, epsrel, result, abserr, neval, ier)

- subroutine qawce (f, a, b, c, epsabs, epsrel, limit, result, abserr, neval, ier, alist, blist, rlist, elist, iord, last)
- subroutine qawf (f, a, omega, integr, epsabs, result, abserr, neval, ier)
- subroutine qawfe (f, a, omega, integr, epsabs, limlst, limit, maxp1, result, abserr, neval, ier, rslst, erlst, ierlst, lst, alist, blist, rlist, elist, iord, nnlog, chebmo)
- subroutine qawo (f, a, b, omega, integr, epsabs, epsrel, result, abserr, neval, ier)
- subroutine qaws (f, a, b, alfa, beta, integr, epsabs, epsrel, result, abserr, neval, ier)
- subroutine qawse (f, a, b, alfa, beta, integr, epsabs, epsrel, limit, result, abserr, neval, ier, alist, blist, rlist, elist, iord, last)
- subroutine qc25c (f, a, b, c, result, abserr, krul, neval)
- subroutine qc25o (f, a, b, omega, integr, nrmom, maxp1, ksave, result, abserr, neval, resabs, resasc, momcom, chebmo)
- subroutine qc25s (f, a, b, bl, br, alfa, beta, ri, rj, rg, rh, result, abserr, resasc, integr, neval)
- subroutine qcheb (x, fval, cheb12, cheb24)
- subroutine qextr (n, epstab, result, abserr, res3la, nres)
- subroutine qfour (f, a, b, omega, integr, epsabs, epsrel, limit, icall, maxp1, result, abserr, neval, ier, alist, blist, rlist, elist, iord, nnlog, momcom, chebmo)
- subroutine qk15 (f, a, b, result, abserr, resabs, resasc)
- subroutine qk15i (f, boun, inf, a, b, result, abserr, resabs, resasc)
- subroutine qk15w (f, w, p1, p2, p3, p4, kp, a, b, result, abserr, resabs, resasc)
- subroutine qk21 (f, a, b, result, abserr, resabs, resasc)
- subroutine qk31 (f, a, b, result, abserr, resabs, resasc)
- subroutine qk41 (f, a, b, result, abserr, resabs, resasc)
- subroutine qk51 (f, a, b, result, abserr, resabs, resasc)
- subroutine qk61 (f, a, b, result, abserr, resabs, resasc)
- subroutine qmomo (alfa, beta, ri, rj, rg, rh, integr)
- subroutine qng (f, a, b, epsabs, epsrel, result, abserr, neval, ier)
- subroutine qsort (limit, last, maxerr, ermax, elist, iord, nrmax)
- real(kind=params_wp) function qwgtc (x, c, p2, p3, p4, kp)
- real(kind=params_wp) function qwgto (x, omega, p2, p3, p4, integr)
- real(kind=params_wp) function qwgts (x, a, b, alfa, beta, integr)
- subroutine timestamp ()

14.37.1 Function/Subroutine Documentation

14.37.1.1 aaaa()

```
subroutine quadpack::aaaa
```

Definition at line 50 of file quadpack.f90.

```
00051
00052 !*****************************************************************************80
00053 !
00054 !! AAAA is a dummy subroutine with QUADPACK documentation in its comments.
00055 !
00056 ! 1. introduction
00057 !
00058 !    quadpack is a fortran subroutine package for the numerical
00059 !    computation of definite one-dimensional integrals. it originated
00060 !    from a joint project of r. piessens and e. de doncker (appl.
00061 !    math. and progr. div.- k.u.leuven, belgium), c. ueberhuber (inst.
00062 !    fuer math.- techn.u.wien, austria), and d. kahaner (nation. bur.
00063 !    of standards- washington d.c., u.s.a.).
00064 !
00065 ! 2. survey
00066 !
00067 !    - qags : is an integrator based on globally adaptive interval
00068 !             subdivision in connection with extrapolation (de doncker,
00069 !             1978) by the epsilon algorithm (wynn, 1956).
00070 !
00071 !    - qagp : serves the same purposes as qags, but also allows
00072 !             for eventual user-supplied information, i.e. the
00073 !             abscissae of internal singularities, discontinuities
00074 !             and other difficulties of the integrand function.
00075 !             the algorithm is a modification of that in qags.
```

```
00076 !
00077 !    - qagi : handles integration over infinite intervals. the
00078 !             infinite range is mapped onto a finite interval and
00079 !             then the same strategy as in qags is applied.
00080 !
00081 !    - qawo : is a routine for the integration of cos(omega*x)*f(x)
00082 !             or sin(omega*x)*f(x) over a finite interval (a,b).
00083 !             omega is is specified by the user
00084 !             the rule evaluation component is based on the
00085 !             modified clenshaw-curtis technique.
00086 !             an adaptive subdivision scheme is used connected with
00087 !             an extrapolation procedure, which is a modification
00088 !             of that in qags and provides the possibility to deal
00089 !             even with singularities in f.
00090 !
00091 !    - qawf : calculates the fourier cosine or fourier sine
00092 !             transform of f(x), for user-supplied interval (a,
00093 !             infinity), omega, and f. the procedure of qawo is
00094 !             used on successive finite intervals, and convergence
00095 !             acceleration by means of the epsilon algorithm (wynn,
00096 !             1956) is applied to the series of the integral
00097 !             contributions.
00098 !
00099 !    - qaws : integrates w(x)*f(x) over (a,b) with a < b finite,
00100 !             and   w(x) = ((x-a)**alfa)*((b-x)**beta)*v(x)
00101 !             where v(x) = 1 or log(x-a) or log(b-x)
00102 !                            or log(x-a)*log(b-x)
00103 !             and   -1 < alfa, -1 < beta.
00104 !             the user specifies a, b, alfa, beta and the type of
00105 !             the function v.
00106 !             a globally adaptive subdivision strategy is applied,
00107 !             with modified clenshaw-curtis integration on the
00108 !             subintervals which contain a or b.
00109 !
00110 !    - qawc : computes the cauchy principal value of f(x)/(x-c)
00111 !             over a finite interval (a,b) and for
00112 !             user-determined c.
00113 !             the strategy is globally adaptive, and modified
00114 !             clenshaw-curtis integration is used on the subranges
00115 !             which contain the point x = c.
00116 !
00117 !  each of the routines above also has a "more detailed" version
00118 !    with a name ending in e, as qage.  these provide more
00119 !    information and control than the easier versions.
00120 !
00121 !
00122 !   the preceeding routines are all automatic.  that is, the user
00123 !      inputs his problem and an error tolerance.  the routine
00124 !      attempts to perform the integration to within the requested
00125 !      absolute or relative error.
00126 !   there are, in addition, a number of non-automatic integrators.
00127 !      these are most useful when the problem is such that the
00128 !      user knows that a fixed rule will provide the accuracy
00129 !      required.  typically they return an error estimate but make
00130 !      no attempt to satisfy any particular input error request.
00131 !
00132 !      qk15
00133 !      qk21
00134 !      qk31
00135 !      qk41
00136 !      qk51
00137 !      qk61
00138 !           estimate the integral on [a,b] using 15, 21,..., 61
00139 !           point rule and return an error estimate.
00140 !      qk15i 15 point rule for (semi)infinite interval.
00141 !      qk15w 15 point rule for special singular weight functions.
00142 !      qc25c 25 point rule for cauchy principal values
00143 !      qc25o 25 point rule for sin/cos integrand.
00144 !      qmomo integrates k-th degree chebychev polynomial times
00145 !            function with various explicit singularities.
00146 !
00147 ! 3. guidelines for the use of quadpack
00148 !
00149 !    here it is not our purpose to investigate the question when
00150 !    automatic quadrature should be used. we shall rather attempt
00151 !    to help the user who already made the decision to use quadpack,
00152 !    with selecting an appropriate routine or a combination of
00153 !    several routines for handling his problem.
00154 !
00155 !    for both quadrature over finite and over infinite intervals,
00156 !    one of the first questions to be answered by the user is
00157 !    related to the amount of computer time he wants to spend,
00158 !    versus his -own- time which would be needed, for example, for
00159 !    manual subdivision of the interval or other analytic
00160 !    manipulations.
00161 !
00162 !    (1) the user may not care about computer time, or not be
```

```
00163 !        willing to do any analysis of the problem. especially when
00164 !        only one or a few integrals must be calculated, this attitude
00165 !        can be perfectly reasonable. in this case it is clear that
00166 !        either the most sophisticated of the routines for finite
00167 !        intervals, qags, must be used, or its analogue for infinite
00168 !        intervals, qagi. these routines are able to cope with
00169 !        rather difficult, even with improper integrals.
00170 !        this way of proceeding may be expensive. but the integrator
00171 !        is supposed to give you an answer in return, with additional
00172 !        information in the case of a failure, through its error
00173 !        estimate and flag. yet it must be stressed that the programs
00174 !        cannot be totally reliable.
00175 !
00176 !    (2) the user may want to examine the integrand function.
00177 !        if bad local difficulties occur, such as a discontinuity, a
00178 !        singularity, derivative singularity or high peak at one or
00179 !        more points within the interval, the first advice is to
00180 !        split up the interval at these points. the integrand must
00181 !        then be examinated over each of the subintervals separately,
00182 !        so that a suitable integrator can be selected for each of
00183 !        them. if this yields problems involving relative accuracies
00184 !        to be imposed on -finite- subintervals, one can make use of
00185 !        qagp, which must be provided with the positions of the local
00186 !        difficulties. however, if strong singularities are present
00187 !        and a high accuracy is requested, application of qags on the
00188 !        subintervals may yield a better result.
00189 !
00190 !        for quadrature over finite intervals we thus dispose of qags
00191 !        and
00192 !        - qng for well-behaved integrands,
00193 !        - qag for functions with an oscillating behavior of a non
00194 !          specific type,
00195 !        - qawo for functions, eventually singular, containing a
00196 !          factor cos(omega*x) or sin(omega*x) where omega is known,
00197 !        - qaws for integrands with algebraico-logarithmic end point
00198 !          singularities of known type,
00199 !        - qawc for cauchy principal values.
00200 !
00201 !        remark
00202 !
00203 !        on return, the work arrays in the argument lists of the
00204 !        adaptive integrators contain information about the interval
00205 !        subdivision process and hence about the integrand behavior:
00206 !        the end points of the subintervals, the local integral
00207 !        contributions and error estimates, and eventually other
00208 !        characteristics. for this reason, and because of its simple
00209 !        globally adaptive nature, the routine qag in particular is
00210 !        well-suited for integrand examination. difficult spots can
00211 !        be located by investigating the error estimates on the
00212 !        subintervals.
00213 !
00214 !        for infinite intervals we provide only one general-purpose
00215 !        routine, qagi. it is based on the qags algorithm applied
00216 !        after a transformation of the original interval into (0,1).
00217 !        yet it may eventuate that another type of transformation is
00218 !        more appropriate, or one might prefer to break up the
00219 !        original interval and use qagi only on the infinite part
00220 !        and so on. these kinds of actions suggest a combined use of
00221 !        different quadpack integrators. note that, when the only
00222 !        difficulty is an integrand singularity at the finite
00223 !        integration limit, it will in general not be necessary to
00224 !        break up the interval, as qagi deals with several types of
00225 !        singularity at the boundary point of the integration range.
00226 !        it also handles slowly convergent improper integrals, on
00227 !        the condition that the integrand does not oscillate over
00228 !        the entire infinite interval. if it does we would advise
00229 !        to sum succeeding positive and negative contributions to
00230 !        the integral -e.g. integrate between the zeros- with one
00231 !        or more of the finite-range integrators, and apply
00232 !        convergence acceleration eventually by means of quadpack
00233 !        subroutine qelg which implements the epsilon algorithm.
00234 !        such quadrature problems include the fourier transform as
00235 !        a special case. yet for the latter we have an automatic
00236 !        integrator available, qawf.
00237 !
00238   return
```

14.37.1.2 qag()

```
subroutine quadpack::qag (
            real(kind=params_wp), external f,
            real(kind=params_wp) a,
```

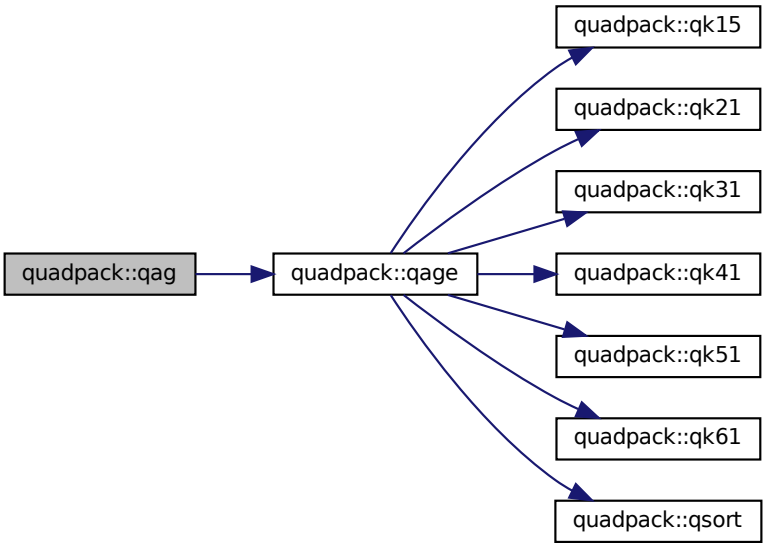
```
            real(kind=params_wp) b,
            real(kind=params_wp) epsabs,
            real(kind=params_wp) epsrel,
            integer key,
            real(kind=params_wp) result,
            real(kind=params_wp) abserr,
            integer neval,
            integer ier )
```

Definition at line 240 of file quadpack.f90.

```
00241
00242 !****************************************************************************80
00243 !
00244 !! QAG approximates an integral over a finite interval.
00245 !
00246 !  Discussion:
00247 !
00248 !    The routine calculates an approximation RESULT to a definite integral
00249 !      I = integral of F over (A,B),
00250 !    hopefully satisfying
00251 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
00252 !
00253 !    QAG is a simple globally adaptive integrator using the strategy of
00254 !    Aind (Piessens, 1973).  It is possible to choose between 6 pairs of
00255 !    Gauss-Kronrod quadrature formulae for the rule evaluation component.
00256 !    The pairs of high degree of precision are suitable for handling
00257 !    integration difficulties due to a strongly oscillating integrand.
00258 !
00259 !  Author:
00260 !
00261 !    Robert Piessens, Elise de Doncker-Kapenger,
00262 !    Christian Ueberhuber, David Kahaner
00263 !
00264 !  Reference:
00265 !
00266 !    Robert Piessens, Elise de Doncker-Kapenger,
00267 !    Christian Ueberhuber, David Kahaner,
00268 !    QUADPACK, a Subroutine Package for Automatic Integration,
00269 !    Springer Verlag, 1983
00270 !
00271 !  Parameters:
00272 !
00273 !    Input, external real F, the name of the function routine, of the form
00274 !      function f ( x )
00275 !      real f
00276 !      real x
00277 !    which evaluates the integrand function.
00278 !
00279 !    Input, real A, B, the limits of integration.
00280 !
00281 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
00282 !
00283 !    Input, integer KEY, chooses the order of the local integration rule:
00284 !    1,  7 Gauss points, 15 Gauss-Kronrod points,
00285 !    2, 10 Gauss points, 21 Gauss-Kronrod points,
00286 !    3, 15 Gauss points, 31 Gauss-Kronrod points,
00287 !    4, 20 Gauss points, 41 Gauss-Kronrod points,
00288 !    5, 25 Gauss points, 51 Gauss-Kronrod points,
00289 !    6, 30 Gauss points, 61 Gauss-Kronrod points.
00290 !
00291 !    Output, real RESULT, the estimated value of the integral.
00292 !
00293 !    Output, real ABSERR, an estimate of || I - RESULT ||.
00294 !
00295 !    Output, integer NEVAL, the number of times the integral was evaluated.
00296 !
00297 !    Output, integer IER, return code.
00298 !    0, normal and reliable termination of the routine.  It is assumed that the
00299 !      requested accuracy has been achieved.
00300 !    1, maximum number of subdivisions allowed has been achieved.  One can
00301 !      allow more subdivisions by increasing the value of LIMIT in QAG.
00302 !      However, if this yields no improvement it is advised to analyze the
00303 !      integrand to determine the integration difficulties.  If the position
00304 !      of a local difficulty can be determined, such as a singularity or
00305 !      discontinuity within the interval) one will probably gain from
00306 !      splitting up the interval at this point and calling the integrator
00307 !      on the subranges.  If possible, an appropriate special-purpose
00308 !      integrator should be used which is designed for handling the type
00309 !      of difficulty involved.
00310 !    2, the occurrence of roundoff error is detected, which prevents the
00311 !      requested tolerance from being achieved.
00312 !    3, extremely bad integrand behavior occurs at some points of the
00313 !      integration interval.
00314 !    6, the input is invalid, because EPSABS < 0 and EPSREL < 0.
```

```
00315 !
00316 !  Local parameters:
00317 !
00318 !    LIMIT is the maximum number of subintervals allowed in
00319 !    the subdivision process of QAGE.
00320 !
00321   implicit none
00322
00323   integer, parameter :: limit = 500
00324
00325   real(kind=params_wp) a
00326   real(kind=params_wp) abserr
00327   real(kind=params_wp) alist(limit)
00328   real(kind=params_wp) b
00329   real(kind=params_wp) blist(limit)
00330   real(kind=params_wp) elist(limit)
00331   real(kind=params_wp) epsabs
00332   real(kind=params_wp) epsrel
00333   real(kind=params_wp), external :: f
00334   integer ier
00335   integer iord(limit)
00336   integer key
00337   integer last
00338   integer neval
00339   real(kind=params_wp) result
00340   real(kind=params_wp) rlist(limit)
00341
00342   call qage ( f, a, b, epsabs, epsrel, key, limit, result, abserr, neval, &
00343     ier, alist, blist, rlist, elist, iord, last )
00344
00345   return
```

Here is the call graph for this function:



14.37.1.3 qage()

```
subroutine quadpack::qage (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) epsrel,
             integer key,
```

```
            integer limit,
            real(kind=params_wp) result,
            real(kind=params_wp) abserr,
            integer neval,
            integer ier,
            real(kind=params_wp), dimension(limit) alist,
            real(kind=params_wp), dimension(limit) blist,
            real(kind=params_wp), dimension(limit) rlist,
            real(kind=params_wp), dimension(limit) elist,
            integer, dimension(limit) iord,
            integer last )
```

Definition at line 347 of file quadpack.f90.

```
00349
00350 !****************************************************************************80
00351 !
00352 !! QAGE estimates a definite integral.
00353 !
00354 !  Discussion:
00355 !
00356 !    The routine calculates an approximation RESULT to a definite integral
00357 !      I = integral of F over (A,B),
00358 !    hopefully satisfying
00359 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
00360 !
00361 !  Author:
00362 !
00363 !    Robert Piessens, Elise de Doncker-Kapenger,
00364 !    Christian Ueberhuber, David Kahaner
00365 !
00366 !  Reference:
00367 !
00368 !    Robert Piessens, Elise de Doncker-Kapenger,
00369 !    Christian Ueberhuber, David Kahaner,
00370 !    QUADPACK, a Subroutine Package for Automatic Integration,
00371 !    Springer Verlag, 1983
00372 !
00373 !  Parameters:
00374 !
00375 !    Input, external real F, the name of the function routine, of the form
00376 !      function f ( x )
00377 !      real f
00378 !      real x
00379 !    which evaluates the integrand function.
00380 !
00381 !    Input, real A, B, the limits of integration.
00382 !
00383 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
00384 !
00385 !    Input, integer KEY, chooses the order of the local integration rule:
00386 !    1,  7 Gauss points, 15 Gauss-Kronrod points,
00387 !    2, 10 Gauss points, 21 Gauss-Kronrod points,
00388 !    3, 15 Gauss points, 31 Gauss-Kronrod points,
00389 !    4, 20 Gauss points, 41 Gauss-Kronrod points,
00390 !    5, 25 Gauss points, 51 Gauss-Kronrod points,
00391 !    6, 30 Gauss points, 61 Gauss-Kronrod points.
00392 !
00393 !    Input, integer LIMIT, the maximum number of subintervals that
00394 !    can be used.
00395 !
00396 !    Output, real RESULT, the estimated value of the integral.
00397 !
00398 !    Output, real ABSERR, an estimate of || I - RESULT ||.
00399 !
00400 !    Output, integer NEVAL, the number of times the integral was evaluated.
00401 !
00402 !    Output, integer IER, return code.
00403 !    0, normal and reliable termination of the routine.  It is assumed that the
00404 !      requested accuracy has been achieved.
00405 !    1, maximum number of subdivisions allowed has been achieved.  One can
00406 !      allow more subdivisions by increasing the value of LIMIT in QAG.
00407 !      However, if this yields no improvement it is advised to analyze the
00408 !      integrand to determine the integration difficulties.  If the position
00409 !      of a local difficulty can be determined, such as a singularity or
00410 !      discontinuity within the interval) one will probably gain from
00411 !      splitting up the interval at this point and calling the integrator
00412 !      on the subranges.  If possible, an appropriate special-purpose
00413 !      integrator should be used which is designed for handling the type
00414 !      of difficulty involved.
00415 !    2, the occurrence of roundoff error is detected, which prevents the
00416 !      requested tolerance from being achieved.
00417 !    3, extremely bad integrand behavior occurs at some points of the
00418 !      integration interval.
```

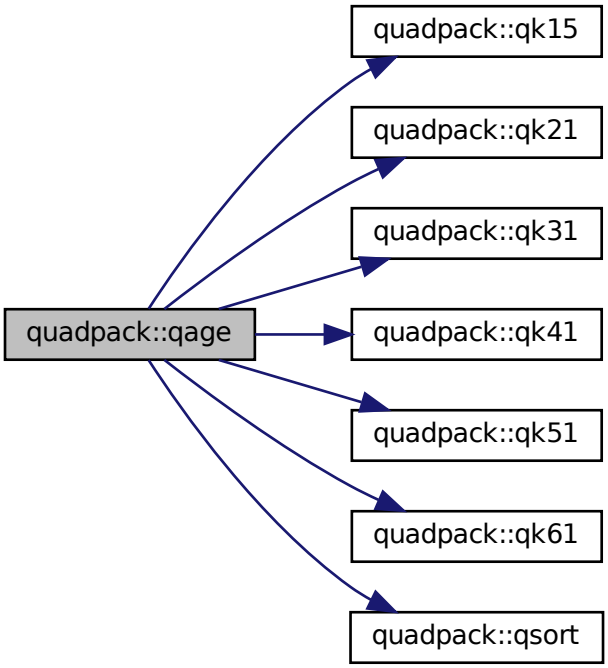
```
00419 !    6, the input is invalid, because EPSABS < 0 and EPSREL < 0.
00420 !
00421 !    Workspace, real ALIST(LIMIT), BLIST(LIMIT), contains in entries 1
00422 !    through LAST the left and right ends of the partition subintervals.
00423 !
00424 !    Workspace, real RLIST(LIMIT), contains in entries 1 through LAST
00425 !    the integral approximations on the subintervals.
00426 !
00427 !    Workspace, real ELIST(LIMIT), contains in entries 1 through LAST
00428 !    the absolute error estimates on the subintervals.
00429 !
00430 !    Output, integer IORD(LIMIT), the first K elements of which are pointers
00431 !    to the error estimates over the subintervals, such that
00432 !    elist(iord(1)), ..., elist(iord(k)) form a decreasing sequence, with
00433 !    k = last if last <= (limit/2+2), and k = limit+1-last otherwise.
00434 !
00435 !    Output, integer LAST, the number of subintervals actually produced
00436 !    in the subdivision process.
00437 !
00438 !  Local parameters:
00439 !
00440 !    alist     - list of left end points of all subintervals
00441 !                    considered up to now
00442 !    blist     - list of right end points of all subintervals
00443 !                    considered up to now
00444 !    elist(i)  - error estimate applying to rlist(i)
00445 !    maxerr    - pointer to the interval with largest error estimate
00446 !    errmax    - elist(maxerr)
00447 !    area      - sum of the integrals over the subintervals
00448 !    errsum    - sum of the errors over the subintervals
00449 !    errbnd    - requested accuracy max(epsabs,epsrel*abs(result))
00450 !    *****1    - variable for the left subinterval
00451 !    *****2    - variable for the right subinterval
00452 !    last      - index for subdivision
00453 !
00454   implicit none
00455
00456   integer limit
00457
00458   real(kind=params_wp) a
00459   real(kind=params_wp) abserr
00460   real(kind=params_wp) alist(limit)
00461   real(kind=params_wp) area
00462   real(kind=params_wp) area1
00463   real(kind=params_wp) area12
00464   real(kind=params_wp) area2
00465   real(kind=params_wp) a1
00466   real(kind=params_wp) a2
00467   real(kind=params_wp) b
00468   real(kind=params_wp) blist(limit)
00469   real(kind=params_wp) b1
00470   real(kind=params_wp) b2
00471   real(kind=params_wp) c
00472   real(kind=params_wp) defabs
00473   real(kind=params_wp) defab1
00474   real(kind=params_wp) defab2
00475   real(kind=params_wp) elist(limit)
00476   real(kind=params_wp) epsabs
00477   real(kind=params_wp) epsrel
00478   real(kind=params_wp) errbnd
00479   real(kind=params_wp) errmax
00480   real(kind=params_wp) error1
00481   real(kind=params_wp) error2
00482   real(kind=params_wp) erro12
00483   real(kind=params_wp) errsum
00484   real(kind=params_wp), external :: f
00485   integer ier
00486   integer iord(limit)
00487   integer iroff1
00488   integer iroff2
00489   integer key
00490   integer keyf
00491   integer last
00492   integer maxerr
00493   integer neval
00494   integer nrmax
00495   real(kind=params_wp) resabs
00496   real(kind=params_wp) result
00497   real(kind=params_wp) rlist(limit)
00498 !
00499 !  Test on validity of parameters.
00500 !
00501   ier = 0
00502   neval = 0
00503   last = 0
00504   result = 0.0e+00
00505   abserr = 0.0e+00
```

```
00506   alist(1) = a
00507   blist(1) = b
00508   rlist(1) = 0.0e+00
00509   elist(1) = 0.0e+00
00510   iord(1) = 0
00511
00512   if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
00513     ier = 6
00514     return
00515   end if
00516 !
00517 !  First approximation to the integral.
00518 !
00519   keyf = key
00520   keyf = max( keyf, 1 )
00521   keyf = min( keyf, 6 )
00522
00523   c = keyf
00524   neval = 0
00525
00526   if ( keyf == 1 ) then
00527     call qk15 ( f, a, b, result, abserr, defabs, resabs )
00528   else if ( keyf == 2 ) then
00529     call qk21 ( f, a, b, result, abserr, defabs, resabs )
00530   else if ( keyf == 3 ) then
00531     call qk31 ( f, a, b, result, abserr, defabs, resabs )
00532   else if ( keyf == 4 ) then
00533     call qk41 ( f, a, b, result, abserr, defabs, resabs )
00534   else if ( keyf == 5 ) then
00535     call qk51 ( f, a, b, result, abserr, defabs, resabs )
00536   else if ( keyf == 6 ) then
00537     call qk61 ( f, a, b, result, abserr, defabs, resabs )
00538   end if
00539
00540   last = 1
00541   rlist(1) = result
00542   elist(1) = abserr
00543   iord(1) = 1
00544 !
00545 !  Test on accuracy.
00546 !
00547   errbnd = max( epsabs, epsrel * abs( result ) )
00548
00549   if ( abserr <= 5.0e+01 * epsilon( defabs ) * defabs .and. &
00550     errbnd < abserr ) then
00551     ier = 2
00552   end if
00553
00554   if ( limit == 1 ) then
00555     ier = 1
00556   end if
00557
00558   if ( ier /= 0 .or. &
00559     ( abserr <= errbnd .and. abserr /= resabs ) .or. &
00560     abserr == 0.0e+00 ) then
00561
00562     if ( keyf /= 1 ) then
00563       neval = (10*keyf+1) * (2*neval+1)
00564     else
00565       neval = 30 * neval + 15
00566     end if
00567
00568     return
00569
00570   end if
00571 !
00572 !  Initialization.
00573 !
00574   errmax = abserr
00575   maxerr = 1
00576   area = result
00577   errsum = abserr
00578   nrmax = 1
00579   iroff1 = 0
00580   iroff2 = 0
00581
00582   do last = 2, limit
00583 !
00584 !  Bisect the subinterval with the largest error estimate.
00585 !
00586     a1 = alist(maxerr)
00587     b1 = 0.5e+00 * ( alist(maxerr) + blist(maxerr) )
00588     a2 = b1
00589     b2 = blist(maxerr)
00590
00591     if ( keyf == 1 ) then
00592       call qk15 ( f, a1, b1, area1, error1, resabs, defab1 )
```

```
00593     else if ( keyf == 2 ) then
00594       call qk21 ( f, a1, b1, area1, error1, resabs, defab1 )
00595     else if ( keyf == 3 ) then
00596       call qk31 ( f, a1, b1, area1, error1, resabs, defab1 )
00597     else if ( keyf == 4 ) then
00598       call qk41 ( f, a1, b1, area1, error1, resabs, defab1)
00599     else if ( keyf == 5 ) then
00600       call qk51 ( f, a1, b1, area1, error1, resabs, defab1 )
00601     else if ( keyf == 6 ) then
00602       call qk61 ( f, a1, b1, area1, error1, resabs, defab1 )
00603     end if
00604
00605     if ( keyf == 1 ) then
00606       call qk15 ( f, a2, b2, area2, error2, resabs, defab2 )
00607     else if ( keyf == 2 ) then
00608       call qk21 ( f, a2, b2, area2, error2, resabs, defab2 )
00609     else if ( keyf == 3 ) then
00610       call qk31 ( f, a2, b2, area2, error2, resabs, defab2 )
00611     else if ( keyf == 4 ) then
00612       call qk41 ( f, a2, b2, area2, error2, resabs, defab2 )
00613     else if ( keyf == 5 ) then
00614       call qk51 ( f, a2, b2, area2, error2, resabs, defab2 )
00615     else if ( keyf == 6 ) then
00616       call qk61 ( f, a2, b2, area2, error2, resabs, defab2 )
00617     end if
00618 !
00619 !  Improve previous approximations to integral and error and
00620 !  test for accuracy.
00621 !
00622     neval = neval + 1
00623     area12 = area1 + area2
00624     erro12 = error1 + error2
00625     errsum = errsum + erro12 - errmax
00626     area = area + area12 - rlist(maxerr)
00627
00628     if ( defab1 /= error1 .and. defab2 /= error2 ) then
00629
00630       if ( abs( rlist(maxerr) - area12 ) <= 1.0e-05 * abs( area12 ) &
00631         .and. 9.9e-01 * errmax <= erro12 ) then
00632         iroff1 = iroff1 + 1
00633       end if
00634
00635       if ( 10 < last .and. errmax < erro12 ) then
00636         iroff2 = iroff2 + 1
00637       end if
00638
00639     end if
00640
00641     rlist(maxerr) = area1
00642     rlist(last) = area2
00643     errbnd = max( epsabs, epsrel * abs( area ) )
00644 !
00645 !  Test for roundoff error and eventually set error flag.
00646 !
00647     if ( errbnd < errsum ) then
00648
00649       if ( 6 <= iroff1 .or. 20 <= iroff2 ) then
00650         ier = 2
00651       end if
00652 !
00653 !  Set error flag in the case that the number of subintervals
00654 !  equals limit.
00655 !
00656       if ( last == limit ) then
00657         ier = 1
00658       end if
00659 !
00660 !  Set error flag in the case of bad integrand behavior
00661 !  at a point of the integration range.
00662 !
00663       if ( max( abs( a1 ), abs( b2 ) ) <= ( 1.0e+00_params_wp + c * 1.0e+03 * &
00664         epsilon( a1 ) ) * ( abs( a2 ) + 1.0e+04 * tiny( a2 ) ) ) then
00665         ier = 3
00666       end if
00667
00668     end if
00669 !
00670 !  Append the newly-created intervals to the list.
00671 !
00672     if ( error2 <= error1 ) then
00673       alist(last) = a2
00674       blist(maxerr) = b1
00675       blist(last) = b2
00676       elist(maxerr) = error1
00677       elist(last) = error2
00678     else
00679       alist(maxerr) = a2
```

```
00680         alist(last) = a1
00681         blist(last) = b1
00682         rlist(maxerr) = area2
00683         rlist(last) = area1
00684         elist(maxerr) = error2
00685         elist(last) = error1
00686       end if
00687 !
00688 !  Call QSORT to maintain the descending ordering
00689 !  in the list of error estimates and select the subinterval
00690 !  with the largest error estimate (to be bisected next).
00691 !
00692       call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
00693
00694       if ( ier /= 0 .or. errsum <= errbnd ) then
00695         exit
00696       end if
00697
00698     end do
00699 !
00700 !  Compute final result.
00701 !
00702     result = sum( rlist(1:last) )
00703
00704     abserr = errsum
00705
00706     if ( keyf /= 1 ) then
00707       neval = ( 10 * keyf + 1 ) * ( 2 * neval + 1 )
00708     else
00709       neval = 30 * neval + 15
00710     end if
00711
00712     return
```

Here is the call graph for this function:



Here is the caller graph for this function:

14.37.1.4 qagi()

```
subroutine quadpack::qagi (
             real(kind=params_wp), external f,
             real(kind=params_wp) bound,
             integer inf,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) epsrel,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             integer neval,
             integer ier )
```

Definition at line 714 of file quadpack.f90.

```
00715
00716 !*****************************************************************************80
00717 !
00718 !! QAGI estimates an integral over a semi-infinite or infinite interval.
00719 !
00720 !  Discussion:
00721 !
00722 !    The routine calculates an approximation RESULT to a definite integral
00723 !      I = integral of F over (A, +Infinity),
00724 !    or
00725 !      I = integral of F over (-Infinity,A)
00726 !    or
00727 !      I = integral of F over (-Infinity,+Infinity),
00728 !    hopefully satisfying
00729 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
00730 !
00731 !  Author:
00732 !
00733 !    Robert Piessens, Elise de Doncker-Kapenger,
00734 !    Christian Ueberhuber, David Kahaner
00735 !
00736 !  Reference:
00737 !
00738 !    Robert Piessens, Elise de Doncker-Kapenger,
00739 !    Christian Ueberhuber, David Kahaner,
00740 !    QUADPACK, a Subroutine Package for Automatic Integration,
00741 !    Springer Verlag, 1983
00742 !
00743 !  Parameters:
00744 !
00745 !    Input, external real F, the name of the function routine, of the form
00746 !      function f ( x )
00747 !      real f
00748 !      real x
00749 !    which evaluates the integrand function.
00750 !
00751 !    Input, real BOUND, the value of the finite endpoint of the integration
00752 !    range, if any, that is, if INF is 1 or -1.
00753 !
00754 !    Input, integer INF, indicates the type of integration range.
00755 !    1:  (  BOUND,    +Infinity),
00756 !    -1: ( -Infinity,  BOUND),
00757 !    2:  ( -Infinity, +Infinity).
00758 !
00759 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
00760 !
00761 !    Output, real RESULT, the estimated value of the integral.
```

```
00762 !
00763 !    Output, real ABSERR, an estimate of || I - RESULT ||.
00764 !
00765 !    Output, integer NEVAL, the number of times the integral was evaluated.
00766 !
00767 !    Output, integer IER, error indicator.
00768 !    0, normal and reliable termination of the routine.  It is assumed that
00769 !      the requested accuracy has been achieved.
00770 !    > 0,  abnormal termination of the routine.  The estimates for result
00771 !      and error are less reliable.  It is assumed that the requested
00772 !      accuracy has not been achieved.
00773 !    1, maximum number of subdivisions allowed has been achieved.  One can
00774 !      allow more subdivisions by increasing the data value of LIMIT in QAGI
00775 !      (and taking the according dimension adjustments into account).
00776 !      However, if this yields no improvement it is advised to analyze the
00777 !      integrand in order to determine the integration difficulties.  If the
00778 !      position of a local difficulty can be determined (e.g. singularity,
00779 !      discontinuity within the interval) one will probably gain from
00780 !      splitting up the interval at this point and calling the integrator
00781 !      on the subranges.  If possible, an appropriate special-purpose
00782 !      integrator should be used, which is designed for handling the type
00783 !      of difficulty involved.
00784 !    2, the occurrence of roundoff error is detected, which prevents the
00785 !      requested tolerance from being achieved.  The error may be
00786 !      under-estimated.
00787 !    3, extremely bad integrand behavior occurs at some points of the
00788 !      integration interval.
00789 !    4, the algorithm does not converge.  Roundoff error is detected in the
00790 !      extrapolation table.  It is assumed that the requested tolerance
00791 !      cannot be achieved, and that the returned result is the best which
00792 !      can be obtained.
00793 !    5, the integral is probably divergent, or slowly convergent.  It must
00794 !      be noted that divergence can occur with any other value of IER.
00795 !    6, the input is invalid, because INF /= 1 and INF /= -1 and INF /= 2, or
00796 !      epsabs < 0 and epsrel < 0.  result, abserr, neval are set to zero.
00797 !
00798 !  Local parameters:
00799 !
00800 !           the dimension of rlist2 is determined by the value of
00801 !           limexp in QEXTR.
00802 !
00803 !           alist     - list of left end points of all subintervals
00804 !                       considered up to now
00805 !           blist     - list of right end points of all subintervals
00806 !                       considered up to now
00807 !           rlist(i)  - approximation to the integral over
00808 !                       (alist(i),blist(i))
00809 !           rlist2    - array of dimension at least (limexp+2),
00810 !                       containing the part of the epsilon table
00811 !                       which is still needed for further computations
00812 !           elist(i)  - error estimate applying to rlist(i)
00813 !           maxerr    - pointer to the interval with largest error
00814 !                       estimate
00815 !           errmax    - elist(maxerr)
00816 !           erlast    - error on the interval currently subdivided
00817 !                       (before that subdivision has taken place)
00818 !           area      - sum of the integrals over the subintervals
00819 !           errsum    - sum of the errors over the subintervals
00820 !           errbnd    - requested accuracy max(epsabs,epsrel*
00821 !                       abs(result))
00822 !           *****1    - variable for the left subinterval
00823 !           *****2    - variable for the right subinterval
00824 !           last      - index for subdivision
00825 !           nres      - number of calls to the extrapolation routine
00826 !           numrl2    - number of elements currently in rlist2. if an
00827 !                       appropriate approximation to the compounded
00828 !                       integral has been obtained, it is put in
00829 !                       rlist2(numrl2) after numrl2 has been increased
00830 !                       by one.
00831 !           small     - length of the smallest interval considered up
00832 !                       to now, multiplied by 1.5
00833 !           erlarg    - sum of the errors over the intervals larger
00834 !                       than the smallest interval considered up to now
00835 !           extrap    - logical variable denoting that the routine
00836 !                       is attempting to perform extrapolation. i.e.
00837 !                       before subdividing the smallest interval we
00838 !                       try to decrease the value of erlarg.
00839 !           noext     - logical variable denoting that extrapolation
00840 !                       is no longer allowed (true-value)
00841 !
00842   implicit none
00843
00844   integer, parameter :: limit = 500
00845
00846   real(kind=params_wp) abseps
00847   real(kind=params_wp) abserr
00848   real(kind=params_wp) alist(limit)
```

```
00849   real(kind=params_wp) area
00850   real(kind=params_wp) area1
00851   real(kind=params_wp) area12
00852   real(kind=params_wp) area2
00853   real(kind=params_wp) a1
00854   real(kind=params_wp) a2
00855   real(kind=params_wp) blist(limit)
00856   real(kind=params_wp) boun
00857   real(kind=params_wp) bound
00858   real(kind=params_wp) b1
00859   real(kind=params_wp) b2
00860   real(kind=params_wp) correc
00861   real(kind=params_wp) defabs
00862   real(kind=params_wp) defab1
00863   real(kind=params_wp) defab2
00864   real(kind=params_wp) dres
00865   real(kind=params_wp) elist(limit)
00866   real(kind=params_wp) epsabs
00867   real(kind=params_wp) epsrel
00868   real(kind=params_wp) erlarg
00869   real(kind=params_wp) erlast
00870   real(kind=params_wp) errbnd
00871   real(kind=params_wp) errmax
00872   real(kind=params_wp) error1
00873   real(kind=params_wp) error2
00874   real(kind=params_wp) erro12
00875   real(kind=params_wp) errsum
00876   real(kind=params_wp) ertest
00877   logical extrap
00878   real(kind=params_wp), external :: f
00879   integer id
00880   integer ier
00881   integer ierro
00882   integer inf
00883   integer iord(limit)
00884   integer iroff1
00885   integer iroff2
00886   integer iroff3
00887   integer jupbnd
00888   integer k
00889   integer ksgn
00890   integer ktmin
00891   integer last
00892   integer maxerr
00893   integer neval
00894   logical noext
00895   integer nres
00896   integer nrmax
00897   integer numrl2
00898   real(kind=params_wp) resabs
00899   real(kind=params_wp) reseps
00900   real(kind=params_wp) result
00901   real(kind=params_wp) res3la(3)
00902   real(kind=params_wp) rlist(limit)
00903   real(kind=params_wp) rlist2(52)
00904   real(kind=params_wp) small
00905 !
00906 !  Test on validity of parameters.
00907 !
00908   ier = 0
00909   neval = 0
00910   last = 0
00911   result = 0.0e+00
00912   abserr = 0.0e+00
00913   alist(1) = 0.0e+00
00914   blist(1) = 1.0e+00_params_wp
00915   rlist(1) = 0.0e+00
00916   elist(1) = 0.0e+00
00917   iord(1) = 0
00918
00919   if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
00920     ier = 6
00921     return
00922   end if
00923 !
00924 !  First approximation to the integral.
00925 !
00926 !  Determine the interval to be mapped onto (0,1).
00927 !  If INF = 2 the integral is computed as i = i1+i2, where
00928 !  i1 = integral of f over (-infinity,0),
00929 !  i2 = integral of f over (0,+infinity).
00930 !
00931   if ( inf == 2 ) then
00932     boun = 0.0e+00_params_wp
00933   else
00934     boun = bound
00935   end if
```

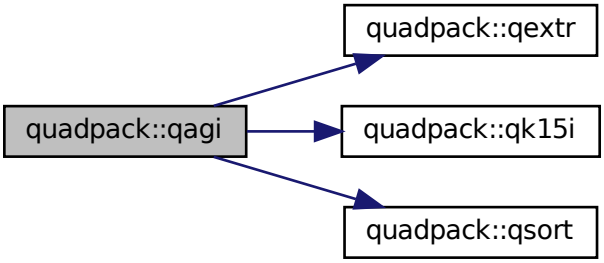
```
00936
00937   call qk15i ( f, boun, inf, 0.0e+00_params_wp, 1.0e+00_params_wp, result, abserr, defabs, resabs )
00938 !
00939 !  Test on accuracy.
00940 !
00941   last = 1
00942   rlist(1) = result
00943   elist(1) = abserr
00944   iord(1) = 1
00945   dres = abs( result )
00946   errbnd = max( epsabs, epsrel * dres )
00947
00948   if ( abserr <= 100.0e+00 * epsilon( defabs ) * defabs .and. &
00949     errbnd < abserr ) then
00950     ier = 2
00951   end if
00952
00953   if ( limit == 1 ) then
00954     ier = 1
00955   end if
00956
00957   if ( ier /= 0 .or. (abserr <= errbnd .and. abserr /= resabs ) .or. &
00958     abserr == 0.0e+00 ) go to 130
00959 !
00960 !  Initialization.
00961 !
00962   rlist2(1) = result
00963   errmax = abserr
00964   maxerr = 1
00965   area = result
00966   errsum = abserr
00967   abserr = huge( abserr )
00968   nrmax = 1
00969   nres = 0
00970   ktmin = 0
00971   numrl2 = 2
00972   extrap = .false.
00973   noext = .false.
00974   ierro = 0
00975   iroff1 = 0
00976   iroff2 = 0
00977   iroff3 = 0
00978
00979   if ( ( 1.0e+00_params_wp - 5.0e+01 * epsilon( defabs ) ) * defabs <= dres ) then
00980     ksgn = 1
00981   else
00982     ksgn = -1
00983   end if
00984
00985   do last = 2, limit
00986 !
00987 !  Bisect the subinterval with nrmax-th largest error estimate.
00988 !
00989     a1 = alist(maxerr)
00990     b1 = 5.0e-01 * ( alist(maxerr) + blist(maxerr) )
00991     a2 = b1
00992     b2 = blist(maxerr)
00993     erlast = errmax
00994     call qk15i ( f, boun, inf, a1, b1, area1, error1, resabs, defab1 )
00995     call qk15i ( f, boun, inf, a2, b2, area2, error2, resabs, defab2 )
00996 !
00997 !  Improve previous approximations to integral and error
00998 !  and test for accuracy.
00999 !
01000     area12 = area1 + area2
01001     erro12 = error1 + error2
01002     errsum = errsum + erro12 - errmax
01003     area = area + area12 - rlist(maxerr)
01004
01005     if ( defab1 /= error1 .and. defab2 /= error2 ) then
01006
01007       if ( abs( rlist(maxerr) - area12 ) <= 1.0e-05 * abs( area12 ) &
01008         .and. 9.9e-01 * errmax <= erro12 ) then
01009
01010         if ( extrap ) then
01011           iroff2 = iroff2 + 1
01012         end if
01013
01014         if ( .not. extrap ) then
01015           iroff1 = iroff1 + 1
01016         end if
01017
01018       end if
01019
01020       if ( 10 < last .and. errmax < erro12 ) then
01021         iroff3 = iroff3 + 1
01022       end if
```

```
01023
01024     end if
01025
01026     rlist(maxerr) = area1
01027     rlist(last) = area2
01028     errbnd = max( epsabs, epsrel * abs( area ) )
01029 !
01030 !  Test for roundoff error and eventually set error flag.
01031 !
01032     if ( 10 <= iroff1 + iroff2 .or. 20 <= iroff3 ) then
01033       ier = 2
01034     end if
01035
01036     if ( 5 <= iroff2 ) then
01037       ierro = 3
01038     end if
01039 !
01040 !  Set error flag in the case that the number of subintervals equals LIMIT.
01041 !
01042     if ( last == limit ) then
01043       ier = 1
01044     end if
01045 !
01046 !  Set error flag in the case of bad integrand behavior
01047 !  at some points of the integration range.
01048 !
01049     if ( max( abs(a1), abs(b2) ) <= (1.0e+00_params_wp + 1.0e+03 * epsilon( a1 ) ) * &
01050     ( abs(a2) + 1.0e+03 * tiny( a2 ) )) then
01051       ier = 4
01052     end if
01053 !
01054 !  Append the newly-created intervals to the list.
01055 !
01056     if ( error2 <= error1 ) then
01057       alist(last) = a2
01058       blist(maxerr) = b1
01059       blist(last) = b2
01060       elist(maxerr) = error1
01061       elist(last) = error2
01062     else
01063       alist(maxerr) = a2
01064       alist(last) = a1
01065       blist(last) = b1
01066       rlist(maxerr) = area2
01067       rlist(last) = area1
01068       elist(maxerr) = error2
01069       elist(last) = error1
01070     end if
01071 !
01072 !  Call QSORT to maintain the descending ordering
01073 !  in the list of error estimates and select the subinterval
01074 !  with NRMAX-th largest error estimate (to be bisected next).
01075 !
01076     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
01077
01078     if ( errsum <= errbnd ) go to 115
01079
01080     if ( ier /= 0 ) then
01081       exit
01082     end if
01083
01084     if ( last == 2 ) then
01085       small = 3.75e-01
01086       erlarg = errsum
01087       ertest = errbnd
01088       rlist2(2) = area
01089       cycle
01090     end if
01091
01092     if ( noext ) then
01093       cycle
01094     end if
01095
01096     erlarg = erlarg - erlast
01097
01098     if ( small < abs( b1 - a1 ) ) then
01099       erlarg = erlarg + erro12
01100     end if
01101 !
01102 !  Test whether the interval to be bisected next is the
01103 !  smallest interval.
01104 !
01105     if ( .not. extrap ) then
01106
01107       if ( small < abs( blist(maxerr) - alist(maxerr) ) ) then
01108         cycle
01109       end if
```

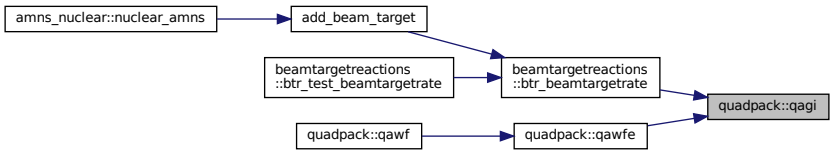
```
01110
01111       extrap = .true.
01112       nrmax = 2
01113
01114     end if
01115
01116     if ( ierro == 3 .or. erlarg <= ertest ) then
01117       go to 60
01118     end if
01119 !
01120 !  The smallest interval has the largest error.
01121 !  before bisecting decrease the sum of the errors over the
01122 !  larger intervals (erlarg) and perform extrapolation.
01123 !
01124     id = nrmax
01125     jupbnd = last
01126
01127     if ( (2+limit/2) < last ) then
01128       jupbnd = limit + 3 - last
01129     end if
01130
01131     do k = id, jupbnd
01132       maxerr = iord(nrmax)
01133       errmax = elist(maxerr)
01134       if ( small < abs( blist(maxerr) - alist(maxerr) ) ) then
01135         go to 90
01136       end if
01137       nrmax = nrmax + 1
01138     end do
01139 !
01140 !  Extrapolate.
01141 !
01142 60  continue
01143
01144     numrl2 = numrl2 + 1
01145     rlist2(numrl2) = area
01146     call qextr ( numrl2, rlist2, reseps, abseps, res3la, nres )
01147     ktmin = ktmin+1
01148
01149     if ( 5 < ktmin .and. abserr < 1.0e-03 * errsum ) then
01150       ier = 5
01151     end if
01152
01153     if ( abseps < abserr ) then
01154
01155       ktmin = 0
01156       abserr = abseps
01157       result = reseps
01158       correc = erlarg
01159       ertest = max( epsabs, epsrel * abs(reseps) )
01160
01161       if ( abserr <= ertest ) then
01162         exit
01163       end if
01164
01165     end if
01166 !
01167 !  Prepare bisection of the smallest interval.
01168 !
01169     if ( numrl2 == 1 ) then
01170       noext = .true.
01171     end if
01172
01173     if ( ier == 5 ) then
01174       exit
01175     end if
01176
01177     maxerr = iord(1)
01178     errmax = elist(maxerr)
01179     nrmax = 1
01180     extrap = .false.
01181     small = small * 5.0e-01
01182     erlarg = errsum
01183
01184 90  continue
01185
01186   end do
01187 !
01188 !  Set final result and error estimate.
01189 !
01190   if ( abserr == huge( abserr ) ) then
01191     go to 115
01192   end if
01193
01194   if ( ( ier + ierro ) == 0 ) then
01195     go to 110
01196   end if
```

```
01197
01198   if ( ierro == 3 ) then
01199     abserr = abserr + correc
01200   end if
01201
01202   if ( ier == 0 ) then
01203     ier = 3
01204   end if
01205
01206   if ( result /= 0.0e+00 .and. area /= 0.0e+00) then
01207     go to 105
01208   end if
01209
01210   if ( errsum < abserr ) then
01211     go to 115
01212   end if
01213
01214   if ( area == 0.0e+00 ) then
01215     go to 130
01216   end if
01217
01218   go to 110
01219
01220 105 continue
01221
01222   if ( errsum / abs( area ) < abserr / abs( result )  ) then
01223     go to 115
01224   end if
01225 !
01226 !  Test on divergence
01227 !
01228 110 continue
01229
01230   if ( ksgn == (-1) .and. &
01231   max( abs(result), abs(area) ) <=  defabs * 1.0e-02) go to 130
01232
01233   if ( 1.0e-02 > (result/area) .or. &
01234     (result/area) > 1.0e+02 .or. &
01235     errsum > abs(area)) then
01236     ier = 6
01237   end if
01238
01239   go to 130
01240 !
01241 !  Compute global integral sum.
01242 !
01243   115 continue
01244
01245   result = sum( rlist(1:last) )
01246
01247   abserr = errsum
01248   130 continue
01249
01250   neval = 30 * last - 15
01251   if ( inf == 2 ) then
01252     neval = 2 * neval
01253   end if
01254
01255   if ( 2 < ier ) then
01256     ier = ier - 1
01257   end if
01258
01259   return
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.37.1.5 qagp()

```
subroutine quadpack::qagp (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             integer npts2,
             real(kind=params_wp), dimension(40) points,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) epsrel,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             integer neval,
             integer ier )
```

Definition at line 1261 of file quadpack.f90.

```
01263
01264 !****************************************************************************80
01265 !
01266 !! QAGP computes a definite integral.
01267 !
01268 !  Discussion:
01269 !
01270 !    The routine calculates an approximation RESULT to a definite integral
01271 !      I = integral of F over (A,B),
01272 !    hopefully satisfying
01273 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
01274 !
01275 !    Interior break points of the integration interval,
01276 !    where local difficulties of the integrand may occur, such as
01277 !    singularities or discontinuities, are provided by the user.
01278 !
01279 !  Author:
```

```
01280 !
01281 !    Robert Piessens, Elise de Doncker-Kapenger,
01282 !    Christian Ueberhuber, David Kahaner
01283 !
01284 !  Reference:
01285 !
01286 !    Robert Piessens, Elise de Doncker-Kapenger,
01287 !    Christian Ueberhuber, David Kahaner,
01288 !    QUADPACK, a Subroutine Package for Automatic Integration,
01289 !    Springer Verlag, 1983
01290 !
01291 !  Parameters:
01292 !
01293 !    Input, external real F, the name of the function routine, of the form
01294 !      function f ( x )
01295 !      real f
01296 !      real x
01297 !    which evaluates the integrand function.
01298 !
01299 !    Input, real A, B, the limits of integration.
01300 !
01301 !    Input, integer NPTS2, the number of user-supplied break points within
01302 !    the integration range, plus 2.  NPTS2 must be at least 2.
01303 !
01304 !    Input/output, real POINTS(NPTS2), contains the user provided interior
01305 !    breakpoints in entries 1 through NPTS2-2.  If these points are not
01306 !    in ascending order on input, they will be sorted.
01307 !
01308 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
01309 !
01310 !    Output, real RESULT, the estimated value of the integral.
01311 !
01312 !    Output, real ABSERR, an estimate of || I - RESULT ||.
01313 !
01314 !    Output, integer NEVAL, the number of times the integral was evaluated.
01315 !
01316 !    Output, integer IER, return flag.
01317 !                      ier = 0 normal and reliable termination of the
01318 !                              routine. it is assumed that the requested
01319 !                              accuracy has been achieved.
01320 !                      ier > 0 abnormal termination of the routine.
01321 !                              the estimates for integral and error are
01322 !                              less reliable. it is assumed that the
01323 !                              requested accuracy has not been achieved.
01324 !                      ier = 1 maximum number of subdivisions allowed
01325 !                              has been achieved. one can allow more
01326 !                              subdivisions by increasing the data value
01327 !                              of limit in qagp(and taking the according
01328 !                              dimension adjustments into account).
01329 !                              however, if this yields no improvement
01330 !                              it is advised to analyze the integrand
01331 !                              in order to determine the integration
01332 !                              difficulties. if the position of a local
01333 !                              difficulty can be determined (i.e.
01334 !                              singularity, discontinuity within the
01335 !                              interval), it should be supplied to the
01336 !                              routine as an element of the vector
01337 !                              points. if necessary, an appropriate
01338 !                              special-purpose integrator must be used,
01339 !                              which is designed for handling the type
01340 !                              of difficulty involved.
01341 !                          = 2 the occurrence of roundoff error is
01342 !                              detected, which prevents the requested
01343 !                              tolerance from being achieved.
01344 !                              the error may be under-estimated.
01345 !                          = 3 extremely bad integrand behavior occurs
01346 !                              at some points of the integration
01347 !                              interval.
01348 !                          = 4 the algorithm does not converge. roundoff
01349 !                              error is detected in the extrapolation
01350 !                              table. it is presumed that the requested
01351 !                              tolerance cannot be achieved, and that
01352 !                              the returned result is the best which
01353 !                              can be obtained.
01354 !                          = 5 the integral is probably divergent, or
01355 !                              slowly convergent. it must be noted that
01356 !                              divergence can occur with any other value
01357 !                              of ier > 0.
01358 !                          = 6 the input is invalid because
01359 !                              npts2 < 2 or
01360 !                              break points are specified outside
01361 !                              the integration range or
01362 !                              epsabs < 0 and epsrel < 0,
01363 !                              or limit < npts2.
01364 !                              result, abserr, neval are set to zero.
01365 !
01366 !  Local parameters:
```

```
01367 !
01368 !            the dimension of rlist2 is determined by the value of
01369 !            limexp in QEXTR (rlist2 should be of dimension
01370 !            (limexp+2) at least).
01371 !
01372 !           alist     - list of left end points of all subintervals
01373 !                       considered up to now
01374 !           blist     - list of right end points of all subintervals
01375 !                       considered up to now
01376 !           rlist(i)  - approximation to the integral over
01377 !                       (alist(i),blist(i))
01378 !           rlist2    - array of dimension at least limexp+2
01379 !                       containing the part of the epsilon table which
01380 !                       is still needed for further computations
01381 !           elist(i)  - error estimate applying to rlist(i)
01382 !           maxerr    - pointer to the interval with largest error
01383 !                       estimate
01384 !           errmax    - elist(maxerr)
01385 !           erlast    - error on the interval currently subdivided
01386 !                       (before that subdivision has taken place)
01387 !           area      - sum of the integrals over the subintervals
01388 !           errsum    - sum of the errors over the subintervals
01389 !           errbnd    - requested accuracy max(epsabs,epsrel*
01390 !                       abs(result))
01391 !           *****1    - variable for the left subinterval
01392 !           *****2    - variable for the right subinterval
01393 !           last      - index for subdivision
01394 !           nres      - number of calls to the extrapolation routine
01395 !           numrl2    - number of elements in rlist2. if an appropriate
01396 !                       approximation to the compounded integral has
01397 !                       obtained, it is put in rlist2(numrl2) after
01398 !                       numrl2 has been increased by one.
01399 !           erlarg    - sum of the errors over the intervals larger
01400 !                       than the smallest interval considered up to now
01401 !           extrap    - logical variable denoting that the routine
01402 !                       is attempting to perform extrapolation. i.e.
01403 !                       before subdividing the smallest interval we
01404 !                       try to decrease the value of erlarg.
01405 !           noext     - logical variable denoting that extrapolation is
01406 !                       no longer allowed (true-value)
01407 !
01408   implicit none
01409
01410   integer, parameter :: limit = 500
01411
01412   real(kind=params_wp) a
01413   real(kind=params_wp) abseps
01414   real(kind=params_wp) abserr
01415   real(kind=params_wp) alist(limit)
01416   real(kind=params_wp) area
01417   real(kind=params_wp) area1
01418   real(kind=params_wp) area12
01419   real(kind=params_wp) area2
01420   real(kind=params_wp) a1
01421   real(kind=params_wp) a2
01422   real(kind=params_wp) b
01423   real(kind=params_wp) blist(limit)
01424   real(kind=params_wp) b1
01425   real(kind=params_wp) b2
01426   real(kind=params_wp) correc
01427   real(kind=params_wp) defabs
01428   real(kind=params_wp) defab1
01429   real(kind=params_wp) defab2
01430   real(kind=params_wp) dres
01431   real(kind=params_wp) elist(limit)
01432   real(kind=params_wp) epsabs
01433   real(kind=params_wp) epsrel
01434   real(kind=params_wp) erlarg
01435   real(kind=params_wp) erlast
01436   real(kind=params_wp) errbnd
01437   real(kind=params_wp) errmax
01438   real(kind=params_wp) error1
01439   real(kind=params_wp) erro12
01440   real(kind=params_wp) error2
01441   real(kind=params_wp) errsum
01442   real(kind=params_wp) ertest
01443   logical extrap
01444   real(kind=params_wp), external :: f
01445   integer i
01446   integer id
01447   integer ier
01448   integer ierro
01449   integer ind1
01450   integer ind2
01451   integer iord(limit)
01452   integer iroff1
01453   integer iroff2
```

```
01454   integer iroff3
01455   integer j
01456   integer jlow
01457   integer jupbnd
01458   integer k
01459   integer ksgn
01460   integer ktmin
01461   integer last
01462   integer levcur
01463   integer level(limit)
01464   integer levmax
01465   integer maxerr
01466   integer ndin(40)
01467   integer neval
01468   integer nint
01469   logical noext
01470   integer npts
01471   integer npts2
01472   integer nres
01473   integer nrmax
01474   integer numrl2
01475   real(kind=params_wp) points(40)
01476   real(kind=params_wp) pts(40)
01477   real(kind=params_wp) resa
01478   real(kind=params_wp) resabs
01479   real(kind=params_wp) reseps
01480   real(kind=params_wp) result
01481   real(kind=params_wp) res3la(3)
01482   real(kind=params_wp) rlist(limit)
01483   real(kind=params_wp) rlist2(52)
01484   real(kind=params_wp) sign
01485   real(kind=params_wp) temp
01486 !
01487 !  Test on validity of parameters.
01488 !
01489   ier = 0
01490   neval = 0
01491   last = 0
01492   result = 0.0e+00
01493   abserr = 0.0e+00
01494   alist(1) = a
01495   blist(1) = b
01496   rlist(1) = 0.0e+00
01497   elist(1) = 0.0e+00
01498   iord(1) = 0
01499   level(1) = 0
01500   npts = npts2 - 2
01501
01502   if ( npts2 < 2 ) then
01503     ier = 6
01504     return
01505   else if ( limit <= npts .or. ( epsabs < 0.0e+00 .and. &
01506     epsrel < 0.0e+00) ) then
01507     ier = 6
01508     return
01509   end if
01510 !
01511 !  If any break points are provided, sort them into an
01512 !  ascending sequence.
01513 !
01514   if ( b < a ) then
01515     sign = -1.0e+00_params_wp
01516   else
01517     sign = +1.0e+00_params_wp
01518   end if
01519
01520   pts(1) = min( a, b )
01521
01522   do i = 1, npts
01523     pts(i+1) = points(i)
01524   end do
01525
01526   pts(npts+2) = max( a, b )
01527   nint = npts+1
01528   a1 = pts(1)
01529
01530   if ( npts /= 0 ) then
01531
01532     do i = 1, nint
01533       do j = i+1, nint+1
01534         if ( pts(j) < pts(i) ) then
01535           temp   = pts(i)
01536           pts(i) = pts(j)
01537           pts(j) = temp
01538         end if
01539       end do
01540     end do
```

```
01541
01542     if ( pts(1) /= min( a, b ) .or. pts(nint+1) /= max( a, b ) ) then
01543       ier = 6
01544       return
01545     end if
01546
01547   end if
01548 !
01549 !  Compute first integral and error approximations.
01550 !
01551   resabs = 0.0e+00
01552
01553   do i = 1, nint
01554
01555     b1 = pts(i+1)
01556     call qk21 ( f, a1, b1, area1, error1, defabs, resa )
01557     abserr = abserr + error1
01558     result = result + area1
01559     ndin(i) = 0
01560
01561     if ( error1 == resa .and. error1 /= 0.0e+00 ) then
01562       ndin(i) = 1
01563     end if
01564
01565     resabs = resabs + defabs
01566     level(i) = 0
01567     elist(i) = error1
01568     alist(i) = a1
01569     blist(i) = b1
01570     rlist(i) = area1
01571     iord(i) = i
01572     a1 = b1
01573
01574   end do
01575
01576   errsum = 0.0e+00
01577
01578   do i = 1, nint
01579     if ( ndin(i) == 1 ) then
01580       elist(i) = abserr
01581     end if
01582     errsum = errsum + elist(i)
01583   end do
01584 !
01585 !  Test on accuracy.
01586 !
01587   last = nint
01588   neval = 21 * nint
01589   dres = abs( result )
01590   errbnd = max( epsabs, epsrel * dres )
01591
01592   if ( abserr <= 1.0e+02 * epsilon( resabs ) * resabs .and. &
01593     abserr > errbnd ) then
01594     ier = 2
01595   end if
01596
01597   if ( nint /= 1 ) then
01598
01599     do i = 1, npts
01600
01601       jlow = i+1
01602       ind1 = iord(i)
01603
01604       do j = jlow, nint
01605         ind2 = iord(j)
01606         if ( elist(ind1) <= elist(ind2) ) then
01607           ind1 = ind2
01608           k = j
01609         end if
01610       end do
01611
01612       if ( ind1 /= iord(i) ) then
01613         iord(k) = iord(i)
01614         iord(i) = ind1
01615       end if
01616
01617     end do
01618
01619     if ( limit < npts2 ) then
01620       ier = 1
01621     end if
01622
01623   end if
01624
01625   if ( ier /= 0 .or. abserr <= errbnd ) then
01626     return
01627   end if
```

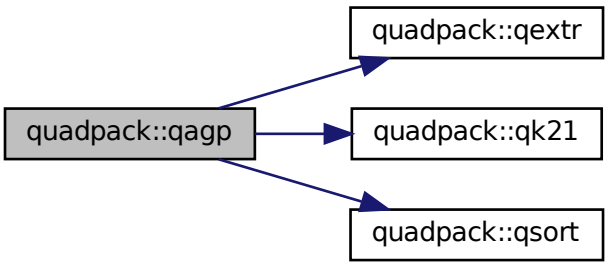
```
01628 !
01629 !  Initialization
01630 !
01631   rlist2(1) = result
01632   maxerr = iord(1)
01633   errmax = elist(maxerr)
01634   area = result
01635   nrmax = 1
01636   nres = 0
01637   numrl2 = 1
01638   ktmin = 0
01639   extrap = .false.
01640   noext = .false.
01641   erlarg = errsum
01642   ertest = errbnd
01643   levmax = 1
01644   iroff1 = 0
01645   iroff2 = 0
01646   iroff3 = 0
01647   ierro = 0
01648   abserr = huge( abserr )
01649
01650   if ( dres >= ( 1.0e+00_params_wp - 0.5e+00 * epsilon( resabs ) ) * resabs ) then
01651     ksgn = 1
01652   else
01653     ksgn = -1
01654   end if
01655
01656   do last = npts2, limit
01657 !
01658 !  Bisect the subinterval with the NRMAX-th largest error estimate.
01659 !
01660     levcur = level(maxerr) + 1
01661     a1 = alist(maxerr)
01662     b1 = 0.5e+00 * ( alist(maxerr) + blist(maxerr) )
01663     a2 = b1
01664     b2 = blist(maxerr)
01665     erlast = errmax
01666     call qk21 ( f, a1, b1, area1, error1, resa, defab1 )
01667     call qk21 ( f, a2, b2, area2, error2, resa, defab2 )
01668 !
01669 !  Improve previous approximations to integral and error
01670 !  and test for accuracy.
01671 !
01672     neval = neval + 42
01673     area12 = area1 + area2
01674     erro12 = error1 + error2
01675     errsum = errsum + erro12 -errmax
01676     area = area + area12 - rlist(maxerr)
01677
01678     if ( defab1 /= error1 .and. defab2 /= error2 ) then
01679
01680       if ( abs( rlist( maxerr ) - area12 ) <= 1.0e-05 * abs(area12) .and. &
01681         erro12 >= 9.9e-01 * errmax ) then
01682
01683         if ( extrap ) then
01684           iroff2 = iroff2+1
01685         else
01686           iroff1 = iroff1+1
01687         end if
01688
01689       end if
01690
01691       if ( last > 10 .and. erro12 > errmax ) then
01692         iroff3 = iroff3 + 1
01693       end if
01694
01695     end if
01696
01697     level(maxerr) = levcur
01698     level(last) = levcur
01699     rlist(maxerr) = area1
01700     rlist(last) = area2
01701     errbnd = max( epsabs, epsrel * abs( area ) )
01702 !
01703 !  Test for roundoff error and eventually set error flag.
01704 !
01705     if ( 10 <= iroff1 + iroff2 .or. 20 <= iroff3 ) then
01706       ier = 2
01707     end if
01708
01709     if ( 5 <= iroff2 ) then
01710       ierro = 3
01711     end if
01712 !
01713 !  Set error flag in the case that the number of subintervals
01714 !  equals limit.
```

```
01715 !
01716     if ( last == limit ) then
01717       ier = 1
01718     end if
01719 !
01720 !  Set error flag in the case of bad integrand behavior
01721 !  at a point of the integration range
01722 !
01723     if ( max( abs(a1), abs(b2)) <= ( 1.0e+00_params_wp + 1.0e+03 * epsilon( a1 ) )* &
01724     ( abs( a2 ) + 1.0e+03 * tiny( a2 ) ) ) then
01725       ier = 4
01726     end if
01727 !
01728 !  Append the newly-created intervals to the list.
01729 !
01730     if ( error2 <= error1 ) then
01731       alist(last) = a2
01732       blist(maxerr) = b1
01733       blist(last) = b2
01734       elist(maxerr) = error1
01735       elist(last) = error2
01736     else
01737       alist(maxerr) = a2
01738       alist(last) = a1
01739       blist(last) = b1
01740       rlist(maxerr) = area2
01741       rlist(last) = area1
01742       elist(maxerr) = error2
01743       elist(last) = error1
01744     end if
01745 !
01746 !  Call QSORT to maintain the descending ordering
01747 !  in the list of error estimates and select the subinterval
01748 !  with nrmax-th largest error estimate (to be bisected next).
01749 !
01750     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
01751
01752     if ( errsum <= errbnd ) then
01753       go to 190
01754     end if
01755
01756     if ( ier /= 0 ) then
01757       exit
01758     end if
01759
01760     if ( noext ) then
01761       cycle
01762     end if
01763
01764     erlarg = erlarg - erlast
01765
01766     if ( levcur+1 <= levmax ) then
01767       erlarg = erlarg + erro12
01768     end if
01769 !
01770 !  Test whether the interval to be bisected next is the
01771 !  smallest interval.
01772 !
01773     if ( .not. extrap ) then
01774
01775       if ( level(maxerr)+1 <= levmax ) then
01776         cycle
01777       end if
01778
01779       extrap = .true.
01780       nrmax = 2
01781
01782     end if
01783 !
01784 !  The smallest interval has the largest error.
01785 !  Before bisecting decrease the sum of the errors over the
01786 !  larger intervals (erlarg) and perform extrapolation.
01787 !
01788     if ( ierro /= 3 .and. erlarg > ertest ) then
01789
01790       id = nrmax
01791       jupbnd = last
01792       if ( last > (2+limit/2) ) then
01793         jupbnd = limit+3-last
01794       end if
01795
01796       do k = id, jupbnd
01797         maxerr = iord(nrmax)
01798         errmax = elist(maxerr)
01799         if ( level(maxerr)+1 <= levmax ) then
01800           go to 160
01801         end if
```

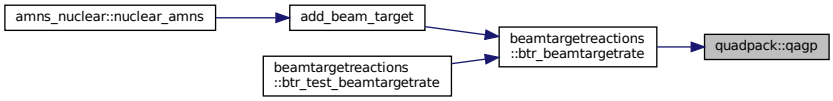
```
01802         nrmax = nrmax + 1
01803       end do
01804
01805     end if
01806 !
01807 !  Perform extrapolation.
01808 !
01809     numrl2 = numrl2 + 1
01810     rlist2(numrl2) = area
01811
01812     if ( numrl2 <= 2 ) then
01813       go to 155
01814     end if
01815
01816     call qextr ( numrl2, rlist2, reseps, abseps, res3la, nres )
01817     ktmin = ktmin+1
01818
01819     if ( 5 < ktmin .and. abserr < 1.0e-03 * errsum ) then
01820       ier = 5
01821     end if
01822
01823     if ( abseps < abserr ) then
01824
01825       ktmin = 0
01826       abserr = abseps
01827       result = reseps
01828       correc = erlarg
01829       ertest = max( epsabs, epsrel * abs(reseps) )
01830
01831       if ( abserr < ertest ) then
01832         exit
01833       end if
01834
01835     end if
01836 !
01837 !  Prepare bisection of the smallest interval.
01838 !
01839     if ( numrl2 == 1 ) then
01840       noext = .true.
01841     end if
01842
01843     if ( 5 <= ier ) then
01844       exit
01845     end if
01846
01847 155 continue
01848
01849     maxerr = iord(1)
01850     errmax = elist(maxerr)
01851     nrmax = 1
01852     extrap = .false.
01853     levmax = levmax + 1
01854     erlarg = errsum
01855
01856 160 continue
01857
01858   end do
01859 !
01860 !  Set the final result.
01861 !
01862   if ( abserr == huge( abserr ) ) then
01863     go to 190
01864   end if
01865
01866   if ( ( ier + ierro ) == 0 ) then
01867     go to 180
01868   end if
01869
01870   if ( ierro == 3 ) then
01871     abserr = abserr + correc
01872   end if
01873
01874   if ( ier == 0 ) then
01875     ier = 3
01876   end if
01877
01878   if ( result /= 0.0e+00 .and. area /= 0.0e+00 ) then
01879     go to 175
01880   end if
01881
01882   if ( errsum < abserr ) then
01883     go to 190
01884   end if
01885
01886   if ( area == 0.0e+00 ) then
01887     go to 210
01888   end if
```

```
01889
01890   go to 180
01891
01892 175 continue
01893
01894   if ( abserr / abs(result) > errsum / abs(area) ) then
01895     go to 190
01896   end if
01897 !
01898 !  Test on divergence.
01899 !
01900   180 continue
01901
01902   if ( ksgn == (-1) .and. max( abs(result),abs(area)) <=  &
01903     resabs*1.0e-02 ) go to 210
01904
01905   if ( 1.0e-02 > (result/area) .or. (result/area) > 1.0e+02 .or. &
01906     errsum > abs(area) ) then
01907     ier = 6
01908   end if
01909
01910   go to 210
01911 !
01912 !  Compute global integral sum.
01913 !
01914 190 continue
01915
01916   result = sum( rlist(1:last) )
01917
01918   abserr = errsum
01919
01920 210 continue
01921
01922   if ( 2 < ier ) then
01923     ier = ier - 1
01924   end if
01925
01926   result = result * sign
01927
01928   return
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.37.1.6 qags()

```
subroutine quadpack::qags (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) epsrel,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             integer neval,
             integer ier )
```

Definition at line 1930 of file quadpack.f90.

```
01931
01932 !*****************************************************************************80
01933 !
01934 !! QAGS estimates the integral of a function.
01935 !
01936 !  Discussion:
01937 !
01938 !    The routine calculates an approximation RESULT to a definite integral
01939 !      I = integral of F over (A,B),
01940 !    hopefully satisfying
01941 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
01942 !
01943 !  Author:
01944 !
01945 !    Robert Piessens, Elise de Doncker-Kapenger,
01946 !    Christian Ueberhuber, David Kahaner
01947 !
01948 !  Reference:
01949 !
01950 !    Robert Piessens, Elise de Doncker-Kapenger,
01951 !    Christian Ueberhuber, David Kahaner,
01952 !    QUADPACK, a Subroutine Package for Automatic Integration,
01953 !    Springer Verlag, 1983
01954 !
01955 !  Parameters:
01956 !
01957 !    Input, external real F, the name of the function routine, of the form
01958 !      function f ( x )
01959 !      real f
01960 !      real x
01961 !    which evaluates the integrand function.
01962 !
01963 !    Input, real A, B, the limits of integration.
01964 !
01965 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
01966 !
01967 !    Output, real RESULT, the estimated value of the integral.
01968 !
01969 !    Output, real ABSERR, an estimate of || I - RESULT ||.
01970 !
01971 !    Output, integer NEVAL, the number of times the integral was evaluated.
01972 !
01973 !    Output, integer IER, error flag.
01974 !                    ier = 0 normal and reliable termination of the
01975 !                            routine. it is assumed that the requested
01976 !                            accuracy has been achieved.
01977 !                    ier > 0 abnormal termination of the routine
01978 !                            the estimates for integral and error are
01979 !                            less reliable. it is assumed that the
01980 !                            requested accuracy has not been achieved.
01981 !                        = 1 maximum number of subdivisions allowed
01982 !                            has been achieved. one can allow more sub-
01983 !                            divisions by increasing the data value of
01984 !                            limit in qags (and taking the according
01985 !                            dimension adjustments into account).
01986 !                            however, if this yields no improvement
01987 !                            it is advised to analyze the integrand
01988 !                            in order to determine the integration
01989 !                            difficulties. if the position of a
01990 !                            local difficulty can be determined (e.g.
01991 !                            singularity, discontinuity within the
01992 !                            interval) one will probably gain from
01993 !                            splitting up the interval at this point
01994 !                            and calling the integrator on the sub-
01995 !                            ranges. if possible, an appropriate
01996 !                            special-purpose integrator should be used,
01997 !                            which is designed for handling the type
01998 !                            of difficulty involved.
01999 !                        = 2 the occurrence of roundoff error is detec-
```

```
02000 !                             ted, which prevents the requested
02001 !                             tolerance from being achieved.
02002 !                             the error may be under-estimated.
02003 !                           = 3 extremely bad integrand behavior occurs
02004 !                             at some  points of the integration
02005 !                             interval.
02006 !                           = 4 the algorithm does not converge. roundoff
02007 !                             error is detected in the extrapolation
02008 !                             table. it is presumed that the requested
02009 !                             tolerance cannot be achieved, and that the
02010 !                             returned result is the best which can be
02011 !                             obtained.
02012 !                           = 5 the integral is probably divergent, or
02013 !                             slowly convergent. it must be noted that
02014 !                             divergence can occur with any other value
02015 !                             of ier.
02016 !                           = 6 the input is invalid, because
02017 !                             epsabs < 0 and epsrel < 0,
02018 !                             result, abserr and neval are set to zero.
02019 !
02020 !  Local Parameters:
02021 !
02022 !           alist     - list of left end points of all subintervals
02023 !                       considered up to now
02024 !           blist     - list of right end points of all subintervals
02025 !                       considered up to now
02026 !           rlist(i)  - approximation to the integral over
02027 !                       (alist(i),blist(i))
02028 !           rlist2    - array of dimension at least limexp+2 containing
02029 !                       the part of the epsilon table which is still
02030 !                       needed for further computations
02031 !           elist(i)  - error estimate applying to rlist(i)
02032 !           maxerr    - pointer to the interval with largest error
02033 !                       estimate
02034 !           errmax    - elist(maxerr)
02035 !           erlast    - error on the interval currently subdivided
02036 !                       (before that subdivision has taken place)
02037 !           area      - sum of the integrals over the subintervals
02038 !           errsum    - sum of the errors over the subintervals
02039 !           errbnd    - requested accuracy max(epsabs,epsrel*
02040 !                       abs(result))
02041 !           *****1    - variable for the left interval
02042 !           *****2    - variable for the right interval
02043 !           last      - index for subdivision
02044 !           nres      - number of calls to the extrapolation routine
02045 !           numrl2    - number of elements currently in rlist2. if an
02046 !                       appropriate approximation to the compounded
02047 !                       integral has been obtained it is put in
02048 !                       rlist2(numrl2) after numrl2 has been increased
02049 !                       by one.
02050 !           small     - length of the smallest interval considered
02051 !                       up to now, multiplied by 1.5
02052 !           erlarg    - sum of the errors over the intervals larger
02053 !                       than the smallest interval considered up to now
02054 !           extrap    - logical variable denoting that the routine is
02055 !                       attempting to perform extrapolation i.e. before
02056 !                       subdividing the smallest interval we try to
02057 !                       decrease the value of erlarg.
02058 !           noext     - logical variable denoting that extrapolation
02059 !                       is no longer allowed (true value)
02060 !
02061   implicit none
02062
02063   integer, parameter :: limit = 500
02064
02065   real(kind=params_wp) a
02066   real(kind=params_wp) abseps
02067   real(kind=params_wp) abserr
02068   real(kind=params_wp) alist(limit)
02069   real(kind=params_wp) area
02070   real(kind=params_wp) area1
02071   real(kind=params_wp) area12
02072   real(kind=params_wp) area2
02073   real(kind=params_wp) a1
02074   real(kind=params_wp) a2
02075   real(kind=params_wp) b
02076   real(kind=params_wp) blist(limit)
02077   real(kind=params_wp) b1
02078   real(kind=params_wp) b2
02079   real(kind=params_wp) correc
02080   real(kind=params_wp) defabs
02081   real(kind=params_wp) defab1
02082   real(kind=params_wp) defab2
02083   real(kind=params_wp) dres
02084   real(kind=params_wp) elist(limit)
02085   real(kind=params_wp) epsabs
02086   real(kind=params_wp) epsrel
```

```
02087   real(kind=params_wp) erlarg
02088   real(kind=params_wp) erlast
02089   real(kind=params_wp) errbnd
02090   real(kind=params_wp) errmax
02091   real(kind=params_wp) error1
02092   real(kind=params_wp) error2
02093   real(kind=params_wp) erro12
02094   real(kind=params_wp) errsum
02095   real(kind=params_wp) ertest
02096   logical extrap
02097   real(kind=params_wp), external :: f
02098   integer id
02099   integer ier
02100   integer ierro
02101   integer iord(limit)
02102   integer iroff1
02103   integer iroff2
02104   integer iroff3
02105   integer jupbnd
02106   integer k
02107   integer ksgn
02108   integer ktmin
02109   integer last
02110   logical noext
02111   integer maxerr
02112   integer neval
02113   integer nres
02114   integer nrmax
02115   integer numrl2
02116   real(kind=params_wp) resabs
02117   real(kind=params_wp) reseps
02118   real(kind=params_wp) result
02119   real(kind=params_wp) res3la(3)
02120   real(kind=params_wp) rlist(limit)
02121   real(kind=params_wp) rlist2(52)
02122   real(kind=params_wp) small
02123 !
02124 !  The dimension of rlist2 is determined by the value of
02125 !  limexp in QEXTR (rlist2 should be of dimension
02126 !  (limexp+2) at least).
02127 !
02128 !  Test on validity of parameters.
02129 !
02130   ier = 0
02131   neval = 0
02132   last = 0
02133   result = 0.0e+00
02134   abserr = 0.0e+00
02135   alist(1) = a
02136   blist(1) = b
02137   rlist(1) = 0.0e+00
02138   elist(1) = 0.0e+00
02139
02140   if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
02141     ier = 6
02142     return
02143   end if
02144 !
02145 !  First approximation to the integral.
02146 !
02147   ierro = 0
02148   call qk21 ( f, a, b, result, abserr, defabs, resabs )
02149 !
02150 !  Test on accuracy.
02151 !
02152   dres = abs( result )
02153   errbnd = max( epsabs, epsrel * dres )
02154   last = 1
02155   rlist(1) = result
02156   elist(1) = abserr
02157   iord(1) = 1
02158
02159   if ( abserr <= 1.0e+02 * epsilon( defabs ) * defabs .and. &
02160     abserr > errbnd ) then
02161     ier = 2
02162   end if
02163
02164   if ( limit == 1 ) then
02165     ier = 1
02166   end if
02167
02168   if ( ier /= 0 .or. (abserr <= errbnd .and. abserr /= resabs ) .or. &
02169     abserr == 0.0e+00 ) go to 140
02170 !
02171 !  Initialization.
02172 !
02173   rlist2(1) = result
```

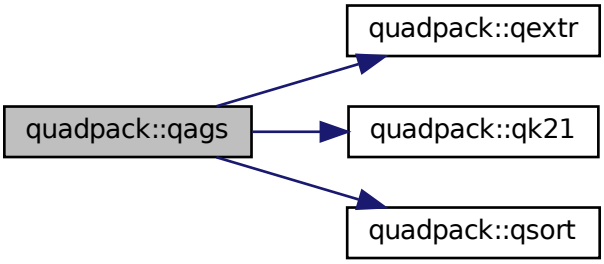
```
02174   errmax = abserr
02175   maxerr = 1
02176   area = result
02177   errsum = abserr
02178   abserr = huge( abserr )
02179   nrmax = 1
02180   nres = 0
02181   numrl2 = 2
02182   ktmin = 0
02183   extrap = .false.
02184   noext = .false.
02185   iroff1 = 0
02186   iroff2 = 0
02187   iroff3 = 0
02188
02189   if ( dres >= (1.0e+00_params_wp-5.0e+01* epsilon( defabs ) ) * defabs ) then
02190     ksgn = 1
02191   else
02192     ksgn = -1
02193   end if
02194
02195   do last = 2, limit
02196 !
02197 !  Bisect the subinterval with the nrmax-th largest error estimate.
02198 !
02199     a1 = alist(maxerr)
02200     b1 = 5.0e-01 * ( alist(maxerr) + blist(maxerr) )
02201     a2 = b1
02202     b2 = blist(maxerr)
02203     erlast = errmax
02204     call qk21 ( f, a1, b1, area1, error1, resabs, defab1 )
02205     call qk21 ( f, a2, b2, area2, error2, resabs, defab2 )
02206 !
02207 !  Improve previous approximations to integral and error
02208 !  and test for accuracy.
02209 !
02210     area12 = area1+area2
02211     erro12 = error1+error2
02212     errsum = errsum+erro12-errmax
02213     area = area+area12-rlist(maxerr)
02214
02215     if ( defab1 == error1 .or. defab2 == error2 ) go to 15
02216
02217     if ( abs( rlist(maxerr) - area12) > 1.0e-05 * abs(area12) &
02218       .or. erro12 < 9.9e-01 * errmax ) go to 10
02219
02220     if ( extrap ) then
02221       iroff2 = iroff2+1
02222     else
02223       iroff1 = iroff1+1
02224     end if
02225
02226 10  continue
02227
02228     if ( last > 10 .and. erro12 > errmax ) then
02229       iroff3 = iroff3+1
02230     end if
02231
02232 15  continue
02233
02234     rlist(maxerr) = area1
02235     rlist(last) = area2
02236     errbnd = max( epsabs, epsrel*abs(area) )
02237 !
02238 !  Test for roundoff error and eventually set error flag.
02239 !
02240     if ( iroff1+iroff2 >= 10 .or. iroff3 >= 20 ) then
02241       ier = 2
02242     end if
02243
02244     if ( iroff2 >= 5 ) then
02245       ierro = 3
02246     end if
02247 !
02248 !  Set error flag in the case that the number of subintervals
02249 !  equals limit.
02250 !
02251     if ( last == limit ) then
02252       ier = 1
02253     end if
02254 !
02255 !  Set error flag in the case of bad integrand behavior
02256 !  at a point of the integration range.
02257 !
02258     if ( max( abs(a1),abs(b2)) <= (1.0e+00_params_wp+1.0e+03* epsilon( a1 ) )* &
02259       (abs(a2)+1.0e+03* tiny( a2 ) ) ) then
02260       ier = 4
```

```
02261     end if
02262 !
02263 !  Append the newly-created intervals to the list.
02264 !
02265     if ( error2 <= error1 ) then
02266       alist(last) = a2
02267       blist(maxerr) = b1
02268       blist(last) = b2
02269       elist(maxerr) = error1
02270       elist(last) = error2
02271     else
02272       alist(maxerr) = a2
02273       alist(last) = a1
02274       blist(last) = b1
02275       rlist(maxerr) = area2
02276       rlist(last) = area1
02277       elist(maxerr) = error2
02278       elist(last) = error1
02279     end if
02280 !
02281 !  Call QSORT to maintain the descending ordering
02282 !  in the list of error estimates and select the subinterval
02283 !  with nrmax-th largest error estimate (to be bisected next).
02284 !
02285     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
02286
02287     if ( errsum <= errbnd ) go to 115
02288
02289     if ( ier /= 0 ) then
02290       exit
02291     end if
02292
02293     if ( last == 2 ) go to 80
02294     if ( noext ) go to 90
02295
02296     erlarg = erlarg-erlast
02297
02298     if ( abs(b1-a1) > small ) then
02299       erlarg = erlarg+erro12
02300     end if
02301 !
02302 !  Test whether the interval to be bisected next is the
02303 !  smallest interval.
02304 !
02305     if ( .not. extrap ) then
02306       if ( abs(blist(maxerr)-alist(maxerr)) > small ) go to 90
02307       extrap = .true.
02308       nrmax = 2
02309     end if
02310
02311 !40  continue
02312 !
02313 !  The smallest interval has the largest error.
02314 !  Before bisecting decrease the sum of the errors over the
02315 !  larger intervals (erlarg) and perform extrapolation.
02316 !
02317     if ( ierro /= 3 .and. erlarg > ertest ) then
02318
02319       id = nrmax
02320       jupbnd = last
02321
02322       if ( last > (2+limit/2) ) then
02323         jupbnd = limit+3-last
02324       end if
02325
02326       do k = id, jupbnd
02327         maxerr = iord(nrmax)
02328         errmax = elist(maxerr)
02329         if ( abs(blist(maxerr)-alist(maxerr)) > small ) then
02330           go to 90
02331         end if
02332         nrmax = nrmax+1
02333       end do
02334
02335     end if
02336 !
02337 !  Perform extrapolation.
02338 !
02339 !60  continue
02340
02341     numrl2 = numrl2+1
02342     rlist2(numrl2) = area
02343     call qextr ( numrl2, rlist2, reseps, abseps, res3la, nres )
02344     ktmin = ktmin+1
02345
02346     if ( ktmin > 5 .and. abserr < 1.0e-03 * errsum ) then
02347       ier = 5
```

```
02348     end if
02349
02350     if ( abseps < abserr ) then
02351
02352       ktmin = 0
02353       abserr = abseps
02354       result = reseps
02355       correc = erlarg
02356       ertest = max( epsabs,epsrel*abs(reseps))
02357
02358       if ( abserr <= ertest ) then
02359         exit
02360       end if
02361
02362     end if
02363 !
02364 !  Prepare bisection of the smallest interval.
02365 !
02366     if ( numrl2 == 1 ) then
02367       noext = .true.
02368     end if
02369
02370     if ( ier == 5 ) then
02371       exit
02372     end if
02373
02374     maxerr = iord(1)
02375     errmax = elist(maxerr)
02376     nrmax = 1
02377     extrap = .false.
02378     small = small * 5.0e-01
02379     erlarg = errsum
02380     go to 90
02381
02382 80  continue
02383
02384     small = abs( b - a ) * 3.75e-01
02385     erlarg = errsum
02386     ertest = errbnd
02387     rlist2(2) = area
02388
02389 90  continue
02390
02391   end do
02392 !
02393 !  Set final result and error estimate.
02394 !
02395   if ( abserr == huge( abserr ) ) then
02396     go to 115
02397   end if
02398
02399   if ( ier + ierro == 0 ) then
02400     go to 110
02401   end if
02402
02403   if ( ierro == 3 ) then
02404     abserr = abserr + correc
02405   end if
02406
02407   if ( ier == 0 ) then
02408     ier = 3
02409   end if
02410
02411   if ( result /= 0.0e+00.and.area /= 0.0e+00 ) then
02412     go to 105
02413   end if
02414
02415   if ( abserr > errsum ) go to 115
02416   if ( area == 0.0e+00 ) go to 130
02417   go to 110
02418
02419 105 continue
02420
02421   if ( abserr/abs(result) > errsum/abs(area) ) go to 115
02422 !
02423 !  Test on divergence.
02424 !
02425 110 continue
02426
02427   if ( ksgn == (-1).and.max( abs(result),abs(area)) <=  &
02428    defabs*1.0e-02 ) go to 130
02429
02430   if ( 1.0e-02 > (result/area) .or. (result/area) > 1.0e+02 &
02431    .or. errsum > abs(area) ) then
02432     ier = 6
02433   end if
02434
```

```
02435   go to 130
02436 !
02437 !  Compute global integral sum.
02438 !
02439 115 continue
02440
02441   result = sum( rlist(1:last) )
02442
02443   abserr = errsum
02444
02445 130 continue
02446
02447   if ( 2 < ier ) then
02448     ier = ier - 1
02449   end if
02450
02451 140 continue
02452
02453   neval = 42*last-21
02454
02455   return
```

Here is the call graph for this function:



14.37.1.7 qawc()

```
subroutine quadpack::qawc (
              real(kind=params_wp), external f,
              real(kind=params_wp) a,
              real(kind=params_wp) b,
              real(kind=params_wp) c,
              real(kind=params_wp) epsabs,
              real(kind=params_wp) epsrel,
              real(kind=params_wp) result,
              real(kind=params_wp) abserr,
              integer neval,
              integer ier )
```

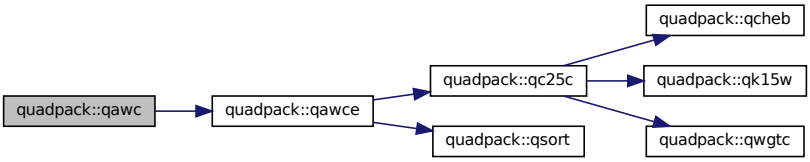
Definition at line 2457 of file quadpack.f90.

```
02458
02459 !*******************************************************************************80
02460 !
02461 !! QAWC computes a Cauchy principal value.
02462 !
02463 !  Discussion:
02464 !
02465 !    The routine calculates an approximation RESULT to a Cauchy principal
02466 !    value
02467 !      I = integral of F*W over (A,B),
02468 !    with
02469 !      W(X) = 1 / (X-C),
02470 !    with C distinct from A and B, hopefully satisfying
02471 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
02472 !
```

```
02473 !  Author:
02474 !
02475 !    Robert Piessens, Elise de Doncker-Kapenger,
02476 !    Christian Ueberhuber, David Kahaner
02477 !
02478 !  Reference:
02479 !
02480 !    Robert Piessens, Elise de Doncker-Kapenger,
02481 !    Christian Ueberhuber, David Kahaner,
02482 !    QUADPACK, a Subroutine Package for Automatic Integration,
02483 !    Springer Verlag, 1983
02484 !
02485 !  Parameters:
02486 !
02487 !    Input, external real F, the name of the function routine, of the form
02488 !      function f ( x )
02489 !      real f
02490 !      real x
02491 !    which evaluates the integrand function.
02492 !
02493 !    Input, real A, B, the limits of integration.
02494 !
02495 !    Input, real C, a parameter in the weight function, which must
02496 !    not be equal to A or B.
02497 !
02498 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
02499 !
02500 !    Output, real RESULT, the estimated value of the integral.
02501 !
02502 !    Output, real ABSERR, an estimate of || I - RESULT ||.
02503 !
02504 !    Output, integer NEVAL, the number of times the integral was evaluated.
02505 !
02506 !            ier    - integer
02507 !                     ier = 0 normal and reliable termination of the
02508 !                             routine. it is assumed that the requested
02509 !                             accuracy has been achieved.
02510 !                     ier > 0 abnormal termination of the routine
02511 !                             the estimates for integral and error are
02512 !                             less reliable. it is assumed that the
02513 !                             requested accuracy has not been achieved.
02514 !                     ier = 1 maximum number of subdivisions allowed
02515 !                             has been achieved. one can allow more sub-
02516 !                             divisions by increasing the data value of
02517 !                             limit in qawc (and taking the according
02518 !                             dimension adjustments into account).
02519 !                             however, if this yields no improvement it
02520 !                             is advised to analyze the integrand in
02521 !                             order to determine the integration
02522 !                             difficulties. if the position of a local
02523 !                             difficulty can be determined (e.g.
02524 !                             singularity, discontinuity within the
02525 !                             interval one will probably gain from
02526 !                             splitting up the interval at this point
02527 !                             and calling appropriate integrators on the
02528 !                             subranges.
02529 !                         = 2 the occurrence of roundoff error is detec-
02530 !                             ted, which prevents the requested
02531 !                             tolerance from being achieved.
02532 !                         = 3 extremely bad integrand behavior occurs
02533 !                             at some points of the integration
02534 !                             interval.
02535 !                         = 6 the input is invalid, because
02536 !                             c = a or c = b or
02537 !                             epsabs < 0 and epsrel < 0,
02538 !                             result, abserr, neval are set to zero.
02539 !
02540 !  Local parameters:
02541 !
02542 !    LIMIT is the maximum number of subintervals allowed in the
02543 !    subdivision process of qawce. take care that limit >= 1.
02544 !
02545   implicit none
02546
02547   integer, parameter :: limit = 500
02548
02549   real(kind=params_wp) a
02550   real(kind=params_wp) abserr
02551   real(kind=params_wp) alist(limit)
02552   real(kind=params_wp) b
02553   real(kind=params_wp) blist(limit)
02554   real(kind=params_wp) elist(limit)
02555   real(kind=params_wp) c
02556   real(kind=params_wp) epsabs
02557   real(kind=params_wp) epsrel
02558   real(kind=params_wp), external :: f
02559   integer ier
```

```
02560   integer iord(limit)
02561   integer last
02562   integer neval
02563   real(kind=params_wp) result
02564   real(kind=params_wp) rlist(limit)
02565
02566   call qawce ( f, a, b, c, epsabs, epsrel, limit, result, abserr, neval, ier, &
02567     alist, blist, rlist, elist, iord, last )
02568
02569   return
```

Here is the call graph for this function:



14.37.1.8 qawce()

```
subroutine quadpack::qawce (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) c,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) epsrel,
             integer limit,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             integer neval,
             integer ier,
             real(kind=params_wp), dimension(limit) alist,
             real(kind=params_wp), dimension(limit) blist,
             real(kind=params_wp), dimension(limit) rlist,
             real(kind=params_wp), dimension(limit) elist,
             integer, dimension(limit) iord,
             integer last )
```

Definition at line 2571 of file quadpack.f90.

```
02573
02574 !*****************************************************************************80
02575 !
02576 !! QAWCE computes a Cauchy principal value.
02577 !
02578 !  Discussion:
02579 !
02580 !    The routine calculates an approximation RESULT to a Cauchy principal
02581 !    value
02582 !      I = integral of F*W over (A,B),
02583 !    with
02584 !      W(X) = 1 / ( X - C ),
02585 !    with C distinct from A and B, hopefully satisfying
02586 !      | I - RESULT | <= max ( EPSABS, EPSREL * |I| ).
02587 !
02588 !  Author:
02589 !
02590 !    Robert Piessens, Elise de Doncker-Kapenger,
02591 !    Christian Ueberhuber, David Kahaner
02592 !
02593 !  Reference:
02594 !
02595 !    Robert Piessens, Elise de Doncker-Kapenger,
02596 !    Christian Ueberhuber, David Kahaner,
```

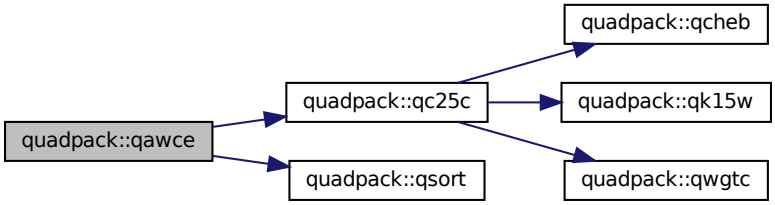
```
02597 !    QUADPACK, a Subroutine Package for Automatic Integration,
02598 !    Springer Verlag, 1983
02599 !
02600 !  Parameters:
02601 !
02602 !    Input, external real F, the name of the function routine, of the form
02603 !      function f ( x )
02604 !      real(kind=params_wp) f
02605 !      real x
02606 !    which evaluates the integrand function.
02607 !
02608 !    Input, real A, B, the limits of integration.
02609 !
02610 !    Input, real C, a parameter in the weight function, which cannot be
02611 !    equal to A or B.
02612 !
02613 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
02614 !
02615 !    Input, integer LIMIT, the upper bound on the number of subintervals that
02616 !    will be used in the partition of [A,B].  LIMIT is typically 500.
02617 !
02618 !    Output, real RESULT, the estimated value of the integral.
02619 !
02620 !    Output, real ABSERR, an estimate of || I - RESULT ||.
02621 !
02622 !    Output, integer NEVAL, the number of times the integral was evaluated.
02623 !
02624 !            ier    - integer
02625 !                     ier = 0 normal and reliable termination of the
02626 !                             routine. it is assumed that the requested
02627 !                             accuracy has been achieved.
02628 !                     ier > 0 abnormal termination of the routine
02629 !                             the estimates for integral and error are
02630 !                             less reliable. it is assumed that the
02631 !                             requested accuracy has not been achieved.
02632 !                     ier = 1 maximum number of subdivisions allowed
02633 !                             has been achieved. one can allow more sub-
02634 !                             divisions by increasing the value of
02635 !                             limit. however, if this yields no
02636 !                             improvement it is advised to analyze the
02637 !                             integrand, in order to determine the
02638 !                             integration difficulties.  if the position
02639 !                             of a local difficulty can be determined
02640 !                             (e.g. singularity, discontinuity within
02641 !                             the interval) one will probably gain
02642 !                             from splitting up the interval at this
02643 !                             point and calling appropriate integrators
02644 !                             on the subranges.
02645 !                         = 2 the occurrence of roundoff error is detec-
02646 !                             ted, which prevents the requested
02647 !                             tolerance from being achieved.
02648 !                         = 3 extremely bad integrand behavior occurs
02649 !                             at some interior points of the integration
02650 !                             interval.
02651 !                         = 6 the input is invalid, because
02652 !                             c = a or c = b or
02653 !                             epsabs < 0 and epsrel < 0,
02654 !                             or limit < 1.
02655 !                             result, abserr, neval, rlist(1), elist(1),
02656 !                             iord(1) and last are set to zero.
02657 !                             alist(1) and blist(1) are set to a and b
02658 !                             respectively.
02659 !
02660 !    Workspace, real ALIST(LIMIT), BLIST(LIMIT), contains in entries 1
02661 !    through LAST the left and right ends of the partition subintervals.
02662 !
02663 !    Workspace, real RLIST(LIMIT), contains in entries 1 through LAST
02664 !    the integral approximations on the subintervals.
02665 !
02666 !    Workspace, real ELIST(LIMIT), contains in entries 1 through LAST
02667 !    the absolute error estimates on the subintervals.
02668 !
02669 !            iord    - integer
02670 !                      vector of dimension at least limit, the first k
02671 !                      elements of which are pointers to the error
02672 !                      estimates over the subintervals, so that
02673 !                      elist(iord(1)), ...,  elist(iord(k)) with
02674 !                      k = last if last <= (limit/2+2), and
02675 !                      k = limit+1-last otherwise, form a decreasing
02676 !                      sequence.
02677 !
02678 !            last    - integer
02679 !                      number of subintervals actually produced in
02680 !                      the subdivision process
02681 !
02682 !  Local parameters:
02683 !
```

```
02684 !           alist     - list of left end points of all subintervals
02685 !                       considered up to now
02686 !           blist     - list of right end points of all subintervals
02687 !                       considered up to now
02688 !           rlist(i)  - approximation to the integral over
02689 !                       (alist(i),blist(i))
02690 !           elist(i)  - error estimate applying to rlist(i)
02691 !           maxerr    - pointer to the interval with largest error
02692 !                       estimate
02693 !           errmax    - elist(maxerr)
02694 !           area      - sum of the integrals over the subintervals
02695 !           errsum    - sum of the errors over the subintervals
02696 !           errbnd    - requested accuracy max(epsabs,epsrel*
02697 !                       abs(result))
02698 !           *****1    - variable for the left subinterval
02699 !           *****2    - variable for the right subinterval
02700 !           last      - index for subdivision
02701 !
02702   implicit none
02703
02704   integer limit
02705
02706   real(kind=params_wp) a
02707   real(kind=params_wp) aa
02708   real(kind=params_wp) abserr
02709   real(kind=params_wp) alist(limit)
02710   real(kind=params_wp) area
02711   real(kind=params_wp) area1
02712   real(kind=params_wp) area12
02713   real(kind=params_wp) area2
02714   real(kind=params_wp) a1
02715   real(kind=params_wp) a2
02716   real(kind=params_wp) b
02717   real(kind=params_wp) bb
02718   real(kind=params_wp) blist(limit)
02719   real(kind=params_wp) b1
02720   real(kind=params_wp) b2
02721   real(kind=params_wp) c
02722   real(kind=params_wp) elist(limit)
02723   real(kind=params_wp) epsabs
02724   real(kind=params_wp) epsrel
02725   real(kind=params_wp) errbnd
02726   real(kind=params_wp) errmax
02727   real(kind=params_wp) error1
02728   real(kind=params_wp) error2
02729   real(kind=params_wp) erro12
02730   real(kind=params_wp) errsum
02731   real(kind=params_wp), external :: f
02732   integer ier
02733   integer iord(limit)
02734   integer iroff1
02735   integer iroff2
02736   integer krule
02737   integer last
02738   integer maxerr
02739   integer nev
02740   integer neval
02741   integer nrmax
02742   real(kind=params_wp) result
02743   real(kind=params_wp) rlist(limit)
02744 !
02745 !  Test on validity of parameters.
02746 !
02747   ier = 0
02748   neval = 0
02749   last = 0
02750   alist(1) = a
02751   blist(1) = b
02752   rlist(1) = 0.0e+00
02753   elist(1) = 0.0e+00
02754   iord(1) = 0
02755   result = 0.0e+00
02756   abserr = 0.0e+00
02757
02758   if ( c == a  ) then
02759     ier = 6
02760     return
02761   else if ( c == b ) then
02762     ier = 6
02763     return
02764   else if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
02765     ier = 6
02766     return
02767   end if
02768 !
02769 !  First approximation to the integral.
02770 !
```

```
02771   if ( a <= b ) then
02772     aa = a
02773     bb = b
02774   else
02775     aa = b
02776     bb = a
02777   end if
02778
02779   krule = 1
02780   call qc25c ( f, aa, bb, c, result, abserr, krule, neval )
02781   last = 1
02782   rlist(1) = result
02783   elist(1) = abserr
02784   iord(1) = 1
02785   alist(1) = a
02786   blist(1) = b
02787 !
02788 !  Test on accuracy.
02789 !
02790   errbnd = max( epsabs, epsrel * abs(result) )
02791
02792   if ( limit == 1 ) then
02793     ier = 1
02794     go to 70
02795   end if
02796
02797   if ( abserr < min( 1.0e-02 * abs(result), errbnd)  ) then
02798     go to 70
02799   end if
02800 !
02801 !  Initialization
02802 !
02803   alist(1) = aa
02804   blist(1) = bb
02805   rlist(1) = result
02806   errmax = abserr
02807   maxerr = 1
02808   area = result
02809   errsum = abserr
02810   nrmax = 1
02811   iroff1 = 0
02812   iroff2 = 0
02813
02814   do last = 2, limit
02815 !
02816 !  Bisect the subinterval with nrmax-th largest error estimate.
02817 !
02818     a1 = alist(maxerr)
02819     b1 = 5.0e-01*(alist(maxerr)+blist(maxerr))
02820     b2 = blist(maxerr)
02821
02822     if ( c <= b1 .and. a1 < c ) then
02823       b1 = 5.0e-01*(c+b2)
02824     end if
02825
02826     if ( b1 < c .and. c < b2 ) then
02827       b1 = 5.0e-01 * ( a1 + c )
02828     end if
02829
02830     a2 = b1
02831     krule = 2
02832
02833     call qc25c ( f, a1, b1, c, area1, error1, krule, nev )
02834     neval = neval+nev
02835
02836     call qc25c ( f, a2, b2, c, area2, error2, krule, nev )
02837     neval = neval+nev
02838 !
02839 !  Improve previous approximations to integral and error
02840 !  and test for accuracy.
02841 !
02842     area12 = area1 + area2
02843     erro12 = error1 + error2
02844     errsum = errsum + erro12 - errmax
02845     area = area + area12 - rlist(maxerr)
02846
02847     if ( abs( rlist(maxerr)-area12) < 1.0e-05 * abs(area12) &
02848       .and. erro12 >= 9.9e-01 * errmax .and. krule == 0 ) &
02849       iroff1 = iroff1+1
02850
02851     if ( last > 10.and.erro12 > errmax .and. krule == 0 ) then
02852       iroff2 = iroff2+1
02853     end if
02854
02855     rlist(maxerr) = area1
02856     rlist(last) = area2
02857     errbnd = max( epsabs, epsrel * abs(area) )
```

```
02858
02859     if ( errsum > errbnd ) then
02860 !
02861 !  Test for roundoff error and eventually set error flag.
02862 !
02863       if ( iroff1 >= 6 .and. iroff2 > 20 ) then
02864         ier = 2
02865       end if
02866 !
02867 !  Set error flag in the case that number of interval
02868 !  bisections exceeds limit.
02869 !
02870       if ( last == limit ) then
02871         ier = 1
02872       end if
02873 !
02874 !  Set error flag in the case of bad integrand behavior at
02875 !  a point of the integration range.
02876 !
02877       if ( max( abs(a1), abs(b2) ) <= ( 1.0e+00_params_wp + 1.0e+03 * epsilon( a1 ) ) &
02878         *( abs(a2)+1.0e+03* tiny( a2 ) )) then
02879         ier = 3
02880       end if
02881
02882     end if
02883 !
02884 !  Append the newly-created intervals to the list.
02885 !
02886     if ( error2 <= error1 ) then
02887       alist(last) = a2
02888       blist(maxerr) = b1
02889       blist(last) = b2
02890       elist(maxerr) = error1
02891       elist(last) = error2
02892     else
02893       alist(maxerr) = a2
02894       alist(last) = a1
02895       blist(last) = b1
02896       rlist(maxerr) = area2
02897       rlist(last) = area1
02898       elist(maxerr) = error2
02899       elist(last) = error1
02900     end if
02901 !
02902 !  Call QSORT to maintain the descending ordering
02903 !  in the list of error estimates and select the subinterval
02904 !  with NRMAX-th largest error estimate (to be bisected next).
02905 !
02906     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
02907
02908     if ( ier /= 0 .or. errsum <= errbnd ) then
02909       exit
02910     end if
02911
02912   end do
02913 !
02914 !  Compute final result.
02915 !
02916   result = sum( rlist(1:last) )
02917
02918   abserr = errsum
02919
02920 70 continue
02921
02922   if ( aa == b ) then
02923     result = - result
02924   end if
02925
02926   return
```

Here is the call graph for this function:



Here is the caller graph for this function:

14.37.1.9 qawf()

```
subroutine quadpack::qawf (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) omega,
             integer integr,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             integer neval,
             integer ier )
```

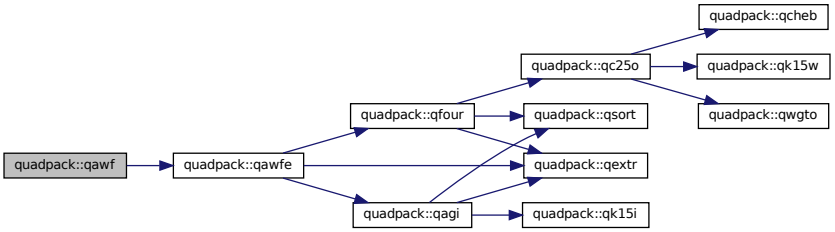
Definition at line 2928 of file quadpack.f90.

```
02929
02930 !*****************************************************************************80
02931 !
02932 !! QAWF computes Fourier integrals over the interval [ A, +Infinity ).
02933 !
02934 !  Discussion:
02935 !
02936 !    The routine calculates an approximation RESULT to a definite integral
02937 !
02938 !      I = integral of F*COS(OMEGA*X)
02939 !    or
02940 !      I = integral of F*SIN(OMEGA*X)
02941 !
02942 !    over the interval [A,+Infinity), hopefully satisfying
02943 !
02944 !      || I - RESULT || <= EPSABS.
02945 !
02946 !    If OMEGA = 0 and INTEGR = 1, the integral is calculated by means
02947 !    of QAGI, and IER has the meaning as described in the comments of QAGI.
02948 !
02949 !  Author:
02950 !
02951 !    Robert Piessens, Elise de Doncker-Kapenger,
02952 !    Christian Ueberhuber, David Kahaner
```

```
02953 !
02954 !  Reference:
02955 !
02956 !    Robert Piessens, Elise de Doncker-Kapenger,
02957 !    Christian Ueberhuber, David Kahaner,
02958 !    QUADPACK, a Subroutine Package for Automatic Integration,
02959 !    Springer Verlag, 1983
02960 !
02961 !  Parameters:
02962 !
02963 !    Input, external real F, the name of the function routine, of the form
02964 !      function f ( x )
02965 !      real(kind=params_wp) f
02966 !      real x
02967 !    which evaluates the integrand function.
02968 !
02969 !    Input, real A, the lower limit of integration.
02970 !
02971 !    Input, real OMEGA, the parameter in the weight function.
02972 !
02973 !    Input, integer INTEGR, indicates which weight functions is used
02974 !    = 1, w(x) = cos(omega*x)
02975 !    = 2, w(x) = sin(omega*x)
02976 !
02977 !    Input, real EPSABS, the absolute accuracy requested.
02978 !
02979 !    Output, real RESULT, the estimated value of the integral.
02980 !
02981 !    Output, real ABSERR, an estimate of || I - RESULT ||.
02982 !
02983 !    Output, integer NEVAL, the number of times the integral was evaluated.
02984 !
02985 !            ier    - integer
02986 !                     ier = 0 normal and reliable termination of the
02987 !                             routine. it is assumed that the
02988 !                             requested accuracy has been achieved.
02989 !                     ier > 0 abnormal termination of the routine.
02990 !                             the estimates for integral and error are
02991 !                             less reliable. it is assumed that the
02992 !                             requested accuracy has not been achieved.
02993 !                    if omega /= 0
02994 !                     ier = 6 the input is invalid because
02995 !                             (integr /= 1 and integr /= 2) or
02996 !                              epsabs <= 0
02997 !                              result, abserr, neval, lst are set to
02998 !                              zero.
02999 !                         = 7 abnormal termination of the computation
03000 !                             of one or more subintegrals
03001 !                         = 8 maximum number of cycles allowed
03002 !                             has been achieved, i.e. of subintervals
03003 !                             (a+(k-1)c,a+kc) where
03004 !                             c = (2*int(abs(omega))+1)*pi/abs(omega),
03005 !                             for k = 1, 2, ...
03006 !                         = 9 the extrapolation table constructed for
03007 !                             convergence acceleration of the series
03008 !                             formed by the integral contributions
03009 !                             over the cycles, does not converge to
03010 !                             within the requested accuracy.
03011 !
03012 !  Local parameters:
03013 !
03014 !    Integer LIMLST, gives an upper bound on the number of cycles, LIMLST >= 3.
03015 !    if limlst < 3, the routine will end with ier = 6.
03016 !
03017 !    Integer MAXP1, an upper bound on the number of Chebyshev moments which
03018 !    can be stored, i.e. for the intervals of lengths abs(b-a)*2**(-l),
03019 !    l = 0,1, ..., maxp1-2, maxp1 >= 1.  if maxp1 < 1, the routine will end
03020 !    with ier = 6.
03021 !
03022   implicit none
03023
03024   integer, parameter :: limit = 500
03025   integer, parameter :: limlst = 50
03026   integer, parameter :: maxp1 = 21
03027
03028   real(kind=params_wp) a
03029   real(kind=params_wp) abserr
03030   real(kind=params_wp) alist(limit)
03031   real(kind=params_wp) blist(limit)
03032   real(kind=params_wp) chebmo(maxp1,25)
03033   real(kind=params_wp) elist(limit)
03034   real(kind=params_wp) epsabs
03035   real(kind=params_wp) erlst(limlst)
03036   real(kind=params_wp), external :: f
03037   integer ier
03038   integer integr
03039   integer iord(limit)
```

```
03040   integer ierlst(limlst)
03041   integer lst
03042   integer neval
03043   integer nnlog(limit)
03044   real(kind=params_wp) omega
03045   real(kind=params_wp) result
03046   real(kind=params_wp) rlist(limit)
03047   real(kind=params_wp) rslst(limlst)
03048
03049   ier = 6
03050   neval = 0
03051   result = 0.0e+00
03052   abserr = 0.0e+00
03053
03054   if ( limlst < 3 .or. maxp1 < 1 ) then
03055     return
03056   end if
03057
03058   call qawfe ( f, a, omega, integr, epsabs, limlst, limit, maxp1, &
03059     result, abserr, neval, ier, rslst, erlst, ierlst, lst, alist, blist, &
03060     rlist, elist, iord, nnlog, chebmo )
03061
03062   return
```

Here is the call graph for this function:



14.37.1.10 qawfe()

```
subroutine quadpack::qawfe (
            real(kind=params_wp), external f,
            real(kind=params_wp) a,
            real(kind=params_wp) omega,
            integer integr,
            real(kind=params_wp) epsabs,
            integer limlst,
            integer limit,
            integer maxp1,
            real(kind=params_wp) result,
            real(kind=params_wp) abserr,
            integer neval,
            integer ier,
            real(kind=params_wp), dimension(limlst) rslst,
            real(kind=params_wp), dimension(limlst) erlst,
            integer, dimension(limlst) ierlst,
            integer lst,
            real(kind=params_wp), dimension(limit) alist,
            real(kind=params_wp), dimension(limit) blist,
            real(kind=params_wp), dimension(limit) rlist,
            real(kind=params_wp), dimension(limit) elist,
            integer, dimension(limit) iord,
            integer, dimension(limit) nnlog,
            real(kind=params_wp), dimension(maxp1,25) chebmo )
```

Definition at line 3064 of file quadpack.f90.

```
03067
03068 !****************************************************************************80
03069 !
03070 !! QAWFE computes Fourier integrals.
03071 !
03072 !  Discussion:
03073 !
03074 !    The routine calculates an approximation RESULT to a definite integral
03075 !      I = integral of F*COS(OMEGA*X) or F*SIN(OMEGA*X) over (A,+Infinity),
03076 !    hopefully satisfying
03077 !      || I - RESULT || <= EPSABS.
03078 !
03079 !  Author:
03080 !
03081 !    Robert Piessens, Elise de Doncker-Kapenger,
03082 !    Christian Ueberhuber, David Kahaner
03083 !
03084 !  Reference:
03085 !
03086 !    Robert Piessens, Elise de Doncker-Kapenger,
03087 !    Christian Ueberhuber, David Kahaner,
03088 !    QUADPACK, a Subroutine Package for Automatic Integration,
03089 !    Springer Verlag, 1983
03090 !
03091 !  Parameters:
03092 !
03093 !    Input, external real F, the name of the function routine, of the form
03094 !      function f ( x )
03095 !      real f
03096 !      real x
03097 !    which evaluates the integrand function.
03098 !
03099 !    Input, real A, the lower limit of integration.
03100 !
03101 !    Input, real OMEGA, the parameter in the weight function.
03102 !
03103 !    Input, integer INTEGR, indicates which weight function is used
03104 !    = 1      w(x) = cos(omega*x)
03105 !    = 2      w(x) = sin(omega*x)
03106 !
03107 !    Input, real EPSABS, the absolute accuracy requested.
03108 !
03109 !    Input, integer LIMLST, an upper bound on the number of cycles.
03110 !    LIMLST must be at least 1.  In fact, if LIMLST < 3, the routine
03111 !    will end with IER= 6.
03112 !
03113 !    Input, integer LIMIT, an upper bound on the number of subintervals
03114 !    allowed in the partition of each cycle, limit >= 1.
03115 !
03116 !            maxp1  - integer
03117 !                     gives an upper bound on the number of
03118 !                     Chebyshev moments which can be stored, i.e.
03119 !                     for the intervals of lengths abs(b-a)*2**(-l),
03120 !                     l=0,1, ..., maxp1-2, maxp1 >= 1
03121 !
03122 !    Output, real RESULT, the estimated value of the integral.
03123 !
03124 !    Output, real ABSERR, an estimate of || I - RESULT ||.
03125 !
03126 !    Output, integer NEVAL, the number of times the integral was evaluated.
03127 !
03128 !            ier    - ier = 0 normal and reliable termination of
03129 !                             the routine. it is assumed that the
03130 !                             requested accuracy has been achieved.
03131 !                     ier > 0 abnormal termination of the routine
03132 !                             the estimates for integral and error
03133 !                             are less reliable. it is assumed that
03134 !                             the requested accuracy has not been
03135 !                             achieved.
03136 !                   if omega /= 0
03137 !                    ier = 6 the input is invalid because
03138 !                             (integr /= 1 and integr /= 2) or
03139 !                              epsabs <= 0 or limlst < 3.
03140 !                             result, abserr, neval, lst are set
03141 !                             to zero.
03142 !                         = 7 bad integrand behavior occurs within
03143 !                             one or more of the cycles. location
03144 !                             and type of the difficulty involved
03145 !                             can be determined from the vector ierlst.
03146 !                             here lst is the number of cycles actually
03147 !                             needed (see below).
03148 !                             ierlst(k) = 1 the maximum number of
03149 !                                           subdivisions (= limit)
03150 !                                           has been achieved on the
03151 !                                           k th cycle.
03152 !                                       = 2 occurence of roundoff
03153 !                                           error is detected and
```

```
03154 !                                        prevents the tolerance
03155 !                                        imposed on the k th cycle
03156 !                                        from being acheived.
03157 !                                    = 3 extremely bad integrand
03158 !                                        behavior occurs at some
03159 !                                        points of the k th cycle.
03160 !                                    = 4 the integration procedure
03161 !                                        over the k th cycle does
03162 !                                        not converge (to within the
03163 !                                        required accuracy) due to
03164 !                                        roundoff in the
03165 !                                        extrapolation procedure
03166 !                                        invoked on this cycle. it
03167 !                                        is assumed that the result
03168 !                                        on this interval is the
03169 !                                        best which can be obtained.
03170 !                                    = 5 the integral over the k th
03171 !                                        cycle is probably divergent
03172 !                                        or slowly convergent. it
03173 !                                        must be noted that
03174 !                                        divergence can occur with
03175 !                                        any other value of
03176 !                                        ierlst(k).
03177 !                           = 8 maximum number of  cycles  allowed
03178 !                               has been achieved, i.e. of subintervals
03179 !                               (a+(k-1)c,a+kc) where
03180 !                               c = (2*int(abs(omega))+1)*pi/abs(omega),
03181 !                               for k = 1, 2, ..., lst.
03182 !                               one can allow more cycles by increasing
03183 !                               the value of limlst (and taking the
03184 !                               according dimension adjustments into
03185 !                               account).
03186 !                               examine the array iwork which contains
03187 !                               the error flags over the cycles, in order
03188 !                               to eventual look for local integration
03189 !                               difficulties.
03190 !                               if the position of a local difficulty can
03191 !                               be determined (e.g. singularity,
03192 !                               discontinuity within the interval)
03193 !                               one will probably gain from splitting
03194 !                               up the interval at this point and
03195 !                               calling appopriate integrators on the
03196 !                               subranges.
03197 !                           = 9 the extrapolation table constructed for
03198 !                               convergence acceleration of the series
03199 !                               formed by the integral contributions
03200 !                               over the cycles, does not converge to
03201 !                               within the required accuracy.
03202 !                               as in the case of ier = 8, it is advised
03203 !                               to examine the array iwork which contains
03204 !                               the error flags on the cycles.
03205 !                     if omega = 0 and integr = 1,
03206 !                     the integral is calculated by means of qagi
03207 !                     and ier = ierlst(1) (with meaning as described
03208 !                     for ierlst(k), k = 1).
03209 !
03210 !            rslst  - real
03211 !                     vector of dimension at least limlst
03212 !                     rslst(k) contains the integral contribution
03213 !                     over the interval (a+(k-1)c,a+kc) where
03214 !                     c = (2*int(abs(omega))+1)*pi/abs(omega),
03215 !                     k = 1, 2, ..., lst.
03216 !                     note that, if omega = 0, rslst(1) contains
03217 !                     the value of the integral over (a,infinity).
03218 !
03219 !            erlst  - real
03220 !                     vector of dimension at least limlst
03221 !                     erlst(k) contains the error estimate
03222 !                     corresponding with rslst(k).
03223 !
03224 !            ierlst - integer
03225 !                     vector of dimension at least limlst
03226 !                     ierlst(k) contains the error flag corresponding
03227 !                     with rslst(k). for the meaning of the local error
03228 !                     flags see description of output parameter ier.
03229 !
03230 !            lst    - integer
03231 !                     number of subintervals needed for the integration
03232 !                     if omega = 0 then lst is set to 1.
03233 !
03234 !            alist, blist, rlist, elist - real
03235 !                     vector of dimension at least limit,
03236 !
03237 !            iord, nnlog - integer
03238 !                     vector of dimension at least limit, providing
03239 !                     space for the quantities needed in the
03240 !                     subdivision process of each cycle
```

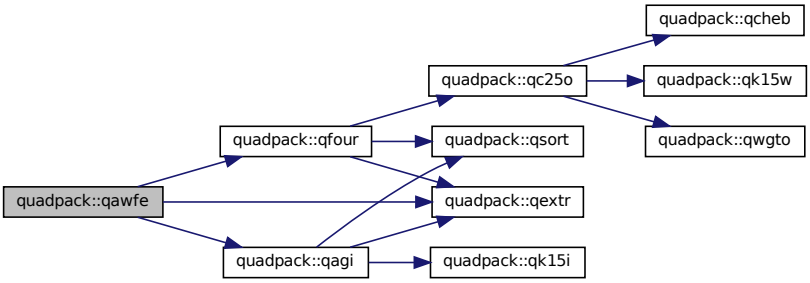
```
03241 !
03242 !            chebmo - real
03243 !                     array of dimension at least (maxp1,25),
03244 !                     providing space for the Chebyshev moments
03245 !                     needed within the cycles
03246 !
03247 !  Local parameters:
03248 !
03249 !           c1, c2    - end points of subinterval (of length
03250 !                       cycle)
03251 !           cycle     - (2*int(abs(omega))+1)*pi/abs(omega)
03252 !           psum      - vector of dimension at least (limexp+2)
03253 !                       (see routine qextr)
03254 !                       psum contains the part of the epsilon table
03255 !                       which is still needed for further computations.
03256 !                       each element of psum is a partial sum of
03257 !                       the series which should sum to the value of
03258 !                       the integral.
03259 !           errsum    - sum of error estimates over the
03260 !                       subintervals, calculated cumulatively
03261 !           epsa      - absolute tolerance requested over current
03262 !                       subinterval
03263 !           chebmo    - array containing the modified Chebyshev
03264 !                       moments (see also routine qc25o)
03265 !
03266   implicit none
03267
03268   integer limit
03269   integer limlst
03270   integer maxp1
03271
03272   real(kind=params_wp) a
03273   real(kind=params_wp) abseps
03274   real(kind=params_wp) abserr
03275   real(kind=params_wp) alist(limit)
03276   real(kind=params_wp) blist(limit)
03277   real(kind=params_wp) chebmo(maxp1,25)
03278   real(kind=params_wp) correc
03279   real(kind=params_wp) cycle
03280   real(kind=params_wp) c1
03281   real(kind=params_wp) c2
03282   real(kind=params_wp) dl
03283 ! real dla
03284   real(kind=params_wp) drl
03285   real(kind=params_wp) elist(limit)
03286   real(kind=params_wp) ep
03287   real(kind=params_wp) eps
03288   real(kind=params_wp) epsa
03289   real(kind=params_wp) epsabs
03290   real(kind=params_wp) erlst(limlst)
03291   real(kind=params_wp) errsum
03292   real(kind=params_wp), external :: f
03293   real(kind=params_wp) fact
03294   integer ier
03295   integer ierlst(limlst)
03296   integer integr
03297   integer iord(limit)
03298   integer ktmin
03299   integer l
03300   integer ll
03301   integer lst
03302   integer momcom
03303   integer nev
03304   integer neval
03305   integer nnlog(limit)
03306   integer nres
03307   integer numrl2
03308   real(kind=params_wp) omega
03309   real(kind=params_wp), parameter :: p = 0.9e+00
03310   real(kind=params_wp), parameter :: pi = 3.1415926535897932e+00
03311   real(kind=params_wp) p1
03312   real(kind=params_wp) psum(52)
03313   real(kind=params_wp) reseps
03314   real(kind=params_wp) result
03315   real(kind=params_wp) res3la(3)
03316   real(kind=params_wp) rlist(limit)
03317   real(kind=params_wp) rslst(limlst)
03318 !
03319 !  The dimension of  psum  is determined by the value of
03320 !  limexp in QEXTR (psum must be
03321 !  of dimension (limexp+2) at least).
03322 !
03323 !  Test on validity of parameters.
03324 !
03325   result = 0.0e+00
03326   abserr = 0.0e+00
03327   neval = 0
```

```
03328   lst = 0
03329   ier = 0
03330
03331   if ( (integr /= 1 .and. integr /= 2 ) .or. &
03332     epsabs <= 0.0e+00 .or. &
03333     limlst < 3 ) then
03334     ier = 6
03335     return
03336   end if
03337
03338   if ( omega == 0.0e+00 ) then
03339
03340     if ( integr == 1 ) then
03341       call qagi ( f, 0.0e+00_params_wp, 1, epsabs, 0.0e+00_params_wp, result, abserr, neval, ier )
03342     else
03343       result = 0.0e+00
03344       abserr = 0.0e+00
03345       neval = 0
03346       ier = 0
03347     end if
03348
03349     rslst(1) = result
03350     erlst(1) = abserr
03351     ierlst(1) = ier
03352     lst = 1
03353
03354     return
03355   end if
03356 !
03357 !  Initializations.
03358 !
03359   l = int( abs( omega ) )
03360   dl = 2 * l + 1
03361   cycle = dl * pi / abs( omega )
03362   ier = 0
03363   ktmin = 0
03364   neval = 0
03365   numrl2 = 0
03366   nres = 0
03367   c1 = a
03368   c2 = cycle+a
03369   p1 = 1.0e+00_params_wp-p
03370   eps = epsabs
03371
03372   if ( epsabs > tiny( epsabs ) / p1 ) then
03373     eps = epsabs * p1
03374   end if
03375
03376   ep = eps
03377   fact = 1.0e+00_params_wp
03378   correc = 0.0e+00
03379   abserr = 0.0e+00
03380   errsum = 0.0e+00
03381
03382   do lst = 1, limlst
03383 !
03384 !  Integrate over current subinterval.
03385 !
03386 !    dla = lst
03387     epsa = eps * fact
03388
03389     call qfour ( f, c1, c2, omega, integr, epsa, 0.0e+00_params_wp, limit, lst, maxp1, &
03390       rslst(lst), erlst(lst), nev, ierlst(lst), alist, blist, rlist, elist, &
03391       iord, nnlog, momcom, chebmo )
03392
03393     neval = neval + nev
03394     fact = fact * p
03395     errsum = errsum + erlst(lst)
03396     drl = 5.0e+01 * abs(rslst(lst))
03397 !
03398 !  Test on accuracy with partial sum.
03399 !
03400     if ((errsum+drl) <= epsabs.and.lst >= 6) then
03401       go to 80
03402     end if
03403
03404     correc = max( correc,erlst(lst))
03405
03406     if ( ierlst(lst) /= 0 ) then
03407       eps = max( ep,correc*p1)
03408       ier = 7
03409     end if
03410
03411     if ( ier == 7 .and. (errsum+drl) <= correc*1.0e+01.and. lst > 5) go to 80
03412
03413     numrl2 = numrl2+1
03414
```

```
03415     if ( lst <= 1 ) then
03416       psum(1) = rslst(1)
03417       go to 40
03418     end if
03419
03420     psum(numrl2) = psum(ll) + rslst(lst)
03421
03422     if ( lst == 2 ) then
03423       go to 40
03424     end if
03425 !
03426 !  Test on maximum number of subintervals
03427 !
03428     if ( lst == limlst ) then
03429       ier = 8
03430     end if
03431 !
03432 !  Perform new extrapolation
03433 !
03434     call qextr ( numrl2, psum, reseps, abseps, res3la, nres )
03435 !
03436 !  Test whether extrapolated result is influenced by roundoff
03437 !
03438     ktmin = ktmin + 1
03439
03440     if ( ktmin >= 15 .and. abserr <= 1.0e-03 * (errsum+drl) ) then
03441       ier = 9
03442     end  if
03443
03444     if ( abseps <= abserr .or. lst == 3 ) then
03445
03446       abserr = abseps
03447       result = reseps
03448       ktmin = 0
03449 !
03450 !  If IER is not 0, check whether direct result (partial
03451 !  sum) or extrapolated result yields the best integral
03452 !  approximation
03453 !
03454       if ( ( abserr + 1.0e+01 * correc ) <= epsabs ) then
03455         exit
03456       end if
03457
03458       if ( abserr <= epsabs .and. 1.0e+01 * correc >= epsabs ) then
03459         exit
03460       end if
03461
03462     end if
03463
03464     if ( ier /= 0 .and. ier /= 7 ) then
03465       exit
03466     end if
03467
03468 40  continue
03469
03470     ll = numrl2
03471     c1 = c2
03472     c2 = c2+cycle
03473
03474   end do
03475 !
03476 !  Set final result and error estimate.
03477 !
03478 !60 continue
03479
03480   abserr = abserr + 1.0e+01 * correc
03481
03482   if ( ier == 0 ) then
03483     return
03484   end if
03485
03486   if ( result /= 0.0e+00 .and. psum(numrl2) /= 0.0e+00) go to 70
03487
03488   if ( abserr > errsum ) then
03489     go to 80
03490   end if
03491
03492   if ( psum(numrl2) == 0.0e+00 ) then
03493     return
03494   end if
03495
03496 70 continue
03497
03498   if ( abserr / abs(result) <= (errsum+drl)/abs(psum(numrl2)) ) then
03499
03500     if ( ier >= 1 .and. ier /= 7 ) then
03501       abserr = abserr + drl
```

```
03502     end if
03503
03504     return
03505
03506   end if
03507
03508 80 continue
03509
03510   result = psum(numrl2)
03511   abserr = errsum + drl
03512
03513   return
```

Here is the call graph for this function:



Here is the caller graph for this function:

14.37.1.11 qawo()

```
subroutine quadpack::qawo (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) omega,
             integer integr,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) epsrel,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             integer neval,
             integer ier )
```

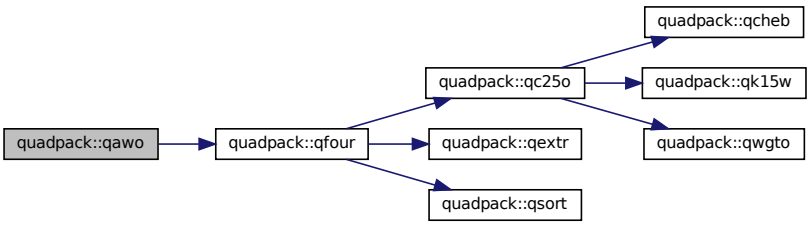
Definition at line 3515 of file quadpack.f90.

```
03517
03518 !***********************************************************************80
03519 !
03520 !! QAWO computes the integrals of oscillatory integrands.
03521 !
03522 !  Discussion:
03523 !
```

```
03524 !    The routine calculates an approximation RESULT to a given
03525 !    definite integral
03526 !      I = Integral ( A <= X <= B ) F(X) * cos ( OMEGA * X ) dx
03527 !    or
03528 !      I = Integral ( A <= X <= B ) F(X) * sin ( OMEGA * X ) dx
03529 !    hopefully satisfying following claim for accuracy
03530 !      | I - RESULT | <= max ( epsabs, epsrel * |I| ).
03531 !
03532 !  Author:
03533 !
03534 !    Robert Piessens, Elise de Doncker-Kapenger,
03535 !    Christian Ueberhuber, David Kahaner
03536 !
03537 !  Reference:
03538 !
03539 !    Robert Piessens, Elise de Doncker-Kapenger,
03540 !    Christian Ueberhuber, David Kahaner,
03541 !    QUADPACK, a Subroutine Package for Automatic Integration,
03542 !    Springer Verlag, 1983
03543 !
03544 !  Parameters:
03545 !
03546 !    Input, external real F, the name of the function routine, of the form
03547 !      function f ( x )
03548 !      real f
03549 !      real x
03550 !    which evaluates the integrand function.
03551 !
03552 !    Input, real A, B, the limits of integration.
03553 !
03554 !    Input, real OMEGA, the parameter in the weight function.
03555 !
03556 !    Input, integer INTEGR, specifies the weight function:
03557 !    1, W(X) = cos ( OMEGA * X )
03558 !    2, W(X) = sin ( OMEGA * X )
03559 !
03560 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
03561 !
03562 !    Output, real RESULT, the estimated value of the integral.
03563 !
03564 !    Output, real ABSERR, an estimate of || I - RESULT ||.
03565 !
03566 !    Output, integer NEVAL, the number of times the integral was evaluated.
03567 !
03568 !            ier    - integer
03569 !                     ier = 0 normal and reliable termination of the
03570 !                             routine. it is assumed that the
03571 !                             requested accuracy has been achieved.
03572 !                   - ier > 0 abnormal termination of the routine.
03573 !                             the estimates for integral and error are
03574 !                             less reliable. it is assumed that the
03575 !                             requested accuracy has not been achieved.
03576 !                     ier = 1 maximum number of subdivisions allowed
03577 !                             (= leniw/2) has been achieved. one can
03578 !                             allow more subdivisions by increasing the
03579 !                             value of leniw (and taking the according
03580 !                             dimension adjustments into account).
03581 !                             however, if this yields no improvement it
03582 !                             is advised to analyze the integrand in
03583 !                             order to determine the integration
03584 !                             difficulties. if the position of a local
03585 !                             difficulty can be determined (e.g.
03586 !                             singularity, discontinuity within the
03587 !                             interval) one will probably gain from
03588 !                             splitting up the interval at this point
03589 !                             and calling the integrator on the
03590 !                             subranges. if possible, an appropriate
03591 !                             special-purpose integrator should
03592 !                             be used which is designed for handling
03593 !                             the type of difficulty involved.
03594 !                         = 2 the occurrence of roundoff error is
03595 !                             detected, which prevents the requested
03596 !                             tolerance from being achieved.
03597 !                             the error may be under-estimated.
03598 !                         = 3 extremely bad integrand behavior occurs
03599 !                             at some interior points of the integration
03600 !                             interval.
03601 !                         = 4 the algorithm does not converge. roundoff
03602 !                             error is detected in the extrapolation
03603 !                             table. it is presumed that the requested
03604 !                             tolerance cannot be achieved due to
03605 !                             roundoff in the extrapolation table,
03606 !                             and that the returned result is the best
03607 !                             which can be obtained.
03608 !                         = 5 the integral is probably divergent, or
03609 !                             slowly convergent. it must be noted that
03610 !                             divergence can occur with any other value
```

```
03611 !                           of ier.
03612 !                       = 6 the input is invalid, because
03613 !                           epsabs < 0 and epsrel < 0,
03614 !                           result, abserr, neval are set to zero.
03615 !
03616 !  Local parameters:
03617 !
03618 !    limit is the maximum number of subintervals allowed in the
03619 !    subdivision process of QFOUR. take care that limit >= 1.
03620 !
03621 !    maxp1 gives an upper bound on the number of Chebyshev moments
03622 !    which can be stored, i.e. for the intervals of lengths
03623 !    abs(b-a)*2**(-l), l = 0, 1, ... , maxp1-2. take care that
03624 !    maxp1 >= 1.
03625
03626   implicit none
03627
03628   integer, parameter :: limit = 500
03629   integer, parameter :: maxp1 = 21
03630
03631   real(kind=params_wp) a
03632   real(kind=params_wp) abserr
03633   real(kind=params_wp) alist(limit)
03634   real(kind=params_wp) b
03635   real(kind=params_wp) blist(limit)
03636   real(kind=params_wp) chebmo(maxp1,25)
03637   real(kind=params_wp) elist(limit)
03638   real(kind=params_wp) epsabs
03639   real(kind=params_wp) epsrel
03640   real(kind=params_wp), external :: f
03641   integer ier
03642   integer integr
03643   integer iord(limit)
03644   integer momcom
03645   integer neval
03646   integer nnlog(limit)
03647   real(kind=params_wp) omega
03648   real(kind=params_wp) result
03649   real(kind=params_wp) rlist(limit)
03650
03651   call qfour ( f, a, b, omega, integr, epsabs, epsrel, limit, 1, maxp1, &
03652     result, abserr, neval, ier, alist, blist, rlist, elist, iord, nnlog, &
03653     momcom, chebmo )
03654
03655   return
```

Here is the call graph for this function:



14.37.1.12 qaws()

```
subroutine quadpack::qaws (
            real(kind=params_wp), external f,
            real(kind=params_wp) a,
            real(kind=params_wp) b,
            real(kind=params_wp) alfa,
            real(kind=params_wp) beta,
            integer integr,
            real(kind=params_wp) epsabs,
            real(kind=params_wp) epsrel,
            real(kind=params_wp) result,
```

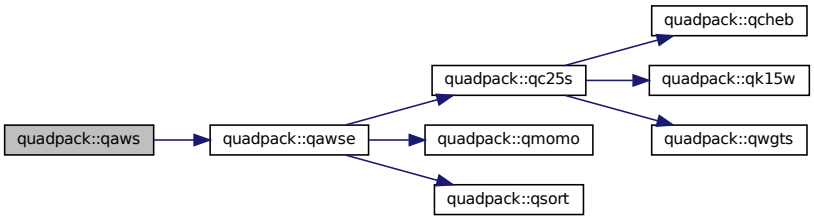
```
              real(kind=params_wp) abserr,
              integer neval,
              integer ier )
```

Definition at line 3657 of file quadpack.f90.

```
03659
03660 !****************************************************************************80
03661 !
03662 !! QAWS estimates integrals with algebraico-logarithmic endpoint singularities.
03663 !
03664 !  Discussion:
03665 !
03666 !    This routine calculates an approximation RESULT to a given
03667 !    definite integral
03668 !      I = integral of f*w over (a,b)
03669 !    where w shows a singular behavior at the end points, see parameter
03670 !    integr, hopefully satisfying following claim for accuracy
03671 !      abs(i-result) <= max(epsabs,epsrel*abs(i)).
03672 !
03673 !  Author:
03674 !
03675 !    Robert Piessens, Elise de Doncker-Kapenger,
03676 !    Christian Ueberhuber, David Kahaner
03677 !
03678 !  Reference:
03679 !
03680 !    Robert Piessens, Elise de Doncker-Kapenger,
03681 !    Christian Ueberhuber, David Kahaner,
03682 !    QUADPACK, a Subroutine Package for Automatic Integration,
03683 !    Springer Verlag, 1983
03684 !
03685 !  Parameters:
03686 !
03687 !    Input, external real F, the name of the function routine, of the form
03688 !      function f ( x )
03689 !      real f
03690 !      real x
03691 !    which evaluates the integrand function.
03692 !
03693 !    Input, real A, B, the limits of integration.
03694 !
03695 !    Input, real ALFA, BETA, parameters used in the weight function.
03696 !    ALFA and BETA should be greater than -1.
03697 !
03698 !    Input, integer INTEGR, indicates which weight function is to be used
03699 !    = 1  (x-a)**alfa*(b-x)**beta
03700 !    = 2  (x-a)**alfa*(b-x)**beta*log(x-a)
03701 !    = 3  (x-a)**alfa*(b-x)**beta*log(b-x)
03702 !    = 4  (x-a)**alfa*(b-x)**beta*log(x-a)*log(b-x)
03703 !
03704 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
03705 !
03706 !    Output, real RESULT, the estimated value of the integral.
03707 !
03708 !    Output, real ABSERR, an estimate of || I - RESULT ||.
03709 !
03710 !    Output, integer NEVAL, the number of times the integral was evaluated.
03711 !
03712 !            ier    - integer
03713 !                     ier = 0 normal and reliable termination of the
03714 !                             routine. it is assumed that the requested
03715 !                             accuracy has been achieved.
03716 !                     ier > 0 abnormal termination of the routine
03717 !                             the estimates for the integral and error
03718 !                             are less reliable. it is assumed that the
03719 !                             requested accuracy has not been achieved.
03720 !                     ier = 1 maximum number of subdivisions allowed
03721 !                             has been achieved. one can allow more
03722 !                             subdivisions by increasing the data value
03723 !                             of limit in qaws (and taking the according
03724 !                             dimension adjustments into account).
03725 !                             however, if this yields no improvement it
03726 !                             is advised to analyze the integrand, in
03727 !                             order to determine the integration
03728 !                             difficulties which prevent the requested
03729 !                             tolerance from being achieved. in case of
03730 !                             a jump discontinuity or a local
03731 !                             singularity of algebraico-logarithmic type
03732 !                             at one or more interior points of the
03733 !                             integration range, one should proceed by
03734 !                             splitting up the interval at these points
03735 !                             and calling the integrator on the
03736 !                             subranges.
03737 !                         = 2 the occurrence of roundoff error is
03738 !                             detected, which prevents the requested
03739 !                             tolerance from being achieved.
03740 !                         = 3 extremely bad integrand behavior occurs
```

```
03741 !                             at some points of the integration
03742 !                             interval.
03743 !                         = 6 the input is invalid, because
03744 !                             b <= a or alfa <= (-1) or beta <= (-1) or
03745 !                             integr < 1 or integr > 4 or
03746 !                             epsabs < 0 and epsrel < 0,
03747 !                             result, abserr, neval are set to zero.
03748 !
03749 !  Local parameters:
03750 !
03751 !    LIMIT is the maximum number of subintervals allowed in the
03752 !    subdivision process of qawse. take care that limit >= 2.
03753 !
03754   implicit none
03755
03756   integer, parameter :: limit = 500
03757
03758   real(kind=params_wp) a
03759   real(kind=params_wp) abserr
03760   real(kind=params_wp) alfa
03761   real(kind=params_wp) alist(limit)
03762   real(kind=params_wp) b
03763   real(kind=params_wp) blist(limit)
03764   real(kind=params_wp) beta
03765   real(kind=params_wp) elist(limit)
03766   real(kind=params_wp) epsabs
03767   real(kind=params_wp) epsrel
03768   real(kind=params_wp), external :: f
03769   integer ier
03770   integer integr
03771   integer iord(limit)
03772   integer last
03773   integer neval
03774   real(kind=params_wp) result
03775   real(kind=params_wp) rlist(limit)
03776
03777   call qawse ( f, a, b, alfa, beta, integr, epsabs, epsrel, limit, result, &
03778     abserr, neval, ier, alist, blist, rlist, elist, iord, last )
03779
03780   return
```

Here is the call graph for this function:



14.37.1.13 qawse()

```
subroutine quadpack::qawse (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) alfa,
             real(kind=params_wp) beta,
             integer integr,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) epsrel,
             integer limit,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             integer neval,
```

```
            integer ier,
            real(kind=params_wp), dimension(limit) alist,
            real(kind=params_wp), dimension(limit) blist,
            real(kind=params_wp), dimension(limit) rlist,
            real(kind=params_wp), dimension(limit) elist,
            integer, dimension(limit) iord,
            integer last )
```

Definition at line 3782 of file quadpack.f90.

```
03784
03785 !*****************************************************************************80
03786 !
03787 !! QAWSE estimates integrals with algebraico-logarithmic endpoint singularities.
03788 !
03789 !  Discussion:
03790 !
03791 !    This routine calculates an approximation RESULT to an integral
03792 !      I = integral of F(X) * W(X) over (a,b),
03793 !    where W(X) shows a singular behavior at the endpoints, hopefully
03794 !    satisfying:
03795 !      | I - RESULT | <= max ( epsabs, epsrel * |I| ).
03796 !
03797 !  Author:
03798 !
03799 !    Robert Piessens, Elise de Doncker-Kapenger,
03800 !    Christian Ueberhuber, David Kahaner
03801 !
03802 !  Reference:
03803 !
03804 !    Robert Piessens, Elise de Doncker-Kapenger,
03805 !    Christian Ueberhuber, David Kahaner,
03806 !    QUADPACK, a Subroutine Package for Automatic Integration,
03807 !    Springer Verlag, 1983
03808 !
03809 !  Parameters:
03810 !
03811 !    Input, external real F, the name of the function routine, of the form
03812 !      function f ( x )
03813 !      real f
03814 !      real x
03815 !    which evaluates the integrand function.
03816 !
03817 !    Input, real A, B, the limits of integration.
03818 !
03819 !    Input, real ALFA, BETA, parameters used in the weight function.
03820 !    ALFA and BETA should be greater than -1.
03821 !
03822 !    Input, integer INTEGR, indicates which weight function is used:
03823 !    = 1  (x-a)**alfa*(b-x)**beta
03824 !    = 2  (x-a)**alfa*(b-x)**beta*log(x-a)
03825 !    = 3  (x-a)**alfa*(b-x)**beta*log(b-x)
03826 !    = 4  (x-a)**alfa*(b-x)**beta*log(x-a)*log(b-x)
03827 !
03828 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
03829 !
03830 !    Input, integer LIMIT, an upper bound on the number of subintervals
03831 !    in the partition of (A,B), LIMIT >= 2.  If LIMIT < 2, the routine
03832 !     will end with IER = 6.
03833 !
03834 !    Output, real RESULT, the estimated value of the integral.
03835 !
03836 !    Output, real ABSERR, an estimate of || I - RESULT ||.
03837 !
03838 !    Output, integer NEVAL, the number of times the integral was evaluated.
03839 !
03840 !            ier    - integer
03841 !                     ier = 0 normal and reliable termination of the
03842 !                             routine. it is assumed that the requested
03843 !                             accuracy has been achieved.
03844 !                     ier > 0 abnormal termination of the routine
03845 !                             the estimates for the integral and error
03846 !                             are less reliable. it is assumed that the
03847 !                             requested accuracy has not been achieved.
03848 !                         = 1 maximum number of subdivisions allowed
03849 !                             has been achieved. one can allow more
03850 !                             subdivisions by increasing the value of
03851 !                             limit. however, if this yields no
03852 !                             improvement it is advised to analyze the
03853 !                             integrand, in order to determine the
03854 !                             integration difficulties which prevent
03855 !                             the requested tolerance from being
03856 !                             achieved. in case of a jump discontinuity
03857 !                             or a local singularity of algebraico-
03858 !                             logarithmic type at one or more interior
03859 !                             points of the integration range, one
```

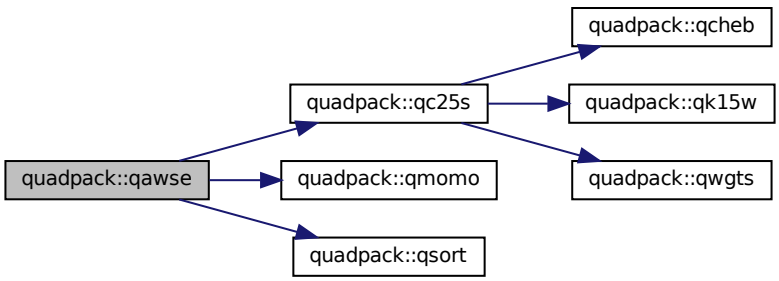
```
03860 !                             should proceed by splitting up the
03861 !                             interval at these points and calling the
03862 !                             integrator on the subranges.
03863 !                           = 2 the occurrence of roundoff error is
03864 !                             detected, which prevents the requested
03865 !                             tolerance from being achieved.
03866 !                           = 3 extremely bad integrand behavior occurs
03867 !                             at some points of the integration
03868 !                             interval.
03869 !                           = 6 the input is invalid, because
03870 !                             b <= a or alfa <= (-1) or beta <= (-1) or
03871 !                             integr < 1 or integr > 4, or
03872 !                             epsabs < 0 and epsrel < 0,
03873 !                             or limit < 2.
03874 !                             result, abserr, neval, rlist(1), elist(1),
03875 !                             iord(1) and last are set to zero.
03876 !                             alist(1) and blist(1) are set to a and b
03877 !                             respectively.
03878 !
03879 !    Workspace, real ALIST(LIMIT), BLIST(LIMIT), contains in entries 1
03880 !    through LAST the left and right ends of the partition subintervals.
03881 !
03882 !    Workspace, real RLIST(LIMIT), contains in entries 1 through LAST
03883 !    the integral approximations on the subintervals.
03884 !
03885 !    Workspace, real ELIST(LIMIT), contains in entries 1 through LAST
03886 !    the absolute error estimates on the subintervals.
03887 !
03888 !            iord   - integer
03889 !                     vector of dimension at least limit, the first k
03890 !                     elements of which are pointers to the error
03891 !                     estimates over the subintervals, so that
03892 !                     elist(iord(1)), ..., elist(iord(k)) with k = last
03893 !                     if last <= (limit/2+2), and k = limit+1-last
03894 !                     otherwise, form a decreasing sequence.
03895 !
03896 !    Output, integer LAST, the number of subintervals actually produced in
03897 !    the subdivision process.
03898 !
03899 !  Local parameters:
03900 !
03901 !           alist     - list of left end points of all subintervals
03902 !                       considered up to now
03903 !           blist     - list of right end points of all subintervals
03904 !                       considered up to now
03905 !           rlist(i)  - approximation to the integral over
03906 !                       (alist(i),blist(i))
03907 !           elist(i)  - error estimate applying to rlist(i)
03908 !           maxerr    - pointer to the interval with largest error
03909 !                       estimate
03910 !           errmax    - elist(maxerr)
03911 !           area      - sum of the integrals over the subintervals
03912 !           errsum    - sum of the errors over the subintervals
03913 !           errbnd    - requested accuracy max(epsabs,epsrel*
03914 !                       abs(result))
03915 !           *****1    - variable for the left subinterval
03916 !           *****2    - variable for the right subinterval
03917 !           last      - index for subdivision
03918 !
03919   implicit none
03920
03921   integer limit
03922
03923   real(kind=params_wp) a
03924   real(kind=params_wp) abserr
03925   real(kind=params_wp) alfa
03926   real(kind=params_wp) alist(limit)
03927   real(kind=params_wp) area
03928   real(kind=params_wp) area1
03929   real(kind=params_wp) area12
03930   real(kind=params_wp) area2
03931   real(kind=params_wp) a1
03932   real(kind=params_wp) a2
03933   real(kind=params_wp) b
03934   real(kind=params_wp) beta
03935   real(kind=params_wp) blist(limit)
03936   real(kind=params_wp) b1
03937   real(kind=params_wp) b2
03938   real(kind=params_wp) centre
03939   real(kind=params_wp) elist(limit)
03940   real(kind=params_wp) epsabs
03941   real(kind=params_wp) epsrel
03942   real(kind=params_wp) errbnd
03943   real(kind=params_wp) errmax
03944   real(kind=params_wp) error1
03945   real(kind=params_wp) erro12
03946   real(kind=params_wp) error2
```

```
03947   real(kind=params_wp) errsum
03948   real(kind=params_wp), external :: f
03949   integer ier
03950   integer integr
03951   integer iord(limit)
03952   integer iroff1
03953   integer iroff2
03954   integer last
03955   integer maxerr
03956   integer nev
03957   integer neval
03958   integer nrmax
03959   real(kind=params_wp) resas1
03960   real(kind=params_wp) resas2
03961   real(kind=params_wp) result
03962   real(kind=params_wp) rg(25)
03963   real(kind=params_wp) rh(25)
03964   real(kind=params_wp) ri(25)
03965   real(kind=params_wp) rj(25)
03966   real(kind=params_wp) rlist(limit)
03967 !
03968 !  Test on validity of parameters.
03969 !
03970   ier = 0
03971   neval = 0
03972   last = 0
03973   rlist(1) = 0.0e+00
03974   elist(1) = 0.0e+00
03975   iord(1) = 0
03976   result = 0.0e+00
03977   abserr = 0.0e+00
03978
03979   if ( b <= a .or. &
03980     (epsabs < 0.0e+00 .and. epsrel < 0.0e+00) .or. &
03981     alfa <= (-1.0e+00_params_wp) .or. &
03982     beta <= (-1.0e+00_params_wp) .or. &
03983     integr < 1 .or. &
03984     integr > 4 .or. &
03985     limit < 2 ) then
03986     ier = 6
03987     return
03988   end if
03989 !
03990 !  Compute the modified Chebyshev moments.
03991 !
03992   call qmomo ( alfa, beta, ri, rj, rg, rh, integr )
03993 !
03994 !  Integrate over the intervals (a,(a+b)/2) and ((a+b)/2,b).
03995 !
03996   centre = 5.0e-01 * ( b + a )
03997
03998   call qc25s ( f, a, b, a, centre, alfa, beta, ri, rj, rg, rh, area1, &
03999     error1, resas1, integr, nev )
04000
04001   neval = nev
04002
04003   call qc25s ( f, a, b, centre, b, alfa, beta, ri, rj, rg, rh, area2, &
04004     error2, resas2, integr, nev )
04005
04006   last = 2
04007   neval = neval+nev
04008   result = area1+area2
04009   abserr = error1+error2
04010 !
04011 !  Test on accuracy.
04012 !
04013   errbnd = max( epsabs, epsrel * abs( result ) )
04014 !
04015 !  Initialization.
04016 !
04017   if ( error2 <= error1 ) then
04018     alist(1) = a
04019     alist(2) = centre
04020     blist(1) = centre
04021     blist(2) = b
04022     rlist(1) = area1
04023     rlist(2) = area2
04024     elist(1) = error1
04025     elist(2) = error2
04026   else
04027     alist(1) = centre
04028     alist(2) = a
04029     blist(1) = b
04030     blist(2) = centre
04031     rlist(1) = area2
04032     rlist(2) = area1
04033     elist(1) = error2
```

```
04034       elist(2) = error1
04035     end if
04036
04037     iord(1) = 1
04038     iord(2) = 2
04039
04040     if ( limit == 2 ) then
04041       ier = 1
04042       return
04043     end if
04044
04045     if ( abserr <= errbnd ) then
04046       return
04047     end if
04048
04049     errmax = elist(1)
04050     maxerr = 1
04051     nrmax = 1
04052     area = result
04053     errsum = abserr
04054     iroff1 = 0
04055     iroff2 = 0
04056
04057     do last = 3, limit
04058   !
04059   !  Bisect the subinterval with largest error estimate.
04060   !
04061       a1 = alist(maxerr)
04062       b1 = 5.0e-01 * ( alist(maxerr) + blist(maxerr) )
04063       a2 = b1
04064       b2 = blist(maxerr)
04065
04066       call qc25s ( f, a, b, a1, b1, alfa, beta, ri, rj, rg, rh, area1, &
04067         error1, resas1, integr, nev )
04068
04069       neval = neval + nev
04070
04071       call qc25s ( f, a, b, a2, b2, alfa, beta, ri, rj, rg, rh, area2, &
04072         error2, resas2, integr, nev )
04073
04074       neval = neval + nev
04075   !
04076   !  Improve previous approximations integral and error and
04077   !  test for accuracy.
04078   !
04079       area12 = area1+area2
04080       erro12 = error1+error2
04081       errsum = errsum+erro12-errmax
04082       area = area+area12-rlist(maxerr)
04083   !
04084   !  Test for roundoff error.
04085   !
04086       if ( a /= a1 .and. b /= b2 ) then
04087
04088         if ( resas1 /= error1 .and. resas2 /= error2 ) then
04089
04090           if ( abs( rlist(maxerr) - area12 ) < 1.0e-05 * abs( area12 ) &
04091             .and.erro12 >= 9.9e-01*errmax ) then
04092             iroff1 = iroff1 + 1
04093           end if
04094
04095           if ( last > 10 .and. erro12 > errmax ) then
04096             iroff2 = iroff2 + 1
04097           end if
04098
04099         end if
04100
04101       end if
04102
04103       rlist(maxerr) = area1
04104       rlist(last) = area2
04105   !
04106   !  Test on accuracy.
04107   !
04108       errbnd = max( epsabs, epsrel * abs( area ) )
04109
04110       if ( errsum > errbnd ) then
04111   !
04112   !  Set error flag in the case that the number of interval
04113   !  bisections exceeds limit.
04114   !
04115         if ( last == limit ) then
04116           ier = 1
04117         end if
04118   !
04119   !  Set error flag in the case of roundoff error.
04120   !
```

```
04121       if ( iroff1 >= 6 .or. iroff2 >= 20 ) then
04122         ier = 2
04123       end if
04124 !
04125 !  Set error flag in the case of bad integrand behavior
04126 !  at interior points of integration range.
04127 !
04128       if ( max( abs(a1),abs(b2)) <= (1.0e+00_params_wp+1.0e+03* epsilon( a1 ) )* &
04129         ( abs(a2) + 1.0e+03* tiny( a2) )) then
04130         ier = 3
04131       end if
04132
04133     end if
04134 !
04135 !  Append the newly-created intervals to the list.
04136 !
04137     if ( error2 <= error1 ) then
04138       alist(last) = a2
04139       blist(maxerr) = b1
04140       blist(last) = b2
04141       elist(maxerr) = error1
04142       elist(last) = error2
04143     else
04144       alist(maxerr) = a2
04145       alist(last) = a1
04146       blist(last) = b1
04147       rlist(maxerr) = area2
04148       rlist(last) = area1
04149       elist(maxerr) = error2
04150       elist(last) = error1
04151     end if
04152 !
04153 !  Call QSORT to maintain the descending ordering
04154 !  in the list of error estimates and select the subinterval
04155 !  with largest error estimate (to be bisected next).
04156 !
04157     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
04158
04159     if ( ier /= 0 .or. errsum <= errbnd ) then
04160       exit
04161     end if
04162
04163   end do
04164 !
04165 !  Compute final result.
04166 !
04167   result = sum( rlist(1:last) )
04168
04169   abserr = errsum
04170
04171   return
```

Here is the call graph for this function:



Here is the caller graph for this function:

14.37.1.14 qc25c()

```
subroutine quadpack::qc25c (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) c,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             integer krul,
             integer neval )
```

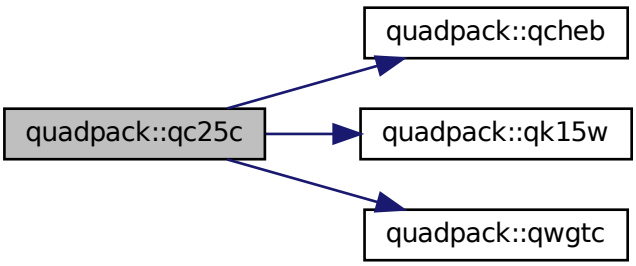
Definition at line 4173 of file quadpack.f90.

```
04174
04175 !*****************************************************************************80
04176 !
04177 !! QC25C returns integration rules for Cauchy Principal Value integrals.
04178 !
04179 !  Discussion:
04180 !
04181 !    This routine estimates
04182 !      I = integral of F(X) * W(X) over (a,b)
04183 !    with error estimate, where
04184 !      w(x) = 1/(x-c)
04185 !
04186 !  Author:
04187 !
04188 !    Robert Piessens, Elise de Doncker-Kapenger,
04189 !    Christian Ueberhuber, David Kahaner
04190 !
04191 !  Reference:
04192 !
04193 !    Robert Piessens, Elise de Doncker-Kapenger,
04194 !    Christian Ueberhuber, David Kahaner,
04195 !    QUADPACK, a Subroutine Package for Automatic Integration,
04196 !    Springer Verlag, 1983
04197 !
04198 !  Parameters:
04199 !
04200 !    Input, external real F, the name of the function routine, of the form
04201 !      function f ( x )
04202 !      real f
04203 !      real x
04204 !    which evaluates the integrand function.
04205 !
04206 !    Input, real A, B, the limits of integration.
04207 !
04208 !    Input, real C, the parameter in the weight function.
04209 !
04210 !    Output, real RESULT, the estimated value of the integral.
04211 !    RESULT is computed by using a generalized Clenshaw-Curtis method if
04212 !    C lies within ten percent of the integration interval.  In the
04213 !    other case the 15-point Kronrod rule obtained by optimal addition
04214 !    of abscissae to the 7-point Gauss rule, is applied.
04215 !
04216 !    Output, real ABSERR, an estimate of || I - RESULT ||.
04217 !
04218 !           krul   - integer
04219 !                    key which is decreased by 1 if the 15-point
04220 !                    Gauss-Kronrod scheme has been used
04221 !
04222 !    Output, integer NEVAL, the number of times the integral was evaluated.
```

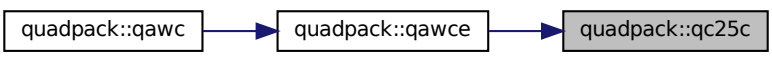
```
04223 !
04224 !  Local parameters:
04225 !
04226 !           fval   - value of the function f at the points
04227 !                    cos(k*pi/24),  k = 0, ..., 24
04228 !           cheb12 - Chebyshev series expansion coefficients, for the
04229 !                    function f, of degree 12
04230 !           cheb24 - Chebyshev series expansion coefficients, for the
04231 !                    function f, of degree 24
04232 !           res12  - approximation to the integral corresponding to the
04233 !                    use of cheb12
04234 !           res24  - approximation to the integral corresponding to the
04235 !                    use of cheb24
04236 !           qwgtc  - external function subprogram defining the weight
04237 !                    function
04238 !           hlgth  - half-length of the interval
04239 !           centr  - mid point of the interval
04240 !
04241   implicit none
04242
04243   real(kind=params_wp) a
04244   real(kind=params_wp) abserr
04245   real(kind=params_wp) ak22
04246   real(kind=params_wp) amom0
04247   real(kind=params_wp) amom1
04248   real(kind=params_wp) amom2
04249   real(kind=params_wp) b
04250   real(kind=params_wp) c
04251   real(kind=params_wp) cc
04252   real(kind=params_wp) centr
04253   real(kind=params_wp) cheb12(13)
04254   real(kind=params_wp) cheb24(25)
04255   real(kind=params_wp), external :: f
04256   real(kind=params_wp) fval(25)
04257   real(kind=params_wp) hlgth
04258   integer i
04259   integer isym
04260   integer k
04261   integer kp
04262   integer krul
04263   integer neval
04264   real(kind=params_wp) p2
04265   real(kind=params_wp) p3
04266   real(kind=params_wp) p4
04267 !  real(kind=params_wp), external :: qwgtc
04268   real(kind=params_wp) resabs
04269   real(kind=params_wp) resasc
04270   real(kind=params_wp) result
04271   real(kind=params_wp) res12
04272   real(kind=params_wp) res24
04273   real(kind=params_wp) u
04274   real(kind=params_wp), parameter, dimension ( 11 ) :: x = (/ &
04275     9.914448613738104e-01, 9.659258262890683e-01, &
04276     9.238795325112868e-01, 8.660254037844386e-01, &
04277     7.933533402912352e-01, 7.071067811865475e-01, &
04278     6.087614290087206e-01, 5.000000000000000e-01, &
04279     3.826834323650898e-01, 2.588190451025208e-01, &
04280     1.305261922200516e-01 /)
04281 !
04282 !  Check the position of C.
04283 !
04284   cc = ( 2.0e+00 * c - b - a ) / ( b - a )
04285 !
04286 !  Apply the 15-point Gauss-Kronrod scheme.
04287 !
04288   if ( abs( cc ) >= 1.1e+00 ) then
04289     krul = krul - 1
04290     call qk15w ( f, qwgtc, c, p2, p3, p4, kp, a, b, result, abserr, &
04291       resabs, resasc )
04292     neval = 15
04293     if ( resasc == abserr ) then
04294       krul = krul+1
04295     end if
04296     return
04297   end if
04298 !
04299 !  Use the generalized Clenshaw-Curtis method.
04300 !
04301   hlgth = 5.0e-01 * ( b - a )
04302   centr = 5.0e-01 * ( b + a )
04303   neval = 25
04304   fval(1) = 5.0e-01 * f(hlgth+centr)
04305   fval(13) = f(centr)
04306   fval(25) = 5.0e-01 * f(centr-hlgth)
04307
04308   do i = 2, 12
04309     u = hlgth * x(i-1)
```

```
04310     isym = 26 - i
04311     fval(i) = f(u+centr)
04312     fval(isym) = f(centr-u)
04313   end do
04314 !
04315 !  Compute the Chebyshev series expansion.
04316 !
04317   call qcheb ( x, fval, cheb12, cheb24 )
04318 !
04319 !  The modified Chebyshev moments are computed by forward
04320 !  recursion, using AMOM0 and AMOM1 as starting values.
04321 !
04322   amom0 = log( abs( ( 1.0e+00_params_wp - cc ) / ( 1.0e+00_params_wp + cc ) ) )
04323   amom1 = 2.0e+00 + cc * amom0
04324   res12 = cheb12(1) * amom0 + cheb12(2) * amom1
04325   res24 = cheb24(1) * amom0 + cheb24(2) * amom1
04326
04327   do k = 3, 13
04328     amom2 = 2.0e+00 * cc * amom1 - amom0
04329     ak22 = ( k - 2 ) * ( k - 2 )
04330     if ( ( k / 2 ) * 2 == k ) then
04331       amom2 = amom2 - 4.0e+00 / ( ak22 - 1.0e+00_params_wp )
04332     end if
04333     res12 = res12 + cheb12(k) * amom2
04334     res24 = res24 + cheb24(k) * amom2
04335     amom0 = amom1
04336     amom1 = amom2
04337   end do
04338
04339   do k = 14, 25
04340     amom2 = 2.0e+00 * cc * amom1 - amom0
04341     ak22 = ( k - 2 ) * ( k - 2 )
04342     if ( ( k / 2 ) * 2 == k ) then
04343       amom2 = amom2 - 4.0e+00 / ( ak22 - 1.0e+00_params_wp )
04344     end if
04345     res24 = res24 + cheb24(k) * amom2
04346     amom0 = amom1
04347     amom1 = amom2
04348   end do
04349
04350   result = res24
04351   abserr = abs( res24 - res12 )
04352
04353   return
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.37.1.15 qc25o()

```
subroutine quadpack::qc25o (
              real(kind=params_wp), external f,
              real(kind=params_wp) a,
              real(kind=params_wp) b,
              real(kind=params_wp) omega,
              integer integr,
              integer nrmom,
              integer maxp1,
              integer ksave,
              real(kind=params_wp) result,
              real(kind=params_wp) abserr,
              integer neval,
              real(kind=params_wp) resabs,
              real(kind=params_wp) resasc,
              integer momcom,
              real(kind=params_wp), dimension(maxp1,25) chebmo )
```

Definition at line 4355 of file quadpack.f90.

```
04357
04358 !*****************************************************************************80
04359 !
04360 !! QC25O returns integration rules for integrands with a COS or SIN factor.
04361 !
04362 !  Discussion:
04363 !
04364 !    This routine estimates the integral
04365 !      I = integral of f(x) * w(x) over (a,b)
04366 !    where
04367 !      w(x) = cos(omega*x)
04368 !    or
04369 !      w(x) = sin(omega*x),
04370 !    and estimates
04371 !      J = integral ( A <= X <= B ) |F(X)| dx.
04372 !
04373 !    For small values of OMEGA or small intervals (a,b) the 15-point
04374 !    Gauss-Kronrod rule is used.  In all other cases a generalized
04375 !    Clenshaw-Curtis method is used, that is, a truncated Chebyshev
04376 !    expansion of the function F is computed on (a,b), so that the
04377 !    integrand can be written as a sum of terms of the form W(X)*T(K,X),
04378 !    where T(K,X) is the Chebyshev polynomial of degree K.  The Chebyshev
04379 !    moments are computed with use of a linear recurrence relation.
04380 !
04381 !  Author:
04382 !
04383 !    Robert Piessens, Elise de Doncker-Kapenger,
04384 !    Christian Ueberhuber, David Kahaner
04385 !
04386 !  Reference:
04387 !
04388 !    Robert Piessens, Elise de Doncker-Kapenger,
04389 !    Christian Ueberhuber, David Kahaner,
04390 !    QUADPACK, a Subroutine Package for Automatic Integration,
04391 !    Springer Verlag, 1983
04392 !
04393 !  Parameters:
04394 !
04395 !    Input, external real F, the name of the function routine, of the form
04396 !      function f ( x )
04397 !      real f
04398 !      real x
04399 !    which evaluates the integrand function.
04400 !
04401 !    Input, real A, B, the limits of integration.
04402 !
04403 !    Input, real OMEGA, the parameter in the weight function.
04404 !
04405 !    Input, integer INTEGR, indicates which weight function is to be used
04406 !    = 1, w(x) = cos(omega*x)
04407 !    = 2, w(x) = sin(omega*x)
04408 !
04409 !    ?, integer NRMOM, the length of interval (a,b) is equal to the length
04410 !    of the original integration interval divided by
04411 !    2**nrmom (we suppose that the routine is used in an
04412 !    adaptive integration process, otherwise set
```

```
04413 !    nrmom = 0).  nrmom must be zero at the first call.
04414 !
04415 !           maxp1  - integer
04416 !                    gives an upper bound on the number of Chebyshev
04417 !                    moments which can be stored, i.e. for the intervals
04418 !                    of lengths abs(bb-aa)*2**(-l), l = 0,1,2, ...,
04419 !                    maxp1-2.
04420 !
04421 !           ksave  - integer
04422 !                    key which is one when the moments for the
04423 !                    current interval have been computed
04424 !
04425 !    Output, real RESULT, the estimated value of the integral.
04426 !
04427 !           abserr - real
04428 !                    estimate of the modulus of the absolute
04429 !                    error, which should equal or exceed abs(i-result)
04430 !
04431 !    Output, integer NEVAL, the number of times the integral was evaluated.
04432 !
04433 !    Output, real RESABS, approximation to the integral J.
04434 !
04435 !    Output, real RESASC, approximation to the integral of abs(F-I/(B-A)).
04436 !
04437 !         on entry and return
04438 !           momcom - integer
04439 !                    for each interval length we need to compute
04440 !                    the Chebyshev moments. momcom counts the number
04441 !                    of intervals for which these moments have already
04442 !                    been computed. if nrmom < momcom or ksave = 1,
04443 !                    the Chebyshev moments for the interval (a,b)
04444 !                    have already been computed and stored, otherwise
04445 !                    we compute them and we increase momcom.
04446 !
04447 !           chebmo - real
04448 !                    array of dimension at least (maxp1,25) containing
04449 !                    the modified Chebyshev moments for the first momcom
04450 !                    interval lengths
04451 !
04452 !  Local parameters:
04453 !
04454 !    maxp1 gives an upper bound
04455 !           on the number of Chebyshev moments which can be
04456 !           computed, i.e. for the interval (bb-aa), ...,
04457 !           (bb-aa)/2**(maxp1-2).
04458 !           should this number be altered, the first dimension of
04459 !           chebmo needs to be adapted.
04460 !
04461 !    x contains the values cos(k*pi/24)
04462 !           k = 1, ...,11, to be used for the Chebyshev expansion of f
04463 !
04464 !           centr  - mid point of the integration interval
04465 !           hlgth  - half length of the integration interval
04466 !           fval   - value of the function f at the points
04467 !                    (b-a)*0.5*cos(k*pi/12) + (b+a)*0.5
04468 !                    k = 0, ...,24
04469 !           cheb12 - coefficients of the Chebyshev series expansion
04470 !                    of degree 12, for the function f, in the
04471 !                    interval (a,b)
04472 !           cheb24 - coefficients of the Chebyshev series expansion
04473 !                    of degree 24, for the function f, in the
04474 !                    interval (a,b)
04475 !           resc12 - approximation to the integral of
04476 !                    cos(0.5*(b-a)*omega*x)*f(0.5*(b-a)*x+0.5*(b+a))
04477 !                    over (-1,+1), using the Chebyshev series
04478 !                    expansion of degree 12
04479 !           resc24 - approximation to the same integral, using the
04480 !                    Chebyshev series expansion of degree 24
04481 !           ress12 - the analogue of resc12 for the sine
04482 !           ress24 - the analogue of resc24 for the sine
04483 !
04484   implicit none
04485
04486   integer maxp1
04487
04488   real(kind=params_wp) a
04489   real(kind=params_wp) abserr
04490   real(kind=params_wp) ac
04491   real(kind=params_wp) an
04492   real(kind=params_wp) an2
04493   real(kind=params_wp) as
04494   real(kind=params_wp) asap
04495   real(kind=params_wp) ass
04496   real(kind=params_wp) b
04497   real(kind=params_wp) centr
04498   real(kind=params_wp) chebmo(maxp1,25)
04499   real(kind=params_wp) cheb12(13)
```

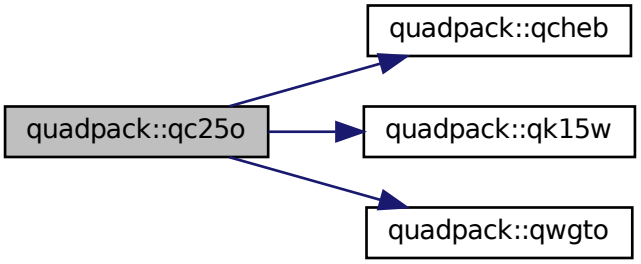
```
04500   real(kind=params_wp) cheb24(25)
04501   real(kind=params_wp) conc
04502   real(kind=params_wp) cons
04503   real(kind=params_wp) cospar
04504   real(kind=params_wp) d(28)
04505   real(kind=params_wp) d1(28)
04506   real(kind=params_wp) d2(28)
04507   real(kind=params_wp) d3(28)
04508   real(kind=params_wp) estc
04509   real(kind=params_wp) ests
04510   real(kind=params_wp), external :: f
04511   real(kind=params_wp) fval(25)
04512   real(kind=params_wp) hlgth
04513   integer i
04514   integer integr
04515   integer isym
04516   integer j
04517   integer k
04518   integer ksave
04519   integer m
04520   integer momcom
04521   integer neval
04522   integer, parameter :: nmac = 28
04523   integer noeq1
04524   integer noequ
04525   integer nrmom
04526   real(kind=params_wp) omega
04527   real(kind=params_wp) parint
04528   real(kind=params_wp) par2
04529   real(kind=params_wp) par22
04530   real(kind=params_wp) p2
04531   real(kind=params_wp) p3
04532   real(kind=params_wp) p4
04533 !  real(kind=params_wp), external :: qwgto
04534   real(kind=params_wp) resabs
04535   real(kind=params_wp) resasc
04536   real(kind=params_wp) resc12
04537   real(kind=params_wp) resc24
04538   real(kind=params_wp) ress12
04539   real(kind=params_wp) ress24
04540   real(kind=params_wp) result
04541   real(kind=params_wp) sinpar
04542   real(kind=params_wp) v(28)
04543   real(kind=params_wp), dimension ( 11 ) :: x = (/ &
04544     9.914448613738104e-01,     9.659258262890683e-01, &
04545     9.238795325112868e-01,     8.660254037844386e-01, &
04546     7.933533402912352e-01,     7.071067811865475e-01, &
04547     6.087614290087206e-01,     5.000000000000000e-01, &
04548     3.826834323650898e-01,     2.588190451025208e-01, &
04549     1.305261922200516e-01 /)
04550
04551   centr = 5.0e-01 * ( b + a )
04552   hlgth = 5.0e-01 * ( b - a )
04553   parint = omega * hlgth
04554 !
04555 !  Compute the integral using the 15-point Gauss-Kronrod
04556 !  formula if the value of the parameter in the integrand
04557 !  is small or if the length of the integration interval
04558 !  is less than (bb-aa)/2**(maxp1-2), where (aa,bb) is the
04559 !  original integration interval.
04560 !
04561   if ( abs( parint ) <= 2.0e+00 ) then
04562
04563     call qk15w ( f, qwgto, omega, p2, p3, p4, integr, a, b, result, &
04564       abserr, resabs, resasc )
04565
04566     neval = 15
04567     return
04568
04569   end if
04570 !
04571 !  Compute the integral using the generalized clenshaw-curtis method.
04572 !
04573   conc = hlgth * cos(centr*omega)
04574   cons = hlgth * sin(centr*omega)
04575   resasc = huge( resasc )
04576   neval = 25
04577 !
04578 !  Check whether the Chebyshev moments for this interval
04579 !  have already been computed.
04580 !
04581   if ( nrmom < momcom .or. ksave == 1 ) then
04582     go to 140
04583   end if
04584 !
04585 !  Compute a new set of Chebyshev moments.
04586 !
```

```
04587   m = momcom + 1
04588   par2 = parint * parint
04589   par22 = par2 + 2.0e+00
04590   sinpar = sin(parint)
04591   cospar = cos(parint)
04592 !
04593 !  Compute the Chebyshev moments with respect to cosine.
04594 !
04595   v(1) = 2.0e+00 * sinpar / parint
04596   v(2) = (8.0e+00*cospar+(par2+par2-8.0e+00)*sinpar/ parint)/par2
04597   v(3) = (3.2e+01*(par2-1.2e+01)*cospar+(2.0e+00* &
04598      ((par2-8.0e+01)*par2+1.92e+02)*sinpar)/ &
04599      parint)/(par2*par2)
04600   ac = 8.0e+00*cospar
04601   as = 2.4e+01*parint*sinpar
04602
04603   if ( abs( parint ) > 2.4e+01 ) then
04604     go to 70
04605   end if
04606 !
04607 !  Compute the Chebyshev moments as the solutions of a boundary value
04608 !  problem with one initial value (v(3)) and one end value computed
04609 !  using an asymptotic formula.
04610 !
04611   noequ = nmac-3
04612   noeq1 = noequ-1
04613   an = 6.0e+00
04614
04615   do k = 1, noeq1
04616     an2 = an*an
04617     d(k) = -2.0e+00*(an2-4.0e+00) * (par22-an2-an2)
04618     d2(k) = (an-1.0e+00_params_wp)*(an-2.0e+00) * par2
04619     d1(k) = (an+3.0e+00)*(an+4.0e+00) * par2
04620     v(k+3) = as-(an2-4.0e+00) * ac
04621     an = an+2.0e+00
04622   end do
04623
04624   an2 = an*an
04625   d(noequ) = -2.0e+00*(an2-4.0e+00) * (par22-an2-an2)
04626   v(noequ+3) = as - ( an2 - 4.0e+00 ) * ac
04627   v(4) = v(4) - 5.6e+01 * par2 * v(3)
04628   ass = parint * sinpar
04629   asap = (((((2.10e+02*par2-1.0e+00_params_wp)*cospar-(1.05e+02*par2 &
04630      -6.3e+01)*ass)/an2-(1.0e+00_params_wp-1.5e+01*par2)*cospar &
04631      +1.5e+01*ass)/an2-cospar+3.0e+00*ass)/an2-cospar)/an2
04632   v(noequ+3) = v(noequ+3)-2.0e+00*asap*par2*(an-1.0e+00_params_wp)* &
04633       (an-2.0e+00)
04634 !
04635 !  Solve the tridiagonal system by means of Gaussian
04636 !  elimination with partial pivoting.
04637 !
04638   d3(1:noequ) = 0.0e+00
04639
04640   d2(noequ) = 0.0e+00
04641
04642   do i = 1, noeq1
04643
04644     if ( abs(d1(i)) > abs(d(i)) ) then
04645       an = d1(i)
04646       d1(i) = d(i)
04647       d(i) = an
04648       an = d2(i)
04649       d2(i) = d(i+1)
04650       d(i+1) = an
04651       d3(i) = d2(i+1)
04652       d2(i+1) = 0.0e+00
04653       an = v(i+4)
04654       v(i+4) = v(i+3)
04655       v(i+3) = an
04656     end if
04657
04658     d(i+1) = d(i+1)-d2(i)*d1(i)/d(i)
04659     d2(i+1) = d2(i+1)-d3(i)*d1(i)/d(i)
04660     v(i+4) = v(i+4)-v(i+3)*d1(i)/d(i)
04661
04662   end do
04663
04664   v(noequ+3) = v(noequ+3) / d(noequ)
04665   v(noequ+2) = (v(noequ+2)-d2(noeq1)*v(noequ+3))/d(noeq1)
04666
04667   do i = 2, noeq1
04668     k = noequ-i
04669     v(k+3) = (v(k+3)-d3(k)*v(k+5)-d2(k)*v(k+4))/d(k)
04670   end do
04671
04672   go to 90
04673 !
```

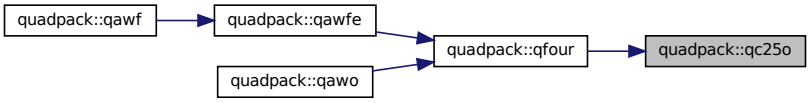
```
04674 !  Compute the Chebyshev moments by means of forward recursion
04675 !
04676 70 continue
04677
04678   an = 4.0e+00
04679
04680   do i = 4, 13
04681     an2 = an*an
04682     v(i) = ((an2-4.0e+00)*(2.0e+00*(par22-an2-an2)*v(i-1)-ac) &
04683      +as-par2*(an+1.0e+00_params_wp)*(an+2.0e+00)*v(i-2))/ &
04684      (par2*(an-1.0e+00_params_wp)*(an-2.0e+00))
04685     an = an+2.0e+00
04686   end do
04687
04688 90 continue
04689
04690   do j = 1, 13
04691     chebmo(m,2*j-1) = v(j)
04692   end do
04693 !
04694 !  Compute the Chebyshev moments with respect to sine.
04695 !
04696   v(1) = 2.0e+00*(sinpar-parint*cospar)/par2
04697   v(2) = (1.8e+01-4.8e+01/par2)*sinpar/par2 &
04698      +(-2.0e+00+4.8e+01/par2)*cospar/parint
04699   ac = -2.4e+01*parint*cospar
04700   as = -8.0e+00*sinpar
04701   chebmo(m,2) = v(1)
04702   chebmo(m,4) = v(2)
04703
04704   if ( abs(parint) <= 2.4e+01 ) then
04705
04706     do k = 3, 12
04707       an = k
04708       chebmo(m,2*k) = -sinpar/(an*(2.0e+00*an-2.0e+00)) &
04709                    -2.5e-01*parint*(v(k+1)/an-v(k)/(an-1.0e+00_params_wp))
04710     end do
04711 !
04712 !  Compute the Chebyshev moments by means of forward recursion.
04713 !
04714   else
04715
04716     an = 3.0e+00
04717
04718     do i = 3, 12
04719       an2 = an*an
04720       v(i) = ((an2-4.0e+00)*(2.0e+00*(par22-an2-an2)*v(i-1)+as) &
04721        +ac-par2*(an+1.0e+00_params_wp)*(an+2.0e+00)*v(i-2)) &
04722        /(par2*(an-1.0e+00_params_wp)*(an-2.0e+00))
04723       an = an+2.0e+00
04724       chebmo(m,2*i) = v(i)
04725     end do
04726
04727   end if
04728
04729 140 continue
04730
04731   if ( nrmom < momcom ) then
04732     m = nrmom + 1
04733   end if
04734
04735   if ( momcom < maxp1 - 1 .and. nrmom >= momcom ) then
04736     momcom = momcom + 1
04737   end if
04738 !
04739 !  Compute the coefficients of the Chebyshev expansions
04740 !  of degrees 12 and 24 of the function F.
04741 !
04742   fval(1) = 5.0e-01 * f(centr+hlgth)
04743   fval(13) = f(centr)
04744   fval(25) = 5.0e-01 * f(centr-hlgth)
04745
04746   do i = 2, 12
04747     isym = 26-i
04748     fval(i) = f(hlgth*x(i-1)+centr)
04749     fval(isym) = f(centr-hlgth*x(i-1))
04750   end do
04751
04752   call qcheb ( x, fval, cheb12, cheb24 )
04753 !
04754 !  Compute the integral and error estimates.
04755 !
04756   resc12 = cheb12(13) * chebmo(m,13)
04757   ress12 = 0.0e+00
04758   estc = abs( cheb24(25)*chebmo(m,25))+abs((cheb12(13)- &
04759     cheb24(13))*chebmo(m,13) )
04760   ests = 0.0e+00
```

```
04761   k = 11
04762
04763   do j = 1, 6
04764     resc12 = resc12+cheb12(k)*chebmo(m,k)
04765     ress12 = ress12+cheb12(k+1)*chebmo(m,k+1)
04766     estc = estc+abs((cheb12(k)-cheb24(k))*chebmo(m,k))
04767     ests = ests+abs((cheb12(k+1)-cheb24(k+1))*chebmo(m,k+1))
04768     k = k-2
04769   end do
04770
04771   resc24 = cheb24(25)*chebmo(m,25)
04772   ress24 = 0.0e+00
04773   resabs = abs(cheb24(25))
04774   k = 23
04775
04776   do j = 1, 12
04777
04778     resc24 = resc24+cheb24(k)*chebmo(m,k)
04779     ress24 = ress24+cheb24(k+1)*chebmo(m,k+1)
04780     resabs = resabs+abs(cheb24(k))+abs(cheb24(k+1))
04781
04782     if ( j <= 5 ) then
04783       estc = estc+abs(cheb24(k)*chebmo(m,k))
04784       ests = ests+abs(cheb24(k+1)*chebmo(m,k+1))
04785     end if
04786
04787     k = k-2
04788
04789   end do
04790
04791   resabs = resabs * abs( hlgth )
04792
04793   if ( integr == 1 ) then
04794     result = conc * resc24-cons*ress24
04795     abserr = abs( conc * estc ) + abs( cons * ests )
04796   else
04797     result = conc*ress24+cons*resc24
04798     abserr = abs(conc*ests)+abs(cons*estc)
04799   end if
04800
04801   return
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.37.1.16 qc25s()

```
subroutine quadpack::qc25s (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) bl,
             real(kind=params_wp) br,
             real(kind=params_wp) alfa,
             real(kind=params_wp) beta,
             real(kind=params_wp), dimension(25) ri,
             real(kind=params_wp), dimension(25) rj,
             real(kind=params_wp), dimension(25) rg,
             real(kind=params_wp), dimension(25) rh,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             real(kind=params_wp) resasc,
             integer integr,
             integer neval )
```

Definition at line 4803 of file quadpack.f90.

```
04805
04806 !****************************************************************************80
04807 !
04808 !! QC25S returns rules for algebraico-logarithmic end point singularities.
04809 !
04810 !  Discussion:
04811 !
04812 !    This routine computes
04813 !      i = integral of F(X) * W(X) over (bl,br),
04814 !    with error estimate, where the weight function W(X) has a singular
04815 !    behavior of algebraico-logarithmic type at the points
04816 !    a and/or b.
04817 !
04818 !    The interval (bl,br) is a subinterval of (a,b).
04819 !
04820 !  Author:
04821 !
04822 !    Robert Piessens, Elise de Doncker-Kapenger,
04823 !    Christian Ueberhuber, David Kahaner
04824 !
04825 !  Reference:
04826 !
04827 !    Robert Piessens, Elise de Doncker-Kapenger,
04828 !    Christian Ueberhuber, David Kahaner,
04829 !    QUADPACK, a Subroutine Package for Automatic Integration,
04830 !    Springer Verlag, 1983
04831 !
04832 !  Parameters:
04833 !
04834 !    Input, external real F, the name of the function routine, of the form
04835 !      function f ( x )
04836 !      real f
04837 !      real x
04838 !    which evaluates the integrand function.
04839 !
04840 !    Input, real A, B, the limits of integration.
04841 !
04842 !    Input, real BL, BR, the lower and upper limits of integration.
04843 !    A <= BL < BR <= B.
04844 !
04845 !    Input, real ALFA, BETA, parameters in the weight function.
04846 !
04847 !    Input, real RI(25), RJ(25), RG(25), RH(25), modified Chebyshev moments
04848 !    for the application of the generalized Clenshaw-Curtis method,
04849 !    computed in QMOMO.
04850 !
04851 !    Output, real RESULT, the estimated value of the integral, computed by
04852 !    using a generalized clenshaw-curtis method if bl = a or br = b.
04853 !    In all other cases the 15-point Kronrod rule is applied, obtained by
04854 !    optimal addition of abscissae to the 7-point Gauss rule.
04855 !
04856 !    Output, real ABSERR, an estimate of || I - RESULT ||.
04857 !
04858 !    Output, real RESASC, approximation to the integral of abs(F*W-I/(B-A)).
04859 !
04860 !    Input, integer INTEGR,  determines the weight function
04861 !    1, w(x) = (x-a)**alfa*(b-x)**beta
04862 !    2, w(x) = (x-a)**alfa*(b-x)**beta*log(x-a)
```

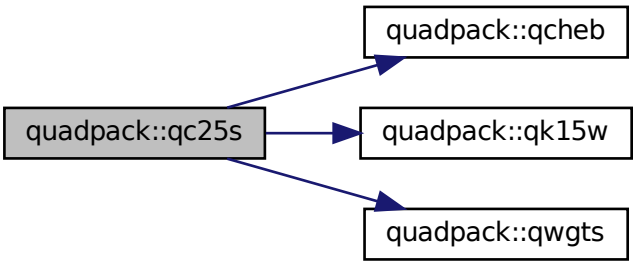
```
04863 !    3, w(x) = (x-a)**alfa*(b-x)**beta*log(b-x)
04864 !    4, w(x) = (x-a)**alfa*(b-x)**beta*log(x-a)*log(b-x)
04865 !
04866 !    Output, integer NEVAL, the number of times the integral was evaluated.
04867 !
04868 !  Local Parameters:
04869 !
04870 !           fval   - value of the function f at the points
04871 !                    (br-bl)*0.5*cos(k*pi/24)+(br+bl)*0.5
04872 !                    k = 0, ..., 24
04873 !           cheb12 - coefficients of the Chebyshev series expansion
04874 !                    of degree 12, for the function f, in the interval
04875 !                    (bl,br)
04876 !           cheb24 - coefficients of the Chebyshev series expansion
04877 !                    of degree 24, for the function f, in the interval
04878 !                    (bl,br)
04879 !           res12  - approximation to the integral obtained from cheb12
04880 !           res24  - approximation to the integral obtained from cheb24
04881 !           qwgts  - external function subprogram defining the four
04882 !                    possible weight functions
04883 !           hlgth  - half-length of the interval (bl,br)
04884 !           centr  - mid point of the interval (bl,br)
04885 !
04886 !           the vector x contains the values cos(k*pi/24)
04887 !           k = 1, ..., 11, to be used for the computation of the
04888 !           Chebyshev series expansion of f.
04889 !
04890   implicit none
04891
04892   real(kind=params_wp) a
04893   real(kind=params_wp) abserr
04894   real(kind=params_wp) alfa
04895   real(kind=params_wp) b
04896   real(kind=params_wp) beta
04897   real(kind=params_wp) bl
04898   real(kind=params_wp) br
04899   real(kind=params_wp) centr
04900   real(kind=params_wp) cheb12(13)
04901   real(kind=params_wp) cheb24(25)
04902   real(kind=params_wp) dc
04903   real(kind=params_wp), external :: f
04904   real(kind=params_wp) factor
04905   real(kind=params_wp) fix
04906   real(kind=params_wp) fval(25)
04907   real(kind=params_wp) hlgth
04908   integer i
04909   integer integr
04910   integer isym
04911   integer neval
04912 !  real(kind=params_wp), external :: qwgts
04913   real(kind=params_wp) resabs
04914   real(kind=params_wp) resasc
04915   real(kind=params_wp) result
04916   real(kind=params_wp) res12
04917   real(kind=params_wp) res24
04918   real(kind=params_wp) rg(25)
04919   real(kind=params_wp) rh(25)
04920   real(kind=params_wp) ri(25)
04921   real(kind=params_wp) rj(25)
04922   real(kind=params_wp) u
04923   real(kind=params_wp), dimension ( 11 ) :: x = (/ &
04924     9.914448613738104e-01,     9.659258262890683e-01, &
04925     9.238795325112868e-01,     8.660254037844386e-01, &
04926     7.933533402912352e-01,     7.071067811865475e-01, &
04927     6.087614290087206e-01,     5.000000000000000e-01, &
04928     3.826834323650898e-01,     2.588190451025208e-01, &
04929     1.305261922200516e-01 /)
04930
04931   neval = 25
04932
04933   if ( bl == a .and. (alfa /= 0.0e+00 .or. integr == 2 .or. integr == 4)) then
04934     go to 10
04935   end if
04936
04937   if ( br == b .and. (beta /= 0.0e+00 .or. integr == 3 .or. integr == 4)) &
04938     go to 140
04939 !
04940 !  If a > bl and b < br, apply the 15-point Gauss-Kronrod scheme.
04941 !
04942   call qk15w ( f, qwgts, a, b, alfa, beta, integr, bl, br, result, abserr, &
04943     resabs, resasc )
04944
04945   neval = 15
04946   return
04947 !
04948 !  This part of the program is executed only if a = bl.
04949 !
```

```
04950 !  Compute the Chebyshev series expansion of the function
04951 !  f1 = (0.5*(b+b-br-a)-0.5*(br-a)*x)**beta*f(0.5*(br-a)*x+0.5*(br+a))
04952 !
04953 10 continue
04954
04955   hlgth = 5.0e-01*(br-bl)
04956   centr = 5.0e-01*(br+bl)
04957   fix = b-centr
04958   fval(1) = 5.0e-01*f(hlgth+centr)*(fix-hlgth)**beta
04959   fval(13) = f(centr)*(fix**beta)
04960   fval(25) = 5.0e-01*f(centr-hlgth)*(fix+hlgth)**beta
04961
04962   do i = 2, 12
04963     u = hlgth*x(i-1)
04964     isym = 26-i
04965     fval(i) = f(u+centr)*(fix-u)**beta
04966     fval(isym) = f(centr-u)*(fix+u)**beta
04967   end do
04968
04969   factor = hlgth**(alfa+1.0e+00_params_wp)
04970   result = 0.0e+00
04971   abserr = 0.0e+00
04972   res12 = 0.0e+00
04973   res24 = 0.0e+00
04974
04975   if ( integr > 2 ) go to 70
04976
04977   call qcheb ( x, fval, cheb12, cheb24 )
04978 !
04979 !  integr = 1  (or 2)
04980 !
04981   do i = 1, 13
04982     res12 = res12+cheb12(i)*ri(i)
04983     res24 = res24+cheb24(i)*ri(i)
04984   end do
04985
04986   do i = 14, 25
04987     res24 = res24 + cheb24(i) * ri(i)
04988   end do
04989
04990   if ( integr == 1 ) go to 130
04991 !
04992 !  integr = 2
04993 !
04994   dc = log( br - bl )
04995   result = res24 * dc
04996   abserr = abs((res24-res12)*dc)
04997   res12 = 0.0e+00
04998   res24 = 0.0e+00
04999
05000   do i = 1, 13
05001     res12 = res12+cheb12(i)*rg(i)
05002     res24 = res24+cheb24(i)*rg(i)
05003   end do
05004
05005   do i = 14, 25
05006     res24 = res24+cheb24(i)*rg(i)
05007   end do
05008
05009   go to 130
05010 !
05011 !  Compute the Chebyshev series expansion of the function
05012 !  F4 = f1*log(0.5*(b+b-br-a)-0.5*(br-a)*x)
05013 !
05014 70 continue
05015
05016   fval(1) = fval(1) * log( fix - hlgth )
05017   fval(13) = fval(13) * log( fix )
05018   fval(25) = fval(25) * log( fix + hlgth )
05019
05020   do i = 2, 12
05021     u = hlgth*x(i-1)
05022     isym = 26-i
05023     fval(i) = fval(i) * log( fix - u )
05024     fval(isym) = fval(isym) * log( fix + u )
05025   end do
05026
05027   call qcheb ( x, fval, cheb12, cheb24 )
05028 !
05029 !  integr = 3  (or 4)
05030 !
05031   do i = 1, 13
05032     res12 = res12+cheb12(i)*ri(i)
05033     res24 = res24+cheb24(i)*ri(i)
05034   end do
05035
05036   do i = 14, 25
```

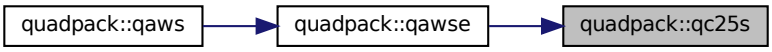
```
05037       res24 = res24+cheb24(i)*ri(i)
05038     end do
05039
05040     if ( integr == 3 ) then
05041       go to 130
05042     end if
05043 !
05044 !    integr = 4
05045 !
05046     dc = log( br - bl )
05047     result = res24*dc
05048     abserr = abs((res24-res12)*dc)
05049     res12 = 0.0e+00
05050     res24 = 0.0e+00
05051
05052     do i = 1, 13
05053       res12 = res12+cheb12(i)*rg(i)
05054       res24 = res24+cheb24(i)*rg(i)
05055     end do
05056
05057     do i = 14, 25
05058       res24 = res24+cheb24(i)*rg(i)
05059     end do
05060
05061 130 continue
05062
05063     result = (result+res24)*factor
05064     abserr = (abserr+abs(res24-res12))*factor
05065     go to 270
05066 !
05067 !    This part of the program is executed only if b = br.
05068 !
05069 !    Compute the Chebyshev series expansion of the function
05070 !    f2 = (0.5*(b+bl-a-a)+0.5*(b-bl)*x)**alfa*f(0.5*(b-bl)*x+0.5*(b+bl))
05071 !
05072 140 continue
05073
05074     hlgth = 5.0e-01*(br-bl)
05075     centr = 5.0e-01*(br+bl)
05076     fix = centr-a
05077     fval(1) = 5.0e-01*f(hlgth+centr)*(fix+hlgth)**alfa
05078     fval(13) = f(centr)*(fix**alfa)
05079     fval(25) = 5.0e-01*f(centr-hlgth)*(fix-hlgth)**alfa
05080
05081     do i = 2, 12
05082       u = hlgth*x(i-1)
05083       isym = 26-i
05084       fval(i) = f(u+centr)*(fix+u)**alfa
05085       fval(isym) = f(centr-u)*(fix-u)**alfa
05086     end do
05087
05088     factor = hlgth**(beta+1.0e+00_params_wp)
05089     result = 0.0e+00
05090     abserr = 0.0e+00
05091     res12 = 0.0e+00
05092     res24 = 0.0e+00
05093
05094     if ( integr == 2 .or. integr == 4 ) then
05095       go to 200
05096     end if
05097 !
05098 !    integr = 1  (or 3)
05099 !
05100     call qcheb ( x, fval, cheb12, cheb24 )
05101
05102     do i = 1, 13
05103       res12 = res12+cheb12(i)*rj(i)
05104       res24 = res24+cheb24(i)*rj(i)
05105     end do
05106
05107     do i = 14, 25
05108       res24 = res24+cheb24(i)*rj(i)
05109     end do
05110
05111     if ( integr == 1 ) go to 260
05112 !
05113 !    integr = 3
05114 !
05115     dc = log( br - bl )
05116     result = res24*dc
05117     abserr = abs((res24-res12)*dc)
05118     res12 = 0.0e+00
05119     res24 = 0.0e+00
05120
05121     do i = 1, 13
05122       res12 = res12+cheb12(i)*rh(i)
05123       res24 = res24+cheb24(i)*rh(i)
```

```
05124   end do
05125
05126   do i = 14, 25
05127     res24 = res24+cheb24(i)*rh(i)
05128   end do
05129
05130   go to 260
05131 !
05132 !  Compute the Chebyshev series expansion of the function
05133 !  f3 = f2*log(0.5*(b-bl)*x+0.5*(b+bl-a-a))
05134 !
05135 200 continue
05136
05137   fval(1) = fval(1) * log( hlgth + fix )
05138   fval(13) = fval(13) * log( fix )
05139   fval(25) = fval(25) * log( fix - hlgth )
05140
05141   do i = 2, 12
05142     u = hlgth*x(i-1)
05143     isym = 26-i
05144     fval(i) = fval(i) * log(u+fix)
05145     fval(isym) = fval(isym) * log(fix-u)
05146   end do
05147
05148   call qcheb ( x, fval, cheb12, cheb24 )
05149 !
05150 !  integr = 2  (or 4)
05151 !
05152   do i = 1, 13
05153     res12 = res12+cheb12(i)*rj(i)
05154     res24 = res24+cheb24(i)*rj(i)
05155   end do
05156
05157   do i = 14, 25
05158     res24 = res24+cheb24(i)*rj(i)
05159   end do
05160
05161   if ( integr == 2 ) go to 260
05162
05163   dc = log(br-bl)
05164   result = res24*dc
05165   abserr = abs((res24-res12)*dc)
05166   res12 = 0.0e+00
05167   res24 = 0.0e+00
05168 !
05169 !  integr = 4
05170 !
05171   do i = 1, 13
05172     res12 = res12+cheb12(i)*rh(i)
05173     res24 = res24+cheb24(i)*rh(i)
05174   end do
05175
05176   do i = 14, 25
05177     res24 = res24+cheb24(i)*rh(i)
05178   end do
05179
05180 260 continue
05181
05182   result = (result+res24)*factor
05183   abserr = (abserr+abs(res24-res12))*factor
05184
05185 270 continue
05186
05187   return
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.37.1.17 qcheb()

```
subroutine quadpack::qcheb (
             real(kind=params_wp), dimension(11) x,
             real(kind=params_wp), dimension(25) fval,
             real(kind=params_wp), dimension(13) cheb12,
             real(kind=params_wp), dimension(25) cheb24 )
```

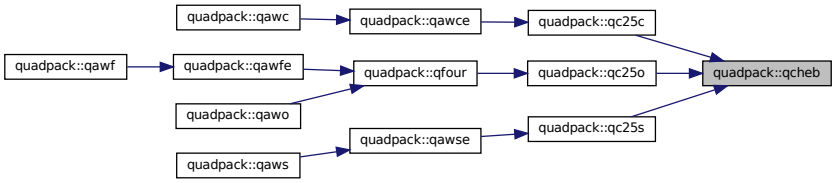
Definition at line 5189 of file quadpack.f90.

```
05190
05191 !*****************************************************************************80
05192 !
05193 !! QCHEB computes the Chebyshev series expansion.
05194 !
05195 !  Discussion:
05196 !
05197 !    This routine computes the Chebyshev series expansion
05198 !    of degrees 12 and 24 of a function using a fast Fourier transform method
05199 !
05200 !      f(x) = sum(k=1, ...,13) (cheb12(k)*t(k-1,x)),
05201 !      f(x) = sum(k=1, ...,25) (cheb24(k)*t(k-1,x)),
05202 !
05203 !    where T(K,X) is the Chebyshev polynomial of degree K.
05204 !
05205 !  Author:
05206 !
05207 !    Robert Piessens, Elise de Doncker-Kapenger,
05208 !    Christian Ueberhuber, David Kahaner
05209 !
05210 !  Reference:
05211 !
05212 !    Robert Piessens, Elise de Doncker-Kapenger,
05213 !    Christian Ueberhuber, David Kahaner,
05214 !    QUADPACK, a Subroutine Package for Automatic Integration,
05215 !    Springer Verlag, 1983
05216 !
05217 !  Parameters:
05218 !
```

```
05219 !    Input, real X(11), contains the values of COS(K*PI/24), for K = 1 to 11.
05220 !
05221 !    Input/output, real FVAL(25), the function values at the points
05222 !    (b+a+(b-a)*cos(k*pi/24))/2, k = 0, ...,24, where (a,b) is the
05223 !    approximation interval.  FVAL(1) and FVAL(25) are divided by two
05224 !    These values are destroyed at output.
05225 !
05226 !    Output, real CHEB12(13), the Chebyshev coefficients for degree 12.
05227 !
05228 !    Output, real CHEB24(25), the Chebyshev coefficients for degree 24.
05229 !
05230   implicit none
05231
05232   real(kind=params_wp) alam
05233   real(kind=params_wp) alam1
05234   real(kind=params_wp) alam2
05235   real(kind=params_wp) cheb12(13)
05236   real(kind=params_wp) cheb24(25)
05237   real(kind=params_wp) fval(25)
05238   integer i
05239   integer j
05240   real(kind=params_wp) part1
05241   real(kind=params_wp) part2
05242   real(kind=params_wp) part3
05243   real(kind=params_wp) v(12)
05244   real(kind=params_wp) x(11)
05245
05246   do i = 1, 12
05247     j = 26-i
05248     v(i) = fval(i)-fval(j)
05249     fval(i) = fval(i)+fval(j)
05250   end do
05251
05252   alam1 = v(1)-v(9)
05253   alam2 = x(6)*(v(3)-v(7)-v(11))
05254   cheb12(4) = alam1+alam2
05255   cheb12(10) = alam1-alam2
05256   alam1 = v(2)-v(8)-v(10)
05257   alam2 = v(4)-v(6)-v(12)
05258   alam = x(3)*alam1+x(9)*alam2
05259   cheb24(4) = cheb12(4)+alam
05260   cheb24(22) = cheb12(4)-alam
05261   alam = x(9)*alam1-x(3)*alam2
05262   cheb24(10) = cheb12(10)+alam
05263   cheb24(16) = cheb12(10)-alam
05264   part1 = x(4)*v(5)
05265   part2 = x(8)*v(9)
05266   part3 = x(6)*v(7)
05267   alam1 = v(1)+part1+part2
05268   alam2 = x(2)*v(3)+part3+x(10)*v(11)
05269   cheb12(2) = alam1+alam2
05270   cheb12(12) = alam1-alam2
05271   alam = x(1)*v(2)+x(3)*v(4)+x(5)*v(6)+x(7)*v(8) &
05272     +x(9)*v(10)+x(11)*v(12)
05273   cheb24(2) = cheb12(2)+alam
05274   cheb24(24) = cheb12(2)-alam
05275   alam = x(11)*v(2)-x(9)*v(4)+x(7)*v(6)-x(5)*v(8) &
05276     +x(3)*v(10)-x(1)*v(12)
05277   cheb24(12) = cheb12(12)+alam
05278   cheb24(14) = cheb12(12)-alam
05279   alam1 = v(1)-part1+part2
05280   alam2 = x(10)*v(3)-part3+x(2)*v(11)
05281   cheb12(6) = alam1+alam2
05282   cheb12(8) = alam1-alam2
05283   alam = x(5)*v(2)-x(9)*v(4)-x(1)*v(6) &
05284     -x(11)*v(8)+x(3)*v(10)+x(7)*v(12)
05285   cheb24(6) = cheb12(6)+alam
05286   cheb24(20) = cheb12(6)-alam
05287   alam = x(7)*v(2)-x(3)*v(4)-x(11)*v(6)+x(1)*v(8) &
05288     -x(9)*v(10)-x(5)*v(12)
05289   cheb24(8) = cheb12(8)+alam
05290   cheb24(18) = cheb12(8)-alam
05291
05292   do i = 1, 6
05293     j = 14-i
05294     v(i) = fval(i)-fval(j)
05295     fval(i) = fval(i)+fval(j)
05296   end do
05297
05298   alam1 = v(1)+x(8)*v(5)
05299   alam2 = x(4)*v(3)
05300   cheb12(3) = alam1+alam2
05301   cheb12(11) = alam1-alam2
05302   cheb12(7) = v(1)-v(5)
05303   alam = x(2)*v(2)+x(6)*v(4)+x(10)*v(6)
05304   cheb24(3) = cheb12(3)+alam
05305   cheb24(23) = cheb12(3)-alam
```

```
05306   alam = x(6)*(v(2)-v(4)-v(6))
05307   cheb24(7) = cheb12(7)+alam
05308   cheb24(19) = cheb12(7)-alam
05309   alam = x(10)*v(2)-x(6)*v(4)+x(2)*v(6)
05310   cheb24(11) = cheb12(11)+alam
05311   cheb24(15) = cheb12(11)-alam
05312
05313   do i = 1, 3
05314     j = 8-i
05315     v(i) = fval(i)-fval(j)
05316     fval(i) = fval(i)+fval(j)
05317   end do
05318
05319   cheb12(5) = v(1)+x(8)*v(3)
05320   cheb12(9) = fval(1)-x(8)*fval(3)
05321   alam = x(4)*v(2)
05322   cheb24(5) = cheb12(5)+alam
05323   cheb24(21) = cheb12(5)-alam
05324   alam = x(8)*fval(2)-fval(4)
05325   cheb24(9) = cheb12(9)+alam
05326   cheb24(17) = cheb12(9)-alam
05327   cheb12(1) = fval(1)+fval(3)
05328   alam = fval(2)+fval(4)
05329   cheb24(1) = cheb12(1)+alam
05330   cheb24(25) = cheb12(1)-alam
05331   cheb12(13) = v(1)-v(3)
05332   cheb24(13) = cheb12(13)
05333   alam = 1.0e+00_params_wp/6.0e+00
05334
05335   do i = 2, 12
05336     cheb12(i) = cheb12(i)*alam
05337   end do
05338
05339   alam = 5.0e-01*alam
05340   cheb12(1) = cheb12(1)*alam
05341   cheb12(13) = cheb12(13)*alam
05342
05343   do i = 2, 24
05344     cheb24(i) = cheb24(i)*alam
05345   end do
05346
05347   cheb24(1) = 0.5e+00 * alam*cheb24(1)
05348   cheb24(25) = 0.5e+00 * alam*cheb24(25)
05349
05350   return
```

Here is the caller graph for this function:



14.37.1.18 qextr()

```
subroutine quadpack::qextr (
             integer n,
             real(kind=params_wp), dimension(52) epstab,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             real(kind=params_wp), dimension(3) res3la,
             integer nres )
```

Definition at line 5352 of file quadpack.f90.

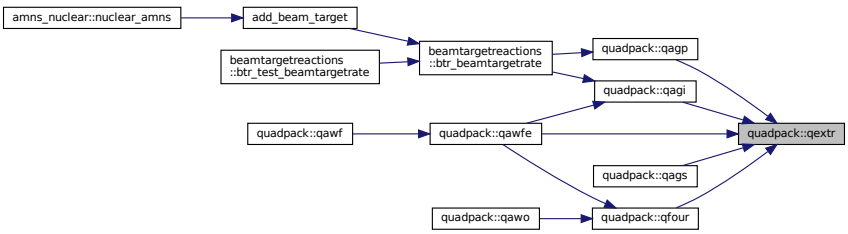
```
05353
05354 !***********************************************************************80
05355 !
05356 !! QEXTR carries out the Epsilon extrapolation algorithm.
05357 !
```

```
05358 !  Discussion:
05359 !
05360 !    The routine determines the limit of a given sequence of approximations,
05361 !    by means of the epsilon algorithm of P. Wynn.  An estimate of the
05362 !    absolute error is also given.  The condensed epsilon table is computed.
05363 !    Only those elements needed for the computation of the next diagonal
05364 !    are preserved.
05365 !
05366 !  Author:
05367 !
05368 !    Robert Piessens, Elise de Doncker-Kapenger,
05369 !    Christian Ueberhuber, David Kahaner
05370 !
05371 !  Reference:
05372 !
05373 !    Robert Piessens, Elise de Doncker-Kapenger,
05374 !    Christian Ueberhuber, David Kahaner,
05375 !    QUADPACK, a Subroutine Package for Automatic Integration,
05376 !    Springer Verlag, 1983
05377 !
05378 !  Parameters:
05379 !
05380 !    Input, integer N, indicates the entry of EPSTAB which contains
05381 !    the new element in the first column of the epsilon table.
05382 !
05383 !    Input/output, real EPSTAB(52), the two lower diagonals of the triangular
05384 !    epsilon table.  The elements are numbered starting at the right-hand
05385 !    corner of the triangle.
05386 !
05387 !    Output, real RESULT, the estimated value of the integral.
05388 !
05389 !    Output, real ABSERR, estimate of the absolute error computed from
05390 !    RESULT and the 3 previous results.
05391 !
05392 !    ?, real RES3LA(3), the last 3 results.
05393 !
05394 !    Input/output, integer NRES, the number of calls to the routine.  This
05395 !    should be zero on the first call, and is automatically updated
05396 !    before return.
05397 !
05398 !  Local Parameters:
05399 !
05400 !           e0     - the 4 elements on which the
05401 !           e1       computation of a new element in
05402 !           e2       the epsilon table is based
05403 !           e3                 e0
05404 !                          e3    e1    new
05405 !                                e2
05406 !           newelm - number of elements to be computed in the new
05407 !                    diagonal
05408 !           error  - error = abs(e1-e0)+abs(e2-e1)+abs(new-e2)
05409 !           result - the element in the new diagonal with least value
05410 !                    of error
05411 !           limexp is the maximum number of elements the epsilon table
05412 !           can contain. if this number is reached, the upper diagonal
05413 !           of the epsilon table is deleted.
05414 !
05415   implicit none
05416
05417   real(kind=params_wp) abserr
05418   real(kind=params_wp) delta1
05419   real(kind=params_wp) delta2
05420   real(kind=params_wp) delta3
05421   real(kind=params_wp) epsinf
05422   real(kind=params_wp) epstab(52)
05423   real(kind=params_wp) error
05424   real(kind=params_wp) err1
05425   real(kind=params_wp) err2
05426   real(kind=params_wp) err3
05427   real(kind=params_wp) e0
05428   real(kind=params_wp) e1
05429   real(kind=params_wp) e1abs
05430   real(kind=params_wp) e2
05431   real(kind=params_wp) e3
05432   integer i
05433   integer ib
05434   integer ib2
05435   integer ie
05436   integer indx
05437   integer k1
05438   integer k2
05439   integer k3
05440   integer limexp
05441   integer n
05442   integer newelm
05443   integer nres
05444   integer num
```

```
05445   real(kind=params_wp) res
05446   real(kind=params_wp) result
05447   real(kind=params_wp) res3la(3)
05448   real(kind=params_wp) ss
05449   real(kind=params_wp) tol1
05450   real(kind=params_wp) tol2
05451   real(kind=params_wp) tol3
05452
05453   nres = nres+1
05454   abserr = huge( abserr )
05455   result = epstab(n)
05456
05457   if ( n < 3 ) then
05458     abserr = max( abserr,0.5e+00* epsilon( result ) *abs(result))
05459     return
05460   end if
05461
05462   limexp = 50
05463   epstab(n+2) = epstab(n)
05464   newelm = (n-1)/2
05465   epstab(n) = huge( epstab(n) )
05466   num = n
05467   k1 = n
05468
05469   do i = 1, newelm
05470
05471     k2 = k1-1
05472     k3 = k1-2
05473     res = epstab(k1+2)
05474     e0 = epstab(k3)
05475     e1 = epstab(k2)
05476     e2 = res
05477     e1abs = abs(e1)
05478     delta2 = e2-e1
05479     err2 = abs(delta2)
05480     tol2 = max( abs(e2),e1abs)* epsilon( e2 )
05481     delta3 = e1-e0
05482     err3 = abs(delta3)
05483     tol3 = max( e1abs,abs(e0))* epsilon( e0 )
05484 !
05485 !  If e0, e1 and e2 are equal to within machine accuracy, convergence
05486 !  is assumed.
05487 !
05488     if ( err2 <= tol2 .and. err3 <= tol3 ) then
05489       result = res
05490       abserr = err2+err3
05491       abserr = max( abserr,0.5e+00* epsilon( result ) *abs(result))
05492       return
05493     end if
05494
05495     e3 = epstab(k1)
05496     epstab(k1) = e1
05497     delta1 = e1-e3
05498     err1 = abs(delta1)
05499     tol1 = max( e1abs,abs(e3))* epsilon( e3 )
05500 !
05501 !  If two elements are very close to each other, omit a part
05502 !  of the table by adjusting the value of N.
05503 !
05504     if ( err1 <= tol1 .or. err2 <= tol2 .or. err3 <= tol3 ) go to 20
05505
05506     ss = 1.0e+00_params_wp/delta1+1.0e+00_params_wp/delta2-1.0e+00_params_wp/delta3
05507     epsinf = abs( ss*e1 )
05508 !
05509 !  Test to detect irregular behavior in the table, and
05510 !  eventually omit a part of the table adjusting the value of N.
05511 !
05512     if ( epsinf > 1.0e-04 ) go to 30
05513
05514 20  continue
05515
05516     n = i+i-1
05517     exit
05518 !
05519 !  Compute a new element and eventually adjust the value of RESULT.
05520 !
05521 30  continue
05522
05523     res = e1+1.0e+00_params_wp/ss
05524     epstab(k1) = res
05525     k1 = k1-2
05526     error = err2+abs(res-e2)+err3
05527
05528     if ( error <= abserr ) then
05529       abserr = error
05530       result = res
05531     end if
```

```
05532
05533   end do
05534 !
05535 !  Shift the table.
05536 !
05537   if ( n == limexp ) then
05538     n = 2*(limexp/2)-1
05539   end if
05540
05541   if ( (num/2)*2 == num ) then
05542     ib = 2
05543   else
05544     ib = 1
05545   end if
05546
05547   ie = newelm+1
05548
05549   do i = 1, ie
05550     ib2 = ib+2
05551     epstab(ib) = epstab(ib2)
05552     ib = ib2
05553   end do
05554
05555   if ( num /= n ) then
05556
05557     indx = num-n+1
05558
05559     do i = 1, n
05560       epstab(i)= epstab(indx)
05561       indx = indx+1
05562     end do
05563
05564   end if
05565
05566   if ( nres < 4 ) then
05567     res3la(nres) = result
05568     abserr = huge( abserr )
05569   else
05570     abserr = abs(result-res3la(3))+abs(result-res3la(2)) &
05571       +abs(result-res3la(1))
05572     res3la(1) = res3la(2)
05573     res3la(2) = res3la(3)
05574     res3la(3) = result
05575   end if
05576
05577   abserr = max( abserr,0.5e+00* epsilon( result ) *abs(result))
05578
05579   return
```

Here is the caller graph for this function:



14.37.1.19 qfour()

```
subroutine quadpack::qfour (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) omega,
             integer integr,
             real(kind=params_wp) epsabs,
             real(kind=params_wp) epsrel,
```

```
            integer limit,
            integer icall,
            integer maxp1,
            real(kind=params_wp) result,
            real(kind=params_wp) abserr,
            integer neval,
            integer ier,
            real(kind=params_wp), dimension(limit) alist,
            real(kind=params_wp), dimension(limit) blist,
            real(kind=params_wp), dimension(limit) rlist,
            real(kind=params_wp), dimension(limit) elist,
            integer, dimension(limit) iord,
            integer, dimension(limit) nnlog,
            integer momcom,
            real(kind=params_wp), dimension(maxp1,25) chebmo )
```

Definition at line 5581 of file quadpack.f90.

```
05584
05585 !*****************************************************************************80
05586 !
05587 !! QFOUR estimates the integrals of oscillatory functions.
05588 !
05589 !  Discussion:
05590 !
05591 !    This routine calculates an approximation RESULT to a definite integral
05592 !      I = integral of F(X) * COS(OMEGA*X)
05593 !    or
05594 !      I = integral of F(X) * SIN(OMEGA*X)
05595 !    over (A,B), hopefully satisfying:
05596 !      | I - RESULT | <= max ( epsabs, epsrel * |I| ) ).
05597 !
05598 !    QFOUR is called by QAWO and QAWF.  It can also be called directly in
05599 !    a user-written program.  In the latter case it is possible for the
05600 !    user to determine the first dimension of array CHEBMO(MAXP1,25).
05601 !    See also parameter description of MAXP1.  Additionally see
05602 !    parameter description of ICALL for eventually re-using
05603 !    Chebyshev moments computed during former call on subinterval
05604 !    of equal length abs(B-A).
05605 !
05606 !  Author:
05607 !
05608 !    Robert Piessens, Elise de Doncker-Kapenger,
05609 !    Christian Ueberhuber, David Kahaner
05610 !
05611 !  Reference:
05612 !
05613 !    Robert Piessens, Elise de Doncker-Kapenger,
05614 !    Christian Ueberhuber, David Kahaner,
05615 !    QUADPACK, a Subroutine Package for Automatic Integration,
05616 !    Springer Verlag, 1983
05617 !
05618 !  Parameters:
05619 !
05620 !    Input, external real F, the name of the function routine, of the form
05621 !      function f ( x )
05622 !      real f
05623 !      real x
05624 !    which evaluates the integrand function.
05625 !
05626 !    Input, real A, B, the limits of integration.
05627 !
05628 !    Input, real OMEGA, the multiplier of X in the weight function.
05629 !
05630 !    Input, integer INTEGR, indicates the weight functions to be used.
05631 !    = 1, w(x) = cos(omega*x)
05632 !    = 2, w(x) = sin(omega*x)
05633 !
05634 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
05635 !
05636 !    Input, integer LIMIT, the maximum number of subintervals of [A,B]
05637 !    that can be generated.
05638 !
05639 !    icall  - integer
05640 !                     if qfour is to be used only once, ICALL must
05641 !                     be set to 1.  assume that during this call, the
05642 !                     Chebyshev moments (for clenshaw-curtis integration
05643 !                     of degree 24) have been computed for intervals of
05644 !                     lenghts (abs(b-a))*2**(-l), l=0,1,2,...momcom-1.
05645 !                     the Chebyshev moments already computed can be
05646 !                     re-used in subsequent calls, if qfour must be
05647 !                     called twice or more times on intervals of the
```

```
05648 !                     same length abs(b-a). from the second call on, one
05649 !                     has to put then ICALL > 1.
05650 !                     if ICALL < 1, the routine will end with ier = 6.
05651 !
05652 !             maxp1  - integer
05653 !                     gives an upper bound on the number of
05654 !                     Chebyshev moments which can be stored, i.e.
05655 !                     for the intervals of lenghts abs(b-a)*2**(-l),
05656 !                     l=0,1, ..., maxp1-2, maxp1 >= 1.
05657 !                     if maxp1 < 1, the routine will end with ier = 6.
05658 !                     increasing (decreasing) the value of maxp1
05659 !                     decreases (increases) the computational time but
05660 !                     increases (decreases) the required memory space.
05661 !
05662 !    Output, real RESULT, the estimated value of the integral.
05663 !
05664 !    Output, real ABSERR, an estimate of || I - RESULT ||.
05665 !
05666 !    Output, integer NEVAL, the number of times the integral was evaluated.
05667 !
05668 !             ier    - integer
05669 !                      ier = 0 normal and reliable termination of the
05670 !                              routine. it is assumed that the
05671 !                              requested accuracy has been achieved.
05672 !                    - ier > 0 abnormal termination of the routine.
05673 !                              the estimates for integral and error are
05674 !                              less reliable. it is assumed that the
05675 !                              requested accuracy has not been achieved.
05676 !                      ier = 1 maximum number of subdivisions allowed
05677 !                              has been achieved. one can allow more
05678 !                              subdivisions by increasing the value of
05679 !                              limit (and taking according dimension
05680 !                              adjustments into account). however, if
05681 !                              this yields no improvement it is advised
05682 !                              to analyze the integrand, in order to
05683 !                              determine the integration difficulties.
05684 !                              if the position of a local difficulty can
05685 !                              be determined (e.g. singularity,
05686 !                              discontinuity within the interval) one
05687 !                              will probably gain from splitting up the
05688 !                              interval at this point and calling the
05689 !                              integrator on the subranges. if possible,
05690 !                              an appropriate special-purpose integrator
05691 !                              should be used which is designed for
05692 !                              handling the type of difficulty involved.
05693 !                          = 2 the occurrence of roundoff error is
05694 !                              detected, which prevents the requested
05695 !                              tolerance from being achieved.
05696 !                              the error may be under-estimated.
05697 !                          = 3 extremely bad integrand behavior occurs
05698 !                              at some points of the integration
05699 !                              interval.
05700 !                          = 4 the algorithm does not converge. roundoff
05701 !                              error is detected in the extrapolation
05702 !                              table. it is presumed that the requested
05703 !                              tolerance cannot be achieved due to
05704 !                              roundoff in the extrapolation table, and
05705 !                              that the returned result is the best which
05706 !                              can be obtained.
05707 !                          = 5 the integral is probably divergent, or
05708 !                              slowly convergent. it must be noted that
05709 !                              divergence can occur with any other value
05710 !                              of ier > 0.
05711 !                          = 6 the input is invalid, because
05712 !                              epsabs < 0 and epsrel < 0,
05713 !                              or (integr /= 1 and integr /= 2) or
05714 !                              ICALL < 1 or maxp1 < 1.
05715 !                              result, abserr, neval, last, rlist(1),
05716 !                              elist(1), iord(1) and nnlog(1) are set to
05717 !                              zero. alist(1) and blist(1) are set to a
05718 !                              and b respectively.
05719 !
05720 !    Workspace, real ALIST(LIMIT), BLIST(LIMIT), contains in entries 1
05721 !    through LAST the left and right ends of the partition subintervals.
05722 !
05723 !    Workspace, real RLIST(LIMIT), contains in entries 1 through LAST
05724 !    the integral approximations on the subintervals.
05725 !
05726 !    Workspace, real ELIST(LIMIT), contains in entries 1 through LAST
05727 !    the absolute error estimates on the subintervals.
05728 !
05729 !             iord   - integer
05730 !                      vector of dimension at least limit, the first k
05731 !                      elements of which are pointers to the error
05732 !                      estimates over the subintervals, such that
05733 !                      elist(iord(1)), ..., elist(iord(k)), form
05734 !                      a decreasing sequence, with k = last
```

```
05735 !                     if last <= (limit/2+2), and
05736 !                     k = limit+1-last otherwise.
05737 !
05738 !            nnlog  - integer
05739 !                     vector of dimension at least limit, indicating the
05740 !                     subdivision levels of the subintervals, i.e.
05741 !                     iwork(i) = l means that the subinterval numbered
05742 !                     i is of length abs(b-a)*2**(1-l)
05743 !
05744 !         on entry and return
05745 !            momcom - integer
05746 !                     indicating that the Chebyshev moments have been
05747 !                     computed for intervals of lengths
05748 !                     (abs(b-a))*2**(-l), l=0,1,2, ..., momcom-1,
05749 !                     momcom < maxp1
05750 !
05751 !            chebmo - real
05752 !                     array of dimension (maxp1,25) containing the
05753 !                     Chebyshev moments
05754 !
05755 !  Local Parameters:
05756 !
05757 !           alist     - list of left end points of all subintervals
05758 !                       considered up to now
05759 !           blist     - list of right end points of all subintervals
05760 !                       considered up to now
05761 !           rlist(i)  - approximation to the integral over
05762 !                       (alist(i),blist(i))
05763 !           rlist2    - array of dimension at least limexp+2 containing
05764 !                       the part of the epsilon table which is still
05765 !                       needed for further computations
05766 !           elist(i)  - error estimate applying to rlist(i)
05767 !           maxerr    - pointer to the interval with largest error
05768 !                       estimate
05769 !           errmax    - elist(maxerr)
05770 !           erlast    - error on the interval currently subdivided
05771 !           area      - sum of the integrals over the subintervals
05772 !           errsum    - sum of the errors over the subintervals
05773 !           errbnd    - requested accuracy max(epsabs,epsrel*
05774 !                       abs(result))
05775 !           *****1    - variable for the left subinterval
05776 !           *****2    - variable for the right subinterval
05777 !           last      - index for subdivision
05778 !           nres      - number of calls to the extrapolation routine
05779 !           numrl2    - number of elements in rlist2. if an appropriate
05780 !                       approximation to the compounded integral has
05781 !                       been obtained it is put in rlist2(numrl2) after
05782 !                       numrl2 has been increased by one
05783 !           small     - length of the smallest interval considered
05784 !                       up to now, multiplied by 1.5
05785 !           erlarg    - sum of the errors over the intervals larger
05786 !                       than the smallest interval considered up to now
05787 !           extrap    - logical variable denoting that the routine is
05788 !                       attempting to perform extrapolation, i.e. before
05789 !                       subdividing the smallest interval we try to
05790 !                       decrease the value of erlarg
05791 !           noext     - logical variable denoting that extrapolation
05792 !                       is no longer allowed (true value)
05793 !
05794   implicit none
05795
05796   integer limit
05797   integer maxp1
05798
05799   real(kind=params_wp) a
05800   real(kind=params_wp) abseps
05801   real(kind=params_wp) abserr
05802   real(kind=params_wp) alist(limit)
05803   real(kind=params_wp) area
05804   real(kind=params_wp) area1
05805   real(kind=params_wp) area12
05806   real(kind=params_wp) area2
05807   real(kind=params_wp) a1
05808   real(kind=params_wp) a2
05809   real(kind=params_wp) b
05810   real(kind=params_wp) blist(limit)
05811   real(kind=params_wp) b1
05812   real(kind=params_wp) b2
05813   real(kind=params_wp) chebmo(maxp1,25)
05814   real(kind=params_wp) correc
05815   real(kind=params_wp) defab1
05816   real(kind=params_wp) defab2
05817   real(kind=params_wp) defabs
05818   real(kind=params_wp) domega
05819   real(kind=params_wp) dres
05820   real(kind=params_wp) elist(limit)
05821   real(kind=params_wp) epsabs
```

```
05822   real(kind=params_wp) epsrel
05823   real(kind=params_wp) erlarg
05824   real(kind=params_wp) erlast
05825   real(kind=params_wp) errbnd
05826   real(kind=params_wp) errmax
05827   real(kind=params_wp) error1
05828   real(kind=params_wp) erro12
05829   real(kind=params_wp) error2
05830   real(kind=params_wp) errsum
05831   real(kind=params_wp) ertest
05832   logical extall
05833   logical extrap
05834   real(kind=params_wp), external :: f
05835   integer icall
05836   integer id
05837   integer ier
05838   integer ierro
05839   integer integr
05840   integer iord(limit)
05841   integer iroff1
05842   integer iroff2
05843   integer iroff3
05844   integer jupbnd
05845   integer k
05846   integer ksgn
05847   integer ktmin
05848   integer last
05849   integer maxerr
05850   integer momcom
05851   integer nev
05852   integer neval
05853   integer nnlog(limit)
05854   logical noext
05855   integer nres
05856   integer nrmax
05857   integer nrmom
05858   integer numrl2
05859   real(kind=params_wp) omega
05860   real(kind=params_wp) resabs
05861   real(kind=params_wp) reseps
05862   real(kind=params_wp) result
05863   real(kind=params_wp) res3la(3)
05864   real(kind=params_wp) rlist(limit)
05865   real(kind=params_wp) rlist2(52)
05866   real(kind=params_wp) small
05867   real(kind=params_wp) width
05868 !
05869 !  the dimension of rlist2 is determined by  the value of
05870 !  limexp in QEXTR (rlist2 should be of dimension
05871 !  (limexp+2) at least).
05872 !
05873 !  Test on validity of parameters.
05874 !
05875   ier = 0
05876   neval = 0
05877   last = 0
05878   result = 0.0e+00
05879   abserr = 0.0e+00
05880   alist(1) = a
05881   blist(1) = b
05882   rlist(1) = 0.0e+00
05883   elist(1) = 0.0e+00
05884   iord(1) = 0
05885   nnlog(1) = 0
05886
05887   if ( (integr /= 1.and.integr /= 2) .or. (epsabs < 0.0e+00.and. &
05888     epsrel < 0.0e+00) .or. icall < 1 .or. maxp1 < 1 ) then
05889     ier = 6
05890     return
05891   end if
05892 !
05893 !  First approximation to the integral.
05894 !
05895   domega = abs( omega )
05896   nrmom = 0
05897
05898   if ( icall <= 1 ) then
05899     momcom = 0
05900   end if
05901
05902   call qc25o ( f, a, b, domega, integr, nrmom, maxp1, 0, result, abserr, &
05903     neval, defabs, resabs, momcom, chebmo )
05904 !
05905 !  Test on accuracy.
05906 !
05907   dres = abs(result)
05908   errbnd = max( epsabs,epsrel*dres)
```

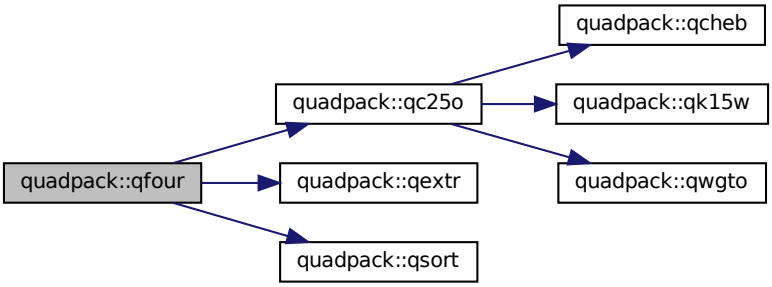
```
05909   rlist(1) = result
05910   elist(1) = abserr
05911   iord(1) = 1
05912   if ( abserr <= 1.0e+02* epsilon( defabs ) *defabs .and. &
05913     abserr > errbnd ) ier = 2
05914
05915   if ( limit == 1 ) then
05916     ier = 1
05917   end if
05918
05919   if ( ier /= 0 .or. abserr <= errbnd ) then
05920     go to 200
05921   end if
05922 !
05923 !  Initializations
05924 !
05925   errmax = abserr
05926   maxerr = 1
05927   area = result
05928   errsum = abserr
05929   abserr = huge( abserr )
05930   nrmax = 1
05931   extrap = .false.
05932   noext = .false.
05933   ierro = 0
05934   iroff1 = 0
05935   iroff2 = 0
05936   iroff3 = 0
05937   ktmin = 0
05938   small = abs(b-a)*7.5e-01
05939   nres = 0
05940   numrl2 = 0
05941   extall = .false.
05942
05943   if ( 5.0e-01*abs(b-a)*domega <= 2.0e+00) then
05944     numrl2 = 1
05945     extall = .true.
05946     rlist2(1) = result
05947   end if
05948
05949   if ( 2.5e-01 * abs(b-a) * domega <= 2.0e+00 ) then
05950     extall = .true.
05951   end if
05952
05953   if ( dres >= (1.0e+00_params_wp-5.0e+01* epsilon( defabs ) )*defabs ) then
05954     ksgn = 1
05955   else
05956     ksgn = -1
05957   end if
05958 !
05959 !  main do-loop
05960 !
05961   do last = 2, limit
05962 !
05963 !  Bisect the subinterval with the nrmax-th largest error estimate.
05964 !
05965     nrmom = nnlog(maxerr)+1
05966     a1 = alist(maxerr)
05967     b1 = 5.0e-01*(alist(maxerr)+blist(maxerr))
05968     a2 = b1
05969     b2 = blist(maxerr)
05970     erlast = errmax
05971
05972     call qc25o ( f, a1, b1, domega, integr, nrmom, maxp1, 0, area1, &
05973       error1, nev, resabs, defab1, momcom, chebmo )
05974
05975     neval = neval+nev
05976
05977     call qc25o ( f, a2, b2, domega, integr, nrmom, maxp1, 1, area2, &
05978       error2, nev, resabs, defab2, momcom, chebmo )
05979
05980     neval = neval+nev
05981 !
05982 !  Improve previous approximations to integral and error and
05983 !  test for accuracy.
05984 !
05985     area12 = area1+area2
05986     erro12 = error1+error2
05987     errsum = errsum+erro12-errmax
05988     area = area+area12-rlist(maxerr)
05989     if ( defab1 == error1 .or. defab2 == error2 ) go to 25
05990     if ( abs(rlist(maxerr)-area12) > 1.0e-05*abs(area12) &
05991     .or. erro12 < 9.9e-01*errmax ) go to 20
05992     if ( extrap ) iroff2 = iroff2+1
05993
05994     if ( .not.extrap ) then
05995       iroff1 = iroff1+1
```

```
05996     end if
05997
05998 20  continue
05999
06000     if ( last > 10.and.erro12 > errmax ) iroff3 = iroff3+1
06001
06002 25  continue
06003
06004     rlist(maxerr) = area1
06005     rlist(last) = area2
06006     nnlog(maxerr) = nrmom
06007     nnlog(last) = nrmom
06008     errbnd = max( epsabs,epsrel*abs(area))
06009 !
06010 !  Test for roundoff error and eventually set error flag
06011 !
06012     if ( iroff1+iroff2 >= 10 .or. iroff3 >= 20 ) ier = 2
06013
06014     if ( iroff2 >= 5) ierro = 3
06015 !
06016 !  Set error flag in the case that the number of subintervals
06017 !  equals limit.
06018 !
06019     if ( last == limit ) then
06020       ier = 1
06021     end if
06022 !
06023 !  Set error flag in the case of bad integrand behavior at
06024 !  a point of the integration range.
06025 !
06026     if ( max( abs(a1),abs(b2)) <= (1.0e+00_params_wp+1.0e+03* epsilon( a1 ) ) &
06027     *(abs(a2)+1.0e+03* tiny( a2 ) )) then
06028       ier = 4
06029     end if
06030 !
06031 !  Append the newly-created intervals to the list.
06032 !
06033     if ( error2 <= error1 ) then
06034       alist(last) = a2
06035       blist(maxerr) = b1
06036       blist(last) = b2
06037       elist(maxerr) = error1
06038       elist(last) = error2
06039     else
06040       alist(maxerr) = a2
06041       alist(last) = a1
06042       blist(last) = b1
06043       rlist(maxerr) = area2
06044       rlist(last) = area1
06045       elist(maxerr) = error2
06046       elist(last) = error1
06047     end if
06048 !
06049 !  Call QSORT to maintain the descending ordering
06050 !  in the list of error estimates and select the subinterval
06051 !  with nrmax-th largest error estimate (to be bisected next).
06052 !
06053
06054     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
06055
06056     if ( errsum <= errbnd ) then
06057       go to 170
06058     end if
06059
06060     if ( ier /= 0 ) then
06061       exit
06062     end if
06063
06064     if ( last == 2 .and. extall ) go to 120
06065
06066     if ( noext ) then
06067       cycle
06068     end if
06069
06070     if ( .not. extall ) go to 50
06071     erlarg = erlarg-erlast
06072     if ( abs(b1-a1) > small ) erlarg = erlarg+erro12
06073     if ( extrap ) go to 70
06074 !
06075 !  Test whether the interval to be bisected next is the
06076 !  smallest interval.
06077 !
06078 50  continue
06079
06080     width = abs(blist(maxerr)-alist(maxerr))
06081
06082     if ( width > small ) then
```

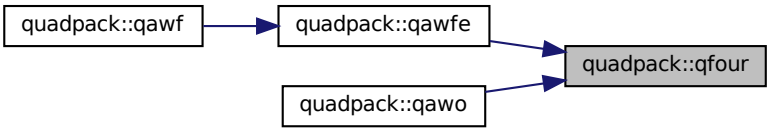
```
06083       cycle
06084     end if
06085
06086     if ( extall ) go to 60
06087 !
06088 !  Test whether we can start with the extrapolation procedure
06089 !  (we do this if we integrate over the next interval with
06090 !  use of a Gauss-Kronrod rule - see QC25O).
06091 !
06092     small = small*5.0e-01
06093
06094     if ( 2.5e-01*width*domega > 2.0e+00 ) then
06095       cycle
06096     end if
06097
06098     extall = .true.
06099     go to 130
06100
06101 60  continue
06102
06103     extrap = .true.
06104     nrmax = 2
06105
06106 70  continue
06107
06108     if ( ierro == 3 .or. erlarg <= ertest ) go to 90
06109 !
06110 !  The smallest interval has the largest error.
06111 !  Before bisecting decrease the sum of the errors over the
06112 !  larger intervals (ERLARG) and perform extrapolation.
06113 !
06114     jupbnd = last
06115
06116     if ( last > (limit/2+2) ) then
06117       jupbnd = limit+3-last
06118     end if
06119
06120     id = nrmax
06121
06122     do k = id, jupbnd
06123       maxerr = iord(nrmax)
06124       errmax = elist(maxerr)
06125       if ( abs(blist(maxerr)-alist(maxerr)) > small ) go to 140
06126       nrmax = nrmax+1
06127     end do
06128 !
06129 !  Perform extrapolation.
06130 !
06131 90  continue
06132
06133     numrl2 = numrl2+1
06134     rlist2(numrl2) = area
06135
06136     if ( numrl2 < 3 ) go to 110
06137
06138     call qextr ( numrl2, rlist2, reseps, abseps, res3la, nres )
06139     ktmin = ktmin+1
06140
06141     if ( ktmin > 5.and.abserr < 1.0e-03*errsum ) then
06142       ier = 5
06143     end if
06144
06145     if ( abseps >= abserr ) go to 100
06146
06147     ktmin = 0
06148     abserr = abseps
06149     result = reseps
06150     correc = erlarg
06151     ertest = max( epsabs, epsrel*abs(reseps))
06152
06153     if ( abserr <= ertest ) then
06154       exit
06155     end if
06156 !
06157 !  Prepare bisection of the smallest interval.
06158 !
06159 100 continue
06160
06161     if ( numrl2 == 1 ) then
06162       noext = .true.
06163     end if
06164
06165     if ( ier == 5 ) then
06166       exit
06167     end if
06168
06169 110 continue
```

```
06170
06171     maxerr = iord(1)
06172     errmax = elist(maxerr)
06173     nrmax = 1
06174     extrap = .false.
06175     small = small*5.0e-01
06176     erlarg = errsum
06177     cycle
06178
06179 120 continue
06180
06181     small = small * 5.0e-01
06182     numrl2 = numrl2 + 1
06183     rlist2(numrl2) = area
06184
06185 130 continue
06186
06187     ertest = errbnd
06188     erlarg = errsum
06189
06190 140 continue
06191
06192   end do
06193 !
06194 !  set the final result.
06195 !
06196   if ( abserr == huge( abserr ) .or. nres == 0 ) then
06197     go to 170
06198   end if
06199
06200   if ( ier+ierro == 0 ) go to 165
06201   if ( ierro == 3 ) abserr = abserr+correc
06202   if ( ier == 0 ) ier = 3
06203   if ( result /= 0.0e+00.and.area /= 0.0e+00 ) go to 160
06204   if ( abserr > errsum ) go to 170
06205   if ( area == 0.0e+00 ) go to 190
06206   go to 165
06207
06208 160 continue
06209
06210   if ( abserr/abs(result) > errsum/abs(area) ) go to 170
06211 !
06212 !  Test on divergence.
06213 !
06214   165 continue
06215
06216   if ( ksgn == (-1) .and. max( abs(result),abs(area)) <=  &
06217    defabs*1.0e-02 ) go to 190
06218
06219   if ( 1.0e-02 > (result/area) .or. (result/area) > 1.0e+02 &
06220    .or. errsum >= abs(area) ) ier = 6
06221
06222   go to 190
06223 !
06224 !  Compute global integral sum.
06225 !
06226 170 continue
06227
06228   result = sum( rlist(1:last) )
06229
06230   abserr = errsum
06231
06232 190 continue
06233
06234   if (ier > 2) ier=ier-1
06235
06236 200 continue
06237
06238   if ( integr == 2 .and. omega < 0.0e+00 ) then
06239     result = -result
06240   end if
06241
06242   return
```

Here is the call graph for this function:



Here is the caller graph for this function:



14.37.1.20 qk15()

```
subroutine quadpack::qk15 (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             real(kind=params_wp) resabs,
             real(kind=params_wp) resasc )
```

Definition at line 6244 of file quadpack.f90.

```
06245
06246 !*****************************************************************************80
06247 !
06248 !! QK15 carries out a 15 point Gauss-Kronrod quadrature rule.
06249 !
06250 !  Discussion:
06251 !
06252 !    This routine approximates
06253 !      I = integral ( A <= X <= B ) F(X) dx
06254 !    with an error estimate, and
06255 !      J = integral ( A <= X <= B ) | F(X) | dx
06256 !
06257 !  Author:
06258 !
06259 !    Robert Piessens, Elise de Doncker-Kapenger,
06260 !    Christian Ueberhuber, David Kahaner
06261 !
06262 !  Reference:
06263 !
06264 !    Robert Piessens, Elise de Doncker-Kapenger,
06265 !    Christian Ueberhuber, David Kahaner,
06266 !    QUADPACK, a Subroutine Package for Automatic Integration,
```

```
06267 !    Springer Verlag, 1983
06268 !
06269 !  Parameters:
06270 !
06271 !    Input, external real F, the name of the function routine, of the form
06272 !      function f ( x )
06273 !      real f
06274 !      real x
06275 !    which evaluates the integrand function.
06276 !
06277 !    Input, real A, B, the limits of integration.
06278 !
06279 !    Output, real RESULT, the estimated value of the integral.
06280 !    RESULT is computed by applying the 15-point Kronrod rule (RESK)
06281 !    obtained by optimal addition of abscissae to the 7-point Gauss rule
06282 !    (RESG).
06283 !
06284 !    Output, real ABSERR, an estimate of | I - RESULT |.
06285 !
06286 !    Output, real RESABS, approximation to the integral of the absolute
06287 !    value of F.
06288 !
06289 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
06290 !    over [A,B].
06291 !
06292 !  Local Parameters:
06293 !
06294 !           the abscissae and weights are given for the interval (-1,1).
06295 !           because of symmetry only the positive abscissae and their
06296 !           corresponding weights are given.
06297 !
06298 !           xgk    - abscissae of the 15-point Kronrod rule
06299 !                    xgk(2), xgk(4), ...  abscissae of the 7-point
06300 !                    Gauss rule
06301 !                    xgk(1), xgk(3), ...  abscissae which are optimally
06302 !                    added to the 7-point Gauss rule
06303 !
06304 !           wgk    - weights of the 15-point Kronrod rule
06305 !
06306 !           wg     - weights of the 7-point Gauss rule
06307 !
06308 !           centr  - mid point of the interval
06309 !           hlgth  - half-length of the interval
06310 !           absc   - abscissa
06311 !           fval*  - function value
06312 !           resg   - result of the 7-point Gauss formula
06313 !           resk   - result of the 15-point Kronrod formula
06314 !           reskh  - approximation to the mean value of f over (a,b),
06315 !                    i.e. to i/(b-a)
06316 !
06317   implicit none
06318
06319   real(kind=params_wp) a
06320   real(kind=params_wp) absc
06321   real(kind=params_wp) abserr
06322   real(kind=params_wp) b
06323   real(kind=params_wp) centr
06324   real(kind=params_wp) dhlgth
06325   real(kind=params_wp), external :: f
06326   real(kind=params_wp) fc
06327   real(kind=params_wp) fsum
06328   real(kind=params_wp) fval1
06329   real(kind=params_wp) fval2
06330   real(kind=params_wp) fv1(7)
06331   real(kind=params_wp) fv2(7)
06332   real(kind=params_wp) hlgth
06333   integer j
06334   integer jtw
06335   integer jtwm1
06336   real(kind=params_wp) resabs
06337   real(kind=params_wp) resasc
06338   real(kind=params_wp) resg
06339   real(kind=params_wp) resk
06340   real(kind=params_wp) reskh
06341   real(kind=params_wp) result
06342   real(kind=params_wp) wg(4)
06343   real(kind=params_wp) wgk(8)
06344   real(kind=params_wp) xgk(8)
06345
06346   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8)/ &
06347        9.914553711208126e-01,   9.491079123427585e-01, &
06348        8.648644233597691e-01,   7.415311855993944e-01, &
06349        5.860872354676911e-01,   4.058451513773972e-01, &
06350        2.077849550078985e-01,   0.0e+00               /
06351   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8)/ &
06352        2.293532201052922e-02,   6.309209262997855e-02, &
06353        1.047900103222502e-01,   1.406532597155259e-01, &
```

```
06354         1.690047266392679e-01,   1.903505780647854e-01, &
06355         2.044329400752989e-01,   2.094821410847278e-01/
06356    data wg(1),wg(2),wg(3),wg(4)/ &
06357         1.294849661688697e-01,   2.797053914892767e-01, &
06358         3.818300505051189e-01,   4.179591836734694e-01/
06359  !
06360    centr = 5.0e-01*(a+b)
06361    hlgth = 5.0e-01*(b-a)
06362    dhlgth = abs(hlgth)
06363  !
06364  !  Compute the 15-point Kronrod approximation to the integral,
06365  !  and estimate the absolute error.
06366  !
06367    fc = f(centr)
06368    resg = fc*wg(4)
06369    resk = fc*wgk(8)
06370    resabs = abs(resk)
06371
06372    do j = 1, 3
06373      jtw = j*2
06374      absc = hlgth*xgk(jtw)
06375      fval1 = f(centr-absc)
06376      fval2 = f(centr+absc)
06377      fv1(jtw) = fval1
06378      fv2(jtw) = fval2
06379      fsum = fval1+fval2
06380      resg = resg+wg(j)*fsum
06381      resk = resk+wgk(jtw)*fsum
06382      resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
06383    end do
06384
06385    do j = 1, 4
06386      jtwm1 = j*2-1
06387      absc = hlgth*xgk(jtwm1)
06388      fval1 = f(centr-absc)
06389      fval2 = f(centr+absc)
06390      fv1(jtwm1) = fval1
06391      fv2(jtwm1) = fval2
06392      fsum = fval1+fval2
06393      resk = resk+wgk(jtwm1)*fsum
06394      resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
06395    end do
06396
06397    reskh = resk * 5.0e-01
06398    resasc = wgk(8)*abs(fc-reskh)
06399
06400    do j = 1, 7
06401      resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
06402    end do
06403
06404    result = resk*hlgth
06405    resabs = resabs*dhlgth
06406    resasc = resasc*dhlgth
06407    abserr = abs((resk-resg)*hlgth)
06408
06409    if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00 ) then
06410      abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02_params_wp*abserr/resasc)**1.5e+00_params_wp)
06411    end if
06412
06413    if ( resabs > tiny( resabs ) / (5.0e+01* epsilon( resabs ) ) ) then
06414      abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
06415    end if
06416
06417    return
```

Here is the caller graph for this function:

14.37.1.21 qk15i()

```
subroutine quadpack::qk15i (
```

```
            real(kind=params_wp), external f,
            real(kind=params_wp) boun,
            integer inf,
            real(kind=params_wp) a,
            real(kind=params_wp) b,
            real(kind=params_wp) result,
            real(kind=params_wp) abserr,
            real(kind=params_wp) resabs,
            real(kind=params_wp) resasc )
```

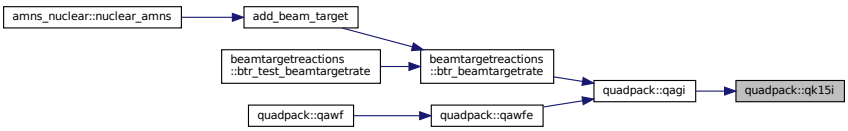
Definition at line 6419 of file quadpack.f90.

```
06420
06421 !*******************************************************************************80
06422 !
06423 !! QK15I applies a 15 point Gauss-Kronrod quadrature on an infinite interval.
06424 !
06425 !  Discussion:
06426 !
06427 !    The original infinite integration range is mapped onto the interval
06428 !    (0,1) and (a,b) is a part of (0,1).  The routine then computes:
06429 !
06430 !    i = integral of transformed integrand over (a,b),
06431 !    j = integral of abs(transformed integrand) over (a,b).
06432 !
06433 !  Author:
06434 !
06435 !    Robert Piessens, Elise de Doncker-Kapenger,
06436 !    Christian Ueberhuber, David Kahaner
06437 !
06438 !  Reference:
06439 !
06440 !    Robert Piessens, Elise de Doncker-Kapenger,
06441 !    Christian Ueberhuber, David Kahaner,
06442 !    QUADPACK, a Subroutine Package for Automatic Integration,
06443 !    Springer Verlag, 1983
06444 !
06445 !  Parameters:
06446 !
06447 !    Input, external real F, the name of the function routine, of the form
06448 !      function f ( x )
06449 !      real f
06450 !      real x
06451 !    which evaluates the integrand function.
06452 !
06453 !    Input, real BOUN, the finite bound of the original integration range,
06454 !    or zero if INF is 2.
06455 !
06456 !    Input, integer INF, indicates the type of the interval.
06457 !    -1: the original interval is (-infinity,BOUN),
06458 !    +1, the original interval is (BOUN,+infinity),
06459 !    +2, the original interval is (-infinity,+infinity) and
06460 !    the integral is computed as the sum of two integrals, one
06461 !    over (-infinity,0) and one over (0,+infinity).
06462 !
06463 !    Input, real A, B, the limits of integration, over a subrange of [0,1].
06464 !
06465 !    Output, real RESULT, the estimated value of the integral.
06466 !    RESULT is computed by applying the 15-point Kronrod rule (RESK) obtained
06467 !    by optimal addition of abscissae to the 7-point Gauss rule (RESG).
06468 !
06469 !    Output, real ABSERR, an estimate of | I - RESULT |.
06470 !
06471 !    Output, real RESABS, approximation to the integral of the absolute
06472 !    value of F.
06473 !
06474 !    Output, real RESASC, approximation to the integral of the
06475 !    transformated integrand | F-I/(B-A) | over [A,B].
06476 !
06477 !  Local Parameters:
06478 !
06479 !           centr  - mid point of the interval
06480 !           hlgth  - half-length of the interval
06481 !           absc*  - abscissa
06482 !           tabsc* - transformed abscissa
06483 !           fval*  - function value
06484 !           resg   - result of the 7-point Gauss formula
06485 !           resk   - result of the 15-point Kronrod formula
06486 !           reskh  - approximation to the mean value of the transformed
06487 !                    integrand over (a,b), i.e. to i/(b-a)
06488 !
06489   implicit none
06490
06491   real(kind=params_wp) a
06492   real(kind=params_wp) absc
```

```
06493   real(kind=params_wp) absc1
06494   real(kind=params_wp) absc2
06495   real(kind=params_wp) abserr
06496   real(kind=params_wp) b
06497   real(kind=params_wp) boun
06498   real(kind=params_wp) centr
06499   real(kind=params_wp) dinf
06500   real(kind=params_wp), external :: f
06501   real(kind=params_wp) fc
06502   real(kind=params_wp) fsum
06503   real(kind=params_wp) fval1
06504   real(kind=params_wp) fval2
06505   real(kind=params_wp) fv1(7)
06506   real(kind=params_wp) fv2(7)
06507   real(kind=params_wp) hlgth
06508   integer inf
06509   integer j
06510   real(kind=params_wp) resabs
06511   real(kind=params_wp) resasc
06512   real(kind=params_wp) resg
06513   real(kind=params_wp) resk
06514   real(kind=params_wp) reskh
06515   real(kind=params_wp) result
06516   real(kind=params_wp) tabsc1
06517   real(kind=params_wp) tabsc2
06518   real(kind=params_wp) wg(8)
06519   real(kind=params_wp) wgk(8)
06520   real(kind=params_wp) xgk(8)
06521 !
06522 !  the abscissae and weights are supplied for the interval
06523 !  (-1,1).  because of symmetry only the positive abscissae and
06524 !  their corresponding weights are given.
06525 !
06526 !           xgk    - abscissae of the 15-point Kronrod rule
06527 !                    xgk(2), xgk(4), ... abscissae of the 7-point Gauss
06528 !                    rule
06529 !                    xgk(1), xgk(3), ...  abscissae which are optimally
06530 !                    added to the 7-point Gauss rule
06531 !
06532 !           wgk    - weights of the 15-point Kronrod rule
06533 !
06534 !           wg     - weights of the 7-point Gauss rule, corresponding
06535 !                    to the abscissae xgk(2), xgk(4), ...
06536 !                    wg(1), wg(3), ... are set to zero.
06537 !
06538   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8)/ &
06539        9.914553711208126e-01,     9.491079123427585e-01, &
06540        8.648644233597691e-01,     7.415311855993944e-01, &
06541        5.860872354676911e-01,     4.058451513773972e-01, &
06542        2.077849550078985e-01,     0.000000000000000e+00/
06543
06544   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8)/ &
06545        2.293532201052922e-02,     6.309209262997855e-02, &
06546        1.047900103222502e-01,     1.406532597155259e-01, &
06547        1.690047266392679e-01,     1.903505780647854e-01, &
06548        2.044329400752989e-01,     2.094821410847278e-01/
06549
06550   data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8)/ &
06551        0.000000000000000e+00,     1.294849661688697e-01, &
06552        0.000000000000000e+00,     2.797053914892767e-01, &
06553        0.000000000000000e+00,     3.818300505051189e-01, &
06554        0.000000000000000e+00,     4.179591836734694e-01/
06555
06556   dinf = min( 1, inf )
06557
06558   centr = 5.0e-01*(a+b)
06559   hlgth = 5.0e-01*(b-a)
06560   tabsc1 = boun+dinf*(1.0e+00_params_wp-centr)/centr
06561   fval1 = f(tabsc1)
06562   if ( inf == 2 ) fval1 = fval1+f(-tabsc1)
06563   fc = (fval1/centr)/centr
06564 !
06565 !  Compute the 15-point Kronrod approximation to the integral,
06566 !  and estimate the error.
06567 !
06568   resg = wg(8)*fc
06569   resk = wgk(8)*fc
06570   resabs = abs(resk)
06571
06572   do j = 1, 7
06573
06574     absc = hlgth*xgk(j)
06575     absc1 = centr-absc
06576     absc2 = centr+absc
06577     tabsc1 = boun+dinf*(1.0e+00_params_wp-absc1)/absc1
06578     tabsc2 = boun+dinf*(1.0e+00_params_wp-absc2)/absc2
06579     fval1 = f(tabsc1)
```

```
06580     fval2 = f(tabsc2)
06581
06582     if ( inf == 2 ) then
06583       fval1 = fval1+f(-tabsc1)
06584       fval2 = fval2+f(-tabsc2)
06585     end if
06586
06587     fval1 = (fval1/absc1)/absc1
06588     fval2 = (fval2/absc2)/absc2
06589     fv1(j) = fval1
06590     fv2(j) = fval2
06591     fsum = fval1+fval2
06592     resg = resg+wg(j)*fsum
06593     resk = resk+wgk(j)*fsum
06594     resabs = resabs+wgk(j)*(abs(fval1)+abs(fval2))
06595   end do
06596
06597   reskh = resk * 5.0e-01
06598   resasc = wgk(8) * abs(fc-reskh)
06599
06600   do j = 1, 7
06601     resasc = resasc + wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
06602   end do
06603
06604   result = resk * hlgth
06605   resasc = resasc * hlgth
06606   resabs = resabs * hlgth
06607   abserr = abs( ( resk - resg ) * hlgth )
06608
06609   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) then
06610     abserr = resasc* min( 1.0e+00_params_wp,(2.0e+02_params_wp*abserr/resasc)**1.5e+00_params_wp)
06611   end if
06612
06613   if ( resabs > tiny( resabs ) / ( 5.0e+01 * epsilon( resabs ) ) ) then
06614     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
06615   end if
06616
06617   return
```

Here is the caller graph for this function:



14.37.1.22 qk15w()

```
subroutine quadpack::qk15w (
             real(kind=params_wp), external f,
             real(kind=params_wp), external w,
             real(kind=params_wp) p1,
             real(kind=params_wp) p2,
             real(kind=params_wp) p3,
             real(kind=params_wp) p4,
             integer kp,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             real(kind=params_wp) resabs,
             real(kind=params_wp) resasc )
```

Definition at line 6619 of file quadpack.f90.

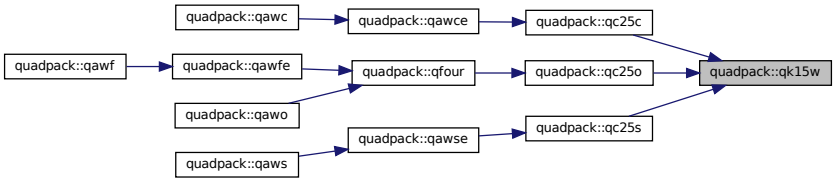
```
06621
06622 !***********************************************************************80
06623 !
06624 !! QK15W applies a 15 point Gauss-Kronrod rule for a weighted integrand.
06625 !
```

```
06626 !  Discussion:
06627 !
06628 !    This routine approximates
06629 !      i = integral of f*w over (a,b),
06630 !    with error estimate, and
06631 !      j = integral of abs(f*w) over (a,b)
06632 !
06633 !  Author:
06634 !
06635 !    Robert Piessens, Elise de Doncker-Kapenger,
06636 !    Christian Ueberhuber, David Kahaner
06637 !
06638 !  Reference:
06639 !
06640 !    Robert Piessens, Elise de Doncker-Kapenger,
06641 !    Christian Ueberhuber, David Kahaner,
06642 !    QUADPACK, a Subroutine Package for Automatic Integration,
06643 !    Springer Verlag, 1983
06644 !
06645 !  Parameters:
06646 !
06647 !    Input, external real F, the name of the function routine, of the form
06648 !      function f ( x )
06649 !      real f
06650 !      real x
06651 !    which evaluates the integrand function.
06652 !
06653 !              w      - real
06654 !                       function subprogram defining the integrand
06655 !                       weight function w(x). the actual name for w
06656 !                       needs to be declared e x t e r n a l in the
06657 !                       calling program.
06658 !
06659 !    ?, real P1, P2, P3, P4, parameters in the weight function
06660 !
06661 !    Input, integer KP, key for indicating the type of weight function
06662 !
06663 !    Input, real A, B, the limits of integration.
06664 !
06665 !    Output, real RESULT, the estimated value of the integral.
06666 !    RESULT is computed by applying the 15-point Kronrod rule (RESK) obtained by
06667 !    optimal addition of abscissae to the 7-point Gauss rule (RESG).
06668 !
06669 !    Output, real ABSERR, an estimate of | I - RESULT |.
06670 !
06671 !    Output, real RESABS, approximation to the integral of the absolute
06672 !    value of F.
06673 !
06674 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
06675 !    over [A,B].
06676 !
06677 !  Local Parameters:
06678 !
06679 !           centr  - mid point of the interval
06680 !           hlgth  - half-length of the interval
06681 !           absc*  - abscissa
06682 !           fval*  - function value
06683 !           resg   - result of the 7-point Gauss formula
06684 !           resk   - result of the 15-point Kronrod formula
06685 !           reskh  - approximation to the mean value of f*w over (a,b),
06686 !                    i.e. to i/(b-a)
06687 !
06688   implicit none
06689
06690   real(kind=params_wp) a
06691   real(kind=params_wp) absc
06692   real(kind=params_wp) absc1
06693   real(kind=params_wp) absc2
06694   real(kind=params_wp) abserr
06695   real(kind=params_wp) b
06696   real(kind=params_wp) centr
06697   real(kind=params_wp) dhlgth
06698   real(kind=params_wp), external :: f
06699   real(kind=params_wp) fc
06700   real(kind=params_wp) fsum
06701   real(kind=params_wp) fval1
06702   real(kind=params_wp) fval2
06703   real(kind=params_wp) fv1(7)
06704   real(kind=params_wp) fv2(7)
06705   real(kind=params_wp) hlgth
06706   integer j
06707   integer jtw
06708   integer jtwm1
06709   integer kp
06710   real(kind=params_wp) p1
06711   real(kind=params_wp) p2
06712   real(kind=params_wp) p3
```

```
06713   real(kind=params_wp) p4
06714   real(kind=params_wp) resabs
06715   real(kind=params_wp) resasc
06716   real(kind=params_wp) resg
06717   real(kind=params_wp) resk
06718   real(kind=params_wp) reskh
06719   real(kind=params_wp) result
06720   real(kind=params_wp), external :: w
06721   real(kind=params_wp), dimension ( 4 ) :: wg = (/ &
06722     1.294849661688697e-01,     2.797053914892767e-01, &
06723     3.818300505051889e-01,     4.179591836734694e-01 /)
06724   real(kind=params_wp) wgk(8)
06725   real(kind=params_wp) xgk(8)
06726 !
06727 !  the abscissae and weights are given for the interval (-1,1).
06728 !  because of symmetry only the positive abscissae and their
06729 !  corresponding weights are given.
06730 !
06731 !           xgk    - abscissae of the 15-point Gauss-Kronrod rule
06732 !                    xgk(2), xgk(4), ... abscissae of the 7-point Gauss
06733 !                    rule
06734 !                    xgk(1), xgk(3), ... abscissae which are optimally
06735 !                    added to the 7-point Gauss rule
06736 !
06737 !           wgk    - weights of the 15-point Gauss-Kronrod rule
06738 !
06739 !           wg     - weights of the 7-point Gauss rule
06740 !
06741   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8)/ &
06742        9.914553711208126e-01,     9.491079123427585e-01, &
06743        8.648644233597691e-01,     7.415311855993944e-01, &
06744        5.860872354676911e-01,     4.058451513773972e-01, &
06745        2.077849550789850e-01,     0.000000000000000e+00/
06746
06747   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8)/ &
06748        2.293532201052922e-02,     6.309209262997855e-02, &
06749        1.047900103222502e-01,     1.406532597155259e-01, &
06750        1.690047266392679e-01,     1.903505780647854e-01, &
06751        2.044329400752989e-01,     2.094821410847278e-01/
06752 !
06753   centr = 5.0e-01*(a+b)
06754   hlgth = 5.0e-01*(b-a)
06755   dhlgth = abs(hlgth)
06756 !
06757 !  Compute the 15-point Kronrod approximation to the integral,
06758 !  and estimate the error.
06759 !
06760   fc = f(centr)*w(centr,p1,p2,p3,p4,kp)
06761   resg = wg(4)*fc
06762   resk = wgk(8)*fc
06763   resabs = abs(resk)
06764
06765   do j = 1, 3
06766     jtw = j*2
06767     absc = hlgth*xgk(jtw)
06768     absc1 = centr-absc
06769     absc2 = centr+absc
06770     fval1 = f(absc1)*w(absc1,p1,p2,p3,p4,kp)
06771     fval2 = f(absc2)*w(absc2,p1,p2,p3,p4,kp)
06772     fv1(jtw) = fval1
06773     fv2(jtw) = fval2
06774     fsum = fval1+fval2
06775     resg = resg+wg(j)*fsum
06776     resk = resk+wgk(jtw)*fsum
06777     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
06778   end do
06779
06780   do j = 1, 4
06781     jtwm1 = j*2-1
06782     absc = hlgth*xgk(jtwm1)
06783     absc1 = centr-absc
06784     absc2 = centr+absc
06785     fval1 = f(absc1)*w(absc1,p1,p2,p3,p4,kp)
06786     fval2 = f(absc2)*w(absc2,p1,p2,p3,p4,kp)
06787     fv1(jtwm1) = fval1
06788     fv2(jtwm1) = fval2
06789     fsum = fval1+fval2
06790     resk = resk+wgk(jtwm1)*fsum
06791     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
06792   end do
06793
06794   reskh = resk*5.0e-01
06795   resasc = wgk(8)*abs(fc-reskh)
06796
06797   do j = 1, 7
06798     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
06799   end do
```

```
06800
06801   result = resk*hlgth
06802   resabs = resabs*dhlgth
06803   resasc = resasc*dhlgth
06804   abserr = abs((resk-resg)*hlgth)
06805
06806   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) then
06807     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00_params_wp)
06808   end if
06809
06810   if ( resabs > tiny( resabs ) /(5.0e+01* epsilon( resabs ) ) ) then
06811     abserr = max( ( epsilon( resabs ) * 5.0e+01)*resabs,abserr)
06812   end if
06813
06814   return
```

Here is the caller graph for this function:



14.37.1.23 qk21()

```
subroutine quadpack::qk21 (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             real(kind=params_wp) resabs,
             real(kind=params_wp) resasc )
```

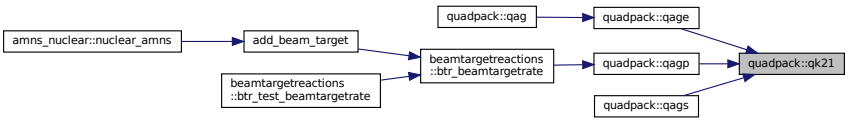
Definition at line 6816 of file quadpack.f90.

```
06817
06818 !*****************************************************************************80
06819 !
06820 !! QK21 carries out a 21 point Gauss-Kronrod quadrature rule.
06821 !
06822 !  Discussion:
06823 !
06824 !    This routine approximates
06825 !      I = integral ( A <= X <= B ) F(X) dx
06826 !    with an error estimate, and
06827 !      J = integral ( A <= X <= B ) | F(X) | dx
06828 !
06829 !  Author:
06830 !
06831 !    Robert Piessens, Elise de Doncker-Kapenger,
06832 !    Christian Ueberhuber, David Kahaner
06833 !
06834 !  Reference:
06835 !
06836 !    Robert Piessens, Elise de Doncker-Kapenger,
06837 !    Christian Ueberhuber, David Kahaner,
06838 !    QUADPACK, a Subroutine Package for Automatic Integration,
06839 !    Springer Verlag, 1983
06840 !
06841 !  Parameters:
06842 !
06843 !    Input, external real F, the name of the function routine, of the form
06844 !      function f ( x )
06845 !      real f
06846 !      real x
06847 !    which evaluates the integrand function.
06848 !
06849 !    Input, real A, B, the limits of integration.
06850 !
```

```
06851 !    Output, real RESULT, the estimated value of the integral.
06852 !    RESULT is computed by applying the 21-point Kronrod rule (resk)
06853 !    obtained by optimal addition of abscissae to the 10-point Gauss
06854 !    rule (resg).
06855 !
06856 !    Output, real ABSERR, an estimate of | I - RESULT |.
06857 !
06858 !    Output, real RESABS, approximation to the integral of the absolute
06859 !    value of F.
06860 !
06861 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
06862 !    over [A,B].
06863 !
06864   implicit none
06865
06866   real(kind=params_wp) a
06867   real(kind=params_wp) absc
06868   real(kind=params_wp) abserr
06869   real(kind=params_wp) b
06870   real(kind=params_wp) centr
06871   real(kind=params_wp) dhlgth
06872   real(kind=params_wp), external :: f
06873   real(kind=params_wp) fc
06874   real(kind=params_wp) fsum
06875   real(kind=params_wp) fval1
06876   real(kind=params_wp) fval2
06877   real(kind=params_wp) fv1(10)
06878   real(kind=params_wp) fv2(10)
06879   real(kind=params_wp) hlgth
06880   integer j
06881   integer jtw
06882   integer jtwm1
06883   real(kind=params_wp) resabs
06884   real(kind=params_wp) resasc
06885   real(kind=params_wp) resg
06886   real(kind=params_wp) resk
06887   real(kind=params_wp) reskh
06888   real(kind=params_wp) result
06889   real(kind=params_wp) wg(5)
06890   real(kind=params_wp) wgk(11)
06891   real(kind=params_wp) xgk(11)
06892 !
06893 !           the abscissae and weights are given for the interval (-1,1).
06894 !           because of symmetry only the positive abscissae and their
06895 !           corresponding weights are given.
06896 !
06897 !           xgk    - abscissae of the 21-point Kronrod rule
06898 !                    xgk(2), xgk(4), ...  abscissae of the 10-point
06899 !                    Gauss rule
06900 !                    xgk(1), xgk(3), ...  abscissae which are optimally
06901 !                    added to the 10-point Gauss rule
06902 !
06903 !           wgk    - weights of the 21-point Kronrod rule
06904 !
06905 !           wg     - weights of the 10-point Gauss rule
06906 !
06907   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
06908     xgk(9),xgk(10),xgk(11)/ &
06909        9.956571630258081e-01,     9.739065285171717e-01, &
06910        9.301574913557082e-01,     8.650633666889845e-01, &
06911        7.808177265864169e-01,     6.794095682990244e-01, &
06912        5.627571346686047e-01,     4.333953941292472e-01, &
06913        2.943928627014602e-01,     1.488743389816312e-01, &
06914        0.000000000000000e+00/
06915 !
06916   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
06917     wgk(9),wgk(10),wgk(11)/ &
06918        1.169463886737187e-02,     3.255816230796473e-02, &
06919        5.475589657435200e-02,     7.503967481091995e-02, &
06920        9.312545458369761e-02,     1.093871588022976e-01, &
06921        1.234919762620659e-01,     1.347092173114733e-01, &
06922        1.427759385770601e-01,     1.477391049013385e-01, &
06923        1.494455540029169e-01/
06924 !
06925   data wg(1),wg(2),wg(3),wg(4),wg(5)/ &
06926        6.667134430868814e-02,     1.494513491505806e-01, &
06927        2.190863625159820e-01,     2.692667193099964e-01, &
06928        2.955242247147529e-01/
06929 !
06930 !
06931 !           list of major variables
06932 !
06933 !           centr  - mid point of the interval
06934 !           hlgth  - half-length of the interval
06935 !           absc   - abscissa
06936 !           fval*  - function value
06937 !           resg   - result of the 10-point Gauss formula
```

```
06938 !           resk   - result of the 21-point Kronrod formula
06939 !           reskh  - approximation to the mean value of f over (a,b),
06940 !                    i.e. to i/(b-a)
06941 !
06942   centr = 5.0e-01*(a+b)
06943   hlgth = 5.0e-01*(b-a)
06944   dhlgth = abs(hlgth)
06945 !
06946 !  Compute the 21-point Kronrod approximation to the
06947 !  integral, and estimate the absolute error.
06948 !
06949   resg = 0.0e+00
06950   fc = f(centr)
06951   resk = wgk(11)*fc
06952   resabs = abs(resk)
06953
06954   do j = 1, 5
06955     jtw = 2*j
06956     absc = hlgth*xgk(jtw)
06957     fval1 = f(centr-absc)
06958     fval2 = f(centr+absc)
06959     fv1(jtw) = fval1
06960     fv2(jtw) = fval2
06961     fsum = fval1+fval2
06962     resg = resg+wg(j)*fsum
06963     resk = resk+wgk(jtw)*fsum
06964     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
06965   end do
06966
06967   do j = 1, 5
06968     jtwm1 = 2*j-1
06969     absc = hlgth*xgk(jtwm1)
06970     fval1 = f(centr-absc)
06971     fval2 = f(centr+absc)
06972     fv1(jtwm1) = fval1
06973     fv2(jtwm1) = fval2
06974     fsum = fval1+fval2
06975     resk = resk+wgk(jtwm1)*fsum
06976     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
06977   end do
06978
06979   reskh = resk*5.0e-01
06980   resasc = wgk(11)*abs(fc-reskh)
06981
06982   do j = 1, 10
06983     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
06984   end do
06985
06986   result = resk*hlgth
06987   resabs = resabs*dhlgth
06988   resasc = resasc*dhlgth
06989   abserr = abs((resk-resg)*hlgth)
06990
06991   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) then
06992     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
06993   end if
06994
06995   if ( resabs > tiny( resabs ) /(5.0e+01* epsilon( resabs ) )) then
06996     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
06997   end if
06998
06999   return
```

Here is the caller graph for this function:



14.37.1.24 qk31()

```
subroutine quadpack::qk31 (
            real(kind=params_wp), external f,
            real(kind=params_wp) a,
```

```
            real(kind=params_wp) b,
            real(kind=params_wp) result,
            real(kind=params_wp) abserr,
            real(kind=params_wp) resabs,
            real(kind=params_wp) resasc )
```

Definition at line 7001 of file quadpack.f90.

```
07002
07003 !*****************************************************************************80
07004 !
07005 !! QK31 carries out a 31 point Gauss-Kronrod quadrature rule.
07006 !
07007 !  Discussion:
07008 !
07009 !    This routine approximates
07010 !      I = integral ( A <= X <= B ) F(X) dx
07011 !    with an error estimate, and
07012 !      J = integral ( A <= X <= B ) | F(X) | dx
07013 !
07014 !  Author:
07015 !
07016 !    Robert Piessens, Elise de Doncker-Kapenger,
07017 !    Christian Ueberhuber, David Kahaner
07018 !
07019 !  Reference:
07020 !
07021 !    Robert Piessens, Elise de Doncker-Kapenger,
07022 !    Christian Ueberhuber, David Kahaner,
07023 !    QUADPACK, a Subroutine Package for Automatic Integration,
07024 !    Springer Verlag, 1983
07025 !
07026 !  Parameters:
07027 !
07028 !    Input, external real F, the name of the function routine, of the form
07029 !      function f ( x )
07030 !      real f
07031 !      real x
07032 !    which evaluates the integrand function.
07033 !
07034 !    Input, real A, B, the limits of integration.
07035 !
07036 !    Output, real RESULT, the estimated value of the integral.
07037 !                        result is computed by applying the 31-point
07038 !                        Gauss-Kronrod rule (resk), obtained by optimal
07039 !                        addition of abscissae to the 15-point Gauss
07040 !                        rule (resg).
07041 !
07042 !    Output, real ABSERR, an estimate of | I - RESULT |.
07043 !
07044 !    Output, real RESABS, approximation to the integral of the absolute
07045 !    value of F.
07046 !
07047 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
07048 !    over [A,B].
07049 !
07050   implicit none
07051
07052   real(kind=params_wp) a
07053   real(kind=params_wp) absc
07054   real(kind=params_wp) abserr
07055   real(kind=params_wp) b
07056   real(kind=params_wp) centr
07057   real(kind=params_wp) dhlgth
07058   real(kind=params_wp), external :: f
07059   real(kind=params_wp) fc
07060   real(kind=params_wp) fsum
07061   real(kind=params_wp) fval1
07062   real(kind=params_wp) fval2
07063   real(kind=params_wp) fv1(15)
07064   real(kind=params_wp) fv2(15)
07065   real(kind=params_wp) hlgth
07066   integer j
07067   integer jtw
07068   integer jtwm1
07069   real(kind=params_wp) resabs
07070   real(kind=params_wp) resasc
07071   real(kind=params_wp) resg
07072   real(kind=params_wp) resk
07073   real(kind=params_wp) reskh
07074   real(kind=params_wp) result
07075   real(kind=params_wp) wg(8)
07076   real(kind=params_wp) wgk(16)
07077   real(kind=params_wp) xgk(16)
07078 !
07079 !           the abscissae and weights are given for the interval (-1,1).
07080 !           because of symmetry only the positive abscissae and their
```

```
07081 !           corresponding weights are given.
07082 !
07083 !           xgk    - abscissae of the 31-point Kronrod rule
07084 !                    xgk(2), xgk(4), ...  abscissae of the 15-point
07085 !                    Gauss rule
07086 !                    xgk(1), xgk(3), ...  abscissae which are optimally
07087 !                    added to the 15-point Gauss rule
07088 !
07089 !           wgk    - weights of the 31-point Kronrod rule
07090 !
07091 !           wg     - weights of the 15-point Gauss rule
07092 !
07093   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
07094     xgk(9),xgk(10),xgk(11),xgk(12),xgk(13),xgk(14),xgk(15),xgk(16)/ &
07095        9.980022986933971e-01,   9.879925180204854e-01, &
07096        9.677390756791391e-01,   9.372733924007059e-01, &
07097        8.972645323440819e-01,   8.482065834104272e-01, &
07098        7.904185014424659e-01,   7.244177313601700e-01, &
07099        6.509967412974170e-01,   5.709721726085388e-01, &
07100        4.850818636402397e-01,   3.941513470775634e-01, &
07101        2.991800071531688e-01,   2.011940939974345e-01, &
07102        1.011420669187175e-01,   0.0e+00               /
07103   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
07104     wgk(9),wgk(10),wgk(11),wgk(12),wgk(13),wgk(14),wgk(15),wgk(16)/ &
07105        5.377479872923349e-03,   1.500794732931612e-02, &
07106        2.546084732671532e-02,   3.534636079137585e-02, &
07107        4.458975132476488e-02,   5.348152469092809e-02, &
07108        6.200956780067064e-02,   6.985412131872826e-02, &
07109        7.684968075772038e-02,   8.308050282313302e-02, &
07110        8.856444305621177e-02,   9.312659817082532e-02, &
07111        9.664272698362368e-02,   9.917359872179196e-02, &
07112        1.007698455238756e-01,   1.013300070147915e-01/
07113   data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8)/ &
07114        3.075324199611727e-02,   7.036604748810812e-02, &
07115        1.071592204671719e-01,   1.395706779261543e-01, &
07116        1.662692058169939e-01,   1.861610000155622e-01, &
07117        1.984314853271116e-01,   2.025782419255613e-01/
07118 !
07119 !
07120 !           list of major variables
07121 !
07122 !           centr  - mid point of the interval
07123 !           hlgth  - half-length of the interval
07124 !           absc   - abscissa
07125 !           fval*  - function value
07126 !           resg   - result of the 15-point Gauss formula
07127 !           resk   - result of the 31-point Kronrod formula
07128 !           reskh  - approximation to the mean value of f over (a,b),
07129 !                    i.e. to i/(b-a)
07130 !
07131   centr = 5.0e-01*(a+b)
07132   hlgth = 5.0e-01*(b-a)
07133   dhlgth = abs(hlgth)
07134 !
07135 !  Compute the 31-point Kronrod approximation to the integral,
07136 !  and estimate the absolute error.
07137 !
07138   fc = f(centr)
07139   resg = wg(8)*fc
07140   resk = wgk(16)*fc
07141   resabs = abs(resk)
07142
07143   do j = 1, 7
07144     jtw = j*2
07145     absc = hlgth*xgk(jtw)
07146     fval1 = f(centr-absc)
07147     fval2 = f(centr+absc)
07148     fv1(jtw) = fval1
07149     fv2(jtw) = fval2
07150     fsum = fval1+fval2
07151     resg = resg+wg(j)*fsum
07152     resk = resk+wgk(jtw)*fsum
07153     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
07154   end do
07155
07156   do j = 1, 8
07157     jtwm1 = j*2-1
07158     absc = hlgth*xgk(jtwm1)
07159     fval1 = f(centr-absc)
07160     fval2 = f(centr+absc)
07161     fv1(jtwm1) = fval1
07162     fv2(jtwm1) = fval2
07163     fsum = fval1+fval2
07164     resk = resk+wgk(jtwm1)*fsum
07165     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
07166   end do
07167
```

```
07168   reskh = resk*5.0e-01
07169   resasc = wgk(16)*abs(fc-reskh)
07170
07171   do j = 1, 15
07172     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
07173  end do
07174
07175   result = resk*hlgth
07176   resabs = resabs*dhlgth
07177   resasc = resasc*dhlgth
07178   abserr = abs((resk-resg)*hlgth)
07179
07180   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) &
07181     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
07182
07183   if ( resabs > tiny( resabs ) /(5.0e+01* epsilon( resabs ) )) then
07184     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
07185   end if
07186
07187   return
```

Here is the caller graph for this function:

14.37.1.25 qk41()

```
subroutine quadpack::qk41 (
             real(kind=params_wp), external f,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) result,
             real(kind=params_wp) abserr,
             real(kind=params_wp) resabs,
             real(kind=params_wp) resasc )
```

Definition at line 7189 of file quadpack.f90.

```
07190
07191 !****************************************************************************80
07192 !
07193 !! QK41 carries out a 41 point Gauss-Kronrod quadrature rule.
07194 !
07195 !  Discussion:
07196 !
07197 !    This routine approximates
07198 !      I = integral ( A <= X <= B ) F(X) dx
07199 !    with an error estimate, and
07200 !      J = integral ( A <= X <= B ) | F(X) | dx
07201 !
07202 !  Author:
07203 !
07204 !    Robert Piessens, Elise de Doncker-Kapenger,
07205 !    Christian Ueberhuber, David Kahaner
07206 !
07207 !  Reference:
07208 !
07209 !    Robert Piessens, Elise de Doncker-Kapenger,
07210 !    Christian Ueberhuber, David Kahaner,
07211 !    QUADPACK, a Subroutine Package for Automatic Integration,
07212 !    Springer Verlag, 1983
07213 !
07214 !  Parameters:
07215 !
07216 !    Input, external real F, the name of the function routine, of the form
07217 !      function f ( x )
07218 !      real f
07219 !      real x
07220 !    which evaluates the integrand function.
07221 !
```

```
07222 !    Input, real A, B, the limits of integration.
07223 !
07224 !    Output, real RESULT, the estimated value of the integral.
07225 !                        result is computed by applying the 41-point
07226 !                        Gauss-Kronrod rule (resk) obtained by optimal
07227 !                        addition of abscissae to the 20-point Gauss
07228 !                        rule (resg).
07229 !
07230 !    Output, real ABSERR, an estimate of | I - RESULT |.
07231 !
07232 !    Output, real RESABS, approximation to the integral of the absolute
07233 !    value of F.
07234 !
07235 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
07236 !    over [A,B].
07237 !
07238 !  Local Parameters:
07239 !
07240 !           centr  - mid point of the interval
07241 !           hlgth  - half-length of the interval
07242 !           absc   - abscissa
07243 !           fval*  - function value
07244 !           resg   - result of the 20-point Gauss formula
07245 !           resk   - result of the 41-point Kronrod formula
07246 !           reskh  - approximation to mean value of f over (a,b), i.e.
07247 !                    to i/(b-a)
07248 !
07249   implicit none
07250
07251   real(kind=params_wp) a
07252   real(kind=params_wp) absc
07253   real(kind=params_wp) abserr
07254   real(kind=params_wp) b
07255   real(kind=params_wp) centr
07256   real(kind=params_wp) dhlgth
07257   real(kind=params_wp), external :: f
07258   real(kind=params_wp) fc
07259   real(kind=params_wp) fsum
07260   real(kind=params_wp) fval1
07261   real(kind=params_wp) fval2
07262   real(kind=params_wp) fv1(20)
07263   real(kind=params_wp) fv2(20)
07264   real(kind=params_wp) hlgth
07265   integer j
07266   integer jtw
07267   integer jtwm1
07268   real(kind=params_wp) resabs
07269   real(kind=params_wp) resasc
07270   real(kind=params_wp) resg
07271   real(kind=params_wp) resk
07272   real(kind=params_wp) reskh
07273   real(kind=params_wp) result
07274   real(kind=params_wp) wg(10)
07275   real(kind=params_wp) wgk(21)
07276   real(kind=params_wp) xgk(21)
07277 !
07278 !           the abscissae and weights are given for the interval (-1,1).
07279 !           because of symmetry only the positive abscissae and their
07280 !           corresponding weights are given.
07281 !
07282 !           xgk    - abscissae of the 41-point Gauss-Kronrod rule
07283 !                    xgk(2), xgk(4), ...  abscissae of the 20-point
07284 !                    Gauss rule
07285 !                    xgk(1), xgk(3), ...  abscissae which are optimally
07286 !                    added to the 20-point Gauss rule
07287 !
07288 !           wgk    - weights of the 41-point Gauss-Kronrod rule
07289 !
07290 !           wg     - weights of the 20-point Gauss rule
07291 !
07292   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
07293     xgk(9),xgk(10),xgk(11),xgk(12),xgk(13),xgk(14),xgk(15),xgk(16), &
07294     xgk(17),xgk(18),xgk(19),xgk(20),xgk(21)/ &
07295        9.988590315882777e-01,   9.931285991850949e-01, &
07296        9.815078774502503e-01,   9.639719272779138e-01, &
07297        9.408226338317548e-01,   9.122344282513259e-01, &
07298        8.782768112522820e-01,   8.391169718222188e-01, &
07299        7.950414288375512e-01,   7.463319064601508e-01, &
07300        6.932376563347514e-01,   6.360536807265150e-01, &
07301        5.751404468197103e-01,   5.108670019508271e-01, &
07302        4.435931752387251e-01,   3.737060887154196e-01, &
07303        3.016278681149130e-01,   2.277858511416451e-01, &
07304        1.526054652409227e-01,   7.652652113349733e-02, &
07305        0.0e+00                /
07306   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
07307     wgk(9),wgk(10),wgk(11),wgk(12),wgk(13),wgk(14),wgk(15),wgk(16), &
07308     wgk(17),wgk(18),wgk(19),wgk(20),wgk(21)/ &
```

```
07309         3.073583718520532e-03,   8.600269855642942e-03, &
07310         1.462616925697125e-02,   2.038837346126652e-02, &
07311         2.588213360495116e-02,   3.128730677703280e-02, &
07312         3.660016975820080e-02,   4.166887332797369e-02, &
07313         4.643482186749767e-02,   5.094457392372869e-02, &
07314         5.519510534828599e-02,   5.911140088063957e-02, &
07315         6.265323755478117e-02,   6.583459713361842e-02, &
07316         6.864867292852162e-02,   7.105442355344407e-02, &
07317         7.303069033278667e-02,   7.458287540049919e-02, &
07318         7.570449768455667e-02,   7.637786767208074e-02, &
07319         7.660071191799966e-02/
07320    data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8),wg(9),wg(10)/ &
07321         1.761400713915212e-02,   4.060142980038694e-02, &
07322         6.267204833410906e-02,   8.327674157670475e-02, &
07323         1.019301198172404e-01,   1.181945319615184e-01, &
07324         1.316886384491766e-01,   1.420961093183821e-01, &
07325         1.491729864726037e-01,   1.527533871307259e-01/
07326 !
07327    centr = 5.0e-01*(a+b)
07328    hlgth = 5.0e-01*(b-a)
07329    dhlgth = abs(hlgth)
07330 !
07331 !  Compute 41-point Gauss-Kronrod approximation to the
07332 !  the integral, and estimate the absolute error.
07333 !
07334    resg = 0.0e+00
07335    fc = f(centr)
07336    resk = wgk(21)*fc
07337    resabs = abs(resk)
07338
07339    do j = 1, 10
07340      jtw = j*2
07341      absc = hlgth*xgk(jtw)
07342      fval1 = f(centr-absc)
07343      fval2 = f(centr+absc)
07344      fv1(jtw) = fval1
07345      fv2(jtw) = fval2
07346      fsum = fval1+fval2
07347      resg = resg+wg(j)*fsum
07348      resk = resk+wgk(jtw)*fsum
07349      resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
07350    end do
07351
07352    do j = 1, 10
07353      jtwm1 = j*2-1
07354      absc = hlgth*xgk(jtwm1)
07355      fval1 = f(centr-absc)
07356      fval2 = f(centr+absc)
07357      fv1(jtwm1) = fval1
07358      fv2(jtwm1) = fval2
07359      fsum = fval1+fval2
07360      resk = resk+wgk(jtwm1)*fsum
07361      resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
07362    end do
07363
07364    reskh = resk*5.0e-01
07365    resasc = wgk(21)*abs(fc-reskh)
07366
07367    do j = 1, 20
07368      resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
07369    end do
07370
07371    result = resk*hlgth
07372    resabs = resabs*dhlgth
07373    resasc = resasc*dhlgth
07374    abserr = abs((resk-resg)*hlgth)
07375
07376    if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) &
07377      abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
07378
07379    if ( resabs > tiny( resabs ) /(5.0e+01* epsilon( resabs ) )) then
07380      abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
07381    end if
07382
07383    return
```

Here is the caller graph for this function:

14.37.1.26 qk51()

```
subroutine quadpack::qk51 (
              real(kind=params_wp), external f,
              real(kind=params_wp) a,
              real(kind=params_wp) b,
              real(kind=params_wp) result,
              real(kind=params_wp) abserr,
              real(kind=params_wp) resabs,
              real(kind=params_wp) resasc )
```

Definition at line 7385 of file quadpack.f90.

```
07386
07387 !*****************************************************************************80
07388 !
07389 !! QK51 carries out a 51 point Gauss-Kronrod quadrature rule.
07390 !
07391 !  Discussion:
07392 !
07393 !    This routine approximates
07394 !      I = integral ( A <= X <= B ) F(X) dx
07395 !    with an error estimate, and
07396 !      J = integral ( A <= X <= B ) | F(X) | dx
07397 !
07398 !  Author:
07399 !
07400 !    Robert Piessens, Elise de Doncker-Kapenger,
07401 !    Christian Ueberhuber, David Kahaner
07402 !
07403 !  Reference:
07404 !
07405 !    Robert Piessens, Elise de Doncker-Kapenger,
07406 !    Christian Ueberhuber, David Kahaner,
07407 !    QUADPACK, a Subroutine Package for Automatic Integration,
07408 !    Springer Verlag, 1983
07409 !
07410 !  Parameters:
07411 !
07412 !    Input, external real F, the name of the function routine, of the form
07413 !      function f ( x )
07414 !      real f
07415 !      real x
07416 !    which evaluates the integrand function.
07417 !
07418 !    Input, real A, B, the limits of integration.
07419 !
07420 !    Output, real RESULT, the estimated value of the integral.
07421 !                        result is computed by applying the 51-point
07422 !                        Kronrod rule (resk) obtained by optimal addition
07423 !                        of abscissae to the 25-point Gauss rule (resg).
07424 !
07425 !    Output, real ABSERR, an estimate of | I - RESULT |.
07426 !
07427 !    Output, real RESABS, approximation to the integral of the absolute
07428 !    value of F.
07429 !
07430 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
07431 !    over [A,B].
07432 !
07433 !  Local Parameters:
07434 !
07435 !           centr  - mid point of the interval
07436 !           hlgth  - half-length of the interval
07437 !           absc   - abscissa
```

```
07438 !           fval*  - function value
07439 !           resg   - result of the 25-point Gauss formula
07440 !           resk   - result of the 51-point Kronrod formula
07441 !           reskh  - approximation to the mean value of f over (a,b),
07442 !                    i.e. to i/(b-a)
07443 !
07444   implicit none
07445
07446   real(kind=params_wp) a
07447   real(kind=params_wp) absc
07448   real(kind=params_wp) abserr
07449   real(kind=params_wp) b
07450   real(kind=params_wp) centr
07451   real(kind=params_wp) dhlgth
07452   real(kind=params_wp), external :: f
07453   real(kind=params_wp) fc
07454   real(kind=params_wp) fsum
07455   real(kind=params_wp) fval1
07456   real(kind=params_wp) fval2
07457   real(kind=params_wp) fv1(25)
07458   real(kind=params_wp) fv2(25)
07459   real(kind=params_wp) hlgth
07460   integer j
07461   integer jtw
07462   integer jtwm1
07463   real(kind=params_wp) resabs
07464   real(kind=params_wp) resasc
07465   real(kind=params_wp) resg
07466   real(kind=params_wp) resk
07467   real(kind=params_wp) reskh
07468   real(kind=params_wp) result
07469   real(kind=params_wp) wg(13)
07470   real(kind=params_wp) wgk(26)
07471   real(kind=params_wp) xgk(26)
07472 !
07473 !           the abscissae and weights are given for the interval (-1,1).
07474 !           because of symmetry only the positive abscissae and their
07475 !           corresponding weights are given.
07476 !
07477 !           xgk    - abscissae of the 51-point Kronrod rule
07478 !                    xgk(2), xgk(4), ...  abscissae of the 25-point
07479 !                    Gauss rule
07480 !                    xgk(1), xgk(3), ...  abscissae which are optimally
07481 !                    added to the 25-point Gauss rule
07482 !
07483 !           wgk    - weights of the 51-point Kronrod rule
07484 !
07485 !           wg     - weights of the 25-point Gauss rule
07486 !
07487   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
07488     xgk(9),xgk(10),xgk(11),xgk(12),xgk(13),xgk(14)/ &
07489        9.992621049926098e-01,   9.955569697904981e-01, &
07490        9.880357945340772e-01,   9.766639214595175e-01, &
07491        9.616149864258425e-01,   9.429745712289743e-01, &
07492        9.207471152817016e-01,   8.949919978782754e-01, &
07493        8.658470652932756e-01,   8.334426287608340e-01, &
07494        7.978737979985001e-01,   7.592592630373576e-01, &
07495        7.177664068130844e-01,   6.735663684734684e-01/
07496    data xgk(15),xgk(16),xgk(17),xgk(18),xgk(19),xgk(20),xgk(21), &
07497     xgk(22),xgk(23),xgk(24),xgk(25),xgk(26)/ &
07498        6.268100990103174e-01,   5.776629302412230e-01, &
07499        5.263252843347192e-01,   4.730027314457150e-01, &
07500        4.178853821930377e-01,   3.611723058093878e-01, &
07501        3.038095389311078e-01,   2.438668837209884e-01, &
07502        1.837189394210489e-01,   1.228646926107104e-01, &
07503        6.154448300568508e-02,   0.0e+00               /
07504   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
07505     wgk(9),wgk(10),wgk(11),wgk(12),wgk(13),wgk(14)/ &
07506        1.987383892330316e-03,   5.561932135356714e-03, &
07507        9.473973386174152e-03,   1.323622919557167e-02, &
07508        1.684781770912830e-02,   2.043537114588284e-02, &
07509        2.400994560695322e-02,   2.747531758785174e-02, &
07510        3.079230016738749e-02,   3.400213027432934e-02, &
07511        3.711627148341554e-02,   4.008382550403238e-02, &
07512        4.287284502017005e-02,   4.550291304992179e-02/
07513    data wgk(15),wgk(16),wgk(17),wgk(18),wgk(19),wgk(20),wgk(21), &
07514     wgk(22),wgk(23),wgk(24),wgk(25),wgk(26)/ &
07515        4.798253713883671e-02,   5.027767908071567e-02, &
07516        5.236288580640748e-02,   5.425112988854549e-02, &
07517        5.595081122041232e-02,   5.743711636156783e-02, &
07518        5.868968002239421e-02,   5.972034032417406e-02, &
07519        6.053945537604586e-02,   6.112850971705305e-02, &
07520        6.147118987142532e-02,   6.158081806783294e-02/
07521   data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8),wg(9),wg(10), &
07522     wg(11),wg(12),wg(13)/ &
07523        1.139379850102629e-02,   2.635498661503214e-02, &
07524        4.093915670130631e-02,   5.490469597583519e-02, &
```

```
07525         6.803833381235692e-02,   8.014070033500102e-02, &
07526         9.102826198296365e-02,   1.005359490670506e-01, &
07527         1.085196244742637e-01,   1.148582591457116e-01, &
07528         1.194557635357848e-01,   1.222424429903100e-01, &
07529         1.231760537267155e-01/
07530 !
07531   centr = 5.0e-01*(a+b)
07532   hlgth = 5.0e-01*(b-a)
07533   dhlgth = abs(hlgth)
07534 !
07535 !  Compute the 51-point Kronrod approximation to the integral,
07536 !  and estimate the absolute error.
07537 !
07538   fc = f(centr)
07539   resg = wg(13)*fc
07540   resk = wgk(26)*fc
07541   resabs = abs(resk)
07542
07543   do j = 1, 12
07544     jtw = j*2
07545     absc = hlgth*xgk(jtw)
07546     fval1 = f(centr-absc)
07547     fval2 = f(centr+absc)
07548     fv1(jtw) = fval1
07549     fv2(jtw) = fval2
07550     fsum = fval1+fval2
07551     resg = resg+wg(j)*fsum
07552     resk = resk+wgk(jtw)*fsum
07553     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
07554   end do
07555
07556   do j = 1, 13
07557     jtwm1 = j*2-1
07558     absc = hlgth*xgk(jtwm1)
07559     fval1 = f(centr-absc)
07560     fval2 = f(centr+absc)
07561     fv1(jtwm1) = fval1
07562     fv2(jtwm1) = fval2
07563     fsum = fval1+fval2
07564     resk = resk+wgk(jtwm1)*fsum
07565     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
07566   end do
07567
07568   reskh = resk*5.0e-01
07569   resasc = wgk(26)*abs(fc-reskh)
07570
07571   do j = 1, 25
07572     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
07573   end do
07574
07575   result = resk*hlgth
07576   resabs = resabs*dhlgth
07577   resasc = resasc*dhlgth
07578   abserr = abs((resk-resg)*hlgth)
07579
07580   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) then
07581     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
07582   end if
07583
07584   if ( resabs > tiny( resabs ) / (5.0e+01* epsilon( resabs ) ) ) then
07585     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
07586   end if
07587
07588   return
```

Here is the caller graph for this function:

14.37.1.27 qk61()

```
subroutine quadpack::qk61 (
```

```
            real(kind=params_wp), external f,
            real(kind=params_wp) a,
            real(kind=params_wp) b,
            real(kind=params_wp) result,
            real(kind=params_wp) abserr,
            real(kind=params_wp) resabs,
            real(kind=params_wp) resasc )
```

Definition at line 7590 of file quadpack.f90.

```
07591
07592 !*****************************************************************************80
07593 !
07594 !! QK61 carries out a 61 point Gauss-Kronrod quadrature rule.
07595 !
07596 !  Discussion:
07597 !
07598 !    This routine approximates
07599 !      I = integral ( A <= X <= B ) F(X) dx
07600 !    with an error estimate, and
07601 !      J = integral ( A <= X <= B ) | F(X) | dx
07602 !
07603 !  Author:
07604 !
07605 !    Robert Piessens, Elise de Doncker-Kapenger,
07606 !    Christian Ueberhuber, David Kahaner
07607 !
07608 !  Reference:
07609 !
07610 !    Robert Piessens, Elise de Doncker-Kapenger,
07611 !    Christian Ueberhuber, David Kahaner,
07612 !    QUADPACK, a Subroutine Package for Automatic Integration,
07613 !    Springer Verlag, 1983
07614 !
07615 !  Parameters:
07616 !
07617 !    Input, external real F, the name of the function routine, of the form
07618 !      function f ( x )
07619 !      real f
07620 !      real x
07621 !    which evaluates the integrand function.
07622 !
07623 !    Input, real A, B, the limits of integration.
07624 !
07625 !    Output, real RESULT, the estimated value of the integral.
07626 !                    result is computed by applying the 61-point
07627 !                    Kronrod rule (resk) obtained by optimal addition of
07628 !                    abscissae to the 30-point Gauss rule (resg).
07629 !
07630 !    Output, real ABSERR, an estimate of | I - RESULT |.
07631 !
07632 !    Output, real RESABS, approximation to the integral of the absolute
07633 !    value of F.
07634 !
07635 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
07636 !    over [A,B].
07637 !
07638 !  Local Parameters:
07639 !
07640 !           centr  - mid point of the interval
07641 !           hlgth  - half-length of the interval
07642 !           absc   - abscissa
07643 !           fval*  - function value
07644 !           resg   - result of the 30-point Gauss rule
07645 !           resk   - result of the 61-point Kronrod rule
07646 !           reskh  - approximation to the mean value of f
07647 !                    over (a,b), i.e. to i/(b-a)
07648 !
07649   implicit none
07650
07651   real(kind=params_wp) a
07652   real(kind=params_wp) absc
07653   real(kind=params_wp) abserr
07654   real(kind=params_wp) b
07655   real(kind=params_wp) centr
07656   real(kind=params_wp) dhlgth
07657   real(kind=params_wp), external :: f
07658   real(kind=params_wp) fc
07659   real(kind=params_wp) fsum
07660   real(kind=params_wp) fval1
07661   real(kind=params_wp) fval2
07662   real(kind=params_wp) fv1(30)
07663   real(kind=params_wp) fv2(30)
07664   real(kind=params_wp) hlgth
07665   integer j
07666   integer jtw
```

```
07667   integer jtwm1
07668   real(kind=params_wp) resabs
07669   real(kind=params_wp) resasc
07670   real(kind=params_wp) resg
07671   real(kind=params_wp) resk
07672   real(kind=params_wp) reskh
07673   real(kind=params_wp) result
07674   real(kind=params_wp) wg(15)
07675   real(kind=params_wp) wgk(31)
07676   real(kind=params_wp) xgk(31)
07677 !
07678 !           the abscissae and weights are given for the
07679 !           interval (-1,1). because of symmetry only the positive
07680 !           abscissae and their corresponding weights are given.
07681 !
07682 !           xgk   - abscissae of the 61-point Kronrod rule
07683 !                   xgk(2), xgk(4)  ... abscissae of the 30-point
07684 !                   Gauss rule
07685 !                   xgk(1), xgk(3)  ... optimally added abscissae
07686 !                   to the 30-point Gauss rule
07687 !
07688 !           wgk   - weights of the 61-point Kronrod rule
07689 !
07690 !           wg    - weigths of the 30-point Gauss rule
07691 !
07692   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
07693      xgk(9),xgk(10)/ &
07694        9.994844100504906e-01,     9.968934840746495e-01, &
07695        9.916309968704046e-01,     9.836681232797472e-01, &
07696        9.731163225011263e-01,     9.600218649683075e-01, &
07697        9.443744447485600e-01,     9.262000474292743e-01, &
07698        9.055733076999078e-01,     8.825605357920527e-01/
07699   data xgk(11),xgk(12),xgk(13),xgk(14),xgk(15),xgk(16),xgk(17), &
07700     xgk(18),xgk(19),xgk(20)/ &
07701        8.572052335460611e-01,     8.295657623827684e-01, &
07702        7.997278358218391e-01,     7.677774321048262e-01, &
07703        7.337900624532268e-01,     6.978504947933158e-01, &
07704        6.600610641266270e-01,     6.205261829892429e-01, &
07705        5.793452358263617e-01,     5.366241481420199e-01/
07706   data xgk(21),xgk(22),xgk(23),xgk(24),xgk(25),xgk(26),xgk(27), &
07707     xgk(28),xgk(29),xgk(30),xgk(31)/ &
07708        4.924804678617786e-01,     4.470337695380892e-01, &
07709        4.004012548303944e-01,     3.527047255308781e-01, &
07710        3.040732022736251e-01,     2.546369261678898e-01, &
07711        2.045251166823099e-01,     1.538699136085835e-01, &
07712        1.028069379667370e-01,     5.147184255531770e-02, &
07713        0.0e+00                   /
07714   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
07715     wgk(9),wgk(10)/ &
07716        1.389013698677008e-03,     3.890461127099884e-03, &
07717        6.630703915931292e-03,     9.273279659517763e-03, &
07718        1.182301525349634e-02,     1.436972950704580e-02, &
07719        1.692088918905327e-02,     1.941414119394238e-02, &
07720        2.182803582160919e-02,     2.419116207808060e-02/
07721   data wgk(11),wgk(12),wgk(13),wgk(14),wgk(15),wgk(16),wgk(17), &
07722     wgk(18),wgk(19),wgk(20)/ &
07723        2.650995488233310e-02,     2.875404876504129e-02, &
07724        3.090725756238776e-02,     3.298144705748373e-02, &
07725        3.497933802806002e-02,     3.688236465182123e-02, &
07726        3.867894562472759e-02,     4.037453895153596e-02, &
07727        4.196981021516425e-02,     4.345253970135607e-02/
07728   data wgk(21),wgk(22),wgk(23),wgk(24),wgk(25),wgk(26),wgk(27), &
07729     wgk(28),wgk(29),wgk(30),wgk(31)/ &
07730        4.481480013316266e-02,     4.605923827100699e-02, &
07731        4.718554656929915e-02,     4.818586175708713e-02, &
07732        4.905543455502978e-02,     4.979568342707421e-02, &
07733        5.040592140278235e-02,     5.088179589874961e-02, &
07734        5.122154784925877e-02,     5.142612853745903e-02, &
07735        5.149472942945157e-02/
07736   data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8)/ &
07737        7.968192496166606e-03,     1.846646831109096e-02, &
07738        2.878470788332337e-02,     3.879919256962705e-02, &
07739        4.840267283059405e-02,     5.749315621761907e-02, &
07740        6.597422988218050e-02,     7.375597473770521e-02/
07741   data wg(9),wg(10),wg(11),wg(12),wg(13),wg(14),wg(15)/ &
07742        8.075589522942022e-02,     8.689978720108298e-02, &
07743        9.212252223778613e-02,     9.636873717464426e-02, &
07744        9.959342058679527e-02,     1.017623897484055e-01, &
07745        1.028526528935588e-01/
07746
07747   centr = 5.0e-01*(b+a)
07748   hlgth = 5.0e-01*(b-a)
07749   dhlgth = abs(hlgth)
07750 !
07751 !  Compute the 61-point Kronrod approximation to the integral,
07752 !  and estimate the absolute error.
07753 !
```

```
07754   resg = 0.0e+00
07755   fc = f(centr)
07756   resk = wgk(31)*fc
07757   resabs = abs(resk)
07758
07759   do j = 1, 15
07760     jtw = j*2
07761     absc = hlgth*xgk(jtw)
07762     fval1 = f(centr-absc)
07763     fval2 = f(centr+absc)
07764     fv1(jtw) = fval1
07765     fv2(jtw) = fval2
07766     fsum = fval1+fval2
07767     resg = resg+wg(j)*fsum
07768     resk = resk+wgk(jtw)*fsum
07769     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
07770   end do
07771
07772   do j = 1, 15
07773     jtwm1 = j*2-1
07774     absc = hlgth*xgk(jtwm1)
07775     fval1 = f(centr-absc)
07776     fval2 = f(centr+absc)
07777     fv1(jtwm1) = fval1
07778     fv2(jtwm1) = fval2
07779     fsum = fval1+fval2
07780     resk = resk+wgk(jtwm1)*fsum
07781     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
07782   end do
07783
07784   reskh = resk * 5.0e-01
07785   resasc = wgk(31)*abs(fc-reskh)
07786
07787   do j = 1, 30
07788     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
07789   end do
07790
07791   result = resk*hlgth
07792   resabs = resabs*dhlgth
07793   resasc = resasc*dhlgth
07794   abserr = abs((resk-resg)*hlgth)
07795
07796   if ( resasc /= 0.0e+00 .and. abserr /= 0.0e+00) then
07797     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
07798   end if
07799
07800   if ( resabs > tiny( resabs ) / (5.0e+01* epsilon( resabs ) )) then
07801     abserr = max( ( epsilon( resabs ) *5.0e+01)*resabs, abserr )
07802   end if
07803
07804   return
```

Here is the caller graph for this function:

14.37.1.28 qmomo()

```
subroutine quadpack::qmomo (
             real(kind=params_wp) alfa,
             real(kind=params_wp) beta,
             real(kind=params_wp), dimension(25) ri,
             real(kind=params_wp), dimension(25) rj,
             real(kind=params_wp), dimension(25) rg,
             real(kind=params_wp), dimension(25) rh,
             integer integr )
```

Definition at line 7806 of file quadpack.f90.

```
07807
```

```
07808 !*****************************************************************************80
07809 !
07810 !! QMOMO computes modified Chebyshev moments.
07811 !
07812 !  Discussion:
07813 !
07814 !    This routine computes modified Chebyshev moments.
07815 !    The K-th modified Chebyshev moment is defined as the
07816 !    integral over (-1,1) of W(X)*T(K,X), where T(K,X) is the
07817 !    Chebyshev polynomial of degree K.
07818 !
07819 !  Author:
07820 !
07821 !    Robert Piessens, Elise de Doncker-Kapenger,
07822 !    Christian Ueberhuber, David Kahaner
07823 !
07824 !  Reference:
07825 !
07826 !    Robert Piessens, Elise de Doncker-Kapenger,
07827 !    Christian Ueberhuber, David Kahaner,
07828 !    QUADPACK, a Subroutine Package for Automatic Integration,
07829 !    Springer Verlag, 1983
07830 !
07831 !  Parameters:
07832 !
07833 !    Input, real ALFA, a parameter in the weight function w(x), ALFA > -1.
07834 !
07835 !    Input, real BETA, a parameter in the weight function w(x), BETA > -1.
07836 !
07837 !           ri     - real
07838 !                    vector of dimension 25
07839 !                    ri(k) is the integral over (-1,1) of
07840 !                    (1+x)**alfa*t(k-1,x), k = 1, ..., 25.
07841 !
07842 !           rj     - real
07843 !                    vector of dimension 25
07844 !                    rj(k) is the integral over (-1,1) of
07845 !                    (1-x)**beta*t(k-1,x), k = 1, ..., 25.
07846 !
07847 !           rg     - real
07848 !                    vector of dimension 25
07849 !                    rg(k) is the integral over (-1,1) of
07850 !                    (1+x)**alfa*log((1+x)/2)*t(k-1,x), k = 1, ...,25.
07851 !
07852 !           rh     - real
07853 !                    vector of dimension 25
07854 !                    rh(k) is the integral over (-1,1) of
07855 !                    (1-x)**beta*log((1-x)/2)*t(k-1,x), k = 1, ..., 25.
07856 !
07857 !           integr - integer
07858 !                    input parameter indicating the modified moments
07859 !                    to be computed
07860 !                    integr = 1 compute ri, rj
07861 !                           = 2 compute ri, rj, rg
07862 !                           = 3 compute ri, rj, rh
07863 !                           = 4 compute ri, rj, rg, rh
07864 !
07865   implicit none
07866
07867   real(kind=params_wp) alfa
07868   real(kind=params_wp) alfp1
07869   real(kind=params_wp) alfp2
07870   real(kind=params_wp) an
07871   real(kind=params_wp) anm1
07872   real(kind=params_wp) beta
07873   real(kind=params_wp) betp1
07874   real(kind=params_wp) betp2
07875   integer i
07876   integer im1
07877   integer integr
07878   real(kind=params_wp) ralf
07879   real(kind=params_wp) rbet
07880   real(kind=params_wp) rg(25)
07881   real(kind=params_wp) rh(25)
07882   real(kind=params_wp) ri(25)
07883   real(kind=params_wp) rj(25)
07884 !
07885   alfp1 = alfa+1.0e+00_params_wp
07886   betp1 = beta+1.0e+00_params_wp
07887   alfp2 = alfa+2.0e+00
07888   betp2 = beta+2.0e+00
07889   ralf = 2.0e+00**alfp1
07890   rbet = 2.0e+00**betp1
07891 !
07892 !  Compute RI, RJ using a forward recurrence relation.
07893 !
07894   ri(1) = ralf/alfp1
```

```
07895   rj(1) = rbet/betp1
07896   ri(2) = ri(1)*alfa/alfp2
07897   rj(2) = rj(1)*beta/betp2
07898   an = 2.0e+00
07899   anm1 = 1.0e+00_params_wp
07900
07901   do i = 3, 25
07902     ri(i) = -(ralf+an*(an-alfp2)*ri(i-1))/(anm1*(an+alfp1))
07903     rj(i) = -(rbet+an*(an-betp2)*rj(i-1))/(anm1*(an+betp1))
07904     anm1 = an
07905     an = an+1.0e+00_params_wp
07906   end do
07907
07908   if ( integr == 1 ) go to 70
07909   if ( integr == 3 ) go to 40
07910 !
07911 !  Compute RG using a forward recurrence relation.
07912 !
07913   rg(1) = -ri(1)/alfp1
07914   rg(2) = -(ralf+ralf)/(alfp2*alfp2)-rg(1)
07915   an = 2.0e+00
07916   anm1 = 1.0e+00_params_wp
07917   im1 = 2
07918
07919   do i = 3, 25
07920     rg(i) = -(an*(an-alfp2)*rg(im1)-an*ri(im1)+anm1*ri(i))/ &
07921     (anm1*(an+alfp1))
07922     anm1 = an
07923     an = an+1.0e+00_params_wp
07924     im1 = i
07925   end do
07926
07927   if ( integr == 2 ) go to 70
07928 !
07929 !  Compute RH using a forward recurrence relation.
07930 !
07931 40 continue
07932
07933   rh(1) = -rj(1) / betp1
07934   rh(2) = -(rbet+rbet)/(betp2*betp2)-rh(1)
07935   an = 2.0e+00
07936   anm1 = 1.0e+00_params_wp
07937   im1 = 2
07938
07939   do i = 3, 25
07940     rh(i) = -(an*(an-betp2)*rh(im1)-an*rj(im1)+ &
07941     anm1*rj(i))/(anm1*(an+betp1))
07942     anm1 = an
07943     an = an+1.0e+00_params_wp
07944     im1 = i
07945   end do
07946
07947   do i = 2, 25, 2
07948     rh(i) = -rh(i)
07949   end do
07950
07951    70 continue
07952
07953   do i = 2, 25, 2
07954     rj(i) = -rj(i)
07955   end do
07956
07957 !  90 continue
07958
07959   return
```

Here is the caller graph for this function:

14.37.1.29 qng()

```
subroutine quadpack::qng (
```

```
            real(kind=params_wp), external f,
            real(kind=params_wp) a,
            real(kind=params_wp) b,
            real(kind=params_wp) epsabs,
            real(kind=params_wp) epsrel,
            real(kind=params_wp) result,
            real(kind=params_wp) abserr,
            integer neval,
            integer ier )
```

Definition at line 7961 of file quadpack.f90.

```
07962
07963 !*******************************************************************************80
07964 !
07965 !! QNG estimates an integral, using non-adaptive integration.
07966 !
07967 !  Discussion:
07968 !
07969 !    The routine calculates an approximation RESULT to a definite integral
07970 !      I = integral of F over (A,B),
07971 !    hopefully satisfying
07972 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
07973 !
07974 !    The routine is a simple non-adaptive automatic integrator, based on
07975 !    a sequence of rules with increasing degree of algebraic
07976 !    precision (Patterson, 1968).
07977 !
07978 !  Author:
07979 !
07980 !    Robert Piessens, Elise de Doncker-Kapenger,
07981 !    Christian Ueberhuber, David Kahaner
07982 !
07983 !  Reference:
07984 !
07985 !    Robert Piessens, Elise de Doncker-Kapenger,
07986 !    Christian Ueberhuber, David Kahaner,
07987 !    QUADPACK, a Subroutine Package for Automatic Integration,
07988 !    Springer Verlag, 1983
07989 !
07990 !  Parameters:
07991 !
07992 !    Input, external real F, the name of the function routine, of the form
07993 !      function f ( x )
07994 !      real(kind=params_wp) f
07995 !      real(kind=params_wp) x
07996 !    which evaluates the integrand function.
07997 !
07998 !    Input, real A, B, the limits of integration.
07999 !
08000 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
08001 !
08002 !    Output, real RESULT, the estimated value of the integral.
08003 !    RESULT is obtained by applying the 21-point Gauss-Kronrod rule (RES21)
08004 !    obtained  by optimal addition of abscissae to the 10-point Gauss rule
08005 !    (RES10), or by applying the 43-point rule (RES43) obtained by optimal
08006 !    addition of abscissae to the 21-point Gauss-Kronrod rule, or by
08007 !    applying the 87-point rule (RES87) obtained by optimal addition of
08008 !    abscissae to the 43-point rule.
08009 !
08010 !    Output, real ABSERR, an estimate of || I - RESULT ||.
08011 !
08012 !    Output, integer NEVAL, the number of times the integral was evaluated.
08013 !
08014 !            ier    - ier = 0 normal and reliable termination of the
08015 !                             routine. it is assumed that the requested
08016 !                             accuracy has been achieved.
08017 !                     ier > 0 abnormal termination of the routine. it is
08018 !                             assumed that the requested accuracy has
08019 !                             not been achieved.
08020 !                     ier = 1 the maximum number of steps has been
08021 !                             executed. the integral is probably too
08022 !                             difficult to be calculated by qng.
08023 !                         = 6 the input is invalid, because
08024 !                             epsabs < 0 and epsrel < 0,
08025 !                             result, abserr and neval are set to zero.
08026 !
08027 !  Local Parameters:
08028 !
08029 !           centr  - mid point of the integration interval
08030 !           hlgth  - half-length of the integration interval
08031 !           fcentr - function value at mid point
08032 !           absc   - abscissa
08033 !           fval   - function value
08034 !           savfun - array of function values which have already
```

```
08035 !                   been computed
08036 !           res10  - 10-point Gauss result
08037 !           res21  - 21-point Kronrod result
08038 !           res43  - 43-point result
08039 !           res87  - 87-point result
08040 !           resabs - approximation to the integral of abs(f)
08041 !           resasc - approximation to the integral of abs(f-i/(b-a))
08042 !
08043   implicit none
08044
08045   real(kind=params_wp) a
08046   real(kind=params_wp) absc
08047   real(kind=params_wp) abserr
08048   real(kind=params_wp) b
08049   real(kind=params_wp) centr
08050   real(kind=params_wp) dhlgth
08051   real(kind=params_wp) epsabs
08052   real(kind=params_wp) epsrel
08053   real(kind=params_wp), external :: f
08054   real(kind=params_wp) fcentr
08055   real(kind=params_wp) fval
08056   real(kind=params_wp) fval1
08057   real(kind=params_wp) fval2
08058   real(kind=params_wp) fv1(5)
08059   real(kind=params_wp) fv2(5)
08060   real(kind=params_wp) fv3(5)
08061   real(kind=params_wp) fv4(5)
08062   real(kind=params_wp) hlgth
08063   integer ier
08064   integer ipx
08065   integer k
08066   integer l
08067   integer neval
08068   real(kind=params_wp) result
08069   real(kind=params_wp) res10
08070   real(kind=params_wp) res21
08071   real(kind=params_wp) res43
08072   real(kind=params_wp) res87
08073   real(kind=params_wp) resabs
08074   real(kind=params_wp) resasc
08075   real(kind=params_wp) reskh
08076   real(kind=params_wp) savfun(21)
08077   real(kind=params_wp) w10(5)
08078   real(kind=params_wp) w21a(5)
08079   real(kind=params_wp) w21b(6)
08080   real(kind=params_wp) w43a(10)
08081   real(kind=params_wp) w43b(12)
08082   real(kind=params_wp) w87a(21)
08083   real(kind=params_wp) w87b(23)
08084   real(kind=params_wp) x1(5)
08085   real(kind=params_wp) x2(5)
08086   real(kind=params_wp) x3(11)
08087   real(kind=params_wp) x4(22)
08088 !
08089 !           the following data statements contain the abscissae
08090 !           and weights of the integration rules used.
08091 !
08092 !           x1      abscissae common to the 10-, 21-, 43- and 87-point
08093 !                   rule
08094 !           x2      abscissae common to the 21-, 43- and 87-point rule
08095 !           x3      abscissae common to the 43- and 87-point rule
08096 !           x4      abscissae of the 87-point rule
08097 !           w10     weights of the 10-point formula
08098 !           w21a    weights of the 21-point formula for abscissae x1
08099 !           w21b    weights of the 21-point formula for abscissae x2
08100 !           w43a    weights of the 43-point formula for absissae x1, x3
08101 !           w43b    weights of the 43-point formula for abscissae x3
08102 !           w87a    weights of the 87-point formula for abscissae x1,
08103 !                   x2 and x3
08104 !           w87b    weights of the 87-point formula for abscissae x4
08105 !
08106   data x1(1),x1(2),x1(3),x1(4),x1(5)/ &
08107        9.739065285171717e-01,     8.650633666889845e-01, &
08108        6.794095682990244e-01,     4.333953941292472e-01, &
08109        1.488743389816312e-01/
08110   data x2(1),x2(2),x2(3),x2(4),x2(5)/ &
08111        9.956571630258081e-01,     9.301574913557082e-01, &
08112        7.808177265864169e-01,     5.627571346686047e-01, &
08113        2.943928627014602e-01/
08114   data x3(1),x3(2),x3(3),x3(4),x3(5),x3(6),x3(7),x3(8),x3(9),x3(10), &
08115     x3(11)/ &
08116        9.993333609019321e-01,     9.874334029080889e-01, &
08117        9.548079348142663e-01,     9.001486957483283e-01, &
08118        8.251983149831142e-01,     7.321483889893050e-01, &
08119        6.228479705377252e-01,     4.994795740710565e-01, &
08120        3.649016613465808e-01,     2.222549197766013e-01, &
08121        7.465061746138332e-02/
```

```
08122   data x4(1),x4(2),x4(3),x4(4),x4(5),x4(6),x4(7),x4(8),x4(9),x4(10), &
08123     x4(11),x4(12),x4(13),x4(14),x4(15),x4(16),x4(17),x4(18),x4(19), &
08124     x4(20),x4(21),x4(22)/         9.990029772627292e-01, &
08125        9.979898959866787e-01,     9.921754978606872e-01, &
08126        9.813581635727128e-01,     9.650576238583846e-01, &
08127        9.431676131336706e-01,     9.158064146855072e-01, &
08128        8.832216577713165e-01,     8.457107484624157e-01, &
08129        8.035576580352310e-01,     7.570057306854956e-01, &
08130        7.062732097873218e-01,     6.515894665011779e-01, &
08131        5.932233740579611e-01,     5.314936059708319e-01, &
08132        4.667636230420228e-01,     3.994248478592188e-01, &
08133        3.298748771061883e-01,     2.585035592021616e-01, &
08134        1.856953965683467e-01,     1.118422131799075e-01, &
08135        3.735212339461987e-02/
08136   data w10(1),w10(2),w10(3),w10(4),w10(5)/ &
08137        6.667134430868814e-02,     1.494513491505806e-01, &
08138        2.190863625159820e-01,     2.692667193099964e-01, &
08139        2.955242247147529e-01/
08140   data w21a(1),w21a(2),w21a(3),w21a(4),w21a(5)/ &
08141        3.255816230796473e-02,     7.503967481091995e-02, &
08142        1.093871588022976e-01,     1.347092173114733e-01, &
08143        1.477391049013385e-01/
08144   data w21b(1),w21b(2),w21b(3),w21b(4),w21b(5),w21b(6)/ &
08145        1.169463886737187e-02,     5.475589657435200e-02, &
08146        9.312545458369761e-02,     1.234919762620659e-01, &
08147        1.427759385770601e-01,     1.494455540029169e-01/
08148   data w43a(1),w43a(2),w43a(3),w43a(4),w43a(5),w43a(6),w43a(7), &
08149     w43a(8),w43a(9),w43a(10)/     1.629673428966656e-02, &
08150        3.752287612086950e-02,     5.469490205825544e-02, &
08151        6.735541460947809e-02,     7.387019963239395e-02, &
08152        5.768556059769796e-03,     2.737189059324884e-02, &
08153        4.656082691042883e-02,     6.174499520144256e-02, &
08154        7.138726726869340e-02/
08155   data w43b(1),w43b(2),w43b(3),w43b(4),w43b(5),w43b(6),w43b(7), &
08156     w43b(8),w43b(9),w43b(10),w43b(11),w43b(12)/ &
08157        1.844477640212414e-03,     1.079868958589165e-02, &
08158        2.189536386779543e-02,     3.259746397534569e-02, &
08159        4.216313793519181e-02,     5.074193960018458e-02, &
08160        5.837939554261925e-02,     6.474640495144589e-02, &
08161        6.956619791235648e-02,     7.282444147183321e-02, &
08162        7.450775101417512e-02,     7.472214751740301e-02/
08163   data w87a(1),w87a(2),w87a(3),w87a(4),w87a(5),w87a(6),w87a(7), &
08164     w87a(8),w87a(9),w87a(10),w87a(11),w87a(12),w87a(13),w87a(14), &
08165     w87a(15),w87a(16),w87a(17),w87a(18),w87a(19),w87a(20),w87a(21)/ &
08166        8.148377384149173e-03,     1.876143820156282e-02, &
08167        2.734745105005229e-02,     3.367770731163793e-02, &
08168        3.693509982042791e-02,     2.884872430211531e-03, &
08169        1.368594602271270e-02,     2.328041350288831e-02, &
08170        3.087249761171336e-02,     3.569363363941877e-02, &
08171        9.152833452022414e-04,     5.399280219300471e-03, &
08172        1.094767960111893e-02,     1.629873169678734e-02, &
08173        2.108156888920384e-02,     2.537096976925383e-02, &
08174        2.918969775647575e-02,     3.237320246720279e-02, &
08175        3.478309895036514e-02,     3.641222073135179e-02, &
08176        3.725387550304771e-02/
08177   data w87b(1),w87b(2),w87b(3),w87b(4),w87b(5),w87b(6),w87b(7), &
08178     w87b(8),w87b(9),w87b(10),w87b(11),w87b(12),w87b(13),w87b(14), &
08179     w87b(15),w87b(16),w87b(17),w87b(18),w87b(19),w87b(20),w87b(21), &
08180     w87b(22),w87b(23)/            2.741455637620724e-04, &
08181        1.807124155057943e-03,     4.096869282759165e-03, &
08182        6.758290051847379e-03,     9.549957672201647e-03, &
08183        1.232944765224485e-02,     1.501044734638895e-02, &
08184        1.754896798624319e-02,     1.993803778644089e-02, &
08185        2.219493596101229e-02,     2.433914712600081e-02, &
08186        2.637450541483921e-02,     2.828691078877120e-02, &
08187        3.005258112809270e-02,     3.164675137143993e-02, &
08188        3.305041341997850e-02,     3.425509970422606e-02, &
08189        3.526241266015668e-02,     3.607698962288870e-02, &
08190        3.669860449845609e-02,     3.712054926983258e-02, &
08191        3.733422875193504e-02,     3.736107376267902e-02/
08192 !
08193 !  Test on validity of parameters.
08194 !
08195   result = 0.0e+00
08196   abserr = 0.0e+00
08197   neval = 0
08198
08199   if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
08200     ier = 6
08201     return
08202   end if
08203
08204   hlgth = 5.0e-01 * ( b - a )
08205   dhlgth = abs( hlgth )
08206   centr = 5.0e-01 * ( b + a )
08207   fcentr = f(centr)
08208   neval = 21
```

```
08209   ier = 1
08210 !
08211 !  Compute the integral using the 10- and 21-point formula.
08212 !
08213   do l = 1, 3
08214
08215     if ( l == 1 ) then
08216
08217       res10 = 0.0e+00
08218       res21 = w21b(6) * fcentr
08219       resabs = w21b(6) * abs(fcentr)
08220
08221       do k = 1, 5
08222         absc = hlgth * x1(k)
08223         fval1 = f(centr+absc)
08224         fval2 = f(centr-absc)
08225         fval = fval1 + fval2
08226         res10 = res10 + w10(k)*fval
08227         res21 = res21 + w21a(k)*fval
08228         resabs = resabs + w21a(k)*(abs(fval1)+abs(fval2))
08229         savfun(k) = fval
08230         fv1(k) = fval1
08231         fv2(k) = fval2
08232       end do
08233
08234       ipx = 5
08235
08236       do k = 1, 5
08237         ipx = ipx + 1
08238         absc = hlgth * x2(k)
08239         fval1 = f(centr+absc)
08240         fval2 = f(centr-absc)
08241         fval = fval1 + fval2
08242         res21 = res21 + w21b(k) * fval
08243         resabs = resabs + w21b(k) * ( abs( fval1 ) + abs( fval2 ) )
08244         savfun(ipx) = fval
08245         fv3(k) = fval1
08246         fv4(k) = fval2
08247       end do
08248 !
08249 !  Test for convergence.
08250 !
08251       result = res21 * hlgth
08252       resabs = resabs * dhlgth
08253       reskh = 5.0e-01 * res21
08254       resasc = w21b(6) * abs( fcentr - reskh )
08255
08256       do k = 1, 5
08257         resasc = resasc+w21a(k)*(abs(fv1(k)-reskh)+abs(fv2(k)-reskh)) &
08258                       +w21b(k)*(abs(fv3(k)-reskh)+abs(fv4(k)-reskh))
08259       end do
08260
08261       abserr = abs( ( res21 - res10 ) * hlgth )
08262       resasc = resasc * dhlgth
08263 !
08264 !  Compute the integral using the 43-point formula.
08265 !
08266     else if ( l == 2 ) then
08267
08268       res43 = w43b(12)*fcentr
08269       neval = 43
08270
08271       do k = 1, 10
08272         res43 = res43 + savfun(k) * w43a(k)
08273       end do
08274
08275       do k = 1, 11
08276         ipx = ipx + 1
08277         absc = hlgth * x3(k)
08278         fval = f(absc+centr) + f(centr-absc)
08279         res43 = res43 + fval * w43b(k)
08280         savfun(ipx) = fval
08281       end do
08282 !
08283 !  Test for convergence.
08284 !
08285       result = res43 * hlgth
08286       abserr = abs((res43-res21)*hlgth)
08287 !
08288 !  Compute the integral using the 87-point formula.
08289 !
08290     else if ( l == 3 ) then
08291
08292       res87 = w87b(23) * fcentr
08293       neval = 87
08294
08295       do k = 1, 21
```

```
08296         res87 = res87 + savfun(k) * w87a(k)
08297       end do
08298
08299       do k = 1, 22
08300         absc = hlgth * x4(k)
08301         res87 = res87 + w87b(k) * ( f(absc+centr) + f(centr-absc) )
08302       end do
08303
08304       result = res87 * hlgth
08305       abserr = abs( ( res87 - res43) * hlgth )
08306
08307     end if
08308
08309     if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00 ) then
08310       abserr = resasc * min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
08311     end if
08312
08313     if ( resabs > tiny( resabs ) / ( 5.0e+01 * epsilon( resabs ) ) ) then
08314       abserr = max(( epsilon( resabs ) *5.0e+01) * resabs, abserr )
08315     end if
08316
08317     if ( abserr <= max( epsabs, epsrel*abs(result))) then
08318       ier = 0
08319     end if
08320
08321     if ( ier == 0 ) then
08322       exit
08323     end if
08324
08325   end do
08326
08327   return
```

14.37.1.30 qsort()

```
subroutine quadpack::qsort (
              integer limit,
              integer last,
              integer maxerr,
              real(kind=params_wp) ermax,
              real(kind=params_wp), dimension(last) elist,
              integer, dimension(last) iord,
              integer nrmax )
```

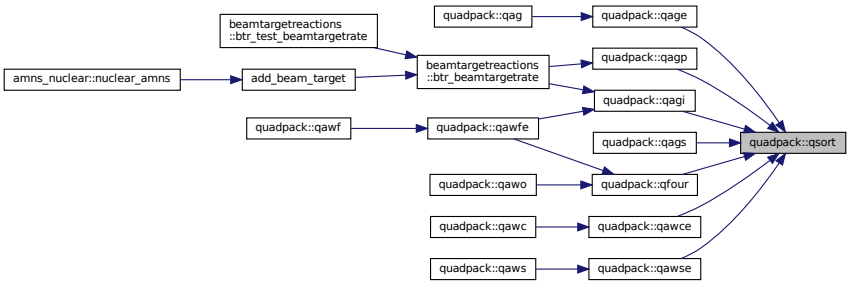
Definition at line 8329 of file quadpack.f90.

```
08330
08331 !***********************************************************************80
08332 !
08333 !! QSORT maintains the order of a list of local error estimates.
08334 !
08335 !  Discussion:
08336 !
08337 !    This routine maintains the descending ordering in the list of the
08338 !    local error estimates resulting from the interval subdivision process.
08339 !    At each call two error estimates are inserted using the sequential
08340 !    search top-down for the largest error estimate and bottom-up for the
08341 !    smallest error estimate.
08342 !
08343 !  Author:
08344 !
08345 !    Robert Piessens, Elise de Doncker-Kapenger,
08346 !    Christian Ueberhuber, David Kahaner
08347 !
08348 !  Reference:
08349 !
08350 !    Robert Piessens, Elise de Doncker-Kapenger,
08351 !    Christian Ueberhuber, David Kahaner,
08352 !    QUADPACK, a Subroutine Package for Automatic Integration,
08353 !    Springer Verlag, 1983
08354 !
08355 !  Parameters:
08356 !
08357 !    Input, integer LIMIT, the maximum number of error estimates the list can
08358 !    contain.
08359 !
08360 !    Input, integer LAST, the current number of error estimates.
08361 !
08362 !    Input/output, integer MAXERR, the index in the list of the NRMAX-th
08363 !    largest error.
08364 !
08365 !    Output, real ERMAX, the NRMAX-th largest error = ELIST(MAXERR).
```

```
08366 !
08367 !    Input, real ELIST(LIMIT), contains the error estimates.
08368 !
08369 !    Input/output, integer IORD(LAST).  The first K elements contain
08370 !    pointers to the error estimates such that ELIST(IORD(1)) through
08371 !    ELIST(IORD(K)) form a decreasing sequence, with
08372 !      K = LAST
08373 !    if
08374 !      LAST <= (LIMIT/2+2),
08375 !    and otherwise
08376 !      K = LIMIT+1-LAST.
08377 !
08378 !    Input/output, integer NRMAX.
08379 !
08380   implicit none
08381
08382   integer last
08383
08384   real(kind=params_wp) elist(last)
08385   real(kind=params_wp) ermax
08386   real(kind=params_wp) errmax
08387   real(kind=params_wp) errmin
08388   integer i
08389   integer ibeg
08390   integer iord(last)
08391   integer isucc
08392   integer j
08393   integer jbnd
08394   integer jupbn
08395   integer k
08396   integer limit
08397   integer maxerr
08398   integer nrmax
08399 !
08400 !  Check whether the list contains more than two error estimates.
08401 !
08402   if ( last <= 2 ) then
08403     iord(1) = 1
08404     iord(2) = 2
08405     go to 90
08406   end if
08407 !
08408 !  This part of the routine is only executed if, due to a
08409 !  difficult integrand, subdivision increased the error
08410 !  estimate. in the normal case the insert procedure should
08411 !  start after the nrmax-th largest error estimate.
08412 !
08413   errmax = elist(maxerr)
08414
08415   do i = 1, nrmax-1
08416
08417     isucc = iord(nrmax-1)
08418
08419     if ( errmax <= elist(isucc) ) then
08420       exit
08421     end if
08422
08423     iord(nrmax) = isucc
08424     nrmax = nrmax-1
08425
08426   end do
08427 !
08428 !  Compute the number of elements in the list to be maintained
08429 !  in descending order.  This number depends on the number of
08430 !  subdivisions still allowed.
08431 !
08432   jupbn = last
08433
08434   if ( (limit/2+2) < last ) then
08435     jupbn = limit+3-last
08436   end if
08437
08438   errmin = elist(last)
08439 !
08440 !  Insert errmax by traversing the list top-down, starting
08441 !  comparison from the element elist(iord(nrmax+1)).
08442 !
08443   jbnd = jupbn-1
08444   ibeg = nrmax+1
08445
08446   do i = ibeg, jbnd
08447     isucc = iord(i)
08448     if ( elist(isucc) <= errmax ) then
08449       go to 60
08450     end if
08451     iord(i-1) = isucc
08452   end do
```

```
08453
08454   iord(jbnd) = maxerr
08455   iord(jupbn) = last
08456   go to 90
08457 !
08458 !  Insert errmin by traversing the list bottom-up.
08459 !
08460 60 continue
08461
08462   iord(i-1) = maxerr
08463   k = jbnd
08464
08465   do j = i, jbnd
08466     isucc = iord(k)
08467     if ( errmin < elist(isucc) ) then
08468       go to 80
08469     end if
08470     iord(k+1) = isucc
08471     k = k-1
08472   end do
08473
08474   iord(i) = last
08475   go to 90
08476
08477 80 continue
08478
08479   iord(k+1) = last
08480 !
08481 !  Set maxerr and ermax.
08482 !
08483 90 continue
08484
08485   maxerr = iord(nrmax)
08486   ermax = elist(maxerr)
08487
08488   return
```

Here is the caller graph for this function:



14.37.1.31 qwgtc()

```
real(kind=params_wp) function quadpack::qwgtc (
            real(kind=params_wp) x,
            real(kind=params_wp) c,
            real(kind=params_wp) p2,
            real(kind=params_wp) p3,
            real(kind=params_wp) p4,
            integer kp )
```

Definition at line 8490 of file quadpack.f90.

```
08491
08492 !*****************************************************************************80
08493 !
08494 !! QWGTC defines the weight function used by QC25C.
08495 !
08496 !  Discussion:
08497 !
08498 !    The weight function has the form 1 / ( X - C ).
08499 !
08500 !  Author:
```

```
08501 !
08502 !    Robert Piessens, Elise de Doncker-Kapenger,
08503 !    Christian Ueberhuber, David Kahaner
08504 !
08505 !  Reference:
08506 !
08507 !    Robert Piessens, Elise de Doncker-Kapenger,
08508 !    Christian Ueberhuber, David Kahaner,
08509 !    QUADPACK, a Subroutine Package for Automatic Integration,
08510 !    Springer Verlag, 1983
08511 !
08512 !  Parameters:
08513 !
08514 !    Input, real X, the point at which the weight function is evaluated.
08515 !
08516 !    Input, real C, the location of the singularity.
08517 !
08518 !    Input, real P2, P3, P4, parameters that are not used.
08519 !
08520 !    Input, integer KP, a parameter that is not used.
08521 !
08522 !    Output, real QWGTC, the value of the weight function at X.
08523 !
08524   implicit none
08525
08526   real(kind=params_wp) c
08527   integer kp
08528   real(kind=params_wp) p2
08529   real(kind=params_wp) p3
08530   real(kind=params_wp) p4
08531   real(kind=params_wp) qwgtc
08532   real(kind=params_wp) x
08533
08534   qwgtc = 1.0e+00_params_wp / ( x - c )
08535
08536   return
```

Here is the caller graph for this function:

14.37.1.32 qwgto()

```
real(kind=params_wp) function quadpack::qwgto (
             real(kind=params_wp) x,
             real(kind=params_wp) omega,
             real(kind=params_wp) p2,
             real(kind=params_wp) p3,
             real(kind=params_wp) p4,
             integer integr )
```

Definition at line 8538 of file quadpack.f90.

```
08539
08540 !****************************************************************************80
08541 !
08542 !! QWGTO defines the weight functions used by QC25O.
08543 !
08544 !  Author:
08545 !
08546 !    Robert Piessens, Elise de Doncker-Kapenger,
08547 !    Christian Ueberhuber, David Kahaner
08548 !
08549 !  Reference:
08550 !
08551 !    Robert Piessens, Elise de Doncker-Kapenger,
08552 !    Christian Ueberhuber, David Kahaner,
08553 !    QUADPACK, a Subroutine Package for Automatic Integration,
08554 !    Springer Verlag, 1983
08555 !
08556 !  Parameters:
08557 !
08558 !    Input, real X, the point at which the weight function is evaluated.
```

```
08559 !
08560 !    Input, real OMEGA, the factor multiplying X.
08561 !
08562 !    Input, real P2, P3, P4, parameters that are not used.
08563 !
08564 !    Input, integer INTEGR, specifies which weight function is used:
08565 !    1. W(X) = cos ( OMEGA * X )
08566 !    2, W(X) = sin ( OMEGA * X )
08567 !
08568 !    Output, real QWGTO, the value of the weight function at X.
08569 !
08570   implicit none
08571
08572   integer integr
08573   real(kind=params_wp) omega
08574   real(kind=params_wp) p2
08575   real(kind=params_wp) p3
08576   real(kind=params_wp) p4
08577   real(kind=params_wp) qwgto
08578   real(kind=params_wp) x
08579
08580   if ( integr == 1 ) then
08581     qwgto = cos( omega * x )
08582   else if ( integr == 2 ) then
08583     qwgto = sin( omega * x )
08584   end if
08585
08586   return
```

Here is the caller graph for this function:

14.37.1.33 qwgts()

```
real(kind=params_wp) function quadpack::qwgts (
             real(kind=params_wp) x,
             real(kind=params_wp) a,
             real(kind=params_wp) b,
             real(kind=params_wp) alfa,
             real(kind=params_wp) beta,
             integer integr )
```

Definition at line 8588 of file quadpack.f90.

```
08589
08590 !*****************************************************************************80
08591 !
08592 !! QWGTS defines the weight functions used by QC25S.
08593 !
08594 !  Author:
08595 !
08596 !    Robert Piessens, Elise de Doncker-Kapenger,
08597 !    Christian Ueberhuber, David Kahaner
08598 !
08599 !  Reference:
08600 !
08601 !    Robert Piessens, Elise de Doncker-Kapenger,
08602 !    Christian Ueberhuber, David Kahaner,
08603 !    QUADPACK, a Subroutine Package for Automatic Integration,
08604 !    Springer Verlag, 1983
08605 !
08606 !  Parameters:
08607 !
08608 !    Input, real X, the point at which the weight function is evaluated.
08609 !
08610 !    Input, real A, B, the endpoints of the integration interval.
08611 !
08612 !    Input, real ALFA, BETA, exponents that occur in the weight function.
08613 !
08614 !    Input, integer INTEGR, specifies which weight function is used:
08615 !    1. W(X) = (X-A)**ALFA * (B-X)**BETA
```

```
08616 !    2, W(X) = (X-A)**ALFA * (B-X)**BETA * log (X-A)
08617 !    3, W(X) = (X-A)**ALFA * (B-X)**BETA * log (B-X)
08618 !    4, W(X) = (X-A)**ALFA * (B-X)**BETA * log (X-A) * log(B-X)
08619 !
08620 !    Output, real QWGTS, the value of the weight function at X.
08621 !
08622   implicit none
08623
08624   real(kind=params_wp) a
08625   real(kind=params_wp) alfa
08626   real(kind=params_wp) b
08627   real(kind=params_wp) beta
08628   integer integr
08629   real(kind=params_wp) qwgts
08630   real(kind=params_wp) x
08631
08632   if ( integr == 1 ) then
08633     qwgts = ( x - a )**alfa * ( b - x )**beta
08634   else if ( integr == 2 ) then
08635     qwgts = ( x - a )**alfa * ( b - x )**beta * log( x - a )
08636   else if ( integr == 3 ) then
08637     qwgts = ( x - a )**alfa * ( b - x )**beta * log( b - x )
08638   else if ( integr == 4 ) then
08639     qwgts = ( x - a )**alfa * ( b - x )**beta * log( x - a ) * log( b - x )
08640   end if
08641
08642   return
```

Here is the caller graph for this function:

14.37.1.34 timestamp()

```
subroutine quadpack::timestamp
```

Definition at line 8644 of file quadpack.f90.

```
08645
08646 !*****************************************************************************80
08647 !
08648 !! TIMESTAMP prints the current YMDHMS date as a time stamp.
08649 !
08650 !  Example:
08651 !
08652 !    May 31 2001   9:45:54.872 AM
08653 !
08654 !  Modified:
08655 !
08656 !    31 May 2001
08657 !
08658 !  Author:
08659 !
08660 !    John Burkardt
08661 !
08662 !  Parameters:
08663 !
08664 !    None
08665 !
08666   implicit none
08667
08668   character ( len = 8 ) ampm
08669   integer d
08670   character ( len = 8 ) date
08671   integer h
08672   integer m
08673   integer mm
08674   character ( len = 9 ), parameter, dimension(12) :: month = (/ &
08675     'January  ', 'February ', 'March    ', 'April    ', &
08676     'May      ', 'June     ', 'July     ', 'August   ', &
08677     'September', 'October  ', 'November ', 'December ' /)
08678   integer n
08679   integer s
08680   character ( len = 10 )  time
08681   integer values(8)
08682   integer y
```

```
08683   character ( len = 5 ) zone
08684
08685   call date_and_time ( date, time, zone, values )
08686
08687   y = values(1)
08688   m = values(2)
08689   d = values(3)
08690   h = values(5)
08691   n = values(6)
08692   s = values(7)
08693   mm = values(8)
08694
08695   if ( h < 12 ) then
08696     ampm = 'AM'
08697   else if ( h == 12 ) then
08698     if ( n == 0 .and. s == 0 ) then
08699       ampm = 'Noon'
08700     else
08701       ampm = 'PM'
08702     end if
08703   else
08704     h = h - 12
08705     if ( h < 12 ) then
08706       ampm = 'PM'
08707     else if ( h == 12 ) then
08708       if ( n == 0 .and. s == 0 ) then
08709         ampm = 'Midnight'
08710       else
08711         ampm = 'AM'
08712       end if
08713     end if
08714   end if
08715
08716   write ( *, '(a,1x,i2,1x,i4,2x,i2,a1,i2.2,a1,i2.2,a1,i3.3,1x,a)' ) &
08717     trim( month(m) ), d, y, h, ':', n, ':', s, '.', mm, trim( ampm )
08718
08719   return
```

14.38 strings Module Reference

strings module from Silvio Gori's grid package

Data Types

- interface operator(//)

Functions/Subroutines

- function print_int_r (char, zahl)
- function print_int_l (zahl, char)
- character(len=len(char)+22) function print_dble_r (char, zahl)
- character(len=len(char)+22) function print_dble_l (zahl, char)
- character(len=len(char)+12) function print_real_r (char, zahl)
- character(len=len(char)+12) function print_real_l (zahl, char)

14.38.1 Detailed Description

strings module from Silvio Gori's grid package
print an integer as a string, which has the length needed to hold the string the definition is rather long and causting, but i don't know how to make it shorter.
over load concatenation operator

Author

Silvio Gori

14.38.2 Function/Subroutine Documentation

14.38.2.1 print_dble_l()

```
character(len=len(char)+22) function strings::print_dble_l (
            real(rkind), intent(in) zahl,
            character(len=*), intent(in) char )
```

Definition at line 83 of file strings.f90.

```
00084     character(len=*), intent(in) :: char
00085     real(rkind) , intent(in) :: zahl
00086     character(len=len(char)+22)::string
00087
00088     write(string,'(1p,e22.15)')zahl
00089     string=char//trim(adjustl(string))
00090
```

14.38.2.2 print_dble_r()

```
character(len=len(char)+22) function strings::print_dble_r (
            character(len=*), intent(in) char,
            real(rkind), intent(in) zahl )
```

Definition at line 71 of file strings.f90.

```
00072     character(len=*), intent(in) :: char
00073     real(rkind) , intent(in) :: zahl
00074     character(len=len(char)+22)::string
00075
00076     write(string,'(1p,e22.15)')zahl
00077     string=char//trim(adjustl(string))
00078
```

14.38.2.3 print_int_l()

```
function strings::print_int_l (
            integer(ikind), intent(in) zahl,
            character(len=*), intent(in) char )
```

Definition at line 59 of file strings.f90.

```
00060     character(len=*), intent(in) :: char
00061     integer(ikind) , intent(in) :: zahl
00062     character(len=len(char)+int(log10(dble(max(1,zahl))))+ &
00063          1-floor(sign(0.1d0,dble(zahl)))) :: string
00064
00065     write(string,'(i0)')zahl
00066     string=trim(adjustl(string))//char
```

14.38.2.4 print_int_r()

```
function strings::print_int_r (
            character(len=*), intent(in) char,
            integer(ikind), intent(in) zahl )
```

Definition at line 45 of file strings.f90.

```
00046
00047     character(len=*), intent(in) :: char
00048     integer(ikind) , intent(in) :: zahl
00049     character(len=len(char)+int(log10(dble(max(1,zahl))))+ &
00050          1-floor(sign(0.1d0,dble(zahl))))::string
00051
00052     write(string,'(i0)')zahl
00053     string=char//trim(adjustl(string))
00054
```

14.38.2.5 print_real_l()

```
character(len=len(char)+12) function strings::print_real_l (
            real, intent(in) zahl,
            character(len=*), intent(in) char )
```

Definition at line 107 of file strings.f90.

```
00108     character(len=*), intent(in) :: char
00109     real , intent(in) :: zahl
```

```
00110     character(len=len(char)+12)::string
00111
00112     write(string,'(1p,e12.6)')dble(zahl)
00113     string=char//trim(adjustl(string))
00114
```

14.38.2.6 print_real_r()

```
character(len=len(char)+12) function strings::print_real_r (
            character(len=*), intent(in) char,
            real, intent(in) zahl )
```

Definition at line 95 of file strings.f90.

```
00096     character(len=*), intent(in) :: char
00097     real , intent(in) :: zahl
00098     character(len=len(char)+12)::string
00099
00100     write(string,'(1p,e12.6)')dble(zahl)
00101     string=char//trim(adjustl(string))
00102
```

14.39 testminimal Namespace Reference

Variables

- amnsdb = amns.Amns()
- r = amns.Reactants()
- lr
- table = amnsdb.get_table(b"CX", r)
- dat = table.data(np.array([100.0]), np.array([1e20]))

14.39.1 Variable Documentation

14.39.1.1 amnsdb

```
testminimal.amnsdb = amns.Amns()
```

Definition at line 5 of file testminimal.py.

14.39.1.2 dat

```
testminimal.dat = table.data(np.array([100.0]), np.array([1e20]))
```

Definition at line 14 of file testminimal.py.

14.39.1.3 lr

```
testminimal.lr
```

Definition at line 9 of file testminimal.py.

14.39.1.4 r

```
testminimal.r = amns.Reactants()
```

Definition at line 6 of file testminimal.py.

14.39.1.5 table

```
testminimal.table = amnsdb.get_table(b"CX", r)
```

Definition at line 12 of file testminimal.py.

14.40 unit_h Module Reference

unit_h module from Silvio Gori's grid package

Functions/Subroutines

- integer(ikind) function, public next_unit ()
- subroutine, public skip_comment_line (iunit)
- subroutine, public read_error (iunit)

14.40.1 Detailed Description

unit_h module from Silvio Gori's grid package

Author

Silvio Gori

14.40.2 Function/Subroutine Documentation

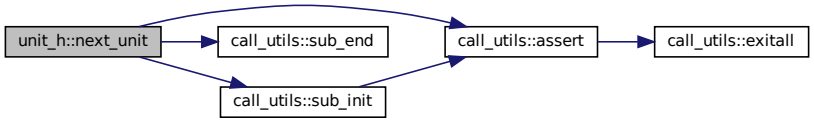
14.40.2.1 next_unit()

```
integer(ikind) function, public unit_h::next_unit
```

Definition at line 28 of file unit_h.f90.

```
00029
00030     integer(ikind)     :: unit_num          !! return the next free unit number
00031
00032     logical            :: op                !! false if unit not in use
00033
00034     call sub_init("next_unit")
00035     do
00036        counter = counter +1
00037        inquire (counter, opened = op)       ! check if unit is open ! bug in f95n
00038        if (.not.op) exit                    ! found next free unit
00039        if (counter > maxcount) &
00040             call assert(.false.,' fatal : next_unit@unit_h could not find free unit number')
00041     enddo
00042
00043     unit_num = counter
00044
00045     call sub_end()
00046
```

Here is the call graph for this function:



14.40.2.2 read_error()

```
subroutine, public unit_h::read_error (
             integer(ikind) iunit )
```

Definition at line 77 of file unit_h.f90.

```
00078     integer(ikind)  :: iunit
00079
00080     character(len=256) :: last_record
00081
00082     backspace(iunit)
```

```
00083
00084     read(iunit,'(a256)') last_record
00085
00086     print *, 'read_error@unit_h: last string read'
00087     print *, trim(last_record)
00088
00089     call exiting(.false., 'read_error@unit_h: last string read:"'//last_record//'"')
00090
```

Here is the call graph for this function:

14.40.2.3 skip_comment_line()

```
subroutine, public unit_h::skip_comment_line (
              integer(ikind), intent(in) iunit )
```

Definition at line 54 of file unit_h.f90.

```
00055
00056     integer(ikind), intent(in) :: iunit
00057
00058     character(skind) :: char
00059
00060     integer :: iostat
00061
00062     do
00063        read(iunit,'(a)',iostat=iostat) char
00064        !print *, '#', trim(char),'#'
00065        if(iostat /= 0) return
00066        if(char(1:1)/='!'.and.len(trim(char))>0) then
00067           backspace iunit
00068           return
00069        endif
00070     enddo
00071
```

Chapter 15

Data Type Documentation

15.1 amns.Amns Class Reference

Public Member Functions

- def __init__ (self, version_string=None, version_number=None, version_backend=None, version_↩ user=None)
- def query (self, queryString)
- def version (self)
- def prop_comment (self)
- def prop_source (self)
- def prop_provider (self)
- def prop_creation (self)
- def code_name (self)
- def code_commit (self)
- def code_version (self)
- def code_repository (self)
- def set (self, setString)
- def finalize (self)
- def get_table (self, reactionString, Reactants, reactants, isotope_resolved=None)
- native long ImasAmnsCcSetupReactants (int idx)
- native long ImasAmnsCcSetupReactantsNumber (int idx, int n_reactants)
- native void ImasAmnsCCSetReactant (long ptr, AmnsReactantType settings)
- native void ImasAmnsCCSetReactantIdx (long ptr, int reactant_idx, AmnsReactantType settings)
- native long ImasAmnsCCSetup (AmnsErrorType error)
- native void ImasAmnsCCGetReactant (long ptr, AmnsReactantType species)
- native double ImasAmnsCCRX0B (long ptr, double arg1, double arg2, AmnsErrorType error)
- native double[ ] ImasAmnsCCRX1A (long ptr, int nx, double[ ] arg1, AmnsErrorType error)
- native double[ ] ImasAmnsCCRX1B (long ptr, int nx, double[ ] arg1, double[ ] arg2, AmnsErrorType error)
- native double[ ] ImasAmnsCCRX1C (long ptr, int nx, double[ ] arg1, double[ ] arg2, double[ ] arg3, AmnsErrorType error)
- native long ImasAmnsCCFinishTable (long ptr, AmnsErrorType error)
- native long ImasAmnsCCFinishReactants (long ptr)
- native long ImasAmnsCCFinish (long ptr, AmnsErrorType error)
- native long ImasAmnsCCSetupTable (long ptrAmns, AmnsReactionType reaction, long ptrReactants, AmnsErrorType error)
- native void ImasAmnsCCSet (long ptrAmns, AmnsSetType setType, AmnsErrorType error)
- native void ImasAmnsCCSetTable (long ptr, AmnsSetType setType, AmnsErrorType error)
- native void ImasAmnsCCQuery (long ptrAmns, AmnsQueryType query, AmnsAnswerType answer, AmnsErrorType error)
- native void ImasAmnsCCQueryTable (long ptrTable, AmnsQueryType query, AmnsAnswerType answer, AmnsErrorType error)

Static Public Member Functions

- static void printErrorCode (AmnsErrorType error, String message)
- static double[ ] reshape2DTo1D (double[ ][ ] input, int nx, int ny)
- static double[ ][ ] reshape1DTo2D (double[ ] input, int nx, int ny)

Static Public Attributes

- string
- tmp_c_str = queryString.encode('UTF-8')

15.1.1 Detailed Description

Provides access to AMNS related codes via JNI interface. It requires libamnsjni.so to be available in $LD_LIBRA↩ RY_PATH (or use -Djava.library.path) - please note that AMNS library provides a JNI interface to AMNS. It means that before you start using Java interface, make sure to load AMNS environment first.
Note that this class contains static block for loading JNI library. Once the first object of the class is created, the JNI library is loaded.
Definition at line 12 of file amns.pyx.

15.1.2 Constructor & Destructor Documentation

15.1.2.1 __init__()

```
def amns.Amns.__init__ (
               self,
               version_string = None,
               version_number = None,
               version_backend = None,
               version_user = None )


Use to initialize the Amns class
        in:
                version_string: requested version (string, default None)
                version_number: requested version (integer, default None)
                version_backend: requested backend (string, default None)
                version_user: requated user (string, default None)
        out:
                instance of the Amns class
```

Definition at line 18 of file amns.pyx.

```
00018     def __init__(self, version_string=None, version_number=None, version_backend=None,
       version_user=None):
00019         """Use to initialize the Amns class
00020         in:
00021                 version_string: requested version (string, default None)
00022                 version_number: requested version (integer, default None)
00023                 version_backend: requested backend (string, default None)
00024                 version_user: requated user (string, default None)
00025         out:
00026                 instance of the Amns class
00027         """
00028         self._handle = NULL
00029         self._setup(version_string=version_string, version_number=version_number,
       version_backend=version_backend, version_user=version_user)
00030
```

Here is the call graph for this function:

15.1.3 Member Function Documentation

15.1.3.1 code_commit()

```
def amns.Amns.code_commit (
              self )

Query for code_commit associated with the AMNS data
```

Definition at line 146 of file amns.pyx.

```
00146     def code_commit(self):
00147         """Query for code_commit associated with the AMNS data"""
00148         cdef camns_interface.amns_c_answer_type answer
00149         answer = self.lquery(b"code_commit")
00150         return answer.string.decode('UTF-8')
00151
```

15.1.3.2 code_name()

```
def amns.Amns.code_name (
              self )

Query for code_name associated with the AMNS data
```

Definition at line 139 of file amns.pyx.

```
00139     def code_name(self):
00140         """Query for code_name associated with the AMNS data"""
00141         cdef camns_interface.amns_c_answer_type answer
00142         answer = self.lquery(b"code_name")
00143         return answer.string.decode('UTF-8')
00144
```

15.1.3.3 code_repository()

```
def amns.Amns.code_repository (
              self )

Query for code_repository associated with the AMNS data
```

Definition at line 160 of file amns.pyx.

```
00160     def code_repository(self):
00161         """Query for code_repository associated with the AMNS data"""
00162         cdef camns_interface.amns_c_answer_type answer
00163         answer = self.lquery(b"code_repository")
00164         return answer.string.decode('UTF-8')
00165
```

15.1.3.4 code_version()

```
def amns.Amns.code_version (
              self )

Query for code_version associated with the AMNS data
```

Definition at line 153 of file amns.pyx.

```
00153     def code_version(self):
00154         """Query for code_version associated with the AMNS data"""
00155         cdef camns_interface.amns_c_answer_type answer
00156         answer = self.lquery(b"code_version")
00157         return answer.string.decode('UTF-8')
00158
```

15.1.3.5 finalize()

```
def amns.Amns.finalize (
              self )

Provide the interface to IMAS_AMNS_FINISH
        in:
                None
        out:
                None
```

Definition at line 185 of file amns.pyx.

```
00185     def finalize(self):
00186         """Provide the interface to IMAS_AMNS_FINISH
00187         in:
00188                 None
00189         out:
00190                 None
00191         """
00192         cdef camns_interface.amns_c_error_type error_status
00193         camns_interface.IMAS_AMNS_CC_FINISH(&self._handle, &error_status)
00194         if error_status.flag:
00195             raise AmnsException(error_status.string.decode('UTF-8'))
00196
```

15.1.3.6 get_table()

```
def amns.Amns.get_table (
              self,
              reactionString,
              Reactants,
              reactants,
              isotope_resolved = None )

Provide the interface to the Table class used for tables
        in:
                reactionString: name of reaction (string, required)
                reactants: reactants object (of type amns.Reactants, required)
                isotope_resolved: 0 if not isotope resolved, 1 if isotope resolved (integer, default None)
        out:
                amns table (of type amns.Table)
```

Definition at line 197 of file amns.pyx.

```
00197     def get_table(self, reactionString, Reactants reactants, isotope_resolved=None):
00198         """Provide the interface to the Table class used for tables
00199         in:
00200                 reactionString: name of reaction (string, required)
00201                 reactants: reactants object (of type amns.Reactants, required)
00202                 isotope_resolved: 0 if not isotope resolved, 1 if isotope resolved (integer, default
      None)
00203         out:
00204                 amns table (of type amns.Table)
00205         """
00206         return Table(reactionString, reactants, self, isotope_resolved)
00207
00208
```

15.1.3.7 ImasAmnsCCFinish()

```
native long amns.Amns.ImasAmnsCCFinish (
            long ptr,
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_FINISH Finalization call for the AMNS package

Parameters

ptr	pointer returned by call to amns.Amns#ImasAmnsCCSetup
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

Returns

Returns new value of the pointer - may be altered by AMNS library (set to null)

15.1.3.8 ImasAmnsCCFinishReactants()

```
native long amns.Amns.ImasAmnsCCFinishReactants (
            long ptr )
```

This method calls native code: IMAS_AMNS_CC_FINISH_REACTANTS This method is just a helper function for C. For details take a look at AMNS doxygen based documentation

Parameters

ptr	pointer returned by call to amns.Amns#ImasAmnsCCSetReactantIdx

Returns

Returns new value of the pointer - may be altered by AMNS library (set to null)

15.1.3.9 ImasAmnsCCFinishTable()

```
native long amns.Amns.ImasAmnsCCFinishTable (
            long ptr,
            AmnsErrorType error )
```

This method calls natice code: IMAS_AMNS_CC_FINISH_TABLE It is used as finalization call for a particular reaction

Parameters

ptr	long type representation of C pointer returned via amns.Amns#ImasAmnsCCSetupTable
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

Returns

Returns new value of the pointer passed as argument; this value may be different as it may be altered by AMNS library

15.1.3.10 ImasAmnsCCGetReactant()

```
native void amns.Amns.ImasAmnsCCGetReactant (
```

```
            long ptr,
            AmnsReactantType species )
```

This method calls native code: IMAS_AMNS_CC_GET_REACTANT

Parameters

ptr	this is the pointer returned from call to amns.Amns#ImasAmnsCcSetupReactants or amns.Amns#ImasAmnsCcSetupReactantsNumber (wrappers for IMAS_AMNS_C_SETUP_REACTANTS)
species	this is the structure that will be filled by AMNS library; for details amns.type.AmnsReactantType

15.1.3.11 ImasAmnsCCQuery()

```
native void amns.Amns.ImasAmnsCCQuery (
            long ptrAmns,
            AmnsQueryType query,
            AmnsAnswerType answer,
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_QUERY Query routine for the AMNS package

Parameters

ptrAmns	C based pointer returned by call to amns.Amns#ImasAmnsCCSetup
query	specifies the query; for details of the type see amns.type.AmnsQueryType
answer	answer from AMNS package; for details of the type see amns.type.AmnsAnswerType
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

15.1.3.12 ImasAmnsCCQueryTable()

```
native void amns.Amns.ImasAmnsCCQueryTable (
            long ptrTable,
            AmnsQueryType query,
            AmnsAnswerType answer,
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_QUERY_TABLE Query routine for a particular reaction

Parameters

ptrTable	C pointer returned from the call to amns.Amns#ImasAmnsCCSetupTable
query	specifies the query; for details of the type see amns.type.AmnsQueryType
answer	answer from AMNS package; for details of the type see amns.type.AmnsAnswerType
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

15.1.3.13 ImasAmnsCCRX0B()

```
native double amns.Amns.ImasAmnsCCRX0B (
            long ptr,
            double arg1,
```

```
            double arg2,
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_RX_0_B Gets the rates associated with the input (0d) args for a particular reaction

Parameters

ptr	this is the pointer returned by call to amns.Amns#ImasAmnsCCSetupTable (please note that this value is a long type representation of C pointer allocated in AMNS library)
arg1	point on 1-st axis (scalar)
arg2	point on 2-nd axis (scalar)
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

Returns

value of the rate associated with input args for a particular reaction

15.1.3.14 ImasAmnsCCRX1A()

```
native double [] amns.Amns.ImasAmnsCCRX1A (
            long ptr,
            int nx,
            double[] arg1,
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_RX_1_A Gets the rates associated with the input (1d) args for a particular reaction

Parameters

ptr	this is the pointer returned by call to amns.Amns#ImasAmnsCCSetupTable (please note that this value is a long type representation of C pointer allocated in AMNS library)
arg1	points on 1-st axis (1d)
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

Returns

values of the rates associated with input args for a particular reaction

15.1.3.15 ImasAmnsCCRX1B()

```
native double [] amns.Amns.ImasAmnsCCRX1B (
            long ptr,
            int nx,
            double[] arg1,
            double[] arg2,
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_RX_1_B Gets the rates associated with the input (1d) args for a particular reaction

Parameters

ptr	this is the pointer returned by call to amns.Amns#ImasAmnsCCSetupTable (please note that this value is a long type representation of C pointer allocated in AMNS library)
arg1	points on 1-st axis (1d)
arg2	points on 2-nd axis (1d)
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

Returns

values of the rates associated with input args for a particular reaction

15.1.3.16 ImasAmnsCCRX1C()

```
native double [] amns.Amns.ImasAmnsCCRX1C (
              long ptr,
              int nx,
              double[] arg1,
              double[] arg2,
              double[] arg3,
              AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_RX_1_B Gets the rates associated with the input (1d) args for a particular reaction

Parameters

ptr	this is the pointer returned by call to amns.Amns#ImasAmnsCCSetupTable (please note that this value is a long type representation of C pointer allocated in AMNS library)
arg1	points on 1-st axis (1d)
arg2	points on 2-nd axis (1d)
arg3	points on 3-rd axis (1d)
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

Returns

values of the rates associated with input args for a particular reaction

15.1.3.17 ImasAmnsCCSet()

```
native void amns.Amns.ImasAmnsCCSet (
              long ptrAmns,
              AmnsSetType setType,
              AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_SET Set a parameter for the AMNS package

Parameters

ptrAmns	C based pointer returned by call to amns.Amns#ImasAmnsCCSetup
setType	parameter we want to set in AMNS package; for details check amns.type.AmnsSetType
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

15.1.3.18 ImasAmnsCCSetReactant()

```
native void amns.Amns.ImasAmnsCCSetReactant (
            long ptr,
            AmnsReactantType settings )
```

This method calls native code: IMAS_AMNS_CC_SET_REACTANT

Parameters

ptr	C pointer returned via amns.Amns#ImasAmnsCcSetupReactantsNumber or amns.Amns#ImasAmnsCcSetupReactants
settings	contains structure of amns.type.AmnsReactantType .note Please note that this code makes assmption regarding reactant number; it always set reactant No. 1

15.1.3.19 ImasAmnsCCSetReactantIdx()

```
native void amns.Amns.ImasAmnsCCSetReactantIdx (
            long ptr,
            int reactant_idx,
            AmnsReactantType settings )
```

This method calls native code: IMAS_AMNS_CC_SET_REACTANT

Parameters

ptr	C pointer returned via amns.Amns#ImasAmnsCcSetupReactantsNumber or amns.Amns#ImasAmnsCcSetupReactants
reactant_idx	number of reactant to be set
settings	contains structure of amns.type.AmnsReactantType

15.1.3.20 ImasAmnsCCSetTable()

```
native void amns.Amns.ImasAmnsCCSetTable (
            long ptr,
            AmnsSetType setType,
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_SET_TABLE Set a parameter for a particular reaction

Parameters

ptr	C based pointer that is returned as a result of call to amns.Amns#ImasAmnsCCSetupTable
setType	parameter we want to set in AMNS package; for details check amns.type.AmnsSetType
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

15.1.3.21 ImasAmnsCCSetup()

```
native long amns.Amns.ImasAmnsCCSetup (
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_SETUP

Parameters

error	structure that returns error code from AMNS call; for details take a look here: amns.type.AmnsErrorType

Returns

returns long value that is a C based pointer; this value must be passed to functions whenever C pointer is required (return value is the equivalent of void **handle_out)

15.1.3.22 ImasAmnsCcSetupReactants()

```
native long amns.Amns.ImasAmnsCcSetupReactants (
            int idx )
```

Interface to native code implemented inside library libamnsjni.so - amns_jni_call.c
This method calls native code: IMAS_AMNS_CC_SETUP_REACTANTS

Parameters

idx	index to be passed to IMAS_AMNS_CC_SETUP_REACTANT

Returns

pointer represented as long value; whenever this pointer is required in JNI code, it must be passed as is .note Please note that this code makes assumptions regarding n_reactants-it always passes "1" as n_reactants .note Please note that this code makes assumptions regarding string_in-it always passes "" as string_in

15.1.3.23 ImasAmnsCcSetupReactantsNumber()

```
native long amns.Amns.ImasAmnsCcSetupReactantsNumber (
            int idx,
            int n_reactants )
```

Interface to native code implemented inside library libamnsjni.so - amns_jni_call.c
This method calls native code: IMAS_AMNS_CC_SETUP_REACTANTS

Parameters

idx	index to be passed to IMAS_AMNS_CC_SETUP_REACTANT
n_reactants	number of reactants passed to IMAS_AMNS_CC_SETUP_REACTANT

Returns

pointer represented as long value; whenever this pointer is required in JNI code, it must be passed as is .note Please note that this code makes assumptions regarding string_in - it always passes ("" - empty string) as string_in

15.1.3.24 ImasAmnsCCSetupTable()

```
native long amns.Amns.ImasAmnsCCSetupTable (
            long ptrAmns,
            AmnsReactionType reaction,
            long ptrReactants,
            AmnsErrorType error )
```

This method calls native code: IMAS_AMNS_CC_SETUP_TABLE Initialization call for a particular reaction

Parameters

ptrAmns	pointer returned by call to amns.Amns#ImasAmnsCCSetup
reaction	reaction type; for details check amns.type.AmnsReactionType
ptrReactants	pointer to reactants returned via call to amns.Amns#ImasAmnsCCSetReactantIdx
error	this structure is filled with error status and error string by AMNS library You can find details regarding this type here: amns.type.AmnsErrorType

Returns

Opaque handle (C pointer) to be passed to table routine calls

15.1.3.25 printErrorCode()

```
static void amns.Amns.printErrorCode (
            AmnsErrorType error,
            String message )  [inline], [static]
```

Prints error message generate by AMNS library. Error message can be extended with user comment passed via message parameter

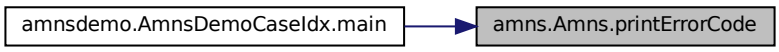
Parameters

error	AMNS error structure that can be passed from AMNS library to Java
message	User based comment that will be printed on stdout; serves as additional description of error

Definition at line 213 of file Amns.java.

```
00213                                                                    {
00214
00215         if(message != null && message.length() != 0 ) {
00216             System.out.println("Error result for: " + message);
00217         }
00218         System.out.println("[AMNS error code] message: " + error.string + " flag: " + error.flag);
00219
00220     }
```

Here is the caller graph for this function:



15.1.3.26 prop_comment()

```
def amns.Amns.prop_comment (
              self )

Query for prop_comment associated with the AMNS data
```

Definition at line 111 of file amns.pyx.

```
00111     def prop_comment(self):
00112         """Query for prop_comment associated with the AMNS data"""
00113         cdef camns_interface.amns_c_answer_type answer
00114         answer = self.lquery(b"prop_comment")
00115         return answer.string.decode('UTF-8')
00116
```

15.1.3.27 prop_creation()

```
def amns.Amns.prop_creation (
              self )

Query for prop_creation associated with the AMNS data
```

Definition at line 132 of file amns.pyx.

```
00132     def prop_creation(self):
00133         """Query for prop_creation associated with the AMNS data"""
00134         cdef camns_interface.amns_c_answer_type answer
00135         answer = self.lquery(b"prop_creation")
00136         return answer.string.decode('UTF-8')
00137
```

15.1.3.28 prop_provider()

```
def amns.Amns.prop_provider (
              self )

Query for prop_provider associated with the AMNS data
```

Definition at line 125 of file amns.pyx.

```
00125     def prop_provider(self):
00126         """Query for prop_provider associated with the AMNS data"""
00127         cdef camns_interface.amns_c_answer_type answer
00128         answer = self.lquery(b"prop_provider")
00129         return answer.string.decode('UTF-8')
00130
```

15.1.3.29 prop_source()

```
def amns.Amns.prop_source (
              self )

Query for prop_source associated with the AMNS data
```

Definition at line 118 of file amns.pyx.

```
00118     def prop_source(self):
00119         """Query for prop_source associated with the AMNS data"""
00120         cdef camns_interface.amns_c_answer_type answer
00121         answer = self.lquery(b"prop_source")
00122         return answer.string.decode('UTF-8')
00123
```

15.1.3.30 query()

```
def amns.Amns.query (
              self,
              queryString )

Provide the interface to IMAS_AMNS_QUERY
        in:
                queryString: string containing query (string, required)
        out:
                answer_string: string containing reply
                answer_number: number containing any numerical answer
```

Definition at line 81 of file amns.pyx.

```
00081     def query(self, queryString):
00082         """Provide the interface to IMAS_AMNS_QUERY
00083         in:
00084                 queryString: string containing query (string, required)
00085         out:
00086                 answer_string: string containing reply
00087                 answer_number: number containing any numerical answer
00088
00089         """
00090         cdef camns_interface.amns_c_query_type query
00091         cdef camns_interface.amns_c_answer_type answer
```

```
00092         cdef camns_interface.amns_c_error_type error_status
00093         if type(queryString) == bytes:
00094             query.string = queryString
00095         else:
00096             tmp_c_str = queryString.encode('UTF-8')
00097             query.string = tmp_c_str
00098         camns_interface.IMAS_AMNS_CC_QUERY(self._handle, &query, &answer, &error_status);
00099         if error_status.flag:
00100             raise AmnsException(error_status.string.decode('UTF-8'))
00101         return answer.string.decode('UTF-8'), answer.number
00102
```

15.1.3.31 reshape1DTo2D()

```
static double [][] amns.Amns.reshape1DTo2D (
              double[] input,
              int nx,
              int ny )  [inline], [static]
```

Reshapes 1D array into 2D array with given height and length there is no sanity check, it is assumed that nx ∗ ny = input.length

Parameters

input	- array that should be reshaped
nx	- first index of element in the array
ny	- second index of element in the array

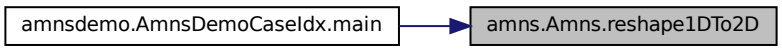
Returns

2D array of doubles where dimmentions match size of input array (nx ∗ ny = input.length)

Definition at line 250 of file Amns.java.

```
00250                                                                                {
00251
00252         double [][] result = new double[nx][ny];
00253
00254         for( int i=0; i<nx; i++) {
00255             for( int p=0; p<ny; p++) {
00256                 result[i][p] = input[ (i * ny) + p];
00257             }
00258         }
00259         return result;
00260     }
```

Here is the caller graph for this function:



15.1.3.32 reshape2DTo1D()

```
static double [] amns.Amns.reshape2DTo1D (
              double input[][],
              int nx,
              int ny )  [inline], [static]
```

Reshapes 2D array with given width and height as 1D array of size nx ∗ ny there is no sanity check, it is assumed that nx and ny are correct sizes of array dimensions

Parameters

input	- array that should be reshaped
nx	- first index of element in the array
ny	- second index of element in the array

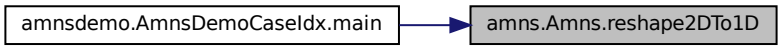
Returns

1D array of double with size equal to nx ∗ ny

Definition at line 230 of file Amns.java.

```
00230                                                                                {
00231         double [] result = new double[nx * ny];
00232
00233         for( int i=0; i<nx; i++) {
00234             for( int p=0; p<ny; p++) {
00235                 result[ (i * ny) + p] = input[i][p];
00236             }
00237         }
00238
00239         return result;
00240     }
```

Here is the caller graph for this function:



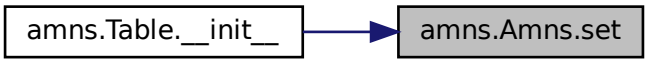
15.1.3.33 set()

```
def amns.Amns.set (
              self,
              setString )

Provide the interface to IMAS_AMNS_SET
        in:
                setString: string containing setting (string, required)
        out:
                None
```

Definition at line 166 of file amns.pyx.

```
00166     def set(self, setString):
00167         """Provide the interface to IMAS_AMNS_SET
00168         in:
00169                 setString: string containing setting (string, required)
00170         out:
00171                 None
00172
00173         """
00174         cdef camns_interface.amns_c_set_type set
00175         cdef camns_interface.amns_c_error_type error_status
00176         if type(setString) == bytes:
00177             set.string = setString
00178         else:
00179             tmp_c_str = setString.encode('UTF-8')
00180             set.string = tmp_c_str
00181         camns_interface.IMAS_AMNS_CC_SET(self._handle, &set, &error_status);
00182         if error_status.flag:
00183             raise AmnsException(error_status.string.decode('UTF-8'))
00184
```

Here is the caller graph for this function:



15.1.3.34 version()

```
def amns.Amns.version (
              self )

Query for version of the AMNS data
```

Definition at line 104 of file amns.pyx.

```
00104     def version(self):
00105         """Query for version of the AMNS data"""
00106         cdef camns_interface.amns_c_answer_type answer
00107         answer = self.lquery(b"version")
00108         return answer.number, answer.string.decode('UTF-8')
00109
```

15.1.4 Member Data Documentation

15.1.4.1 string

```
amns.Amns.string  [static]
```

Definition at line 72 of file amns.pyx.

15.1.4.2 tmp_c_str

```
amns.Amns.tmp_c_str = queryString.encode('UTF-8')  [static]
```

Definition at line 74 of file amns.pyx.

The documentation for this class was generated from the following files:

- src/py/amns/amns.pyx
- src/java/src/amns/Amns.java

15.2 amns_types::amns_answer_type Type Reference

Type for answers from queries in the AMNS package (not interoperable)

Public Attributes

- character(len=answer_length) string

  string version of the answer to a query

- integer(c_int) number

  integer version of the answer to a query

15.2.1 Detailed Description

Type for answers from queries in the AMNS package (not interoperable)
Definition at line 201 of file amns_types.f90.

15.2.2 Member Data Documentation

15.2.2.1 number

```
integer(c_int) amns_types::amns_answer_type::number
```

integer version of the answer to a query
Definition at line 205 of file amns_types.f90.

```
00205     integer(c_int) :: number
```

15.2.2.2 string

```
character(len=answer_length) amns_types::amns_answer_type::string
```

string version of the answer to a query
Definition at line 203 of file amns_types.f90.

```
00203     character(len=answer_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.3 amns_answer_type Struct Reference

Type for answers from queries in the AMNS package ("interoperable" version)
`#include <amns_interface.h>`

Public Attributes

- char string [answer_length]

  string version of the answer to a query

- int number

  integer version of the answer to a query

15.3.1 Detailed Description

Type for answers from queries in the AMNS package ("interoperable" version)
Definition at line 254 of file amns_interface.h.

15.3.2 Member Data Documentation

15.3.2.1 number

```
int amns_answer_type::number
```

integer version of the answer to a query
Definition at line 256 of file amns_interface.h.

15.3.2.2 string

`char amns_answer_type::string[answer_length]`
string version of the answer to a query
Definition at line 255 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.4 amns_c_answer_type Struct Reference

Type for answers from queries in the AMNS package ("C" version)
`#include <amns_interface.h>`

Public Attributes

- char * string

  string version of the answer to a query
- int number

  integer version of the answer to a query

15.4.1 Detailed Description

Type for answers from queries in the AMNS package ("C" version)
Definition at line 263 of file amns_interface.h.

15.4.2 Member Data Documentation

15.4.2.1 number

`int amns_c_answer_type::number`
integer version of the answer to a query
Definition at line 265 of file amns_interface.h.

15.4.2.2 string

`char* amns_c_answer_type::string`
string version of the answer to a query
Definition at line 264 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.5 amns_c_error_type Struct Reference

Type for error returns from the AMNS interface ("C" version)
`#include <amns_interface.h>`

Public Attributes

- bool flag

  True if an error occurred.
- char * string

  text describing the error if flag was True

15.5.1 Detailed Description

Type for error returns from the AMNS interface ("C" version)
Definition at line 135 of file amns_interface.h.

15.5.2 Member Data Documentation

15.5.2.1 flag

`bool amns_c_error_type::flag`
True if an error occurred.
Definition at line 136 of file amns_interface.h.

15.5.2.2 string

`char* amns_c_error_type::string`
text describing the error if flag was True
Definition at line 137 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.6 amns_c_query_type Struct Reference

Type for querying parameters in the AMNS package ("C" version)
`#include <amns_interface.h>`

Public Attributes

- char * string

  used to pass a query about settings

15.6.1 Detailed Description

Type for querying parameters in the AMNS package ("C" version)
Definition at line 235 of file amns_interface.h.

15.6.2 Member Data Documentation

15.6.2.1 string

`char* amns_c_query_type::string`
used to pass a query about settings
Definition at line 236 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.7 amns_c_reaction_type Struct Reference

Type used for specifying reactions when using the AMNS interface ("C" version)
`#include <amns_interface.h>`

Public Attributes

- char * string

  name of the reaction (e.g. RC)
- int isotope_resolved

  if the reaction is different for different isotopes, isotope_resolved shoukld be set to 1

15.7.1 Detailed Description

Type used for specifying reactions when using the AMNS interface ("C" version)
Definition at line 172 of file amns_interface.h.

15.7.2 Member Data Documentation

15.7.2.1 isotope_resolved

```
int amns_c_reaction_type::isotope_resolved
```

if the reaction is different for different isotopes, isotope_resolved shoukld be set to 1
Definition at line 174 of file amns_interface.h.

15.7.2.2 string

```
char* amns_c_reaction_type::string
```

name of the reaction (e.g. RC)
Definition at line 173 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.8 amns_c_set_type Struct Reference

Type for setting parameters in the AMNS package ("C" version)
`#include <amns_interface.h>`

Public Attributes

- char * string

  used to pass a change in the settings

15.8.1 Detailed Description

Type for setting parameters in the AMNS package ("C" version)
Definition at line 208 of file amns_interface.h.

15.8.2 Member Data Documentation

15.8.2.1 string

```
char* amns_c_set_type::string
```

used to pass a change in the settings
Definition at line 209 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.9 amns_c_version_type Struct Reference

Type for specifying the AMNS version ("C" version)
`#include <amns_interface.h>`

Public Attributes

- char * string
  ascii specification of the version
- int number
  integer specification of the version number (primary specification at the moment)
- char * backend
  specify the backend to be used to access the CPOs
- char * user
  specify the username of the data to be obtained (defaults to tghe person running the code)

15.9.1 Detailed Description

Type for specifying the AMNS version ("C" version)
Definition at line 65 of file amns_interface.h.

15.9.2 Member Data Documentation

15.9.2.1 backend

`char* amns_c_version_type::backend`
specify the backend to be used to access the CPOs
Definition at line 68 of file amns_interface.h.

15.9.2.2 number

`int amns_c_version_type::number`
integer specification of the version number (primary specification at the moment)
Definition at line 67 of file amns_interface.h.

15.9.2.3 string

`char* amns_c_version_type::string`
ascii specification of the version
Definition at line 66 of file amns_interface.h.

15.9.2.4 user

`char* amns_c_version_type::user`
specify the username of the data to be obtained (defaults to tghe person running the code)
Definition at line 69 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.10 amns_error_type Struct Reference

Type for error returns from the AMNS interface ("interoperable" version)
`#include <amns_interface.h>`

Public Attributes

- bool flag

  True if an error occurred.
- char string [answer_length]

  text describing the error if flag was True

15.10.1 Detailed Description

Type for error returns from the AMNS interface ("interoperable" version)
Definition at line 126 of file amns_interface.h.

15.10.2 Member Data Documentation

15.10.2.1 flag

```
bool amns_error_type::flag
```

True if an error occurred.
Definition at line 127 of file amns_interface.h.

15.10.2.2 string

```
char amns_error_type::string[answer_length]
```

text describing the error if flag was True
Definition at line 128 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.11 amns_types::amns_error_type Type Reference

Type for error returns from the AMNS interface (not interoperable)

Public Attributes

- logical flag

  True if an error occurred.
- character(len=answer_length) string

  text describing the error if flag was True

15.11.1 Detailed Description

Type for error returns from the AMNS interface (not interoperable)
Definition at line 137 of file amns_types.f90.

15.11.2 Member Data Documentation

15.11.2.1 flag

```
logical amns_types::amns_error_type::flag
```

True if an error occurred.
Definition at line 139 of file amns_types.f90.

```
00139      logical :: flag
```

15.11.2.2 string

```
character(len=answer_length) amns_types::amns_error_type::string
```

text describing the error if flag was True
Definition at line 141 of file amns_types.f90.

```
00141      character(len=answer_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.12 amns_types::amns_fc_answer_type Type Reference

Type for answers from queries in the AMNS package (interoperable with c)

Public Attributes

- character(len=1, kind=c_char), dimension(answer_length) string

  string version of the answer to a query
- integer(c_int) number

  integer version of the answer to a query

15.12.1 Detailed Description

Type for answers from queries in the AMNS package (interoperable with c)
Definition at line 210 of file amns_types.f90.

15.12.2 Member Data Documentation

15.12.2.1 number

```
integer(c_int) amns_types::amns_fc_answer_type::number
```

integer version of the answer to a query
Definition at line 214 of file amns_types.f90.

```
00214      integer(c_int) :: number
```

15.12.2.2 string

```
character(len=1, kind=c_char), dimension(answer_length) amns_types::amns_fc_answer_type↩
::string
```

string version of the answer to a query
Definition at line 212 of file amns_types.f90.

```
00212      character(len=1, kind=c_char), dimension(answer_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.13 amns_types::amns_fc_error_type Type Reference

Type for error returns from the AMNS interface (interoperable with c)

Public Attributes

- logical(c_bool) flag

  True if an error occurred.
- character(len=1, kind=c_char), dimension(answer_length) string

  text describing the error if flag was True

15.13.1 Detailed Description

Type for error returns from the AMNS interface (interoperable with c)
Definition at line 146 of file amns_types.f90.

15.13.2 Member Data Documentation

15.13.2.1 flag

```
logical(c_bool) amns_types::amns_fc_error_type::flag
```

True if an error occurred.
Definition at line 148 of file amns_types.f90.

```
00148      logical(c_bool) :: flag
```

15.13.2.2 string

```
character(len=1, kind=c_char), dimension(answer_length) amns_types::amns_fc_error_type::string
```

text describing the error if flag was True
Definition at line 150 of file amns_types.f90.

```
00150      character(len=1, kind=c_char), dimension(answer_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.14 amns_types::amns_fc_query_type Type Reference

Type for querying parameters in the AMNS package (interoperable with c)

Public Attributes

- character(len=1, kind=c_char), dimension(query_length) string

  used to pass a query about settings

15.14.1 Detailed Description

Type for querying parameters in the AMNS package (interoperable with c)
Definition at line 194 of file amns_types.f90.

15.14.2 Member Data Documentation

15.14.2.1 string

```
character(len=1, kind=c_char), dimension(query_length) amns_types::amns_fc_query_type::string
```

used to pass a query about settings
Definition at line 196 of file amns_types.f90.

```
00196      character(len=1, kind=c_char), dimension(query_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.15 amns_types::amns_fc_reaction_type Type Reference

Type used for specifying reactions when using the AMNS interface (interoperable with c)

Public Attributes

- character(len=1, kind=c_char), dimension(reaction_length) string

  name of the reaction (e.g. RC)
- integer(c_int) isotope_resolved =0

  if the reaction is different for different isotopes, isotope_resolved should be set to 1

15.15.1 Detailed Description

Type used for specifying reactions when using the AMNS interface (interoperable with c)
Definition at line 164 of file amns_types.f90.

15.15.2 Member Data Documentation

15.15.2.1 isotope_resolved

```
integer(c_int) amns_types::amns_fc_reaction_type::isotope_resolved =0
```

if the reaction is different for different isotopes, isotope_resolved should be set to 1
Definition at line 168 of file amns_types.f90.

```
00168      integer(c_int) :: isotope_resolved=0
```

15.15.2.2 string

```
character(len=1, kind=c_char), dimension(reaction_length) amns_types::amns_fc_reaction_type↩
::string
```

name of the reaction (e.g. RC)
Definition at line 166 of file amns_types.f90.

```
00166      character(len=1, kind=c_char), dimension(reaction_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.16 amns_types::amns_fc_set_type Type Reference

Type for setting parameters in the AMNS package (interoperable with c)

Public Attributes

- character(len=1, kind=c_char), dimension(set_length) string

  used to pass a change in the settings

15.16.1 Detailed Description

Type for setting parameters in the AMNS package (interoperable with c)
Definition at line 180 of file amns_types.f90.

15.16.2 Member Data Documentation

15.16.2.1 string

```
character(len=1, kind=c_char), dimension(set_length) amns_types::amns_fc_set_type::string
```

used to pass a change in the settings
Definition at line 182 of file amns_types.f90.

```
00182      character(len=1, kind=c_char), dimension(set_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.17 amns_types::amns_fc_version_type Type Reference

Type for specifying the AMNS version (interoperable with c)

Public Attributes

- character(len=1, kind=c_char), dimension(version_length) string = ''

  ascii specification of the version
- integer(c_int) number = 0

  integer specification of the version number (primary specification at the moment)
- character(len=1, kind=c_char), dimension(version_length) backend = 'mdsplus'

  specify the backend to be used to access the IDSs
- character(len=1, kind=c_char), dimension(version_length) user = ''

  specify the username of the data to be obtained (defaults to the person running the code)

15.17.1 Detailed Description

Type for specifying the AMNS version (interoperable with c)
Definition at line 57 of file amns_types.f90.

15.17.2 Member Data Documentation

15.17.2.1 backend

```
character(len=1, kind=c_char), dimension(version_length) amns_types::amns_fc_version_type↩
::backend = 'mdsplus'
```

specify the backend to be used to access the IDSs
Definition at line 63 of file amns_types.f90.

```
00063      character(len=1, kind=c_char), dimension(version_length) :: backend = 'mdsplus'
```

15.17.2.2 number

```
integer(c_int) amns_types::amns_fc_version_type::number = 0
```

integer specification of the version number (primary specification at the moment)
Definition at line 61 of file amns_types.f90.

```
00061      integer(c_int) :: number = 0
```

15.17.2.3 string

```
character(len=1, kind=c_char), dimension(version_length) amns_types::amns_fc_version_type↩
::string = ''
```

ascii specification of the version
Definition at line 59 of file amns_types.f90.

```
00059      character(len=1, kind=c_char), dimension(version_length) :: string = ''
```

15.17.2.4 user

```
character(len=1, kind=c_char), dimension(version_length) amns_types::amns_fc_version_type↩
::user = ''
```

specify the username of the data to be obtained (defaults to the person running the code)
Definition at line 65 of file amns_types.f90.

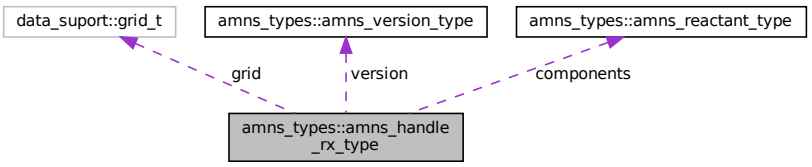
```
00065      character(len=1, kind=c_char), dimension(version_length) :: user = ”
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.18 amns_types::amns_handle_rx_type Type Reference

Type for the AMNS RX handle (opaque for user codes) NOT interoperable with C.
Collaboration diagram for amns_types::amns_handle_rx_type:



Public Attributes

- character(len=answer_length) properties_comment = ''
- character(len=answer_length) properties_source = ''
- character(len=answer_length) properties_provider = ''
- character(len=answer_length) properties_creation_date = ''
- character(len=answer_length) code_name = ''
- character(len=answer_length) code_commit = ''
- character(len=answer_length) code_version = ''
- character(len=answer_length) code_repository = ''
- character(len=answer_length) source = ''
- character(len=answer_length) provider = ''
- character(len=answer_length) citation = ''
- character(len=reaction_length) reaction_type = ''
- integer no_of_reactants = 0
- type(amns_version_type) version
- type(amns_reactant_type), dimension(:), allocatable components
- character(len=reaction_length), dimension(:), allocatable string
- integer index = 0
- logical debug
- logical initialized
- logical filled = .false.
- type(grid_t) grid

15.18.1 Detailed Description

Type for the AMNS RX handle (opaque for user codes) NOT interoperable with C.
Definition at line 116 of file amns_types.f90.

15.18.2 Member Data Documentation

15.18.2.1 citation

```
character(len=answer_length) amns_types::amns_handle_rx_type::citation = ''
```
Definition at line 123 of file amns_types.f90.

15.18.2.2 code_commit

```
character(len=answer_length) amns_types::amns_handle_rx_type::code_commit = ''
```
Definition at line 118 of file amns_types.f90.

15.18.2.3 code_name

```
character(len=answer_length) amns_types::amns_handle_rx_type::code_name = ''
```
Definition at line 118 of file amns_types.f90.

15.18.2.4 code_repository

```
character(len=answer_length) amns_types::amns_handle_rx_type::code_repository = ''
```
Definition at line 118 of file amns_types.f90.

15.18.2.5 code_version

```
character(len=answer_length) amns_types::amns_handle_rx_type::code_version = ''
```
Definition at line 118 of file amns_types.f90.

15.18.2.6 components

```
type (amns_reactant_type), dimension(:), allocatable amns_types::amns_handle_rx_type::components
```
Definition at line 128 of file amns_types.f90.
```
00128      type (amns_reactant_type), allocatable :: components(:)
```

15.18.2.7 debug

```
logical amns_types::amns_handle_rx_type::debug
```
Definition at line 131 of file amns_types.f90.
```
00131      logical :: debug, initialized, filled = .false.
```

15.18.2.8 filled

```
logical amns_types::amns_handle_rx_type::filled = .false.
```
Definition at line 131 of file amns_types.f90.

15.18.2.9 grid

```
type (grid_t) amns_types::amns_handle_rx_type::grid
```
Definition at line 132 of file amns_types.f90.
```
00132      type (grid_t) :: grid
```

15.18.2.10 index

```
integer amns_types::amns_handle_rx_type::index = 0
```

Definition at line 130 of file amns_types.f90.

```
00130      integer :: index = 0
```

15.18.2.11 initialized

```
logical amns_types::amns_handle_rx_type::initialized
```

Definition at line 131 of file amns_types.f90.

15.18.2.12 no_of_reactants

```
integer amns_types::amns_handle_rx_type::no_of_reactants = 0
```

Definition at line 126 of file amns_types.f90.

```
00126      integer :: no_of_reactants = 0
```

15.18.2.13 properties_comment

```
character(len=answer_length) amns_types::amns_handle_rx_type::properties_comment = ''
```

Definition at line 118 of file amns_types.f90.

```
00118      character(len=answer_length) :: &
00119           properties_comment = '', properties_source = '', &
00120           properties_provider = '', properties_creation_date = '',  &
00121           code_name = '', code_commit = '', &
00122           code_version = '', code_repository = ''
```

15.18.2.14 properties_creation_date

```
character(len=answer_length) amns_types::amns_handle_rx_type::properties_creation_date = ''
```

Definition at line 118 of file amns_types.f90.

15.18.2.15 properties_provider

```
character(len=answer_length) amns_types::amns_handle_rx_type::properties_provider = ''
```

Definition at line 118 of file amns_types.f90.

15.18.2.16 properties_source

```
character(len=answer_length) amns_types::amns_handle_rx_type::properties_source = ''
```

Definition at line 118 of file amns_types.f90.

15.18.2.17 provider

```
character(len=answer_length) amns_types::amns_handle_rx_type::provider = ''
```

Definition at line 123 of file amns_types.f90.

15.18.2.18 reaction_type

```
character(len=reaction_length) amns_types::amns_handle_rx_type::reaction_type = ''
```

Definition at line 125 of file amns_types.f90.

```
00125      character(len=reaction_length) :: reaction_type = ''
```

15.18.2.19 source

```
character(len=answer_length) amns_types::amns_handle_rx_type::source = ''
```

Definition at line 123 of file amns_types.f90.

```
00123      character(len=answer_length) :: &
00124           source = ", provider = ", citation = "
```

15.18.2.20 string

```
character(len=reaction_length), dimension(:), allocatable amns_types::amns_handle_rx_type↩
::string
```

Definition at line 129 of file amns_types.f90.

```
00129      character(len=reaction_length), allocatable :: string(:)
```

15.18.2.21 version

```
type (amns_version_type) amns_types::amns_handle_rx_type::version
```

Definition at line 127 of file amns_types.f90.

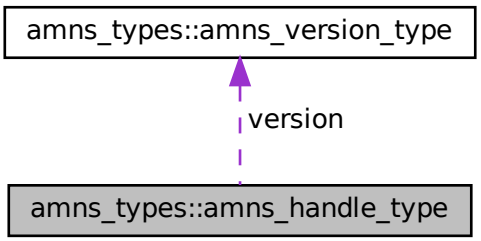
```
00127      type (amns_version_type) :: version
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.19 amns_types::amns_handle_type Type Reference

type for the AMNS handle (opaque for user codes) NOT interoperable with C.
Collaboration diagram for amns_types::amns_handle_type:



Public Attributes

- type(amns_version_type) version
- integer no_of_errors = 0
- logical debug = .false.
- logical initialized = .false.
- character(len=answer_length) properties_comment = ''
- character(len=answer_length) properties_source = ''
- character(len=answer_length) properties_provider = ''
- character(len=answer_length) properties_creation_date = ''
- character(len=answer_length) code_name = ''
- character(len=answer_length) code_commit = ''
- character(len=answer_length) code_version = ''
- character(len=answer_length) code_repository = ''

15.19.1 Detailed Description

type for the AMNS handle (opaque for user codes) NOT interoperable with C.
Definition at line 100 of file amns_types.f90.

15.19.2 Member Data Documentation

15.19.2.1 code_commit

```
character(len=answer_length) amns_types::amns_handle_type::code_commit = ''
```
Definition at line 106 of file amns_types.f90.

15.19.2.2 code_name

```
character(len=answer_length) amns_types::amns_handle_type::code_name = ''
```
Definition at line 106 of file amns_types.f90.

15.19.2.3 code_repository

```
character(len=answer_length) amns_types::amns_handle_type::code_repository = ''
```
Definition at line 106 of file amns_types.f90.

15.19.2.4 code_version

```
character(len=answer_length) amns_types::amns_handle_type::code_version = ''
```
Definition at line 106 of file amns_types.f90.

15.19.2.5 debug

```
logical amns_types::amns_handle_type::debug = .false.
```
Definition at line 104 of file amns_types.f90.
```
00104      logical :: debug = .false.
```

15.19.2.6 initialized

```
logical amns_types::amns_handle_type::initialized = .false.
```
Definition at line 105 of file amns_types.f90.
```
00105      logical :: initialized = .false.
```

15.19.2.7 no_of_errors

```
integer amns_types::amns_handle_type::no_of_errors = 0
```
Definition at line 103 of file amns_types.f90.
```
00103      integer :: no_of_errors = 0
```

15.19.2.8 properties_comment

```
character(len=answer_length) amns_types::amns_handle_type::properties_comment = ''
```
Definition at line 106 of file amns_types.f90.
```
00106      character(len=answer_length) :: &
00107           properties_comment = '', properties_source = '', &
00108           properties_provider = '', properties_creation_date = '',  &
00109           code_name = '', code_commit = '', &
```

```
00110           code_version = '', code_repository = ''
```

15.19.2.9 properties_creation_date

```
character(len=answer_length) amns_types::amns_handle_type::properties_creation_date = ''
```
Definition at line 106 of file amns_types.f90.

15.19.2.10 properties_provider

```
character(len=answer_length) amns_types::amns_handle_type::properties_provider = ''
```
Definition at line 106 of file amns_types.f90.

15.19.2.11 properties_source

```
character(len=answer_length) amns_types::amns_handle_type::properties_source = ''
```
Definition at line 106 of file amns_types.f90.

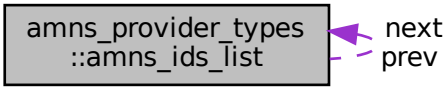
15.19.2.12 version

```
type (amns_version_type) amns_types::amns_handle_type::version
```
Definition at line 102 of file amns_types.f90.
```
00102      type (amns_version_type) :: version
```
The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.20 amns_provider_types::amns_ids_list Type Reference

type for linked list of amns cpos
Collaboration diagram for amns_provider_types::amns_ids_list:



Public Attributes

- type(ids_amns_data) amns_ids
- type(amns_ids_list), pointer prev
- type(amns_ids_list), pointer next => null()

15.20.1 Detailed Description

type for linked list of amns cpos
Definition at line 13 of file amns_provider_types.f90.

15.20.2 Member Data Documentation

15.20.2.1 amns_ids

```
type (ids_amns_data) amns_provider_types::amns_ids_list::amns_ids
```

Definition at line 14 of file amns_provider_types.f90.

```
00014      type (ids_amns_data) ::     amns_ids
```

15.20.2.2 next

```
type (amns_ids_list), pointer amns_provider_types::amns_ids_list::next => null()
```

Definition at line 15 of file amns_provider_types.f90.

15.20.2.3 prev

```
type (amns_ids_list), pointer amns_provider_types::amns_ids_list::prev
```

Definition at line 15 of file amns_provider_types.f90.

```
00015      type (amns_ids_list), pointer :: prev, next => null()
```

The documentation for this type was generated from the following file:

- src/amns_driver/amns_provider_types.f90

15.21 amns_types::amns_ids_list Type Reference

Type for linked list of amns idss NOT interoperable with C.
Collaboration diagram for amns_types::amns_ids_list:

Public Attributes

- integer shot =0
- integer run = 0
- type(ids_amns_data) amns_ids
- type(amns_ids_list), pointer prev
- type(amns_ids_list), pointer next => null()

15.21.1 Detailed Description

Type for linked list of amns idss NOT interoperable with C.
Definition at line 219 of file amns_types.f90.

15.21.2 Member Data Documentation

15.21.2.1 amns_ids

```
type (ids_amns_data) amns_types::amns_ids_list::amns_ids
```

Definition at line 221 of file amns_types.f90.

```
00221      type (ids_amns_data) ::     amns_ids
```

15.21.2.2 next

```
type (amns_ids_list), pointer amns_types::amns_ids_list::next => null()
```

Definition at line 222 of file amns_types.f90.

15.21.2.3 prev

```
type (amns_ids_list), pointer amns_types::amns_ids_list::prev
```

Definition at line 222 of file amns_types.f90.

```
00222      type (amns_ids_list), pointer :: prev, next => null()
```

15.21.2.4 run

```
integer amns_types::amns_ids_list::run = 0
```

Definition at line 220 of file amns_types.f90.

15.21.2.5 shot

```
integer amns_types::amns_ids_list::shot =0
```

Definition at line 220 of file amns_types.f90.

```
00220      integer :: shot=0, run = 0
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.22 amns_types::amns_query_type Type Reference

Type for querying parameters in the AMNS package (not interoperable)

Public Attributes

- character(len=query_length) string

  used to pass a query about settings

15.22.1 Detailed Description

Type for querying parameters in the AMNS package (not interoperable)
Definition at line 187 of file amns_types.f90.

15.22.2 Member Data Documentation

15.22.2.1 string

```
character(len=query_length) amns_types::amns_query_type::string
```

used to pass a query about settings
Definition at line 189 of file amns_types.f90.

```
00189      character(len=query_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.23 amns_query_type Struct Reference

Type for querying parameters in the AMNS package ("interoperable" version)
`#include <amns_interface.h>`

Public Attributes

- char string [query_length]

  used to pass a query about settings

15.23.1 Detailed Description

Type for querying parameters in the AMNS package ("interoperable" version)
Definition at line 227 of file amns_interface.h.

15.23.2 Member Data Documentation

15.23.2.1 string

`char amns_query_type::string[query_length]`
used to pass a query about settings
Definition at line 228 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.24 amns_reactant_type Struct Reference

Type for indicating a single reactant or product when using the AMNS interface.
`#include <amns_interface.h>`

Public Attributes

- double ZN

  Nuclear charge.
- double ZA

  Atomic charge.
- double MI

  Atomic mass.
- int LR

  reactant (LR=0) or product (LR=1)
- double real_specifier

  a place holder to specify an optional real further characterising a reactant/product
- int int_specifier

  a place holder to specify an optional integer further characterising a reactant/product

15.24.1 Detailed Description

Type for indicating a single reactant or product when using the AMNS interface.
Definition at line 99 of file amns_interface.h.

15.24.2 Member Data Documentation

15.24.2.1 int_specifier

`int amns_reactant_type::int_specifier`
a place holder to specify an optional integer further characterising a reactant/product
Definition at line 105 of file amns_interface.h.

15.24.2.2 LR

`int amns_reactant_type::LR`
reactant (LR=0) or product (LR=1)
Definition at line 103 of file amns_interface.h.

15.24.2.3 MI

`double amns_reactant_type::MI`
Atomic mass.
Definition at line 102 of file amns_interface.h.

15.24.2.4 real_specifier

`double amns_reactant_type::real_specifier`
a place holder to specify an optional real further characterising a reactant/product
Definition at line 104 of file amns_interface.h.

15.24.2.5 ZA

`double amns_reactant_type::ZA`
Atomic charge.
Definition at line 101 of file amns_interface.h.

15.24.2.6 ZN

`double amns_reactant_type::ZN`
Nuclear charge.
Definition at line 100 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.25 amns_types::amns_reactant_type Type Reference

Type for indicating a single reactant or product when using the AMNS interface.

Public Attributes

- real(c_double) zn

  nuclear charge of the reactant/product
- real(c_double) za

  atomic charge of the reactant/product
- real(c_double) mi = 0.0_ids_real

  atomic mass of the reactant/product
- integer(c_int) lr = 0

  reactant (LR=0) or product (LR=1)

- real(c_double) real_specifier = ids_real_invalid

  a place holder to specify an optional real further characterising a reactant/product
- integer(c_int) int_specifier = ids_int_invalid

  a place holder to specify an optional integer further characterising a reactant/product

15.25.1 Detailed Description

Type for indicating a single reactant or product when using the AMNS interface.
Definition at line 70 of file amns_types.f90.

15.25.2 Member Data Documentation

15.25.2.1 int_specifier

```
integer (c_int) amns_types::amns_reactant_type::int_specifier = ids_int_invalid
```

a place holder to specify an optional integer further characterising a reactant/product
Definition at line 82 of file amns_types.f90.

```
00082      integer (c_int) :: int_specifier = ids_int_invalid
```

15.25.2.2 lr

```
integer (c_int) amns_types::amns_reactant_type::lr = 0
```

reactant (LR=0) or product (LR=1)
Definition at line 78 of file amns_types.f90.

```
00078      integer (c_int) :: LR = 0      ! LR=0 implies LHS; LR=1 implies RHS
```

15.25.2.3 mi

```
real (c_double) amns_types::amns_reactant_type::mi = 0.0_ids_real
```

atomic mass of the reactant/product
Definition at line 76 of file amns_types.f90.

```
00076      real (c_double) :: MI = 0.0_ids_real
```

15.25.2.4 real_specifier

```
real (c_double) amns_types::amns_reactant_type::real_specifier = ids_real_invalid
```

a place holder to specify an optional real further characterising a reactant/product
Definition at line 80 of file amns_types.f90.

```
00080      real (c_double) :: real_specifier = ids_real_invalid
```

15.25.2.5 za

```
real (c_double) amns_types::amns_reactant_type::za
```

atomic charge of the reactant/product
Definition at line 74 of file amns_types.f90.

```
00074      real (c_double) :: ZA
```

15.25.2.6 zn

```
real (c_double) amns_types::amns_reactant_type::zn
```

nuclear charge of the reactant/product
Definition at line 72 of file amns_types.f90.

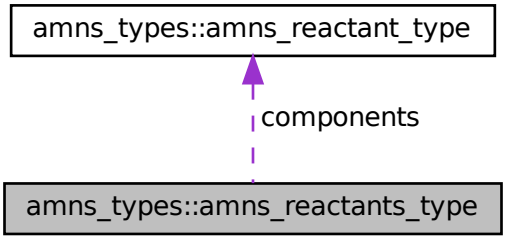
```
00072      real (c_double) :: ZN
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.26 amns_types::amns_reactants_type Type Reference

Type for indicating the reactants when using the AMNS interface NOT interoperable with C.
Collaboration diagram for amns_types::amns_reactants_type:



Public Attributes

- type(amns_reactant_type), dimension(:), allocatable components

  an array of reactants & products
- character(len=reaction_length), dimension(:), allocatable string

  ??
- integer index = 0

  ??

15.26.1 Detailed Description

Type for indicating the reactants when using the AMNS interface NOT interoperable with C.
Definition at line 88 of file amns_types.f90.

15.26.2 Member Data Documentation

15.26.2.1 components

```
type (amns_reactant_type), dimension(:), allocatable amns_types::amns_reactants_type::components
```

an array of reactants & products
Definition at line 90 of file amns_types.f90.

```
00090      type (amns_reactant_type), allocatable :: components(:)
```

15.26.2.2 index

```
integer amns_types::amns_reactants_type::index = 0
```

??
Definition at line 94 of file amns_types.f90.

```
00094      integer :: index = 0
```

15.26.2.3 string

```
character(len=reaction_length), dimension(:), allocatable amns_types::amns_reactants_type↩
::string
```

??

Definition at line 92 of file amns_types.f90.

```
00092      character(len=reaction_length), allocatable :: string(:)
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.27 amns_reaction_type Struct Reference

Type used for specifying reactions when using the AMNS interface ("interoperable" version)

```
#include <amns_interface.h>
```

Public Attributes

- char string [reaction_length]

  name of the reaction (e.g. RC)

- int isotope_resolved

  if the reaction is different for different isotopes, isotope_resolved shoukld be set to 1

15.27.1 Detailed Description

Type used for specifying reactions when using the AMNS interface ("interoperable" version)

Definition at line 163 of file amns_interface.h.

15.27.2 Member Data Documentation

15.27.2.1 isotope_resolved

```
int amns_reaction_type::isotope_resolved
```

if the reaction is different for different isotopes, isotope_resolved shoukld be set to 1

Definition at line 165 of file amns_interface.h.

15.27.2.2 string

```
char amns_reaction_type::string[reaction_length]
```

name of the reaction (e.g. RC)

Definition at line 164 of file amns_interface.h.

The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.28 amns_types::amns_reaction_type Type Reference

Type used for specifying reactions when using the AMNS interface (not interoperable)

Public Attributes

- character(len=reaction_length) string

  name of the reaction (e.g. RC)

- integer(c_int) isotope_resolved =0

  if the reaction is different for different isotopes, isotope_resolved should be set to 1

15.28.1 Detailed Description

Type used for specifying reactions when using the AMNS interface (not interoperable)
Definition at line 155 of file amns_types.f90.

15.28.2 Member Data Documentation

15.28.2.1 isotope_resolved

```
integer(c_int) amns_types::amns_reaction_type::isotope_resolved =0
```

if the reaction is different for different isotopes, isotope_resolved should be set to 1
Definition at line 159 of file amns_types.f90.

```
00159      integer(c_int) :: isotope_resolved=0
```

15.28.2.2 string

```
character(len=reaction_length) amns_types::amns_reaction_type::string
```

name of the reaction (e.g. RC)
Definition at line 157 of file amns_types.f90.

```
00157      character(len=reaction_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.29 amns_set_type Struct Reference

Type for setting parameters in the AMNS package ("interoperable" version)

```
#include <amns_interface.h>
```

Public Attributes

- char string [set_length]

  used to pass a change in the settings

15.29.1 Detailed Description

Type for setting parameters in the AMNS package ("interoperable" version)
Definition at line 200 of file amns_interface.h.

15.29.2 Member Data Documentation

15.29.2.1 string

```
char amns_set_type::string[set_length]
```

used to pass a change in the settings
Definition at line 201 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.30 amns_types::amns_set_type Type Reference

Type for setting parameters in the AMNS package (not interoperable)

Public Attributes

- character(len=set_length) string

  used to pass a change in the settings

15.30.1 Detailed Description

Type for setting parameters in the AMNS package (not interoperable)
Definition at line 173 of file amns_types.f90.

15.30.2 Member Data Documentation

15.30.2.1 string

```
character(len=set_length) amns_types::amns_set_type::string
```

used to pass a change in the settings
Definition at line 175 of file amns_types.f90.

```
00175      character(len=set_length) :: string
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.31 amns_types::amns_version_type Type Reference

Type for specifying the AMNS version (not interoperable)

Public Attributes

- character(len=version_length) string = ''

  ascii specification of the version

- integer(c_int) number = 0

  integer specification of the version number (primary specification at the moment)

- character(len=version_length) backend = 'mdsplus'

  specify the backend to be used to access the IDSs

- character(len=version_length) user = ''

  specify the username of the data to be obtained (defaults to the person running the code)

15.31.1 Detailed Description

Type for specifying the AMNS version (not interoperable)
Definition at line 44 of file amns_types.f90.

15.31.2 Member Data Documentation

15.31.2.1 backend

```
character(len=version_length) amns_types::amns_version_type::backend = 'mdsplus'
```

specify the backend to be used to access the IDSs
Definition at line 50 of file amns_types.f90.

```
00050      character(len=version_length) :: backend = 'mdsplus'
```

15.31.2.2 number

```
integer(c_int) amns_types::amns_version_type::number = 0
```

integer specification of the version number (primary specification at the moment)
Definition at line 48 of file amns_types.f90.

```
00048      integer(c_int) :: number = 0
```

15.31.2.3 string

```
character(len=version_length) amns_types::amns_version_type::string = ''
```

ascii specification of the version
Definition at line 46 of file amns_types.f90.

```
00046      character(len=version_length) :: string = ''
```

15.31.2.4 user

```
character(len=version_length) amns_types::amns_version_type::user = ''
```

specify the username of the data to be obtained (defaults to the person running the code)
Definition at line 52 of file amns_types.f90.

```
00052      character(len=version_length) :: user = ''
```

The documentation for this type was generated from the following file:

- src/libamns/amns_types.f90

15.32 amns_version_type Struct Reference

Type for specifying the AMNS version ("interoperable" version)
`#include <amns_interface.h>`

Public Attributes

- char string [version_length]

  ascii specification of the version
- int number

  integer specification of the version number (primary specification at the moment)
- char backend [version_length]

  specify the backend to be used to access the CPOs
- char user [version_length]

  specify the username of the data to be obtained (defaults to tghe person running the code)

15.32.1 Detailed Description

Type for specifying the AMNS version ("interoperable" version)
Definition at line 54 of file amns_interface.h.

15.32.2 Member Data Documentation

15.32.2.1 backend

```
char amns_version_type::backend[version_length]
```

specify the backend to be used to access the CPOs
Definition at line 57 of file amns_interface.h.

15.32.2.2 number

`int amns_version_type::number`
integer specification of the version number (primary specification at the moment)
Definition at line 56 of file amns_interface.h.

15.32.2.3 string

`char amns_version_type::string[version_length]`
ascii specification of the version
Definition at line 55 of file amns_interface.h.

15.32.2.4 user

`char amns_version_type::user[version_length]`
specify the username of the data to be obtained (defaults to tghe person running the code)
Definition at line 58 of file amns_interface.h.
The documentation for this struct was generated from the following file:

- include/amns_interface.h

15.33 amns.type.AmnsAnswerType Class Reference

Public Attributes

- String string
- int number

15.33.1 Detailed Description

Contains definition of type that is used for answer passing between JNI and C. This type is a Java based version of amns_answer_type This is just a simple JavaBean with publicly accessible fields.
type, bind(c) :: amns_answer_type - is defined in amns_types.F90
It contains two fields `string` and `number`. Depending on context, you can get output as string, number or both.
Definition at line 12 of file AmnsAnswerType.java.

15.33.2 Member Data Documentation

15.33.2.1 number

`int amns.type.AmnsAnswerType.number`
Contains number based output from AMNS library
Definition at line 16 of file AmnsAnswerType.java.

15.33.2.2 string

`String amns.type.AmnsAnswerType.string`
Contains string based output from AMNS library
Definition at line 14 of file AmnsAnswerType.java.
The documentation for this class was generated from the following file:

- src/java/src/amns/type/AmnsAnswerType.java

15.34 amnsdemo.AmnsDemoCaseIdx Class Reference

Static Public Member Functions

- static void main (String[ ] args) throws Exception

15.34.1 Detailed Description

This class implements demo case based on amns_demo_c.c code. It contains the same code (in terms of the logic) but uses Java to JNI calls.
All AMNS related calls are done via Java - JNI based interface to AMNS. Interface is defined in

See also

amns.Amns

Definition at line 12 of file AmnsDemoCaseIdx.java.

15.34.2 Member Function Documentation

15.34.2.1 main()

```
static void amnsdemo.AmnsDemoCaseIdx.main (
            String[ ] args ) throws Exception  [inline], [static]
```

Performs amns related calls that invoke native methods defined in

See also

amns.Amns class

Definition at line 16 of file AmnsDemoCaseIdx.java.

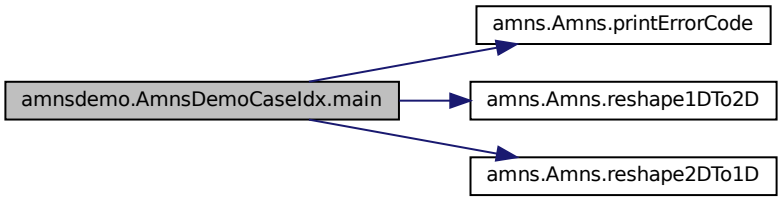
```
00016                                                   {
00017
00018         int ns = 7; // number of species
00019         int nx=101, ny=101; // size of the arrays used in computations
00020         int is, ix, iy; // variables used in for loops later on in the code
00021
00022         // this is a pointer from C
00023         long ptrAmnsHandle = 0;
00024         AmnsErrorType error_stat = new AmnsErrorType();
00025
00026         // these are the pointers as well
00027         // we have to remember that in Java there is no such concept
00028         // as pointer. We will get all the pointers from C as long values
00029         //
00030         long ptrReactantsHandle[] = new long[ns];
00031         long ptrAmnsCxHandle[] = new long[ns];
00032
00033         boolean doNuclear;
00034         // Dimensions of multi-dimensional arrays have to be specified
00035         // in reverse order compared to Fortran
00036         // there is one more step in here, before we will pass arrays to
00037         // JNI we will "flattern" them by storing in 1D array
00038         double te[][] = new double[ny][nx];
00039         double ne[][] = new double [ny][nx];
00040         double rate[][][] = new double[ns][ny][nx];
00041         double rate1[][] = new double[ny][nx];
00042
00043         Amns amns = new Amns();
00044
00045         // Setup
00046         String imas_amns_debug = System.getenv("IMAS_AMNS_DEBUG");
00047         Boolean amns_debug = (imas_amns_debug != null && !imas_amns_debug.trim().isEmpty() &&
      !imas_amns_debug.equals("no") && !imas_amns_debug.equals("NO") );
00048
00049         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP");
00050         ptrAmnsHandle = amns.ImasAmnsCCSetup(error_stat);
00051         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " +
      ptrAmnsHandle);
00052         if (amns_debug) System.err.println("[JVM] Error.flag= " + error_stat.flag + " Error.string= "
      + error_stat.string);
00053
00054         // Set up species
00055         for ( is=0; is < ns; is++) {
```

```
00056             // string and index are unused here
00057             int idx = 0;
00058             if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP_REACTANTS: " + (is +
      1) + "/" + ns);
00059             long tempPtrReactantsHandle = amns.ImasAmnsCcSetupReactantsNumber(idx, 4);
00060             if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      tempPtrReactantsHandle);
00061
00062             ptrReactantsHandle[is] = tempPtrReactantsHandle;
00063
00064             // setup first product
00065             AmnsReactantType species = new AmnsReactantType();
00066             species.ZN = 6;
00067             species.ZA = is;
00068             species.MI = 12;
00069             species.LR = 0;
00070
00071             amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle[is], 1, species);
00072
00073             // setup second product
00074             species.ZN = 1;
00075             species.ZA = 0;
00076             species.MI = 2;
00077             species.LR = 0;
00078
00079             amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle[is], 2, species);
00080
00081             // setup third product
00082             species.ZN = 6;
00083             species.ZA = is - 1;
00084             species.MI = 12;
00085             species.LR = 1;
00086
00087             amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle[is], 3, species);
00088
00089             // setup fourth product
00090             species.ZN = 1;
00091             species.ZA = 1;
00092             species.MI = 2;
00093             species.LR = 1;
00094
00095             amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle[is], 4, species);
00096
00097             // sanity check for the GET routine
00098             species.ZN = 0;
00099             species.ZA = 0;
00100             species.MI = 0;
00101
00102             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00103
00104             System.out.println( "[JVM] Values from C: \n"
00105                                 + " species.ZN = " + species.ZN
00106                                 + " species.ZA = " + species.ZA
00107                                 + " species.MI = " + species.MI);
00108         }
00109
00110
00111         for (ix=0; ix<nx; ix++) {
00112             for (iy=0; iy<ny; iy++) {
00113                 te[iy][ix] = java.lang.Math.pow( 10.0, (double)ix / ((double)(nx - 1) / 5.0) ) * 0.1;
00114             }
00115         }
00116
00117         for (iy=0; iy<ny; iy++) {
00118             for (ix=0; ix<nx; ix++) {
00119                 ne[iy][ix] = java.lang.Math.pow( 10.0, (double)iy / (((double)ny - 1) / 10.0) ) *
      1.0e15;
00120             }
00121         }
00122
00123         AmnsSetType set = new AmnsSetType();
00124         AmnsErrorType error = new AmnsErrorType();
00125         set.string = "backend=mdsplus";
00126
00127         amns.ImasAmnsCCSet(ptrAmnsHandle, set, error);
00128
00129         AmnsQueryType query = new AmnsQueryType();
00130         query.string = "version";
00131
00132         AmnsAnswerType answer = new AmnsAnswerType();
00133
00134         amns.ImasAmnsCCQuery(ptrAmnsHandle, query, answer, error);
00135         System.out.println("[JVM] AMNS data base version = '" + answer.string + "'," + answer.number);
00136
00137         AmnsReactionType xx_rx = new AmnsReactionType();
00138         xx_rx.string = "CX";
00139
```

```
00140         // set up tables for charge-exchange
00141         for (is=0; is < ns; is++) {
00142 
00143             AmnsReactantType species = new AmnsReactantType();
00144 
00145             // get the species
00146             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00147             if (species.ZA == 0) continue;
00148 
00149             long ptrCXHandle = 0;
00150 
00151             if (amns_debug) System.err.println("[JVM] Setting up table: " + is);
00152             ptrCXHandle = amns.ImasAmnsCCSetupTable(
00153                                                     ptrAmnsHandle
00154                                                     , xx_rx
00155                                                     , ptrReactantsHandle[is]
00156                                                     , error_stat);
00157             ptrAmnsCxHandle[is] = ptrCXHandle;
00158             if (amns_debug) System.err.println("[JVM] Pointer to table: " + ptrCXHandle);
00159             Amns.printErrorCode( error_stat, "ImasAmnsCCSetupTable");
00160 
00161             // we will use query and answer objects declared above
00162             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00163             Amns.printErrorCode( error_stat, "ImasAmnsCCQueryTable");
00164 
00165             System.out.println("[JVM] AMNS table version = " + answer.number + " for " + answer.string
      + ", IS = " + is);
00166         }
00167 
00168         // query the tables (for one given process)
00169         for (is=0; is < ns; is++) {
00170 
00171             AmnsReactantType species = new AmnsReactantType();
00172 
00173             // get the species
00174             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00175 
00176             if (species.ZA == 0) continue;
00177 
00178             // we will reuse previous query and answer objects
00179             query.string = "no_of_reactants";
00180             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00181             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00182             System.out.println("[JVM] is, " + query.string + " = " + is + ", " + answer.string);
00183 
00184             query.string = "source";
00185             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00186             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00187             System.out.println("[JVM] is, " + query.string + " = " + is + ", " + answer.string);
00188 
00189             query.string = "filled";
00190             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00191             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00192             System.out.println("[JVM] is, " + query.string + " = " + is + ", " + answer.string);
00193 
00194             query.string = "reaction_type";
00195             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00196             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00197             System.out.println("[JVM] is, " + query.string + " = " + is + ", " + answer.string);
00198 
00199             query.string = "reactants";
00200             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00201             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00202             if (amns_debug) System.err.println("[JVM] is, " + query.string + " = " + is + ", " +
      answer.string);
00203 
00204         }
00205 
00206         // interpolate charge-exchange data, and output the results
00207         set.string = "nowarn";
00208 
00209         for (is=0; is < ns; is++) {
00210 
00211             AmnsReactantType species = new AmnsReactantType();
00212 
00213             // get the species
00214             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00215             if (species.ZA == 0) continue;
00216 
00217             if (amns_debug) System.err.println("[JVM] Interpolate charge-exchange. Loop: " + is);
00218             if (amns_debug) System.err.println("[JVM] Calling ImasAmnsCCSetTable: " +
      ptrAmnsCxHandle[is]);
```

```
00219             amns.ImasAmnsCCSetTable(ptrAmnsCxHandle[is], set, error_stat);
00220             Amns.printErrorCode( error_stat, " after ImasAmnsCCSetTable");
00221 
00222             // we have to reshape table to 1D from 2D
00223             // number of elements in 1D array will be nx * ny
00224             // we have to simply copy row by row an create 1D array here
00225             // NOTE! the result of this call is 1D array of rates. We have to reshape it to proper
      shape
00226             double [] te1D = Amns.reshape2DTo1D(te, ny, nx);
00227             double [] ne1D = Amns.reshape2DTo1D(ne, ny, nx);
00228 
00229             if (amns_debug) System.err.println("[JVM] before calling ImasAmnsCCRX1B: " +
      ptrAmnsCxHandle[is]);
00230             double [] rates1D = amns.ImasAmnsCCRX1B(ptrAmnsCxHandle[is], nx * ny, te1D, ne1D,
      error_stat);
00231 
00232             double [][] rates2D = Amns.reshape1DTo2D( rates1D, ny, nx);
00233             System.out.println("Some CX rates for species is=" + is + "\n");
00234             for (ix=0; ix<10; ix++) {
00235                 System.out.println("is=" + is + " rate(is,0," + ix + ")=" + rates2D[0][ix] + "\n");
00236             }
00237         }
00238 
00239         // finalize the tables
00240         for ( is=0; is < ns; is++) {
00241             // get the species
00242             AmnsReactantType species = new AmnsReactantType();
00243 
00244             // get the species
00245             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00246             if (species.ZA == 0) continue;
00247 
00248             amns.ImasAmnsCCFinishTable(ptrAmnsCxHandle[is], error_stat);
00249             Amns.printErrorCode( error_stat, " after ImasAmnsCCFinishTable");
00250         }
00251 
00252         // Clean up reactants data structures
00253         for (is=0; is < ns; is++) {
00254             amns.ImasAmnsCCFinishReactants(ptrReactantsHandle[is]);
00255         }
00256 
00257         amns.ImasAmnsCCFinish(ptrAmnsHandle, error);
00258     }
```

Here is the call graph for this function:



The documentation for this class was generated from the following file:

- examples/java/src/amnsdemo/AmnsDemoCaseIdx.java

15.35 amns.type.AmnsErrorType Class Reference

Public Attributes

- boolean flag
- String string

15.35.1 Detailed Description

Contains definition of type that is used for error passing between JNI and C. This type is a Java based version of amns_error_type This is just a simple JavaBean with publicly accessible fields.

type, bind(c) :: amns_error_type - is defined in amns_types.F90
It contains two fields `flag` and `string`.
Definition at line 12 of file AmnsErrorType.java.

15.35.2 Member Data Documentation

15.35.2.1 flag

`boolean amns.type.AmnsErrorType.flag`
determines error code - 1: error; 0: no error
Definition at line 14 of file AmnsErrorType.java.

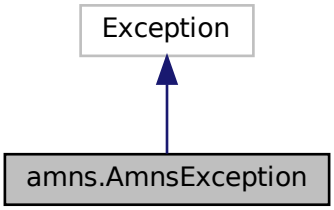
15.35.2.2 string

`String amns.type.AmnsErrorType.string`
contains message passed from AMNS library
Definition at line 16 of file AmnsErrorType.java.
The documentation for this class was generated from the following file:

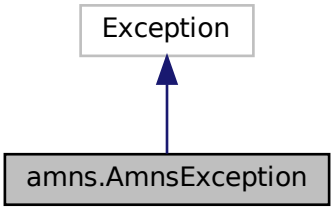
- src/java/src/amns/type/AmnsErrorType.java

15.36 amns.AmnsException Class Reference

Inheritance diagram for amns.AmnsException:



Collaboration diagram for amns.AmnsException:



15.36.1 Detailed Description

Definition at line 736 of file amns.pyx.
The documentation for this class was generated from the following file:

- src/py/amns/amns.pyx

15.37 AmnsMinimal Class Reference

Static Public Member Functions

- static void main (String[] args)

15.37.1 Detailed Description

Class provides implementation of AMNS testing code - equivalent for testminimal.c This is a minimal case for AMNS calls that can be used for testing. Result from the code
`"[JVM] Value of rate:  " + rate` should produce exactly the same value as testminimal.c code.
Definition at line 10 of file AmnsMinimal.java.

15.37.2 Member Function Documentation

15.37.2.1 main()

```
static void AmnsMinimal.main (
            String[] args )  [inline], [static]
```

This method performs calls to AMNS library that are the equivalent of testminimal.c code
Definition at line 13 of file AmnsMinimal.java.

```
00013                                       {
00014
00015         // this is a pointer from C
00016         long ptrAmnsHandle = 0;
00017         AmnsErrorType error_stat = new AmnsErrorType();
00018
00019         // this is a pointer from C
00020         long ptrReactantsHandle = 0;
00021
00022         // Definition of species
00023         AmnsReactantType species1 = new AmnsReactantType();
00024         species1.ZN = 6;
00025         species1.ZA = 1;
00026         species1.MI = 12;
00027         species1.LR = 0;
00028
00029         AmnsReactantType species2 = new AmnsReactantType();
00030         species2.ZN = 1;
00031         species2.ZA = 0;
00032         species2.MI = 2;
00033         species2.LR = 0;
00034
00035         AmnsReactantType species3 = new AmnsReactantType();
00036         species3.ZN = 6;
00037         species3.ZA = 0;
00038         species3.MI = 12;
00039         species3.LR = 1;
00040
00041         AmnsReactantType species4 = new AmnsReactantType();
00042         species4.ZN = 1;
00043         species4.ZA = 1;
00044         species4.MI = 2;
00045         species4.LR = 1;
00046
00047         long ptrCXHandle = 0;
00048         AmnsReactionType reactionType = new AmnsReactionType();
00049         reactionType.string = "CX";
00050         reactionType.isotopeResolved = 0;
00051
00052         double rate = 0;
00053
00054         // Class to manage Amns calls to low level
00055         Amns amns = new Amns();
```

```
00056
00057         // Setup
00058         String imas_amns_debug = System.getenv("IMAS_AMNS_JAVA_DEBUG");
00059         Boolean amns_debug = (imas_amns_debug != null && !imas_amns_debug.trim().isEmpty() &&
      !imas_amns_debug.equals("no") && !imas_amns_debug.equals("NO") );
00060
00061         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP");
00062         ptrAmnsHandle = amns.ImasAmnsCCSetup(error_stat);
00063         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " +
      ptrAmnsHandle);
00064         if (amns_debug) System.err.println("[JVM] Error.flag= " + error_stat.flag + " Error.string= "
      + error_stat.string);
00065
00066         int idx = 0;
00067         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP_REACTANTS");
00068         ptrReactantsHandle = amns.ImasAmnsCcSetupReactantsNumber(idx, 4);
00069         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      ptrReactantsHandle);
00070
00071         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species1");
00072         amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 1, species1);
00073
00074         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species2");
00075         amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 2, species2);
00076
00077         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species3");
00078         amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 3, species3);
00079
00080         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species4");
00081         amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 4, species4);
00082
00083         // we are calling now SETUP_TABLE using all the elements above
00084         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP_TABLE");
00085         ptrCXHandle = amns.ImasAmnsCCSetupTable(ptrAmnsHandle, reactionType
00086                                                 , ptrReactantsHandle
00087                                                 , error_stat);
00088         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
00089
00090         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_RX_0_B");
00091         rate = amns.ImasAmnsCCRX0B( ptrCXHandle, 100, 1.0e20, error_stat);
00092         System.out.println("Value of rate: " + rate);
00093
00094         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_TABLE");
00095         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
00096         ptrCXHandle =amns.ImasAmnsCCFinishTable( ptrCXHandle, error_stat);
00097         if (amns_debug) System.err.println("[JVM] After IMAS_AMNS_CC_FINISH_TABLE");
00098         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
00099
00100         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_REACTANTS");
00101         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      ptrReactantsHandle);
00102         ptrReactantsHandle = amns.ImasAmnsCCFinishReactants(ptrReactantsHandle);
00103         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_REACTANTS");
00104         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      ptrReactantsHandle);
00105
00106         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH");
00107         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " +
      ptrAmnsHandle);
00108         ptrAmnsHandle = amns.ImasAmnsCCFinish(ptrAmnsHandle, error_stat);
00109         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH");
00110         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " +
      ptrAmnsHandle);
00111     }
```

The documentation for this class was generated from the following file:

- tests/java/AmnsMinimal.java

15.38 amns.type.AmnsQueryType Class Reference

Public Attributes

- String string

15.38.1 Detailed Description

Contains definition of type that is used for query passing between JNI and C. This type is a Java based version of amns_query_type This is just a simple JavaBean with publicly accessible fields.
type, bind(c) :: amns_query_type - is defined in amns_types.F90

It contains one field: `string` This field must be filled in before you proceed with the call to AMNS that requires query as argument.
Definition at line 12 of file AmnsQueryType.java.

15.38.2 Member Data Documentation

15.38.2.1 string

`String amns.type.AmnsQueryType.string`
Query to be passed to AMNS library
Query string can have one of the values: "reaction_type", "reactants", "version", "state_label","result_unit","result↩
_label", "ndim","interp_fun","source","no_of_reactants","index","filled"
Please note, that string values are subject to change in the AMNS library.
In case you are not sure whether given string value is correct, refer to amns_module.F90<p> subroutine IMAS_↩
AMNS_QUERY_TABLE
Definition at line 22 of file AmnsQueryType.java.
The documentation for this class was generated from the following file:

- src/java/src/amns/type/AmnsQueryType.java

15.39 amns.type.AmnsReactantType Class Reference

Public Attributes

- double ZN
- double ZA
- double MI
- int LR
- double real_specifier
- int int_specifier

15.39.1 Detailed Description

Contains definition of type that defines Reactant. This type is a Java based equivalent for amns_reactant_type This is just a simple JavaBean with publicly accessible fields.
type, bind(c) :: amns_reactant_type - is defined in amns_types.F90
Definition at line 10 of file AmnsReactantType.java.

15.39.2 Member Data Documentation

15.39.2.1 int_specifier

`int amns.type.AmnsReactantType.int_specifier`
TODO
Definition at line 22 of file AmnsReactantType.java.

15.39.2.2 LR

`int amns.type.AmnsReactantType.LR`
LR=0 implies LHS; LR=1 implies RHS
Definition at line 18 of file AmnsReactantType.java.

15.39.2.3 MI

`double amns.type.AmnsReactantType.MI`
TODO
Definition at line 16 of file AmnsReactantType.java.

15.39.2.4 real_specifier

`double amns.type.AmnsReactantType.real_specifier`
TODO
Definition at line 20 of file AmnsReactantType.java.

15.39.2.5 ZA

`double amns.type.AmnsReactantType.ZA`
TODO
Definition at line 14 of file AmnsReactantType.java.

15.39.2.6 ZN

`double amns.type.AmnsReactantType.ZN`
TODO
Definition at line 12 of file AmnsReactantType.java.
The documentation for this class was generated from the following file:

- src/java/src/amns/type/AmnsReactantType.java

15.40 amns.type.AmnsReactionType Class Reference

Public Attributes

- String string
- int isotopeResolved

15.40.1 Detailed Description

Contains definition of reaction type passing between JNI and C. This type is a Java based version of amns_reaction_type (Type used for specifying reactions when using the AMNS interface). This is just a simple JavaBean with publicly accessible fields.
type, bind(c) :: amns_reaction_type - is defined in amns_types.F90
It contains two fields `string` and `isotopeResolved`.
Definition at line 11 of file AmnsReactionType.java.

15.40.2 Member Data Documentation

15.40.2.1 isotopeResolved

`int amns.type.AmnsReactionType.isotopeResolved`
Definition at line 13 of file AmnsReactionType.java.

15.40.2.2 string

`String amns.type.AmnsReactionType.string`
Definition at line 12 of file AmnsReactionType.java.
The documentation for this class was generated from the following file:

- src/java/src/amns/type/AmnsReactionType.java

15.41 amns.type.AmnsSetType Class Reference

Public Attributes

- String string

15.41.1 Detailed Description

Contains definition of type that defines settings. This type is a Java based equivalent for amns_set_type. Type for setting parameters in the AMNS package. This is just a simple JavaBean with publicly accessible fields.
type, bind(c) :: amns_set_type - is defined in amns_types.F90
Definition at line 10 of file AmnsSetType.java.

15.41.2 Member Data Documentation

15.41.2.1 string

`String amns.type.AmnsSetType.string`
Settings passed to AMNS library
string can contain one of the values: "debug", "nodebug", "backend=mdsplus", "backend=hdf5", "backend=ascii"
In case you are in doubt, check the subroutine IMAS_AMNS_SET inside amns_module.F90
Definition at line 16 of file AmnsSetType.java.
The documentation for this class was generated from the following file:

- src/java/src/amns/type/AmnsSetType.java

15.42 amns_module_isoc::copy Interface Reference

Public Member Functions

- pure character(size(a)) function copy_a2s (a)
- pure character function, dimension(len(s)) copy_s2a (s)

15.42.1 Detailed Description

Definition at line 18 of file amns_module_isoc.f90.

15.42.2 Member Function/Subroutine Documentation

15.42.2.1 copy_a2s()

```
pure character(size(a)) function amns_module_isoc::copy::copy_a2s (
             character, dimension(:), intent(in) a )
```

Definition at line 25 of file amns_module_isoc.f90.

```
00026     character,intent(in) :: a(:)
00027     character(size(a)) :: s
00028     integer :: i
00029     do i = 1,size(a)
```

```
00030         s(i:i) = a(i)
00031       end do
```

15.42.2.2 copy_s2a()

```
pure character function, dimension(len(s)) amns_module_isoc::copy::copy_s2a (
              character(*), intent(in) s )
```

Definition at line 35 of file amns_module_isoc.f90.

```
00036       character(*),intent(in) :: s
00037       character :: a(len(s))
00038       integer :: i
00039       do i = 1,len(s)
00040          a(i) = s(i:i)
00041       end do
```

The documentation for this interface was generated from the following file:

- src/libamns/amns_module_isoc.f90

15.43 amns_external_functions::fun_err_t Type Reference

Public Attributes

- integer ierr
- character(len=128) cerr

15.43.1 Detailed Description

Definition at line 12 of file amns_external_functions.f90.

15.43.2 Member Data Documentation

15.43.2.1 cerr

```
character (len=128) amns_external_functions::fun_err_t::cerr
```

Definition at line 14 of file amns_external_functions.f90.

```
00014       character (len=128)          :: cerr
```

15.43.2.2 ierr

```
integer amns_external_functions::fun_err_t::ierr
```

Definition at line 13 of file amns_external_functions.f90.

```
00013       integer                      :: ierr
```

The documentation for this type was generated from the following file:

- src/libamns/amns_external_functions.f90

15.44 amns_module::imas_amns_rx Interface Reference

get the rates associated with the input args for a particular reaction (generic interface for 1d, 2d & 3d arrays)

Public Member Functions

- subroutine imas_amns_rx_0 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (0d) args for a particular reaction
- subroutine imas_amns_rx_1 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (1d) args for a particular reaction

- subroutine imas_amns_rx_2 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (2d) args for a particular reaction
- subroutine imas_amns_rx_3 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (3d) args for a particular reaction

15.44.1 Detailed Description

get the rates associated with the input args for a particular reaction (generic interface for 1d, 2d & 3d arrays)

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system; scalar, 1d, 2d or 3d
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; rank and dimensionality the same as out
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; rank and dimensionality the same as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; rank and dimensionality the same as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; rank and dimensionality the same as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

Todo Update this interface for 4D and 5D

Definition at line 28 of file amns_module.f90.

15.44.2 Member Function/Subroutine Documentation

15.44.2.1 imas_amns_rx_0()

```
subroutine amns_module::imas_amns_rx::imas_amns_rx_0 (
             type(amns_handle_rx_type), intent(inout) handle_rx,
             real (kind=ids_real), intent(out) out,
             real (kind=ids_real), intent(in) arg1,
             real (kind=ids_real), intent(in), optional arg2,
             real (kind=ids_real), intent(in), optional arg3,
             real (kind=ids_real), intent(in), optional arg4,
             type(amns_error_type), intent(out), optional error_status )
```

get the rates associated with the input (0d) args for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system (scalar)
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; (scalar) of the same size as out
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; (scalar) of the same size as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; (scalar) of the same size as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; (scalar) of the same size as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

Definition at line 785 of file amns_module.f90.

```
00786     use ids_types  ! IGNORE
00787     use amns_types
00788     use data_suport
00789     implicit none
00790     optional arg2,arg3,arg4,error_status
00791     type(amns_handle_rx_type), intent(inout) :: handle_rx
00792     real (kind=ids_real), intent(out) :: out
00793     real (kind=ids_real), intent(in) :: arg1,arg2,arg3,arg4
00794     type(amns_error_type), intent(out) :: error_status
00795
00796     type(data_error_t) :: data_error
00797     real (kind=ids_real) :: out_d(1),arg1_d(1),arg2_d(1),arg3_d(1),arg4_d(1)
00798
00799     if(present(error_status)) then
00800         error_status%flag=.false.
00801         error_status%string="No error"
00802     end if
00803
00804     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_0 start'
00805     if(present(arg4)) then
00806        if(present(arg3).and.present(arg2)) then
00807           if(handle_rx%filled) then
00808              arg1_d(1)=arg1
00809              arg2_d(1)=arg2
00810              arg3_d(1)=arg3
00811              arg4_d(1)=arg4
00812              call interpol( &
00813                   reshape(arg1_d,(/size(arg1_d)/)), &
00814                   reshape(arg2_d,(/size(arg2_d)/)), &
00815                   reshape(arg3_d,(/size(arg3_d)/)), &
00816                   reshape(arg4_d,(/size(arg4_d)/)), &
00817                   fd1=out_d, grid=handle_rx%grid, &
00818                   data_error=data_error)
00819              if(present(error_status)) then
00820                 error_status%flag = data_error%ierr .ne. 0
00821                 error_status%string = data_error%cerr
00822              else
00823                 if(data_error%ierr.ne.0) then
00824                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
00825                    call errorstop('interpol returned an error')
00826                 endif
00827              endif
00828              out=out_d(1)
00829           else
00830              write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00831           endif
00832        else
00833           write(*,*) 'IMAS_AMNS_RX_0: arg4 present but not arg2/arg3!'
00834        endif
00835     else
00836        if(present(arg3)) then
00837           if(present(arg2)) then
00838              if(handle_rx%filled) then
00839                 arg1_d(1)=arg1
00840                 arg2_d(1)=arg2
00841                 arg3_d(1)=arg3
00842                 call interpol( &
00843                      reshape(arg1_d,(/size(arg1_d)/)), &
00844                      reshape(arg2_d,(/size(arg2_d)/)), &
00845                      reshape(arg3_d,(/size(arg3_d)/)), &
00846                      fd1=out_d, grid=handle_rx%grid, &
00847                      data_error=data_error)
00848                 if(present(error_status)) then
00849                    error_status%flag = data_error%ierr .ne. 0
00850                    error_status%string = data_error%cerr
00851                 else
00852                    if(data_error%ierr.ne.0) then
00853                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
00854                       call errorstop('interpol returned an error')
00855                    endif
00856                 endif
00857                 out=out_d(1)
00858              else
00859                 write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00860              endif
00861           else
00862              write(*,*) 'IMAS_AMNS_RX_0: arg3 present but not arg2!'
00863           endif
00864        else
00865           if(present(arg2)) then
00866              if(handle_rx%filled) then
00867                 arg1_d(1)=arg1
00868                 arg2_d(1)=arg2
00869                 call interpol( &
00870                      reshape(arg1_d,(/size(arg1_d)/)), &
```

```
00871                     reshape(arg2_d,(/size(arg2_d)/)), &
00872                     fd1=out_d, grid=handle_rx%grid, &
00873                     data_error=data_error)
00874                if(present(error_status)) then
00875                   error_status%flag = data_error%ierr .ne. 0
00876                   error_status%string = data_error%cerr
00877                else
00878                   if(data_error%ierr.ne.0) then
00879                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
00880                      call errorstop('interpol returned an error')
00881                   endif
00882                endif
00883                out=out_d(1)
00884             else
00885                write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00886             endif
00887          else
00888             if(handle_rx%filled) then
00889                arg1_d(1)=arg1
00890                call interpol( &
00891                     reshape(arg1_d,(/size(arg1_d)/)), &
00892                     fd1=out_d, grid=handle_rx%grid, &
00893                     data_error=data_error)
00894                if(present(error_status)) then
00895                   error_status%flag = data_error%ierr .ne. 0
00896                   error_status%string = data_error%cerr
00897                else
00898                   if(data_error%ierr.ne.0) then
00899                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
00900                      call errorstop('interpol returned an error')
00901                   endif
00902                endif
00903                out=out_d(1)
00904             else
00905                write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00906             endif
00907          endif
00908       endif
00909    endif
00910    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_0 end'
00911
```

15.44.2.2 imas_amns_rx_1()

```
subroutine amns_module::imas_amns_rx::imas_amns_rx_1 (
             type(amns_handle_rx_type), intent(inout) handle_rx,
             real (kind=ids_real), dimension(:), intent(out) out,
             real (kind=ids_real), dimension(:), intent(in) arg1,
             real (kind=ids_real), dimension(:), intent(in), optional arg2,
             real (kind=ids_real), dimension(:), intent(in), optional arg3,
             real (kind=ids_real), dimension(:), intent(in), optional arg4,
             type(amns_error_type), intent(out), optional error_status )
```

get the rates associated with the input (1d) args for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system (1d array)
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; (1d array) of the same size as out
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; (1d array) of the same size as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; (1d array) of the same size as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; (1d array) of the same size as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

Definition at line 923 of file amns_module.f90.

```
00924     use ids_types  ! IGNORE
00925     use amns_types
00926     use data_suport
00927     implicit none
00928     optional arg2,arg3,arg4,error_status
00929     type(amns_handle_rx_type), intent(inout) :: handle_rx
00930     real (kind=ids_real), intent(out) :: out(:)
00931     real (kind=ids_real), intent(in) :: arg1(:),arg2(:),arg3(:),arg4(:)
00932     type(amns_error_type), intent(out) :: error_status
00933
00934     type(data_error_t) :: data_error
00935
00936     if(present(error_status)) then
00937         error_status%flag=.false.
00938         error_status%string="No error"
00939     end if
00940
00941     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1 start'
00942     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(out) = ',lbound(out),ubound(out)
00943     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg1) = ',lbound(arg1),ubound(arg1)
00944     if(present(arg4)) then
00945        if(present(arg3).and.present(arg2)) then
00946           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) = ',lbound(arg2),ubound(arg2)
00947           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg3) = ',lbound(arg3),ubound(arg3)
00948           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg4) = ',lbound(arg4),ubound(arg4)
00949           if(handle_rx%filled) then
00950              call interpol( &
00951                   reshape(arg1,(/size(arg1)/)), &
00952                   reshape(arg2,(/size(arg2)/)), &
00953                   reshape(arg3,(/size(arg3)/)), &
00954                   reshape(arg4,(/size(arg4)/)), &
00955                   fd1=out, grid=handle_rx%grid, &
00956                   data_error=data_error)
00957              if(present(error_status)) then
00958                 error_status%flag = data_error%ierr .ne. 0
00959                 error_status%string = data_error%cerr
00960              else
00961                 if(data_error%ierr.ne.0) then
00962                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
00963                    call errorstop('interpol returned an error')
00964                 endif
00965              endif
00966           else
00967              write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
00968           endif
00969        else
00970           write(*,*) 'IMAS_AMNS_RX_1: arg4 present but not arg2/arg3!'
00971        endif
00972     else
00973        if(present(arg3)) then
00974           if(present(arg2)) then
00975              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) =
       ',lbound(arg2),ubound(arg2)
00976              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg3) =
       ',lbound(arg3),ubound(arg3)
00977              if(handle_rx%filled) then
00978                 call interpol( &
00979                      reshape(arg1,(/size(arg1)/)), &
00980                      reshape(arg2,(/size(arg2)/)), &
00981                      reshape(arg3,(/size(arg3)/)), &
00982                      fd1=out, grid=handle_rx%grid, &
00983                      data_error=data_error)
00984                 if(present(error_status)) then
00985                    error_status%flag = data_error%ierr .ne. 0
00986                    error_status%string = data_error%cerr
00987                 else
00988                    if(data_error%ierr.ne.0) then
00989                       write(*,*) 'interpol returned an error: ', data_error%ierr,
       trim(data_error%cerr)
00990                       call errorstop('interpol returned an error')
00991                    endif
00992                 endif
00993              else
00994                 write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
00995              endif
00996           else
00997              write(*,*) 'IMAS_AMNS_RX_1: arg3 present but not arg2!'
00998           endif
00999        else
01000           if(present(arg2)) then
01001              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) =
       ',lbound(arg2),ubound(arg2)
01002              if(handle_rx%filled) then
01003                 call interpol( &
01004                      reshape(arg1,(/size(arg1)/)), &
01005                      reshape(arg2,(/size(arg2)/)), &
```

```
01006                      fd1=out, grid=handle_rx%grid, &
01007                      data_error=data_error)
01008                if(present(error_status)) then
01009                   error_status%flag = data_error%ierr .ne. 0
01010                   error_status%string = data_error%cerr
01011                else
01012                   if(data_error%ierr.ne.0) then
01013                      write(*,*) 'interpol returned an error: ', data_error%ierr,
     trim(data_error%cerr)
01014                      call errorstop('interpol returned an error')
01015                   endif
01016                endif
01017             else
01018                write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
01019             endif
01020          else
01021             if(handle_rx%filled) then
01022                call interpol( &
01023                     reshape(arg1,(/size(arg1)/)), &
01024                     fd1=out, grid=handle_rx%grid, &
01025                     data_error=data_error)
01026                if(present(error_status)) then
01027                   error_status%flag = data_error%ierr .ne. 0
01028                   error_status%string = data_error%cerr
01029                else
01030                   if(data_error%ierr.ne.0) then
01031                      write(*,*) 'interpol returned an error: ', data_error%ierr,
     trim(data_error%cerr)
01032                      call errorstop('interpol returned an error')
01033                   endif
01034                endif
01035             else
01036                write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
01037             endif
01038          endif
01039       endif
01040    endif
01041    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1 end'
01042
```

15.44.2.3 imas_amns_rx_2()

```
subroutine amns_module::imas_amns_rx::imas_amns_rx_2 (
             type(amns_handle_rx_type), intent(inout) handle_rx,
             real (kind=ids_real), dimension(:,:), intent(out) out,
             real (kind=ids_real), dimension(:,:), intent(in) arg1,
             real (kind=ids_real), dimension(:,:), intent(in), optional arg2,
             real (kind=ids_real), dimension(:,:), intent(in), optional arg3,
             real (kind=ids_real), dimension(:,:), intent(in), optional arg4,
             type(amns_error_type), intent(out), optional error_status )
```

get the rates associated with the input (2d) args for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system (2d array)
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; (2d array) of the same size as out
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; (2d array) of the same size as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; (2d array) of the same size as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; (2d array) of the same size as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

Definition at line 1054 of file amns_module.f90.

```
01055     use ids_types  ! IGNORE
01056     use amns_types
01057     use data_suport
```

```
01058     implicit none
01059     optional arg2,arg3,arg4,error_status
01060     type(amns_handle_rx_type), intent(inout) :: handle_rx
01061     real (kind=ids_real), intent(out) :: out(:,:)
01062     real (kind=ids_real), intent(in) :: arg1(:,:),arg2(:,:),arg3(:,:),arg4(:,:)
01063     type(amns_error_type), intent(out) :: error_status
01064
01065     real (kind=ids_real) :: tmp_out(size(out))
01066
01067     type(data_error_t) :: data_error
01068
01069     if(present(error_status)) then
01070         error_status%flag=.false.
01071         error_status%string="No error"
01072     end if
01073
01074     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2 start'
01075     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(out) = ',lbound(out),ubound(out)
01076     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg1) = ',lbound(arg1),ubound(arg1)
01077     if(present(arg4)) then
01078        if(present(arg3).and.present(arg2)) then
01079           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) = ',lbound(arg2),ubound(arg2)
01080           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg3) = ',lbound(arg3),ubound(arg3)
01081           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg4) = ',lbound(arg4),ubound(arg4)
01082           if(handle_rx%filled) then
01083              call interpol( &
01084                   reshape(arg1,(/size(arg1)/)), &
01085                   reshape(arg2,(/size(arg2)/)), &
01086                   reshape(arg3,(/size(arg3)/)), &
01087                   reshape(arg4,(/size(arg4)/)), &
01088                   fd1=tmp_out, grid=handle_rx%grid, &
01089                   data_error=data_error)
01090              if(present(error_status)) then
01091                 error_status%flag = data_error%ierr .ne. 0
01092                 error_status%string = data_error%cerr
01093              else
01094                 if(data_error%ierr.ne.0) then
01095                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
01096                    call errorstop('interpol returned an error')
01097                 endif
01098              endif
01099              out=reshape(tmp_out,shape(out))
01100           else
01101              write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01102           endif
01103        else
01104           write(*,*) 'IMAS_AMNS_RX_2: arg4 present but not arg2/arg3!'
01105        endif
01106     else
01107        if(present(arg3)) then
01108           if(present(arg2)) then
01109              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01110              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg3) =
      ',lbound(arg3),ubound(arg3)
01111              if(handle_rx%filled) then
01112                 call interpol( &
01113                      reshape(arg1,(/size(arg1)/)), &
01114                      reshape(arg2,(/size(arg2)/)), &
01115                      reshape(arg3,(/size(arg3)/)), &
01116                      fd1=tmp_out, grid=handle_rx%grid, &
01117                      data_error=data_error)
01118                 if(present(error_status)) then
01119                    error_status%flag = data_error%ierr .ne. 0
01120                    error_status%string = data_error%cerr
01121                 else
01122                    if(data_error%ierr.ne.0) then
01123                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01124                       call errorstop('interpol returned an error')
01125                    endif
01126                 endif
01127                 out=reshape(tmp_out,shape(out))
01128              else
01129                 write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01130              endif
01131           else
01132              write(*,*) 'IMAS_AMNS_RX_2: arg3 present but not arg2!'
01133           endif
01134        else
01135           if(present(arg2)) then
01136              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01137              if(handle_rx%filled) then
01138                 call interpol( &
01139                      reshape(arg1,(/size(arg1)/)), &
01140                      reshape(arg2,(/size(arg2)/)), &
```

```
01141                      fd1=tmp_out, grid=handle_rx%grid, &
01142                      data_error=data_error)
01143                if(present(error_status)) then
01144                   error_status%flag = data_error%ierr .ne. 0
01145                   error_status%string = data_error%cerr
01146                else
01147                   if(data_error%ierr.ne.0) then
01148                      write(*,*) 'interpol returned an error: ', data_error%ierr,
     trim(data_error%cerr)
01149                      call errorstop('interpol returned an error')
01150                   endif
01151                endif
01152                out=reshape(tmp_out,shape(out))
01153             else
01154                write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01155             endif
01156          else
01157             if(handle_rx%filled) then
01158                call interpol( &
01159                     reshape(arg1,(/size(arg1)/)), &
01160                     fd1=tmp_out, grid=handle_rx%grid, &
01161                     data_error=data_error)
01162                if(present(error_status)) then
01163                   error_status%flag = data_error%ierr .ne. 0
01164                   error_status%string = data_error%cerr
01165                else
01166                   if(data_error%ierr.ne.0) then
01167                      write(*,*) 'interpol returned an error: ', data_error%ierr,
     trim(data_error%cerr)
01168                      call errorstop('interpol returned an error')
01169                   endif
01170                endif
01171                out=reshape(tmp_out,shape(out))
01172             else
01173                write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01174             endif
01175          endif
01176       endif
01177    endif
01178    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2 end'
01179
```

15.44.2.4 imas_amns_rx_3()

```
subroutine amns_module::imas_amns_rx::imas_amns_rx_3 (
             type(amns_handle_rx_type), intent(inout) handle_rx,
             real (kind=ids_real), dimension(:,:,:), intent(out) out,
             real (kind=ids_real), dimension(:,:,:), intent(in) arg1,
             real (kind=ids_real), dimension(:,:,:), intent(in), optional arg2,
             real (kind=ids_real), dimension(:,:,:), intent(in), optional arg3,
             real (kind=ids_real), dimension(:,:,:), intent(in), optional arg4,
             type(amns_error_type), intent(out), optional error_status )
```

get the rates associated with the input (3d) args for a particular reaction

Parameters

in	*handle_rx*	Opaque handle from call to IMAS_AMNS_SETUP_TABLE
out	*out*	Results from the AMNS system (3d array)
in	*arg1*	Output to be evaluated at the points specified by this argument, possibly in combination with arg2, arg3 and arg4; (3d array) of the same size as out
in	*arg2*	Output to be evaluated at the points specified by this argument, in combination with arg1 and possibly arg3 and arg4; (3d array) of the same size as out (optional)
in	*arg3*	Output to be evaluated at the points specified by this argument, in combination with arg1 and arg2 and possibly arg4; (3d array) of the same size as out (optional)
in	*arg4*	Output to be evaluated at the points specified by this argument, in combination with arg1, arg2 and arg3; (3d array) of the same size as out (optional)
out	*error_status*	If an error occurs, this will be set (optional)

Definition at line 1191 of file amns_module.f90.

```
01192     use ids_types  ! IGNORE
```

```
01193     use amns_types
01194     use data_suport
01195     implicit none
01196     optional arg2,arg3,arg4,error_status
01197     type(amns_handle_rx_type), intent(inout) :: handle_rx
01198     real (kind=ids_real), intent(out) :: out(:,:,:)
01199     real (kind=ids_real), intent(in) :: arg1(:,:,:),arg2(:,:,:),arg3(:,:,:),arg4(:,:,:)
01200     type(amns_error_type), intent(out) :: error_status
01201
01202     real (kind=ids_real) :: tmp_out(size(out))
01203
01204     type(data_error_t) :: data_error
01205
01206     if(present(error_status)) then
01207         error_status%flag=.false.
01208         error_status%string="No error"
01209     end if
01210
01211     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3 start'
01212     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(out) = ',lbound(out),ubound(out)
01213     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg1) = ',lbound(arg1),ubound(arg1)
01214     if(present(arg4)) then
01215        if(present(arg3).and.present(arg2)) then
01216           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) = ',lbound(arg2),ubound(arg2)
01217           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg3) = ',lbound(arg3),ubound(arg3)
01218           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg4) = ',lbound(arg4),ubound(arg4)
01219           if(handle_rx%filled) then
01220              call interpol( &
01221                   reshape(arg1,(/size(arg1)/)), &
01222                   reshape(arg2,(/size(arg2)/)), &
01223                   reshape(arg3,(/size(arg3)/)), &
01224                   reshape(arg4,(/size(arg4)/)), &
01225                   fd1=tmp_out, grid=handle_rx%grid, &
01226                   data_error=data_error)
01227              if(present(error_status)) then
01228                 error_status%flag = data_error%ierr .ne. 0
01229                 error_status%string = data_error%cerr
01230              else
01231                 if(data_error%ierr.ne.0) then
01232                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
01233                    call errorstop('interpol returned an error')
01234                 endif
01235              endif
01236              out=reshape(tmp_out,shape(out))
01237           else
01238              write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01239           endif
01240        else
01241           write(*,*) 'IMAS_AMNS_RX_3: arg4 present but not arg2/arg3!'
01242        endif
01243     else
01244        if(present(arg3)) then
01245           if(present(arg2)) then
01246              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01247              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg3) =
      ',lbound(arg3),ubound(arg3)
01248              if(handle_rx%filled) then
01249                 call interpol( &
01250                      reshape(arg1,(/size(arg1)/)), &
01251                      reshape(arg2,(/size(arg2)/)), &
01252                      reshape(arg3,(/size(arg3)/)), &
01253                      fd1=tmp_out, grid=handle_rx%grid, &
01254                      data_error=data_error)
01255                 if(present(error_status)) then
01256                    error_status%flag = data_error%ierr .ne. 0
01257                    error_status%string = data_error%cerr
01258                 else
01259                    if(data_error%ierr.ne.0) then
01260                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01261                       call errorstop('interpol returned an error')
01262                    endif
01263                 endif
01264                 out=reshape(tmp_out,shape(out))
01265              else
01266                 write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01267              endif
01268           else
01269              write(*,*) 'IMAS_AMNS_RX_3: arg3 present but not arg2!'
01270           endif
01271        else
01272           if(present(arg2)) then
01273              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01274              if(handle_rx%filled) then
01275                 call interpol( &
```

```
01276                     reshape(arg1,(/size(arg1)/)), &
01277                     reshape(arg2,(/size(arg2)/)), &
01278                     fd1=tmp_out, grid=handle_rx%grid, &
01279                     data_error=data_error)
01280                if(present(error_status)) then
01281                   error_status%flag = data_error%ierr .ne. 0
01282                   error_status%string = data_error%cerr
01283                else
01284                   if(data_error%ierr.ne.0) then
01285                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01286                      call errorstop('interpol returned an error')
01287                   endif
01288                endif
01289                out=reshape(tmp_out,shape(out))
01290             else
01291                write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01292             endif
01293          else
01294             if(handle_rx%filled) then
01295                call interpol( &
01296                     reshape(arg1,(/size(arg1)/)), &
01297                     fd1=tmp_out, grid=handle_rx%grid, &
01298                     data_error=data_error)
01299                if(present(error_status)) then
01300                   error_status%flag = data_error%ierr .ne. 0
01301                   error_status%string = data_error%cerr
01302                else
01303                   if(data_error%ierr.ne.0) then
01304                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01305                      call errorstop('interpol returned an error')
01306                   endif
01307                endif
01308                out=reshape(tmp_out,shape(out))
01309             else
01310                write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01311             endif
01312          endif
01313       endif
01314    endif
01315    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3 end'
01316
```

The documentation for this interface was generated from the following file:

- src/libamns/amns_module.f90

15.45 m_mrgrnk::mrgrnk Interface Reference

Public Member Functions

- subroutine d_mrgrnk (XDONT, IRNGT)
- subroutine r_mrgrnk (XDONT, IRNGT)
- subroutine i_mrgrnk (XDONT, IRNGT)

15.45.1 Detailed Description

Definition at line 11 of file m_mrgrnk.f90.

15.45.2 Member Function/Subroutine Documentation

15.45.2.1 d_mrgrnk()

```
subroutine m_mrgrnk::mrgrnk::d_mrgrnk (
             real (kind=kdp), dimension (:), intent(in) XDONT,
             integer, dimension (:), intent(out) IRNGT )
```

Definition at line 16 of file m_mrgrnk.f90.

```
00017 ! ____________________________________________________________
00018 !   MRGRNK = Merge-sort ranking of an array
00019 !   For performance reasons, the first 2 passes are taken
00020 !   out of the standard loop, and use dedicated coding.
00021 ! ____________________________________________________________
```

```
00022 ! ____________________________________________________________
00023       Real (kind=kdp), Dimension (:), Intent (In) :: xdont
00024       Integer, Dimension (:), Intent (Out) :: IRNGT
00025 ! ____________________________________________________________
00026       Real (kind=kdp) :: xvala, xvalb
00027 !
00028       Integer, Dimension (SIZE(IRNGT)) :: JWRKT
00029       Integer :: LMTNA, LMTNC, IRNG1, IRNG2
00030       Integer :: NVAL, IIND, IWRKD, IWRK, IWRKF, JINDA, IINDA, IINDB
00031 !
00032       nval = min(SIZE(xdont), SIZE(irngt))
00033       Select Case (nval)
00034       Case (:0)
00035          Return
00036       Case (1)
00037          irngt(1) = 1
00038          Return
00039       Case Default
00040          Continue
00041       End Select
00042 !
00043 !  Fill-in the index array, creating ordered couples
00044 !
00045       Do iind = 2, nval, 2
00046          If (xdont(iind-1) <= xdont(iind)) Then
00047             irngt(iind-1) = iind - 1
00048             irngt(iind) = iind
00049          Else
00050             irngt(iind-1) = iind
00051             irngt(iind) = iind - 1
00052          End If
00053       End Do
00054       If (modulo(nval, 2) /= 0) Then
00055          irngt(nval) = nval
00056       End If
00057 !
00058 !  We will now have ordered subsets A - B - A - B - ...
00059 !  and merge A and B couples into     C   -   C   - ...
00060 !
00061       lmtna = 2
00062       lmtnc = 4
00063 !
00064 !  First iteration. The length of the ordered subsets goes from 2 to 4
00065 !
00066       Do
00067          If (nval <= 2) Exit
00068 !
00069 !   Loop on merges of A and B into C
00070 !
00071          Do iwrkd = 0, nval - 1, 4
00072             If ((iwrkd+4) > nval) Then
00073                If ((iwrkd+2) >= nval) Exit
00074 !
00075 !   1 2 3
00076 !
00077                If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) Exit
00078 !
00079 !   1 3 2
00080 !
00081                If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00082                   irng2 = irngt(iwrkd+2)
00083                   irngt(iwrkd+2) = irngt(iwrkd+3)
00084                   irngt(iwrkd+3) = irng2
00085 !
00086 !   3 1 2
00087 !
00088                Else
00089                   irng1 = irngt(iwrkd+1)
00090                   irngt(iwrkd+1) = irngt(iwrkd+3)
00091                   irngt(iwrkd+3) = irngt(iwrkd+2)
00092                   irngt(iwrkd+2) = irng1
00093                End If
00094                Exit
00095             End If
00096 !
00097 !   1 2 3 4
00098 !
00099             If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) cycle
00100 !
00101 !   1 3 x x
00102 !
00103             If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00104                irng2 = irngt(iwrkd+2)
00105                irngt(iwrkd+2) = irngt(iwrkd+3)
00106                If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00107 !   1 3 2 4
00108                   irngt(iwrkd+3) = irng2
```

```
00109                Else
00110 !   1 3 4 2
00111                   irngt(iwrkd+3) = irngt(iwrkd+4)
00112                   irngt(iwrkd+4) = irng2
00113                End If
00114 !
00115 !   3 x x x
00116 !
00117             Else
00118                irng1 = irngt(iwrkd+1)
00119                irng2 = irngt(iwrkd+2)
00120                irngt(iwrkd+1) = irngt(iwrkd+3)
00121                If (xdont(irng1) <= xdont(irngt(iwrkd+4))) Then
00122                   irngt(iwrkd+2) = irng1
00123                   If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00124 !   3 1 2 4
00125                      irngt(iwrkd+3) = irng2
00126                   Else
00127 !   3 1 4 2
00128                      irngt(iwrkd+3) = irngt(iwrkd+4)
00129                      irngt(iwrkd+4) = irng2
00130                   End If
00131                Else
00132 !   3 4 1 2
00133                   irngt(iwrkd+2) = irngt(iwrkd+4)
00134                   irngt(iwrkd+3) = irng1
00135                   irngt(iwrkd+4) = irng2
00136                End If
00137             End If
00138          End Do
00139 !
00140 !  The Cs become As and Bs
00141 !
00142          lmtna = 4
00143          Exit
00144       End Do
00145 !
00146 !  Iteration loop. Each time, the length of the ordered subsets
00147 !  is doubled.
00148 !
00149       Do
00150          If (lmtna >= nval) Exit
00151          iwrkf = 0
00152          lmtnc = 2 * lmtnc
00153 !
00154 !   Loop on merges of A and B into C
00155 !
00156          Do
00157             iwrk = iwrkf
00158             iwrkd = iwrkf + 1
00159             jinda = iwrkf + lmtna
00160             iwrkf = iwrkf + lmtnc
00161             If (iwrkf >= nval) Then
00162                If (jinda >= nval) Exit
00163                iwrkf = nval
00164             End If
00165             iinda = 1
00166             iindb = jinda + 1
00167 !
00168 !   Shortcut for the case when the max of A is smaller
00169 !   than the min of B. This line may be activated when the
00170 !   initial set is already close to sorted.
00171 !
00172 !          IF (XDONT(IRNGT(JINDA)) <= XDONT(IRNGT(IINDB))) CYCLE
00173 !
00174 !  One steps in the C subset, that we build in the final rank array
00175 !
00176 !  Make a copy of the rank array for the merge iteration
00177 !
00178             jwrkt(1:lmtna) = irngt(iwrkd:jinda)
00179 !
00180             xvala = xdont(jwrkt(iinda))
00181             xvalb = xdont(irngt(iindb))
00182 !
00183             Do
00184                iwrk = iwrk + 1
00185 !
00186 !  We still have unprocessed values in both A and B
00187 !
00188                If (xvala > xvalb) Then
00189                   irngt(iwrk) = irngt(iindb)
00190                   iindb = iindb + 1
00191                   If (iindb > iwrkf) Then
00192 !  Only A still with unprocessed values
00193                      irngt(iwrk+1:iwrkf) = jwrkt(iinda:lmtna)
00194                      Exit
00195                   End If
```

```
00196                   xvalb = xdont(irngt(iindb))
00197                Else
00198                   irngt(iwrk) = jwrkt(iinda)
00199                   iinda = iinda + 1
00200                   If (iinda > lmtna) exit! Only B still with unprocessed values
00201                   xvala = xdont(jwrkt(iinda))
00202                End If
00203 !
00204             End Do
00205          End Do
00206 !
00207 !  The Cs become As and Bs
00208 !
00209          lmtna = 2 * lmtna
00210       End Do
00211 !
00212       Return
00213 !
```

15.45.2.2 i_mrgrnk()

```
subroutine m_mrgrnk::mrgrnk::i_mrgrnk (
             integer, dimension (:), intent(in) XDONT,
             integer, dimension (:), intent(out) IRNGT )
```

Definition at line 415 of file m_mrgrnk.f90.

```
00416 ! ____________________________________________________________
00417 !   MRGRNK = Merge-sort ranking of an array
00418 !   For performance reasons, the first 2 passes are taken
00419 !   out of the standard loop, and use dedicated coding.
00420 ! ____________________________________________________________
00421 ! ____________________________________________________________
00422       Integer, Dimension (:), Intent (In)  :: XDONT
00423       Integer, Dimension (:), Intent (Out) :: IRNGT
00424 ! ____________________________________________________________
00425       Integer :: XVALA, XVALB
00426 !
00427       Integer, Dimension (SIZE(IRNGT)) :: JWRKT
00428       Integer :: LMTNA, LMTNC, IRNG1, IRNG2
00429       Integer :: NVAL, IIND, IWRKD, IWRK, IWRKF, JINDA, IINDA, IINDB
00430 !
00431       nval = min(SIZE(xdont), SIZE(irngt))
00432       Select Case (nval)
00433       Case (:0)
00434          Return
00435       Case (1)
00436          irngt(1) = 1
00437          Return
00438       Case Default
00439          Continue
00440       End Select
00441 !
00442 !  Fill-in the index array, creating ordered couples
00443 !
00444       Do iind = 2, nval, 2
00445          If (xdont(iind-1) <= xdont(iind)) Then
00446             irngt(iind-1) = iind - 1
00447             irngt(iind) = iind
00448          Else
00449             irngt(iind-1) = iind
00450             irngt(iind) = iind - 1
00451          End If
00452       End Do
00453       If (modulo(nval, 2) /= 0) Then
00454          irngt(nval) = nval
00455       End If
00456 !
00457 !  We will now have ordered subsets A - B - A - B - ...
00458 !  and merge A and B couples into     C   -   C   - ...
00459 !
00460       lmtna = 2
00461       lmtnc = 4
00462 !
00463 !  First iteration. The length of the ordered subsets goes from 2 to 4
00464 !
00465       Do
00466          If (nval <= 2) Exit
00467 !
00468 !   Loop on merges of A and B into C
00469 !
00470          Do iwrkd = 0, nval - 1, 4
00471             If ((iwrkd+4) > nval) Then
00472                If ((iwrkd+2) >= nval) Exit
```

```
00473 !
00474 !   1 2 3
00475 !
00476                If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) Exit
00477 !
00478 !   1 3 2
00479 !
00480                If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00481                   irng2 = irngt(iwrkd+2)
00482                   irngt(iwrkd+2) = irngt(iwrkd+3)
00483                   irngt(iwrkd+3) = irng2
00484 !
00485 !   3 1 2
00486 !
00487                Else
00488                   irng1 = irngt(iwrkd+1)
00489                   irngt(iwrkd+1) = irngt(iwrkd+3)
00490                   irngt(iwrkd+3) = irngt(iwrkd+2)
00491                   irngt(iwrkd+2) = irng1
00492                End If
00493                Exit
00494             End If
00495 !
00496 !   1 2 3 4
00497 !
00498             If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) cycle
00499 !
00500 !   1 3 x x
00501 !
00502             If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00503                irng2 = irngt(iwrkd+2)
00504                irngt(iwrkd+2) = irngt(iwrkd+3)
00505                If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00506 !   1 3 2 4
00507                   irngt(iwrkd+3) = irng2
00508                Else
00509 !   1 3 4 2
00510                   irngt(iwrkd+3) = irngt(iwrkd+4)
00511                   irngt(iwrkd+4) = irng2
00512                End If
00513 !
00514 !   3 x x x
00515 !
00516             Else
00517                irng1 = irngt(iwrkd+1)
00518                irng2 = irngt(iwrkd+2)
00519                irngt(iwrkd+1) = irngt(iwrkd+3)
00520                If (xdont(irng1) <= xdont(irngt(iwrkd+4))) Then
00521                   irngt(iwrkd+2) = irng1
00522                   If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00523 !   3 1 2 4
00524                      irngt(iwrkd+3) = irng2
00525                   Else
00526 !   3 1 4 2
00527                      irngt(iwrkd+3) = irngt(iwrkd+4)
00528                      irngt(iwrkd+4) = irng2
00529                   End If
00530                Else
00531 !   3 4 1 2
00532                   irngt(iwrkd+2) = irngt(iwrkd+4)
00533                   irngt(iwrkd+3) = irng1
00534                   irngt(iwrkd+4) = irng2
00535                End If
00536             End If
00537          End Do
00538 !
00539 !  The Cs become As and Bs
00540 !
00541          lmtna = 4
00542          Exit
00543       End Do
00544 !
00545 !  Iteration loop. Each time, the length of the ordered subsets
00546 !  is doubled.
00547 !
00548       Do
00549          If (lmtna >= nval) Exit
00550          iwrkf = 0
00551          lmtnc = 2 * lmtnc
00552 !
00553 !   Loop on merges of A and B into C
00554 !
00555          Do
00556             iwrk = iwrkf
00557             iwrkd = iwrkf + 1
00558             jinda = iwrkf + lmtna
00559             iwrkf = iwrkf + lmtnc
```

```
00560             If (iwrkf >= nval) Then
00561                If (jinda >= nval) Exit
00562                iwrkf = nval
00563             End If
00564             iinda = 1
00565             iindb = jinda + 1
00566 !
00567 !   Shortcut for the case when the max of A is smaller
00568 !   than the min of B. This line may be activated when the
00569 !   initial set is already close to sorted.
00570 !
00571 !          IF (XDONT(IRNGT(JINDA)) <= XDONT(IRNGT(IINDB))) CYCLE
00572 !
00573 !  One steps in the C subset, that we build in the final rank array
00574 !
00575 !  Make a copy of the rank array for the merge iteration
00576 !
00577              jwrkt(1:lmtna) = irngt(iwrkd:jinda)
00578 !
00579             xvala = xdont(jwrkt(iinda))
00580             xvalb = xdont(irngt(iindb))
00581 !
00582             Do
00583                iwrk = iwrk + 1
00584 !
00585 !  We still have unprocessed values in both A and B
00586 !
00587                If (xvala > xvalb) Then
00588                   irngt(iwrk) = irngt(iindb)
00589                   iindb = iindb + 1
00590                   If (iindb > iwrkf) Then
00591 !  Only A still with unprocessed values
00592                      irngt(iwrk+1:iwrkf) = jwrkt(iinda:lmtna)
00593                      Exit
00594                   End If
00595                   xvalb = xdont(irngt(iindb))
00596                Else
00597                   irngt(iwrk) = jwrkt(iinda)
00598                   iinda = iinda + 1
00599                   If (iinda > lmtna) exit! Only B still with unprocessed values
00600                   xvala = xdont(jwrkt(iinda))
00601                End If
00602 !
00603             End Do
00604          End Do
00605 !
00606 !  The Cs become As and Bs
00607 !
00608          lmtna = 2 * lmtna
00609       End Do
00610 !
00611       Return
00612 !
```

15.45.2.3 r_mrgrnk()

```
subroutine m_mrgrnk::mrgrnk::r_mrgrnk (
             real, dimension (:), intent(in) XDONT,
             integer, dimension (:), intent(out) IRNGT )
```

Definition at line 216 of file m_mrgrnk.f90.

```
00217 ! ____________________________________________________________
00218 !   MRGRNK = Merge-sort ranking of an array
00219 !   For performance reasons, the first 2 passes are taken
00220 !   out of the standard loop, and use dedicated coding.
00221 ! ____________________________________________________________
00222 ! ____________________________________________________________
00223       Real, Dimension (:), Intent (In) :: XDONT
00224       Integer, Dimension (:), Intent (Out) :: IRNGT
00225 ! ____________________________________________________________
00226       Real :: XVALA, XVALB
00227 !
00228       Integer, Dimension (SIZE(IRNGT)) :: JWRKT
00229       Integer :: LMTNA, LMTNC, IRNG1, IRNG2
00230       Integer :: NVAL, IIND, IWRKD, IWRK, IWRKF, JINDA, IINDA, IINDB
00231 !
00232       nval = min(SIZE(xdont), SIZE(irngt))
00233       Select Case (nval)
00234       Case (:0)
00235          Return
00236       Case (1)
00237          irngt(1) = 1
00238          Return
```

```
00239       Case Default
00240          Continue
00241       End Select
00242 !
00243 !  Fill-in the index array, creating ordered couples
00244 !
00245       Do iind = 2, nval, 2
00246          If (xdont(iind-1) <= xdont(iind)) Then
00247             irngt(iind-1) = iind - 1
00248             irngt(iind) = iind
00249          Else
00250             irngt(iind-1) = iind
00251             irngt(iind) = iind - 1
00252          End If
00253       End Do
00254       If (modulo(nval, 2) /= 0) Then
00255          irngt(nval) = nval
00256       End If
00257 !
00258 !  We will now have ordered subsets A - B - A - B - ...
00259 !  and merge A and B couples into     C   -   C   - ...
00260 !
00261       lmtna = 2
00262       lmtnc = 4
00263 !
00264 !  First iteration. The length of the ordered subsets goes from 2 to 4
00265 !
00266       Do
00267          If (nval <= 2) Exit
00268 !
00269 !   Loop on merges of A and B into C
00270 !
00271          Do iwrkd = 0, nval - 1, 4
00272             If ((iwrkd+4) > nval) Then
00273                If ((iwrkd+2) >= nval) Exit
00274 !
00275 !   1 2 3
00276 !
00277                If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) Exit
00278 !
00279 !   1 3 2
00280 !
00281                If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00282                   irng2 = irngt(iwrkd+2)
00283                   irngt(iwrkd+2) = irngt(iwrkd+3)
00284                   irngt(iwrkd+3) = irng2
00285 !
00286 !   3 1 2
00287 !
00288                Else
00289                   irng1 = irngt(iwrkd+1)
00290                   irngt(iwrkd+1) = irngt(iwrkd+3)
00291                   irngt(iwrkd+3) = irngt(iwrkd+2)
00292                   irngt(iwrkd+2) = irng1
00293                End If
00294                Exit
00295             End If
00296 !
00297 !   1 2 3 4
00298 !
00299             If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) cycle
00300 !
00301 !   1 3 x x
00302 !
00303             If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00304                irng2 = irngt(iwrkd+2)
00305                irngt(iwrkd+2) = irngt(iwrkd+3)
00306                If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00307 !   1 3 2 4
00308                   irngt(iwrkd+3) = irng2
00309                Else
00310 !   1 3 4 2
00311                   irngt(iwrkd+3) = irngt(iwrkd+4)
00312                   irngt(iwrkd+4) = irng2
00313                End If
00314 !
00315 !   3 x x x
00316 !
00317             Else
00318                irng1 = irngt(iwrkd+1)
00319                irng2 = irngt(iwrkd+2)
00320                irngt(iwrkd+1) = irngt(iwrkd+3)
00321                If (xdont(irng1) <= xdont(irngt(iwrkd+4))) Then
00322                   irngt(iwrkd+2) = irng1
00323                   If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00324 !   3 1 2 4
00325                      irngt(iwrkd+3) = irng2
```

```
00326                   Else
00327 !   3 1 4 2
00328                      irngt(iwrkd+3) = irngt(iwrkd+4)
00329                      irngt(iwrkd+4) = irng2
00330                   End If
00331                Else
00332 !   3 4 1 2
00333                   irngt(iwrkd+2) = irngt(iwrkd+4)
00334                   irngt(iwrkd+3) = irng1
00335                   irngt(iwrkd+4) = irng2
00336                End If
00337             End If
00338          End Do
00339 !
00340 !  The Cs become As and Bs
00341 !
00342          lmtna = 4
00343          Exit
00344       End Do
00345 !
00346 !  Iteration loop. Each time, the length of the ordered subsets
00347 !  is doubled.
00348 !
00349       Do
00350          If (lmtna >= nval) Exit
00351          iwrkf = 0
00352          lmtnc = 2 * lmtnc
00353 !
00354 !   Loop on merges of A and B into C
00355 !
00356          Do
00357             iwrk = iwrkf
00358             iwrkd = iwrkf + 1
00359             jinda = iwrkf + lmtna
00360             iwrkf = iwrkf + lmtnc
00361             If (iwrkf >= nval) Then
00362                If (jinda >= nval) Exit
00363                iwrkf = nval
00364             End If
00365             iinda = 1
00366             iindb = jinda + 1
00367 !
00368 !   Shortcut for the case when the max of A is smaller
00369 !   than the min of B. This line may be activated when the
00370 !   initial set is already close to sorted.
00371 !
00372 !          IF (XDONT(IRNGT(JINDA)) <= XDONT(IRNGT(IINDB))) CYCLE
00373 !
00374 !  One steps in the C subset, that we build in the final rank array
00375 !
00376 !  Make a copy of the rank array for the merge iteration
00377 !
00378             jwrkt(1:lmtna) = irngt(iwrkd:jinda)
00379 !
00380             xvala = xdont(jwrkt(iinda))
00381             xvalb = xdont(irngt(iindb))
00382 !
00383             Do
00384                iwrk = iwrk + 1
00385 !
00386 !  We still have unprocessed values in both A and B
00387 !
00388                If (xvala > xvalb) Then
00389                   irngt(iwrk) = irngt(iindb)
00390                   iindb = iindb + 1
00391                   If (iindb > iwrkf) Then
00392 !  Only A still with unprocessed values
00393                      irngt(iwrk+1:iwrkf) = jwrkt(iinda:lmtna)
00394                      Exit
00395                   End If
00396                   xvalb = xdont(irngt(iindb))
00397                Else
00398                   irngt(iwrk) = jwrkt(iinda)
00399                   iinda = iinda + 1
00400                   If (iinda > lmtna) exit! Only B still with unprocessed values
00401                   xvala = xdont(jwrkt(iinda))
00402                End If
00403 !
00404             End Do
00405          End Do
00406 !
00407 !  The Cs become As and Bs
00408 !
00409          lmtna = 2 * lmtna
00410       End Do
00411 !
00412       Return
```

```
00413 !
```

The documentation for this interface was generated from the following file:

- src/libamns/m_mrgrnk.f90

15.46 strings::operator(//) Interface Reference

Public Member Functions

- function print_int_r (char, zahl)
- function print_int_l (zahl, char)
- character(len=len(char)+22) function print_dble_r (char, zahl)
- character(len=len(char)+22) function print_dble_l (zahl, char)
- character(len=len(char)+12) function print_real_r (char, zahl)
- character(len=len(char)+12) function print_real_l (zahl, char)

15.46.1 Detailed Description

Definition at line 26 of file strings.f90.

15.46.2 Member Function/Subroutine Documentation

15.46.2.1 print_dble_l()

```
character(len=len(char)+22) function strings::operator(//)::print_dble_l (
            real(rkind), intent(in) zahl,
            character(len=*), intent(in) char )
```

Definition at line 83 of file strings.f90.

```
00084     character(len=*), intent(in) :: char
00085     real(rkind) , intent(in) :: zahl
00086     character(len=len(char)+22)::string
00087
00088     write(string,'(1p,e22.15)')zahl
00089     string=char//trim(adjustl(string))
00090
```

15.46.2.2 print_dble_r()

```
character(len=len(char)+22) function strings::operator(//)::print_dble_r (
            character(len=*), intent(in) char,
            real(rkind), intent(in) zahl )
```

Definition at line 71 of file strings.f90.

```
00072     character(len=*), intent(in) :: char
00073     real(rkind) , intent(in) :: zahl
00074     character(len=len(char)+22)::string
00075
00076     write(string,'(1p,e22.15)')zahl
00077     string=char//trim(adjustl(string))
00078
```

15.46.2.3 print_int_l()

```
function strings::operator(//)::print_int_l (
            integer(ikind), intent(in) zahl,
            character(len=*), intent(in) char )
```

Definition at line 59 of file strings.f90.

```
00060     character(len=*), intent(in) :: char
00061     integer(ikind) , intent(in) :: zahl
00062     character(len=len(char)+int(log10(dble(max(1,zahl))))+ &
00063          1-floor(sign(0.1d0,dble(zahl)))) :: string
00064
```

```
00065     write(string,'(i0)')zahl
00066     string=trim(adjustl(string))//char
```

15.46.2.4 print_int_r()

```
function strings::operator(//)::print_int_r (
             character(len=*), intent(in) char,
             integer(ikind), intent(in) zahl )
```

Definition at line 45 of file strings.f90.

```
00046
00047     character(len=*), intent(in) :: char
00048     integer(ikind) , intent(in) :: zahl
00049     character(len=len(char)+int(log10(dble(max(1,zahl))))+ &
00050          1-floor(sign(0.1d0,dble(zahl))))::string
00051
00052     write(string,'(i0)')zahl
00053     string=char//trim(adjustl(string))
00054
```

15.46.2.5 print_real_l()

```
character(len=len(char)+12) function strings::operator(//)::print_real_l (
             real, intent(in) zahl,
             character(len=*), intent(in) char )
```

Definition at line 107 of file strings.f90.

```
00108     character(len=*), intent(in) :: char
00109     real , intent(in) :: zahl
00110     character(len=len(char)+12)::string
00111
00112     write(string,'(1p,e12.6)')dble(zahl)
00113     string=char//trim(adjustl(string))
00114
```

15.46.2.6 print_real_r()

```
character(len=len(char)+12) function strings::operator(//)::print_real_r (
             character(len=*), intent(in) char,
             real, intent(in) zahl )
```

Definition at line 95 of file strings.f90.

```
00096     character(len=*), intent(in) :: char
00097     real , intent(in) :: zahl
00098     character(len=len(char)+12)::string
00099
00100     write(string,'(1p,e12.6)')dble(zahl)
00101     string=char//trim(adjustl(string))
00102
```

The documentation for this interface was generated from the following file:

- src/libamns/strings.f90

15.47 amns.Reactants Class Reference

Public Member Functions

- def __cinit__ (self)
- def __init__ (self)
- def add (self, zn, za, mi, lr=None, real_specifier=None, int_specifier=None)
- def __len__ (self)
- def __str__ (self)
- def test (self)
- def value (self)

15.47.1 Detailed Description

Definition at line 658 of file amns.pyx.

15.47.2 Constructor & Destructor Documentation

15.47.2.1 __init__()

```
def amns.Reactants.__init__ (
              self )

Use to initialize the Reactants class
        in:
                None
        out:
                instance of the Reactants class
```

Definition at line 666 of file amns.pyx.

```
00666     def __init__(self):
00667         """Use to initialize the Reactants class
00668         in:
00669                 None
00670         out:
00671                 instance of the Reactants class
00672         """
00673
00674         self._handle = NULL
00675         self._reactants= []
00676         pass
00677
```

15.47.3 Member Function Documentation

15.47.3.1 __cinit__()

```
def amns.Reactants.__cinit__ (
              self )
```

Definition at line 663 of file amns.pyx.

```
00663     def __cinit__(self):
00664         pass
00665
```

15.47.3.2 __len__()

```
def amns.Reactants.__len__ (
              self )

Return the number of reactants/products
```

Definition at line 714 of file amns.pyx.

```
00714     def __len__(self):
00715         """Return the number of reactants/products"""
00716         return len(self._reactants)
00717
```

15.47.3.3 __str__()

```
def amns.Reactants.__str__ (
              self )

Return a string version of all the reactants/products
```

Definition at line 718 of file amns.pyx.

```
00718     def __str__(self):
00719         """Return a string version of all the reactants/products"""
00720         return str(self._reactants)
00721
```

15.47.3.4 add()

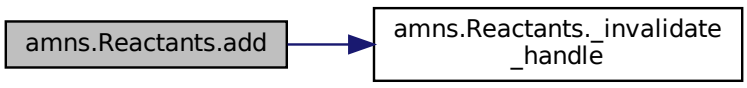
```
def amns.Reactants.add (
              self,
              zn,
              za,
              mi,
              lr = None,
              real_specifier = None,
              int_specifier = None )
```

```
Add to the list of reactants/products
        in:
                zn: nuclear charge (float, required)
                za: ion charge (float, required)
                mi: nuclear mass (float, required)
                lr: if 0, reactant; if 1, product (integer, default None)
                real_specifier: optional additional real (float, default None)
                int_specifier: optional additional integer (integer, default None)
        out:
                None
```

Definition at line 691 of file amns.pyx.

```
00691     def add(self, zn, za, mi, lr=None, real_specifier=None, int_specifier=None):
00692         """Add to the list of reactants/products
00693         in:
00694                 zn: nuclear charge (float, required)
00695                 za: ion charge (float, required)
00696                 mi: nuclear mass (float, required)
00697                 lr: if 0, reactant; if 1, product (integer, default None)
00698                 real_specifier: optional additional real (float, default None)
00699                 int_specifier: optional additional integer (integer, default None)
00700         out:
00701                 None
00702         """
00703         cdef camns_interface.amns_c_reactant_type r
00704         r = camns_interface.get_default_amns_c_reactant_type()
00705         self._invalidate_handle()
00706         r.ZN = zn
00707         r.ZA = za
00708         r.MI = mi
00709         if lr: r.LR = lr
00710         if real_specifier: r.real_specifier = real_specifier
00711         if int_specifier: r.int_specifier = int_specifier
00712         self._reactants.append(r)
00713
```

Here is the call graph for this function:



15.47.3.5 test()

```
def amns.Reactants.test (
              self )
```

Definition at line 722 of file amns.pyx.

```
00722     def test(self):
00723         cdef void* handle
00724         handle = self.handle()
00725
```

15.47.3.6 value()

```
def amns.Reactants.value (
              self )

Return all of the reactants/products
        in:
                None
        out:
                current list of reactants
```

Definition at line 726 of file amns.pyx.

```
00726     def value(self):
00727         """Return all of the reactants/products
00728         in:
00729                 None
00730         out:
00731                 current list of reactants
00732         """
00733         return self._reactants
00734
00735
```

The documentation for this class was generated from the following file:

- src/py/amns/amns.pyx

15.48 amns_utility::string Interface Reference

Public Member Functions

- character ∗24 function int_to_string (int)

  convert an integer to a string

15.48.1 Detailed Description

Definition at line 12 of file amns_utility.f90.

15.48.2 Member Function/Subroutine Documentation

15.48.2.1 int_to_string()

```
character*24 function amns_utility::string::int_to_string (
             integer, intent(in) int )
```

convert an integer to a string

Definition at line 23 of file amns_utility.f90.

```
00024     integer, intent(in) :: int
00025     write(int_to_string,'(I24)') int
00026     int_to_string=adjustl(int_to_string)
```

The documentation for this interface was generated from the following file:

- src/libamns/amns_utility.f90

15.49 amns.Table Class Reference

Public Member Functions

- def __init__ (self, reactionString, Reactants, reactants, Amns, parentAmns, isotope_resolved=None)

- def query (self, queryString)
- def set (self, setString)
- def finalize (self)
- def no_of_reactants (self)
- def ndim (self)
- def source (self)
- def provider (self)
- def citation (self)
- def filled (self)
- def reaction_type (self)
- def reactants (self)
- def coordinates (self)
- def version (self)
- def state_label (self)
- def result_unit (self)
- def result_label (self)
- def interp_fun (self)
- def prop_comment (self)
- def prop_source (self)
- def prop_provider (self)
- def prop_creation (self)
- def code_name (self)
- def code_commit (self)
- def code_version (self)
- def code_repository (self)
- def data (self, p1, p2=None, p3=None, p4=None)

Static Public Attributes

- string
- tmp_c_str = queryString.encode('UTF-8')

15.49.1 Detailed Description

Definition at line 209 of file amns.pyx.

15.49.2 Constructor & Destructor Documentation

15.49.2.1 __init__()

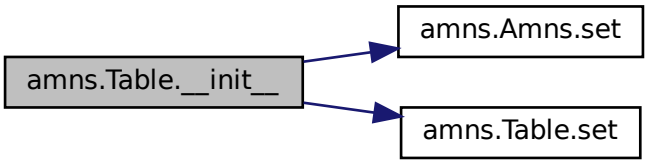
```
def amns.Table.__init__ (
              self,
              reactionString,
              Reactants,
              reactants,
              Amns,
              parentAmns,
              isotope_resolved = None )

Create a new table

        called by AMNS.get_table

        in:
                reactionString: name of reaction (string, required)
                reactants: reactants object (of type amns.Reactants, required)
```

```
                Amns: object of class Amns (of type Amns, required)
                isotope_resolved: 0 if not isotope resolved, 1 if isotope resolved (integer, default None)
        out:
                instance of Table class
```

Definition at line 215 of file amns.pyx.

```
00215     def __init__(self, reactionString, Reactants reactants, Amns parentAmns, isotope_resolved=None):
00216         """Create a new table
00217 
00218         called by AMNS.get_table
00219 
00220         in:
00221                 reactionString: name of reaction (string, required)
00222                 reactants: reactants object (of type amns.Reactants, required)
00223                 Amns: object of class Amns (of type Amns, required)
00224                 isotope_resolved: 0 if not isotope resolved, 1 if isotope resolved (integer, default
      None)
00225         out:
00226                 instance of Table class
00227         """
00228         cdef camns_interface.amns_c_error_type error_status
00229         cdef camns_interface.amns_c_reaction_type reaction
00230         reaction = camns_interface.get_default_amns_c_reaction_type()
00231         self._parentAmns = parentAmns
00232         if type(reactionString) == bytes:
00233             reaction.string = reactionString
00234         else:
00235             tmp_c_str = reactionString.encode('UTF-8')
00236             reaction.string = tmp_c_str
00237         if isotope_resolved : reaction.isotope_resolved = isotope_resolved
00238         camns_interface.IMAS_AMNS_CC_SETUP_TABLE(self._parentAmns.handle(),
00239                                                  &reaction,
00240                                                  reactants.handle(), &self._handle,
00241                                                  &error_status)
00242         if error_status.flag:
00243             raise AmnsException(error_status.string.decode('UTF-8'))
00244 
00245         self.set(b"nowarn")
00246 
00247 
```

Here is the call graph for this function:



15.49.3 Member Function Documentation

15.49.3.1 citation()

```
def amns.Table.citation (
              self )
```

Query for citation associated with the AMNS table

Definition at line 344 of file amns.pyx.

```
00344     def citation(self):
00345         """Query for citation associated with the AMNS table"""
00346         cdef camns_interface.amns_c_answer_type answer
00347         answer = self.lquery(b"citation")
00348         return answer.string.decode('UTF-8')
00349 
```

15.49.3.2 code_commit()

```
def amns.Table.code_commit (
              self )
```

Query for code_commit associated with the AMNS table

Definition at line 449 of file amns.pyx.

```
00449     def code_commit(self):
00450         """Query for code_commit associated with the AMNS table"""
00451         cdef camns_interface.amns_c_answer_type answer
00452         answer = self.lquery(b"code_commit")
00453         return answer.string.decode('UTF-8')
00454
```

15.49.3.3 code_name()

```
def amns.Table.code_name (
              self )
```

Query for code_name associated with the AMNS table

Definition at line 442 of file amns.pyx.

```
00442     def code_name(self):
00443         """Query for code_name associated with the AMNS table"""
00444         cdef camns_interface.amns_c_answer_type answer
00445         answer = self.lquery(b"code_name")
00446         return answer.string.decode('UTF-8')
00447
```

15.49.3.4 code_repository()

```
def amns.Table.code_repository (
              self )
```

Query for code_repository associated with the AMNS table

Definition at line 463 of file amns.pyx.

```
00463     def code_repository(self):
00464         """Query for code_repository associated with the AMNS table"""
00465         cdef camns_interface.amns_c_answer_type answer
00466         answer = self.lquery(b"code_repository")
00467         return answer.string.decode('UTF-8')
00468
```

15.49.3.5 code_version()

```
def amns.Table.code_version (
              self )
```

Query for code_version associated with the AMNS table

Definition at line 456 of file amns.pyx.

```
00456     def code_version(self):
00457         """Query for code_version associated with the AMNS table"""
00458         cdef camns_interface.amns_c_answer_type answer
00459         answer = self.lquery(b"code_version")
00460         return answer.string.decode('UTF-8')
00461
```

15.49.3.6 coordinates()

```
def amns.Table.coordinates (
              self )
```

Query for coordinates associated with the AMNS table

Definition at line 372 of file amns.pyx.

```
00372     def coordinates(self):
00373         """Query for coordinates associated with the AMNS table"""
00374         cdef camns_interface.amns_c_answer_type answer
00375         answer = self.lquery(b"coordinates")
00376         return answer.string.decode('UTF-8')
00377
```

15.49.3.7 data()

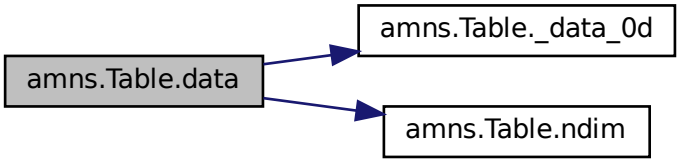
```
def amns.Table.data (
              self,
              p1,
              p2 = None,
              p3 = None,
              p4 = None )
```

```
Return the AMNS data based on the arguments
        in:
                p1: array of input values for argument 1 (float array, required)
                p2: array of input values for argument 2 (optional, float array of the same shape as p1)
                p3: array of input values for argument 3 (optional, float array of the same shape as p1)
                p4: array of input values for argument 4 (optional, float array of the same shape as p1)
        out:
                data evaluated (float array of the same shape as p1)
```

Definition at line 469 of file amns.pyx.

```
00469     def data(self, p1, p2 = None, p3 = None, p4 = None):
00470         """Return the AMNS data based on the arguments
00471         in:
00472                 p1: array of input values for argument 1 (float array, required)
00473                 p2: array of input values for argument 2 (optional, float array of the same shape as
      p1)
00474                 p3: array of input values for argument 3 (optional, float array of the same shape as
      p1)
00475                 p4: array of input values for argument 4 (optional, float array of the same shape as
      p1)
00476         out:
00477                 data evaluated (float array of the same shape as p1)
00478         """
00479         nargs = 1
00480         if p2 is not None: nargs += 1
00481         if p3 is not None: nargs += 1
00482         if p4 is not None: nargs += 1
00483         if nargs != self.ndim:
00484             raise AmnsException("Number of parameters does not match table dimensions (ndim="
00485                                 + str(self.ndim) + ")")
00486
00487         refShape = p1.shape
00488         if p2 is not None and p2.shape != refShape:
00489             raise AmnsException("Shape of parameter 2 inconsistent with shape of parameter 1")
00490         if p3 is not None and p3.shape != refShape:
00491             raise AmnsException("Shape of parameter 3 inconsistent with shape of parameter 1")
00492         if p4 is not None and p4.shape != refShape:
00493             raise AmnsException("Shape of parameter 4 inconsistent with shape of parameter 1")
00494
00495         if len(refShape) == 0:
00496             res = self._data_0d(nargs,p1,p2,p3,p4)
00497         elif len(refShape) == 1:
00498             res = self._data_1d(nargs,p1,p2,p3,p4)
00499         elif len(refShape) == 2:
00500             res = self._data_2d(nargs,p1,p2,p3,p4)
00501         elif len(refShape) == 3:
00502             res = self._data_3d(nargs,p1,p2,p3,p4)
00503         else:
00504             raise AmnsException("Unsupported rank of input arguments: %s" % (len(refShape),))
00505
00506         return res
00507
```

Here is the call graph for this function:



15.49.3.8 filled()

```
def amns.Table.filled (
              self )

Query for filled status associated with the AMNS table
```

Definition at line 351 of file amns.pyx.

```
00351     def filled(self):
00352         """Query for filled status associated with the AMNS table"""
00353         cdef camns_interface.amns_c_answer_type answer
00354         answer = self.lquery(b"filled")
00355         return answer.string.decode('UTF-8')
00356
```

15.49.3.9 finalize()

```
def amns.Table.finalize (
              self )

Provide the interface to IMAS_AMNS_FINISH_TABLE
        in:
                None
        out:
                None
```

Definition at line 303 of file amns.pyx.

```
00303     def finalize(self):
00304         """Provide the interface to IMAS_AMNS_FINISH_TABLE
00305         in:
00306                 None
00307         out:
00308                 None
00309         """
00310         cdef camns_interface.amns_c_error_type error_status
00311         camns_interface.IMAS_AMNS_CC_FINISH_TABLE(&self._handle, &error_status)
00312         if error_status.flag:
00313             raise AmnsException(error_status.string.decode('UTF-8'))
00314
```

15.49.3.10 interp_fun()

```
def amns.Table.interp_fun (
              self )

Query for interp_fun associated with the AMNS table
```

Definition at line 407 of file amns.pyx.

```
00407     def interp_fun(self):
00408         """Query for interp_fun associated with the AMNS table"""
00409         cdef camns_interface.amns_c_answer_type answer
00410         answer = self.lquery(b"interp_fun")
00411         return answer.number
00412
```

15.49.3.11 ndim()

```
def amns.Table.ndim (
               self )
```

Query for ndim associated with the AMNS table

Definition at line 323 of file amns.pyx.

```
00323     def ndim(self):
00324         """Query for ndim associated with the AMNS table"""
00325         cdef camns_interface.amns_c_answer_type answer
00326         answer = self.lquery(b"ndim")
00327         return answer.number
00328
```

Here is the caller graph for this function:

15.49.3.12 no_of_reactants()

```
def amns.Table.no_of_reactants (
               self )
```

Query for no_of_reactants associated with the AMNS table

Definition at line 316 of file amns.pyx.

```
00316     def no_of_reactants(self):
00317         """Query for no_of_reactants associated with the AMNS table"""
00318         cdef camns_interface.amns_c_answer_type answer
00319         answer = self.lquery(b"no_of_reactants")
00320         return answer.number
00321
```

15.49.3.13 prop_comment()

```
def amns.Table.prop_comment (
               self )
```

Query for prop_comment associated with the AMNS table

Definition at line 414 of file amns.pyx.

```
00414     def prop_comment(self):
00415         """Query for prop_comment associated with the AMNS table"""
00416         cdef camns_interface.amns_c_answer_type answer
00417         answer = self.lquery(b"prop_comment")
00418         return answer.string.decode('UTF-8')
00419
```

15.49.3.14 prop_creation()

```
def amns.Table.prop_creation (
              self )
```

Query for prop_creation associated with the AMNS table

Definition at line 435 of file amns.pyx.

```
00435     def prop_creation(self):
00436         """Query for prop_creation associated with the AMNS table"""
00437         cdef camns_interface.amns_c_answer_type answer
00438         answer = self.lquery(b"prop_creation")
00439         return answer.string.decode('UTF-8')
00440
```

15.49.3.15 prop_provider()

```
def amns.Table.prop_provider (
              self )
```

Query for prop_provider associated with the AMNS table

Definition at line 428 of file amns.pyx.

```
00428     def prop_provider(self):
00429         """Query for prop_provider associated with the AMNS table"""
00430         cdef camns_interface.amns_c_answer_type answer
00431         answer = self.lquery(b"prop_provider")
00432         return answer.string.decode('UTF-8')
00433
```

15.49.3.16 prop_source()

```
def amns.Table.prop_source (
              self )
```

Query for prop_source associated with the AMNS table

Definition at line 421 of file amns.pyx.

```
00421     def prop_source(self):
00422         """Query for prop_source associated with the AMNS table"""
00423         cdef camns_interface.amns_c_answer_type answer
00424         answer = self.lquery(b"prop_source")
00425         return answer.string.decode('UTF-8')
00426
```

15.49.3.17 provider()

```
def amns.Table.provider (
              self )
```

Query for provider associated with the AMNS table

Definition at line 337 of file amns.pyx.

```
00337     def provider(self):
00338         """Query for provider associated with the AMNS table"""
00339         cdef camns_interface.amns_c_answer_type answer
00340         answer = self.lquery(b"provider")
00341         return answer.string.decode('UTF-8')
00342
```

15.49.3.18 query()

```
def amns.Table.query (
              self,
              queryString )
```

```
Provide the interface to IMAS_AMNS_QUERY_TABLE
        in:
                queryString: string containing query (string, required)
        out:
                answer_string: string containing reply
                answer_number: number containing any numerical answer
```

Definition at line 262 of file amns.pyx.

```
00262     def query(self, queryString):
00263         """Provide the interface to IMAS_AMNS_QUERY_TABLE
00264         in:
00265                 queryString: string containing query (string, required)
00266         out:
00267                 answer_string: string containing reply
00268                 answer_number: number containing any numerical answer
00269 
00270         """
00271         cdef camns_interface.amns_c_query_type query
00272         cdef camns_interface.amns_c_answer_type answer
00273         cdef camns_interface.amns_c_error_type error_status
00274         if type(queryString) == bytes:
00275             query.string = queryString
00276         else:
00277             tmp_c_str = queryString.encode('UTF-8')
00278             query.string = tmp_c_str
00279         camns_interface.IMAS_AMNS_CC_QUERY_TABLE(self._handle, &query, &answer, &error_status);
00280         if error_status.flag:
00281             raise AmnsException(error_status.string.decode('UTF-8'))
00282         return answer.string.decode('UTF-8'), answer.number
00283 
```

15.49.3.19 reactants()

```
def amns.Table.reactants (
              self )
```

Query for reactants associated with the AMNS table

Definition at line 365 of file amns.pyx.

```
00365     def reactants(self):
00366         """Query for reactants associated with the AMNS table"""
00367         cdef camns_interface.amns_c_answer_type answer
00368         answer = self.lquery(b"reactants")
00369         return answer.string.decode('UTF-8')
00370 
```

15.49.3.20 reaction_type()

```
def amns.Table.reaction_type (
              self )
```

Query for reaction_type associated with the AMNS table

Definition at line 358 of file amns.pyx.

```
00358     def reaction_type(self):
00359         """Query for reaction_type associated with the AMNS table"""
00360         cdef camns_interface.amns_c_answer_type answer
00361         answer = self.lquery(b"reaction_type")
00362         return answer.string.decode('UTF-8')
00363 
```

15.49.3.21 result_label()

```
def amns.Table.result_label (
              self )
```

Query for result_label associated with the AMNS table

Definition at line 400 of file amns.pyx.

```
00400     def result_label(self):
00401         """Query for result_label associated with the AMNS table"""
```

```
00402         cdef camns_interface.amns_c_answer_type answer
00403         answer = self.lquery(b"result_label")
00404         return answer.string.decode('UTF-8')
00405
```

15.49.3.22 result_unit()

```
def amns.Table.result_unit (
              self )

Query for result_unit associated with the AMNS table
```

Definition at line 393 of file amns.pyx.

```
00393     def result_unit(self):
00394         """Query for result_unit associated with the AMNS table"""
00395         cdef camns_interface.amns_c_answer_type answer
00396         answer = self.lquery(b"result_unit")
00397         return answer.string.decode('UTF-8')
00398
```

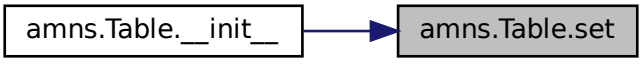
15.49.3.23 set()

```
def amns.Table.set (
              self,
              setString )

Provide the interface to IMAS_AMNS_SET_TABLE
        in:
                setString: string containing setting (string, required)
        out:
                None
```

Definition at line 284 of file amns.pyx.

```
00284     def set(self, setString):
00285         """Provide the interface to IMAS_AMNS_SET_TABLE
00286         in:
00287                 setString: string containing setting (string, required)
00288         out:
00289                 None
00290
00291         """
00292         cdef camns_interface.amns_c_set_type set
00293         cdef camns_interface.amns_c_error_type error_status
00294         if type(setString) == bytes:
00295             set.string = setString
00296         else:
00297             tmp_c_str = setString.encode('UTF-8')
00298             set.string = tmp_c_str
00299         camns_interface.IMAS_AMNS_CC_SET_TABLE(self._handle, &set, &error_status);
00300         if error_status.flag:
00301             raise AmnsException(error_status.string.decode('UTF-8'))
00302
```

Here is the caller graph for this function:



15.49.3.24 source()

```
def amns.Table.source (
              self )

Query for source associated with the AMNS table
```

Definition at line 330 of file amns.pyx.

```
00330     def source(self):
00331         """Query for source associated with the AMNS table"""
00332         cdef camns_interface.amns_c_answer_type answer
00333         answer = self.lquery(b"source")
00334         return answer.string.decode('UTF-8')
00335
```

15.49.3.25 state_label()

```
def amns.Table.state_label (
              self )

Query for state_label associated with the AMNS table
```

Definition at line 386 of file amns.pyx.

```
00386     def state_label(self):
00387         """Query for state_label associated with the AMNS table"""
00388         cdef camns_interface.amns_c_answer_type answer
00389         answer = self.lquery(b"state_label")
00390         return answer.string.decode('UTF-8')
00391
```

15.49.3.26 version()

```
def amns.Table.version (
              self )

Query for version associated with the AMNS table
```

Definition at line 379 of file amns.pyx.

```
00379     def version(self):
00380         """Query for version associated with the AMNS table"""
00381         cdef camns_interface.amns_c_answer_type answer
00382         answer = self.lquery(b"version")
00383         return answer.string.decode('UTF-8'), answer.number
00384
```

15.49.4 Member Data Documentation

15.49.4.1 string

```
amns.Table.string  [static]
```

Definition at line 253 of file amns.pyx.

15.49.4.2 tmp_c_str

```
amns.Table.tmp_c_str = queryString.encode('UTF-8')  [static]
```

Definition at line 255 of file amns.pyx.

The documentation for this class was generated from the following file:

- src/py/amns/amns.pyx

15.50 type_list_data_release Type Reference

Collaboration diagram for type_list_data_release:

Public Attributes

- type(ids_amns_data_data_entry), pointer data_entry => null()
- type(type_list_data_release), pointer next => null()

15.50.1 Detailed Description

Definition at line 70 of file amns_driver.f90.

15.50.2 Member Data Documentation

15.50.2.1 data_entry

```
type (ids_amns_data_data_entry), pointer type_list_data_release::data_entry => null()
```

Definition at line 71 of file amns_driver.f90.

```
00071      type (ids_amns_data_data_entry), pointer :: data_entry => null()
```

15.50.2.2 next

```
type (type_list_data_release), pointer type_list_data_release::next => null()
```

Definition at line 72 of file amns_driver.f90.

```
00072      type (type_list_data_release), pointer :: next => null()
```

The documentation for this type was generated from the following file:

- src/amns_driver/amns_driver.f90

Chapter 16

File Documentation

16.1 examples/coronal_f90/src/coronal.f90 File Reference

Functions/Subroutines

- program coronal

  demo program using the ITM AMNS interface

- subroutine solve_tridiag (a, b, c, v, x, n)

16.1.1 Function/Subroutine Documentation

16.1.1.1 coronal()

`program coronal`
demo program using the ITM AMNS interface
this program calculates the ionization/recombination equilibrium distribution of charge states

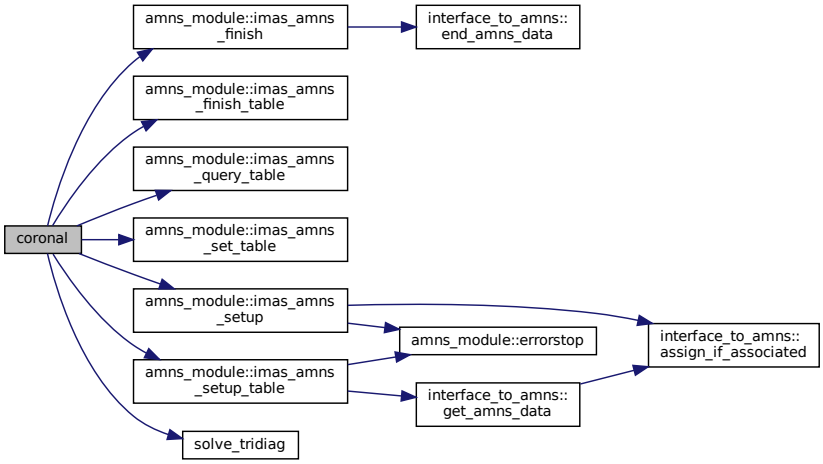
Author

David Coster

Version

"$Id: coronal.F90 463 2015-07-03 22:45:32Z dpc $"

Definition at line 10 of file coronal.f90.
Here is the call graph for this function:



16.1.1.2 solve_tridiag()

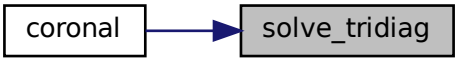
```
subroutine solve_tridiag (
             real(kind=ids_real), dimension(n), intent(in) a,
             real(kind=ids_real), dimension(n), intent(in) b,
             real(kind=ids_real), dimension(n), intent(in) c,
             real(kind=ids_real), dimension(n), intent(in) v,
             real(kind=ids_real), dimension(n), intent(out) x,
             integer, intent(in) n )
```

Definition at line 245 of file coronal.f90.

```
00246   use ids_types
00247   implicit none
00248 !       a - sub-diagonal (means it is the diagonal below the main diagonal)
00249 !       b - the main diagonal
00250 !       c - sup-diagonal (means it is the diagonal above the main diagonal)
00251 !       v - right part
00252 !       x - the answer
00253 !       n - number of equations
00254
00255   integer,intent(in) :: n
00256   real(kind=ids_real),dimension(n),intent(in) :: a,b,c,v
00257   real(kind=ids_real),dimension(n),intent(out) :: x
00258   real(kind=ids_real),dimension(n) :: bp,vp
00259   real(kind=ids_real) :: m
00260   integer i
00261
00262 ! Make copies of the b and v variables so that they are unaltered by this sub
00263   bp(1) = b(1)
00264   vp(1) = v(1)
00265
00266   !The first pass (setting coefficients):
00267   firstpass: do i = 2,n
00268      m = a(i)/bp(i-1)
00269      bp(i) = b(i) - m*c(i-1)
00270      vp(i) = v(i) - m*vp(i-1)
00271   end do firstpass
00272
00273   x(n) = vp(n)/bp(n)
00274   !The second pass (back-substition)
00275   backsub:do i = n-1, 1, -1
00276      x(i) = (vp(i) - c(i)*x(i+1))/bp(i)
```

```
00277   end do backsub
00278
```

Here is the caller graph for this function:



16.2 coronal.f90

```
00001
00009
00010 program coronal
00011   use ids_types
00012   use amns_types
00013   use amns_module
00014
00015   implicit none
00016
00017   integer, parameter :: nt = 120
00018   type (amns_handle_type) :: amns
00019   type (amns_handle_rx_type), allocatable :: amns_ei(:), amns_rc(:), amns_lr(:), amns_br(:)
00020   type (amns_reaction_type) :: xx_rx
00021   type (amns_reactants_type), allocatable :: species_m1(:), species_p1(:), species_cx(:), species(:)
00022   type (amns_query_type) :: query
00023   type (amns_answer_type) :: answer
00024   type (amns_set_type) :: set
00025   real (kind=ids_real), allocatable :: te(:), ne(:), na(:), rhs(:)
00026   real (kind=ids_real), allocatable :: rate_ei(:,:), rate_rc(:,:), rate_lr(:,:), rate_br(:,:)
00027   real (kind=ids_real), allocatable :: l(:), d(:), u(:)
00028   real (kind=ids_real) :: zn, mi
00029   real (kind=ids_real) :: test, norm, line_radiation, recombination_radiation
00030   character (len=12), allocatable :: state_labels(:)
00031   integer ns, is, it, isref
00032
00033   integer :: iargc
00034   character (len=256) :: arg
00035
00036 ! the command line can specify zn and am for the species to be modelled
00037   if(iargc().le.0) then
00038      zn=6
00039      mi=12
00040      write(*,*)
      '-------------------------------------------------------------------------------------------------------'
00041      write(*,*) 'Assumed carbon --- if not OK specify the nuclear change and atomic mass as arguemnts
      to the program'
00042      write(*,*)
      '-------------------------------------------------------------------------------------------------------'
00043   else
00044      call getarg(1,arg)
00045      read(arg,*) zn
00046      if(iargc().ge.2) then
00047         call getarg(2,arg)
00048         read(arg,*) mi
00049      else
00050         mi = 2*zn
00051      write(*,*)
      '-------------------------------------------------------------------------------------------'
00052         write(*,*) 'Approximated the atomic mass as 2 * ZN = ', mi
00053      write(*,*)
      '-------------------------------------------------------------------------------------------------------'
00054      endif
00055   endif
00056
00057   ns=nint(zn)
00058
00059 ! some initializations
00060   allocate(species_m1(0:ns), species_p1(0:ns), species_cx(0:ns), species(0:ns), amns_ei(0:ns),
      amns_rc(0:ns), amns_lr(0:ns), amns_br(0:ns))
00061   allocate(state_labels(0:ns))
00062   allocate(l(0:ns), d(0:ns), u(0:ns), na(0:ns), rhs(0:ns), te(0:nt), ne(0:nt))
00063   allocate(rate_ei(0:nt,0:ns), rate_rc(0:nt,0:ns), rate_lr(0:nt,0:ns), rate_br(0:nt,0:ns))
00064   do is=0, ns
```

```
00065       allocate(species_m1(is)%components(4))
00066       species_m1(is)%components = (/ amns_reactant_type(zn, is, mi, 0), amns_reactant_type(0, -1, 0,
      0), amns_reactant_type(zn, is-1, mi, 1), amns_reactant_type(0, -1, 0, 1) /)
00067       allocate(species_p1(is)%components(4))
00068       species_p1(is)%components = (/ amns_reactant_type(zn, is, mi, 0), amns_reactant_type(0, -1, 0,
      0), amns_reactant_type(zn, is+1, mi, 1), amns_reactant_type(0, -1, 0, 1) /)
00069       allocate(species_cx(is)%components(4))
00070       species_cx(is)%components = (/ amns_reactant_type(zn, is, mi, 0), amns_reactant_type(1,  0, 2,
      0), amns_reactant_type(zn, is-1, mi, 1), amns_reactant_type(1,  1, 2, 1) /)
00071       allocate(species(is)%components(2))
00072       species(is)%components    = (/ amns_reactant_type(zn ,is, mi, 0),
     amns_reactant_type(zn,   is, mi, 1) /)
00073     enddo
00074
00075     do it=0,nt
00076       te(it) = 10.0_ids_real**(it/120.0_ids_real*7.0_ids_real-1.0_ids_real)
00077     enddo
00078     ne=1.0e20_ids_real
00079
00080   ! set up the AMNS system
00081
00082     call imas_amns_setup(amns)
00083     write(*,*) 'Done IMAS_AMNS_SETUP'
00084
00085   ! set up tables for ionization
00086     xx_rx%string='EI'
00087
00088     do is=0,ns
00089       if(species_p1(is)%components(1)%za .ne. species_p1(is)%components(1)%zn) then
00090         call imas_amns_setup_table(amns, xx_rx, species_p1(is), amns_ei(is))
00091         query%string='state_label'
00092         call imas_amns_query_table(amns_ei(is),query,answer)
00093         state_labels(is)=answer%string
00094       endif
00095     enddo
00096
00097   ! set up tables for line radiation
00098     xx_rx%string='LR'
00099     do is=0,ns
00100       if(species(is)%components(1)%za .ne. species(is)%components(1)%zn) then
00101         call imas_amns_setup_table(amns, xx_rx, species(is), amns_lr(is))
00102         query%string='state_label'
00103         call imas_amns_query_table(amns_lr(is),query,answer)
00104         state_labels(is)=answer%string
00105       endif
00106     enddo
00107
00108   ! set up tables for recombination
00109     xx_rx%string='RC'
00110     do is=0,ns
00111       if(species_m1(is)%components(1)%za .ne. 0) then
00112         call imas_amns_setup_table(amns, xx_rx, species_m1(is), amns_rc(is))
00113         query%string='state_label'
00114         call imas_amns_query_table(amns_rc(is),query,answer)
00115         state_labels(is)=answer%string
00116       endif
00117     enddo
00118
00119   ! set up tables for recombination radiation
00120     xx_rx%string='BR'
00121     do is=0,ns
00122       if(species(is)%components(1)%za .ne. 0) then
00123         call imas_amns_setup_table(amns, xx_rx, species(is), amns_br(is))
00124         query%string='state_label'
00125         call imas_amns_query_table(amns_br(is),query,answer)
00126         state_labels(is)=answer%string
00127       endif
00128     enddo
00129
00130     write(*,*) 'Done IMAS_AMNS_SETUP_TABLE'
00131
00132   ! interpolate ionization and line radiation data
00133     set%string='nowarn'
00134     do is=0,ns
00135       if(species(is)%components(1)%za .ne. species(is)%components(1)%zn) then
00136           call imas_amns_set_table(amns_ei(is),set)
00137           call imas_amns_rx(amns_ei(is),rate_ei(:,is),te,ne)
00138           call imas_amns_set_table(amns_lr(is),set)
00139           call imas_amns_rx(amns_lr(is),rate_lr(:,is),te,ne)
00140       endif
00141     enddo
00142
00143   ! interpolate recombination and recombination radiation data
00144     set%string='nowarn'
00145     do is=0,ns
00146       if(species(is)%components(1)%za .ne. 0) then
00147           call imas_amns_set_table(amns_rc(is),set)
```

```
00148          call imas_amns_rx(amns_rc(is),rate_rc(:,is),te,ne)
00149          call imas_amns_set_table(amns_br(is),set)
00150          call imas_amns_rx(amns_br(is),rate_br(:,is),te,ne)
00151       endif
00152    enddo
00153
00154    write(*,*) 'Done IMAS_AMNS_RX'
00155
00156 ! finalize the tables
00157    do is=0,ns
00158       if(species(is)%components(1)%za .ne. species(is)%components(1)%zn) then
00159          call imas_amns_finish_table(amns_ei(is))
00160          call imas_amns_finish_table(amns_lr(is))
00161       endif
00162    enddo
00163    do is=0,ns
00164       if(species(is)%components(1)%za .ne. 0) then
00165          call imas_amns_finish_table(amns_rc(is))
00166          call imas_amns_finish_table(amns_br(is))
00167       endif
00168    enddo
00169
00170    write(*,*) 'Done IMAS_AMNS_FINISH_TABLE'
00171
00172 ! finalize the system
00173    call imas_amns_finish(amns)
00174
00175    write(*,*) 'Done IMAS_AMNS_FINISH'
00176
00177    open(10,file='coronal.out')
00178    write(10,'(100a15)') '#te            ',state_labels, &
00179                         '   LR          ',  &
00180                         '   BR          '
00181
00182    do it=0, nt
00183
00184 ! set up the matrix
00185       l(0)=0.0_ids_real
00186       d(0)=-rate_ei(it,0)
00187       u(0)=rate_rc(it,1)
00188       rhs(0)=0.0_ids_real
00189       do is=1,ns-1
00190          l(is)=rate_ei(it,is-1)
00191          d(is)=-rate_ei(it,is)-rate_rc(it,is)
00192          u(is)=rate_rc(it,is+1)
00193          rhs(is)=0.0_ids_real
00194       enddo
00195       l(ns)=rate_ei(it,ns-1)
00196       d(ns)=-rate_rc(it,ns)
00197       u(ns)=0.0_ids_real
00198       rhs(ns)=0.0_ids_real
00199
00200 ! we need to set the value of 1 charge state
00201       isref=0
00202       do is=1, ns-1
00203          if(rate_ei(it,is) .LT. rate_rc(it,is)) then
00204             exit
00205          else
00206             isref=isref+1
00207          endif
00208       enddo
00209 !      write(*,*) te(it), isref
00210       u(isref)=0.0_ids_real
00211       d(isref)=1.0_ids_real
00212       l(isref)=0.0_ids_real
00213       rhs(isref)=1.0_ids_real
00214
00215 ! solve (l,d,u) na = rhs
00216       call solve_tridiag(l,d,u,rhs,na,ns+1)
00217       na(:) = na(:) / sum(na)
00218
00219       line_radiation = sum(na(0:ns-1)*rate_lr(it,0:ns-1))
00220       recombination_radiation = sum(na(1:ns)*rate_br(it,1:ns))
00221
00222       write(10,'(1p,100g15.6E3)') te(it), na(:), line_radiation, recombination_radiation
00223
00224 ! check the answer
00225
00226       norm=max(maxval(rate_ei(it,0:ns-1) * na(0:ns-1)), maxval(rate_rc(it,1:ns) * na(1:ns)))
00227       is=0
00228       test = (-rate_ei(it,is) * na(is) + rate_rc(it,is+1) * na(is+1))/norm
00229       if(abs(test).gt.1e-15_ids_real) write(*,*) 'LARGE ERROR: ',it, is, test
00230       do is=1, ns-1
00231          test = (rate_ei(it,is-1) * na(is-1) - (rate_rc(it,is) + rate_ei(it,is)) * na(is) +
      rate_rc(it,is+1) * na(is+1))/norm
00232          if(abs(test).gt.1e-15_ids_real) write(*,*) 'LARGE ERROR: ',it, is, test
00233       enddo
```

```
00234       is=ns
00235       test = (rate_ei(it,is-1) * na(is-1) - rate_rc(it,is) * na(is))/norm
00236       if(abs(test).gt.1e-15_ids_real) write(*,*) 'LARGE ERROR: ',it, is, test
00237
00238
00239     enddo
00240     close(10)
00241
00242 end program coronal
00243
00244
00245 subroutine solve_tridiag(a,b,c,v,x,n)
00246     use ids_types
00247     implicit none
00248 !       a - sub-diagonal (means it is the diagonal below the main diagonal)
00249 !       b - the main diagonal
00250 !       c - sup-diagonal (means it is the diagonal above the main diagonal)
00251 !       v - right part
00252 !       x - the answer
00253 !       n - number of equations
00254
00255     integer,intent(in) :: n
00256     real(kind=ids_real),dimension(n),intent(in) :: a,b,c,v
00257     real(kind=ids_real),dimension(n),intent(out) :: x
00258     real(kind=ids_real),dimension(n) :: bp,vp
00259     real(kind=ids_real) :: m
00260     integer i
00261
00262 ! Make copies of the b and v variables so that they are unaltered by this sub
00263     bp(1) = b(1)
00264     vp(1) = v(1)
00265
00266     !The first pass (setting coefficients):
00267     firstpass: do i = 2,n
00268        m = a(i)/bp(i-1)
00269        bp(i) = b(i) - m*c(i-1)
00270        vp(i) = v(i) - m*vp(i-1)
00271     end do firstpass
00272
00273     x(n) = vp(n)/bp(n)
00274     !The second pass (back-substition)
00275     backsub:do i = n-1, 1, -1
00276        x(i) = (vp(i) - c(i)*x(i+1))/bp(i)
00277     end do backsub
00278
00279 end subroutine solve_tridiag
```

16.3 examples/coronal_py/coronal.py File Reference

Namespaces

- coronal

Functions

- def coronal.te_ne (te, ne)
- def coronal.TDMASolve (a, b, c, d)
- def coronal.point (ei, rc)
- def coronal.distribution (rates, te, ne)
- def coronal.rates (ZN)

16.4 coronal.py

```
00001 '''
00002 Tools to work with the coronal distribution
00003 David.Coster@ipp.mpg.de
00004 '''
00005 import numpy as np
00006 import amns
00007
00008 def te_ne (te, ne):
00009     ''' Given 1d arrays of te and ne, return the appropriate outer product arrays of TE and NE '''
00010     TE=np.array(te.repeat(len(ne)).reshape([len(te), len(ne)]),order='F')
00011     NE=np.array(ne.repeat(len(te)).reshape([len(ne), len(te)]).transpose(),order='F')
00012     return TE, NE
00013
```

```
00014 def TDMASolve(a, b, c, d):
00015     ''' Solve the tridiagonal system '''
00016     n = len(d) # n is the numbers of rows, a and c has length n-1
00017     bl = b.copy()
00018     dl = d.copy()
00019     for i in range(n-1):
00020         dl[i+1] -= dl[i] * a[i+1] / bl[i]
00021         bl[i+1] -= c[i] * a[i+1] / bl[i]
00022     for i in reversed(range(n-1)):
00023         dl[i] -= dl[i+1] * c[i] / bl[i+1]
00024     return dl / bl # return the solution
00025
00026 def point(ei, rc):
00027     ''' return the coronal fractional densities determined by the balance of ionization and
      recombination '''
00028     n=len(ei)
00029
00030     l = np.zeros(n)
00031     d = np.zeros(n)
00032     u = np.zeros(n)
00033     na = np.zeros(n)
00034
00035     l[0] = 0.0
00036     d[0] = -ei[0]
00037     u[0] = rc[1]
00038     na[0] = 0.0
00039
00040     l[1:-1] = ei[0:-2]
00041     d[1:-1] = -ei[1:-1] - rc[1:-1]
00042     u[1:-1] = rc[2:]
00043     na[1:-1] = 0.0
00044
00045     l[-1] = ei[-2]
00046     d[-1] = -rc[-1]
00047     u[-1] = 0.0
00048     na[-1] = 0.0
00049
00050     try:
00051         isref = list(ei[:] < rc[:]).index(True)
00052     except:
00053         isref = len(ei[:])-1
00054     l[isref] = 0.0
00055     d[isref] = 1.0
00056     u[isref] = 0.0
00057     na[isref] = 1.0
00058
00059     na = TDMASolve(l,d,u,na)
00060     na = na / na.sum()
00061
00062     return na
00063
00064 def distribution (rates, te, ne):
00065     '''
00066     inputs: rates dictionary, te and ne
00067     outputs: a dictionary containing:
00068       the coronal fractional densities
00069       average charge
00070       line radiation efficiency
00071       recombination (including Bremsstrahlung) radiation efficiency
00072     '''
00073
00074     n = len(rates['EI'])
00075     nte = te.shape[0]
00076     nne = ne.shape[1]
00077
00078     ei = np.zeros([n, nte, nne])
00079     rc = np.zeros([n, nte, nne])
00080     br = np.zeros([n, nte, nne])
00081     lr = np.zeros([n, nte, nne])
00082     na = np.zeros([n, nte, nne])
00083
00084     for i in range(n):
00085         ei[i,:,:] = rates['EI'][i].data(te,ne)
00086         rc[i,:,:] = rates['RC'][i].data(te,ne)
00087         br[i,:,:] = rates['BR'][i].data(te,ne)
00088         lr[i,:,:] = rates['LR'][i].data(te,ne)
00089
00090     for iit in range(nte):
00091         for iin in range(nne):
00092             na[:, iit, iin] = point(ei[:, iit, iin], rc[:, iit, iin])
00093
00094     ZA=(np.arange(n).repeat(nte).repeat(nne)).reshape([n,nte,nne])
00095
00096     Z=(na[:,:,:]*ZA).sum(axis=0)/na[:,:,:].sum(axis=0)
00097
00098     LR_rad = (na*lr).sum(axis=0)*ne
00099     BR_rad = (na*br).sum(axis=0)*ne
```

```
00100
00101     return dict(na=na, Z=Z, LR_rad=LR_rad, BR_rad=BR_rad)
00102
00103 def rates(ZN):
00104     ''' return the rates dictionary for the specified nuclear charge '''
00105     reactantsRC=[]
00106     reactantsEI=[]
00107     reactantsRD=[]
00108     for i in range(ZN+1):
00109         reactantsRC.append(amns.Reactants())
00110         reactantsRC[i].add(ZN,i,0)
00111         reactantsRC[i].add(0,-1,0)
00112         reactantsRC[i].add(ZN,i-1,0,lr=1)
00113         reactantsRC[i].add(0,-1,0,lr=1)
00114         reactantsEI.append(amns.Reactants())
00115         reactantsEI[i].add(ZN,i,0)
00116         reactantsEI[i].add(0,-1,0)
00117         reactantsEI[i].add(ZN,i+1,0,lr=1)
00118         reactantsEI[i].add(0,-1,0,lr=1)
00119         reactantsRD.append(amns.Reactants())
00120         reactantsRD[i].add(ZN,i,0)
00121         reactantsRD[i].add(ZN,i,0,lr=1)
00122
00123     amnsdb = amns.Amns()
00124
00125     print('Setting up tables ...')
00126
00127     vals=dict(SP=[], EI=[], RC=[], BR=[], LR=[])
00128     for i in range(ZN+1):
00129         vals['EI'].append(amnsdb.get_table("EI", reactantsEI[i]))
00130         vals['RC'].append(amnsdb.get_table("RC", reactantsRC[i]))
00131         vals['BR'].append(amnsdb.get_table("BR", reactantsRD[i]))
00132         vals['LR'].append(amnsdb.get_table("LR", reactantsRD[i]))
00133         vals['SP'].append(vals['EI'][i].state_label)
00134
00135     return vals
```

16.5 examples/coronal_py/coronal_charge_state_edge.py File Reference

Namespaces

- coronal_charge_state_edge

Variables

- coronal_charge_state_edge.te
- coronal_charge_state_edge.ne
- coronal_charge_state_edge.rates = coronal.rates(Z)
- coronal_charge_state_edge.dist = coronal.distribution(rates, te, ne)
- coronal_charge_state_edge.label
- coronal_charge_state_edge.loc

16.6 coronal_charge_state_edge.py

```
00001 #! /usr/bin/env python
00002 """
00003 Calculate coronal charge state in edge plasma conditions
00004 David.Coster@ipp.mpg.de
00005 """
00006 import coronal
00007 import numpy as np
00008 import matplotlib.pylab as plt
00009
00010 te, ne = coronal.te_ne(np.logspace(0,4,num=121), np.array([3e19]))
00011
00012 plt.ion()
00013
00014 plt.clf()
00015 for Z in (74, 54, 36, 18, 10, 7, 4, 2, 1):
00016   rates = coronal.rates(Z)
00017   dist = coronal.distribution(rates, te, ne)
00018   plt.semilogx(te[:,0], dist['Z'], label=rates['SP'][0][:-1])
00019 plt.xlabel('Te [$eV$]')
00020 plt.ylabel('Average charge')
00021 plt.legend(loc=0)
00022 plt.title('Average charge for ne = %s' % (np.unique(ne),))
```

```
00023 plt.savefig('coronal_charge_state_edge.pdf')
00024 plt.savefig('coronal_charge_state_edge.png')
```

16.7 examples/coronal_py/coronal_comparison_N+Ne.py File Reference

Namespaces

- coronal_comparison_N+Ne

Variables

- coronal_comparison_N+Ne.te
- coronal_comparison_N+Ne.ne
- coronal_comparison_N+Ne.rates_N = coronal.rates(7)
- coronal_comparison_N+Ne.rates_Ne = coronal.rates(10)
- coronal_comparison_N+Ne.dist_N = coronal.distribution(rates_N, te, ne)
- coronal_comparison_N+Ne.dist_Ne = coronal.distribution(rates_Ne, te, ne)
- coronal_comparison_N+Ne.NUM_COLORS = np.maximum(dist_N['na'].shape[0], dist_Ne['na'].shape[0])
- coronal_comparison_N+Ne.cm = plt.get_cmap('gist_rainbow')
- coronal_comparison_N+Ne.color
- coronal_comparison_N+Ne.label
- coronal_comparison_N+Ne.loc
- coronal_comparison_N+Ne.ncol

16.8 coronal_comparison_N+Ne.py

```
00001 #! /usr/bin/env python
00002 import coronal
00003 import numpy as np
00004 import matplotlib.pylab as plt
00005
00006 te, ne = coronal.te_ne(np.logspace(-1,6,num=121), np.logspace(16,22,num=121))
00007
00008 rates_N  = coronal.rates(7)
00009 rates_Ne = coronal.rates(10)
00010
00011 dist_N  = coronal.distribution(rates_N,  te, ne)
00012 dist_Ne = coronal.distribution(rates_Ne, te, ne)
00013
00014 plt.ion()
00015
00016 NUM_COLORS=np.maximum(dist_N['na'].shape[0], dist_Ne['na'].shape[0])
00017 cm = plt.get_cmap('gist_rainbow')
00018 plt.clf()
00019 for i in range(dist_N['na'].shape[0]):
       plt.semilogx(te[:,ne.shape[1]//2],dist_N['na'][i,:,ne.shape[1]//2],       color=cm(1.*i/NUM_COLORS),
       label='%s' % (rates_N['SP'][i]))
00020 for i in range(dist_Ne['na'].shape[0]):
       plt.semilogx(te[:,ne.shape[1]//2],dist_Ne['na'][i,:,ne.shape[1]//2], '--', color=cm(1.*i/NUM_COLORS),
       label='%s' % (rates_Ne['SP'][i]))
00021 plt.xlabel('Te')
00022 plt.xlim(0.1,1e7)
00023 plt.ylabel('Average charge')
00024 plt.title('ne = %s' % ((np.ravel(ne[0,ne.shape[1]//2]))))
00025 plt.legend(loc=0, ncol=2)
00026 plt.savefig('coronal_comparison_N+Ne_charge_state_distribution.png')
00027
00028 plt.clf()
00029 plt.semilogx(te[:,ne.shape[1]//2],dist_N['Z'][:,ne.shape[1]//2], label='N')
00030 plt.semilogx(te[:,ne.shape[1]//2],dist_Ne['Z'][:,ne.shape[1]//2], label='Ne')
00031 plt.xlabel('Te')
00032 plt.ylabel('Average charge')
00033 plt.title('ne = %s' % (ne[0,ne.shape[1]//2],))
00034 plt.legend(loc=0)
00035 plt.savefig('coronal_comparison_N+Ne_average_charge.png')
00036
00037 plt.clf()
00038 plt.loglog(te[:,ne.shape[1]//2],dist_N['LR_rad'][:,ne.shape[1]//2], label='LR N')
00039 plt.loglog(te[:,ne.shape[1]//2],dist_Ne['LR_rad'][:,ne.shape[1]//2], label='LR Ne')
00040 plt.xlabel('Te')
00041 plt.title('ne = %s' % (ne[0,ne.shape[1]//2],))
00042 plt.legend(loc=0)
```

```
00043 plt.savefig('coronal_comparison_N+Ne_LR_rad.png')
00044
00045 plt.clf()
00046 plt.loglog(te[:,ne.shape[1]//2],dist_N['BR_rad'][:,ne.shape[1]//2], label='BR N')
00047 plt.loglog(te[:,ne.shape[1]//2],dist_Ne['BR_rad'][:,ne.shape[1]//2], label='BR Ne')
00048 plt.xlabel('Te')
00049 plt.title('ne = %s' % (ne[0,ne.shape[1]//2],))
00050 plt.legend(loc=0)
00051 plt.savefig('coronal_comparison_N+Ne_BR_rad.png')
00052
00053 plt.clf()
00054 plt.loglog(te[:,ne.shape[1]//2],dist_N['LR_rad'][:,ne.shape[1]//2]+dist_N['BR_rad'][:,ne.shape[1]//2],
       label='LR+Br N')
00055
       plt.loglog(te[:,ne.shape[1]//2],dist_Ne['LR_rad'][:,ne.shape[1]//2]+dist_Ne['BR_rad'][:,ne.shape[1]//2],
       label='LR+Br Ne')
00056 plt.xlabel('Te')
00057 plt.title('ne = %s' % (ne[0,ne.shape[1]//2],))
00058 plt.ylim(1e-16,1e-12)
00059 plt.legend(loc=0)
00060 plt.savefig('coronal_comparison_N+Ne_LR+BR_rad.png')
```

16.9 examples/coronal_py/coronal_info.py File Reference

Namespaces

- coronal_info

Variables

- coronal_info.parser
- coronal_info.type
- coronal_info.int
- coronal_info.help
- coronal_info.default
- coronal_info.args = parser.parse_args()
- coronal_info.rates = coronal.rates(args.nuclear_charge)
- coronal_info.te
- coronal_info.ne
- coronal_info.dist = coronal.distribution(rates, te, ne)
- coronal_info.header
- coronal_info.loc
- coronal_info.labels
- coronal_info.lw
- coronal_info.fontsize
- coronal_info.ncol

16.10 coronal_info.py

```
00001 #! /usr/bin/env python
00002 """
00003 Calculate coronal data
00004 David.Coster@ipp.mpg.de
00005 """
00006 import coronal
00007 import numpy as np
00008 import matplotlib.pylab as plt
00009 from matplotlib.colors import LogNorm
00010 import argparse
00011
00012 parser = argparse.ArgumentParser(description="Provide coronal information", epilog
00013 ="""
00014 """, formatter_class=argparse.RawTextHelpFormatter)
00015 parser.add_argument("--nuclear_charge", "-z", type=int, help="Nuclear charge", default=74)
00016 args=parser.parse_args()
00017
00018 rates = coronal.rates(args.nuclear_charge)
00019
00020 te, ne = coronal.te_ne(np.logspace(-1,6,num=121), np.array([1e18, 1e19, 1e20, 1e21]))
00021
```

```
00022 dist = coronal.distribution(rates, te, ne)
00023
00024 for i, D in enumerate(ne[0,:]):
00025     np.savetxt('coronal_info_ne=%.2e.out' % (ne[0,i]), np.concatenate((te[:,i:i+1],
      dist['na'][:,:,i].T, dist['LR_rad'][:,i:i+1]/ne[0,i], dist['BR_rad'][:,i:i+1]/ne[0,i]), axis=1),
      header='te '+' '.join(rates['SP'])+' LR BR')
00026
00027 plt.ion()
00028 plt.clf()
00029
00030 plt.clf()
00031 plt.semilogx(te[:,0], dist['Z'])
00032 plt.xlabel('Te [$eV$]')
00033 plt.ylabel('Average charge')
00034 plt.legend(loc=0, labels=ne[0,:])
00035 plt.title('Average charge for %s with varying ne' % (rates['SP'][0][:-1],))
00036 plt.savefig('coronal_info_average_charge.pdf')
00037 plt.savefig('coronal_info_average_charge.png')
00038
00039 plt.clf()
00040 plt.loglog(te,(dist['LR_rad']+dist['BR_rad'])/ne, lw=3)
00041 plt.xlabel('Te [$eV$]')
00042 plt.ylabel('Radiation efficiency')
00043 plt.ylim(1e-37,1e-29)
00044 plt.title('Radiation efficiency for %s with varying ne' % (rates['SP'][0][:-1],))
00045 plt.legend(loc=0, labels=ne[0,:])
00046 plt.savefig('coronal_info_radiation_efficiency.pdf')
00047 plt.savefig('coronal_info_radiation_efficiency.png')
00048
00049 for i, D in enumerate(ne[0,:]):
00050     plt.clf()
00051     plt.loglog(te[:,0], dist['na'][:,:,i].T)
00052     plt.xlabel('Te [$eV$]')
00053     plt.ylabel('Fractional abundance')
00054     plt.ylim(1e-3,1)
00055     plt.title('Fractional abundance for %s with ne = %.2e' % (rates['SP'][0][:-1], ne[0,i]))
00056     plt.legend(loc='lower center', fontsize=5, labels=rates['SP'], ncol=9)
00057     plt.savefig('coronal_info_fractional_abundance_ne=%.2e.pdf' % (ne[0,i]))
00058     plt.savefig('coronal_info_fractional_abundance_ne=%.2e.png' % (ne[0,i]))
00059
```

16.11 examples/coronal_py/coronal_radiation_efficiency.py File Reference

Namespaces

- coronal_radiation_efficiency

Variables

- list coronal_radiation_efficiency.L = [1,2,3,4,6,7,10,18,36,54,74]
- list coronal_radiation_efficiency.rates = [ ]
- coronal_radiation_efficiency.te
- coronal_radiation_efficiency.ne
- list coronal_radiation_efficiency.dist = [ ]
- coronal_radiation_efficiency.label
- coronal_radiation_efficiency.lw
- coronal_radiation_efficiency.loc

16.12 coronal_radiation_efficiency.py

```
00001 #! /usr/bin/env python
00002 """
00003 Calculate the coronal radiation efficiencies
00004 David.Coster@ipp.mpg.de
00005 """
00006 import coronal
00007 import numpy as np
00008 import matplotlib.pylab as plt
00009 from matplotlib.colors import LogNorm
00010
00011 # we will consider the following species
00012 L = [1,2,3,4,6,7,10,18,36,54,74]
```

```
00013
00014 rates = []
00015 for i in L:
00016     print(i)
00017     rates.append(coronal.rates(i))
00018
00019 te, ne = coronal.te_ne(np.logspace(-1,6,num=121), np.array([1e20]))
00020
00021 dist = []
00022 for r in rates:
00023     print(r['SP'][0][:-1])
00024     dist.append(coronal.distribution(r, te, ne))
00025
00026 plt.ion()
00027 plt.clf()
00028
00029 for i, r, d in zip(L, rates, dist):
00030     print(i)
00031     plt.loglog(te[:,0],(d['LR_rad'][:,0]+d['BR_rad'][:,0])/ne[0,0], label=r['SP'][0][:-1], lw=3)
00032
00033 plt.xlabel('Te [$eV$]')
00034 plt.ylabel('Radiation Efficiency')
00035 plt.ylim(1e-37,1e-30)
00036 plt.title('ne = %s' % (ne[0,0],))
00037 plt.legend(loc=0)
00038 plt.savefig('coronal_radiation_efficiency.pdf')
00039 plt.savefig('coronal_radiation_efficiency.png')
```

16.13 examples/coronal_py/coronal_version_comparison.py File Reference

Namespaces

- coronal_version_comparison

Variables

- coronal_version_comparison.parser
- coronal_version_comparison.type
- coronal_version_comparison.int
- coronal_version_comparison.help
- coronal_version_comparison.default
- coronal_version_comparison.str
- coronal_version_comparison.args = parser.parse_args()
- int coronal_version_comparison.nte = 141
- int coronal_version_comparison.nne = 121
- coronal_version_comparison.te
- coronal_version_comparison.ne
- coronal_version_comparison.ZN = args.nuclear_charge
- list coronal_version_comparison.reactantsRC = [ ]
- list coronal_version_comparison.reactantsEI = [ ]
- list coronal_version_comparison.reactantsRD = [ ]
- coronal_version_comparison.lr
- coronal_version_comparison.amnsdb_0 = amns.Amns(version_user=args.user, version_number=args.↩ version_1)
- coronal_version_comparison.amnsdb_1 = amns.Amns(version_user=args.user, version_number=args.↩ version_2)
- list coronal_version_comparison.SP = [ ]
- list coronal_version_comparison.EI_0 = [ ]
- list coronal_version_comparison.EI_1 = [ ]
- coronal_version_comparison.dist_0 = coronal.distribution(dict(EI=EI_0, RC=RC_0, BR=BR_0, LR=LR_0), te, ne)
- coronal_version_comparison.dist_1 = coronal.distribution(dict(EI=EI_1, RC=RC_1, BR=BR_1, LR=LR_1), te, ne)

- coronal_version_comparison.NUM_COLORS = dist_0['na'].shape[0]
- coronal_version_comparison.cm = plt.get_cmap('gist_rainbow')
- coronal_version_comparison.color
- coronal_version_comparison.label
- coronal_version_comparison.loc
- coronal_version_comparison.norm

16.14 coronal_version_comparison.py

```
00001 #! /usr/bin/env python
00002 import amns
00003 import coronal
00004 import numpy as np
00005 from matplotlib.backends.backend_pdf import PdfPages
00006 import matplotlib.pylab as plt
00007 from matplotlib.colors import LogNorm
00008 import os
00009
00010 import argparse
00011
00012 parser = argparse.ArgumentParser(description="Provide a comparison of the coronal results for two
       different versions of the AMNS data", epilog
00013 ="""
00014 """, formatter_class=argparse.RawTextHelpFormatter)
00015 parser.add_argument("--nuclear_charge", "-z", type=int, help="Nuclear charge", default=10)
00016 parser.add_argument("--version_1", "-v", type=int, help="version index for the 1st case", default=1)
00017 parser.add_argument("--version_2", "-V", type=int, help="version index for the 2nd case", default=0)
00018 parser.add_argument("--user", "-u", type=str, help="username of the database owner",
       default=os.getenv("USER"))
00019 args=parser.parse_args()
00020
00021 nte = 141
00022 nne = 121
00023 te, ne = coronal.te_ne(np.logspace(-1,6,nte),np.logspace(16,22,nne))
00024
00025 ZN = args.nuclear_charge
00026
00027 print('Setting up reactants ...')
00028 reactantsRC=[]
00029 reactantsEI=[]
00030 reactantsRD=[]
00031 for i in range(ZN+1):
00032     reactantsRC.append(amns.Reactants())
00033     reactantsRC[i].add(ZN,i,0)
00034     reactantsRC[i].add(0,-1,0)
00035     reactantsRC[i].add(ZN,i-1,0,lr=1)
00036     reactantsRC[i].add(0,-1,0,lr=1)
00037     reactantsEI.append(amns.Reactants())
00038     reactantsEI[i].add(ZN,i,0)
00039     reactantsEI[i].add(0,-1,0)
00040     reactantsEI[i].add(ZN,i+1,0,lr=1)
00041     reactantsEI[i].add(0,-1,0,lr=1)
00042     reactantsRD.append(amns.Reactants())
00043     reactantsRD[i].add(ZN,i,0)
00044     reactantsRD[i].add(ZN,i,0,lr=1)
00045
00046 amnsdb_0 = amns.Amns(version_user=args.user, version_number=args.version_1)
00047 amnsdb_1 = amns.Amns(version_user=args.user, version_number=args.version_2)
00048
00049 print('Setting up tables ...')
00050
00051 SP=[]
00052 EI_0=[]; RC_0=[]; BR_0=[]; LR_0=[]
00053 EI_1=[]; RC_1=[]; BR_1=[]; LR_1=[]
00054 for i in range(ZN+1):
00055     EI_0.append(amnsdb_0.get_table("EI", reactantsEI[i]))
00056     RC_0.append(amnsdb_0.get_table("RC", reactantsRC[i]))
00057     BR_0.append(amnsdb_0.get_table("BR", reactantsRD[i]))
00058     LR_0.append(amnsdb_0.get_table("LR", reactantsRD[i]))
00059     EI_1.append(amnsdb_1.get_table("EI", reactantsEI[i]))
00060     RC_1.append(amnsdb_1.get_table("RC", reactantsRC[i]))
00061     BR_1.append(amnsdb_1.get_table("BR", reactantsRD[i]))
00062     LR_1.append(amnsdb_1.get_table("LR", reactantsRD[i]))
00063     SP.append(EI_0[i].state_label)
00064
00065
00066 dist_0 = coronal.distribution(dict(EI=EI_0, RC=RC_0, BR=BR_0, LR=LR_0), te, ne)
00067 dist_1 = coronal.distribution(dict(EI=EI_1, RC=RC_1, BR=BR_1, LR=LR_1), te, ne)
00068
00069 NUM_COLORS=dist_0['na'].shape[0]
00070 cm = plt.get_cmap('gist_rainbow')
00071
```

```
00072 with PdfPages('coronal_version_comparison.pdf') as pdf:
00073     plt.clf()
00074     for i in range(dist_0['na'].shape[0]):
00075         plt.semilogx(te[:,nne//2],dist_0['na'][i,:,nne//2],       color=cm(1.*i/NUM_COLORS),
      label='%s' % (SP[i]))
00076         plt.semilogx(te[:,nne//2],dist_1['na'][i,:,nne//2], '--', color=cm(1.*i/NUM_COLORS))
00077     plt.xlabel('Te')
00078     plt.ylabel('Fractional density')
00079     plt.title('ne = %s \n REF=(%s, %s, %s) \n ALT=(%s, %s, %s)' % (tuple(np.ravel(ne[0,nne//2])) +
      amnsdb_0.version + (EI_0[0].prop_creation,) + amnsdb_1.version + (EI_1[0].prop_creation,)))
00080     plt.legend(loc=0)
00081     pdf.savefig()
00082     plt.yscale('log') ; plt.ylim(1e-3,1)
00083     pdf.savefig() ; plt.close()
00084
00085     plt.clf()
00086     for i in range(dist_0['na'].shape[0]):
00087         plt.semilogx(te[:,nne//2], dist_1['na'][i,:,nne//2] -
      dist_0['na'][i,:,nne//2],color=cm(1.*i/NUM_COLORS), label='%s' % (SP[i]))
00088     plt.xlabel('Te')
00089     plt.ylabel('Difference in fractional density')
00090     plt.title('ne = %s \n REF=(%s, %s, %s) \n ALT=(%s, %s, %s)' % (tuple(np.ravel(ne[0,nne//2])) +
      amnsdb_0.version + (EI_0[0].prop_creation,) + amnsdb_1.version + (EI_1[0].prop_creation,)))
00091     plt.legend(loc=0)
00092     pdf.savefig() ; plt.close()
00093
00094     plt.clf()
00095     plt.semilogx(te[:,nne//2], dist_0['Z'][:,nne//2], label='%s %s' %amnsdb_0.version)
00096     plt.semilogx(te[:,nne//2], dist_1['Z'][:,nne//2], label='%s %s' %amnsdb_1.version)
00097     plt.xlabel('Te')
00098     plt.ylabel('Average charge')
00099     plt.title('ne = %s' % (ne[0,nne//2],))
00100     plt.legend(loc=0)
00101     pdf.savefig() ; plt.close()
00102
00103
00104     for NE in [0, nne//2, nne-1]:
00105         plt.clf()
00106
      plt.semilogx(te[:,NE],2*(dist_1['LR_rad'][:,NE]-dist_0['LR_rad'][:,NE])/(dist_1['LR_rad'][:,NE]+dist_0['LR_rad'][:,NE])
      label='LR fractional difference')
00107
      plt.semilogx(te[:,NE],2*(dist_1['BR_rad'][:,NE]-dist_0['BR_rad'][:,NE])/(dist_1['BR_rad'][:,NE]+dist_0['BR_rad'][:,NE])
      label='BR fractional difference')
00108         plt.xlabel('Te')
00109         plt.title('ne = %s' % (ne[0,NE],))
00110         plt.legend(loc=0)
00111         pdf.savefig() ; plt.close()
00112
00113     plt.clf()
00114     plt.loglog(te[:,nne//2],dist_0['LR_rad'][:,nne//2], label='LR %s %s' %amnsdb_0.version)
00115     plt.loglog(te[:,nne//2],dist_1['LR_rad'][:,nne//2], label='LR %s %s' %amnsdb_1.version)
00116     plt.xlabel('Te')
00117     plt.title('ne = %s' % (ne[0,nne//2],))
00118     plt.legend(loc=0)
00119     pdf.savefig() ; plt.close()
00120
00121     plt.clf()
00122     plt.loglog(te[:,nne//2],dist_0['BR_rad'][:,nne//2], label='BR %s %s' %amnsdb_0.version)
00123     plt.loglog(te[:,nne//2],dist_1['BR_rad'][:,nne//2], label='BR %s %s' %amnsdb_1.version)
00124     plt.xlabel('Te')
00125     plt.title('ne = %s' % (ne[0,nne//2],))
00126     plt.legend(loc=0)
00127     pdf.savefig() ; plt.close()
00128
00129     plt.clf()
00130     plt.loglog(te[:,nne//2],dist_0['LR_rad'][:,nne//2], label='LR %s %s' %amnsdb_0.version)
00131     plt.loglog(te[:,nne//2],dist_1['LR_rad'][:,nne//2], label='LR %s %s' %amnsdb_1.version)
00132     plt.loglog(te[:,nne//2],dist_0['BR_rad'][:,nne//2], label='BR %s %s' %amnsdb_0.version)
00133     plt.loglog(te[:,nne//2],dist_1['BR_rad'][:,nne//2], label='BR %s %s' %amnsdb_1.version)
00134     plt.xlabel('Te')
00135     plt.title('ne = %s' % (ne[0,nne//2],))
00136     plt.legend(loc=0)
00137     pdf.savefig() ; plt.close()
00138
00139     plt.clf()
00140     plt.contourf(te,ne,dist_0['Z']-dist_1['Z'])
00141     plt.xscale('log')
00142     plt.xlabel('Te')
00143     plt.yscale('log')
00144     plt.ylabel('ne')
00145     plt.colorbar()
00146     plt.title('Difference of average charge')
00147     pdf.savefig() ; plt.close()
00148
00149     plt.clf()
00150     plt.contourf(te,ne,dist_0['LR_rad'],norm=LogNorm())
```

```
00151     plt.xscale('log')
00152     plt.xlabel('Te')
00153     plt.yscale('log')
00154     plt.ylabel('ne')
00155     plt.colorbar()
00156     plt.title('dist_0["LR"]')
00157     pdf.savefig() ; plt.close()
00158
00159     plt.clf()
00160     plt.contourf(te,ne,dist_1['LR_rad'],norm=LogNorm())
00161     plt.xscale('log')
00162     plt.xlabel('Te')
00163     plt.yscale('log')
00164     plt.ylabel('ne')
00165     plt.colorbar()
00166     plt.title('dist_1["LR"]')
00167     pdf.savefig() ; plt.close()
00168
00169     plt.clf()
00170     plt.contourf(te,ne,dist_0['BR_rad'],norm=LogNorm())
00171     plt.xscale('log')
00172     plt.xlabel('Te')
00173     plt.yscale('log')
00174     plt.ylabel('ne')
00175     plt.colorbar()
00176     plt.title('BR_0')
00177     pdf.savefig() ; plt.close()
00178
00179     plt.clf()
00180     plt.contourf(te,ne,dist_1['BR_rad'],norm=LogNorm())
00181     plt.xscale('log')
00182     plt.xlabel('Te')
00183     plt.yscale('log')
00184     plt.ylabel('ne')
00185     plt.colorbar()
00186     plt.title('BR_1')
00187     pdf.savefig() ; plt.close()
00188
00189 amnsdb_0.finalize()
00190 amnsdb_1.finalize()
```

16.15 examples/coronal_py/README.md File Reference

16.16 examples/py/README.md File Reference

16.17 tests/java/README.md File Reference

16.18 tests/matlab/README.md File Reference

16.19 verification/README.md File Reference

16.20 examples/java/src/amnsdemo/AmnsDemoCaseIdx.java File Reference

Classes

- class amnsdemo.AmnsDemoCaseIdx

Packages

- package amnsdemo

16.21 AmnsDemoCaseIdx.java

```
00001 package amnsdemo;
00002
00003 import amns.*;
00004 import amns.type.*;
00005
```

```
00012 public class AmnsDemoCaseIdx {
00013
00016     public static void main(String[] args) throws Exception {
00017
00018         int ns = 7; // number of species
00019         int nx=101, ny=101; // size of the arrays used in computations
00020         int is, ix, iy; // variables used in for loops later on in the code
00021
00022         // this is a pointer from C
00023         long ptrAmnsHandle = 0;
00024         AmnsErrorType error_stat = new AmnsErrorType();
00025
00026         // these are the pointers as well
00027         // we have to remember that in Java there is no such concept
00028         // as pointer. We will get all the pointers from C as long values
00029         //
00030         long ptrReactantsHandle[] = new long[ns];
00031         long ptrAmnsCxHandle[] = new long[ns];
00032
00033         boolean doNuclear;
00034         // Dimensions of multi-dimensional arrays have to be specified
00035         // in reverse order compared to Fortran
00036         // there is one more step in here, before we will pass arrays to
00037         // JNI we will "flattern" them by storing in 1D array
00038         double te[][] = new double[ny][nx];
00039         double ne[][] = new double [ny][nx];
00040         double rate[][][] = new double[ns][ny][nx];
00041         double rate1[][] = new double[ny][nx];
00042
00043         Amns amns = new Amns();
00044
00045         // Setup
00046         String imas_amns_debug = System.getenv("IMAS_AMNS_DEBUG");
00047         Boolean amns_debug = (imas_amns_debug != null && !imas_amns_debug.trim().isEmpty() &&
     !imas_amns_debug.equals("no") && !imas_amns_debug.equals("NO") );
00048
00049         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP");
00050         ptrAmnsHandle = amns.ImasAmnsCCSetup(error_stat);
00051         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " +
     ptrAmnsHandle);
00052         if (amns_debug) System.err.println("[JVM] Error.flag= " + error_stat.flag + " Error.string= "
     + error_stat.string);
00053
00054         // Set up species
00055         for ( is=0; is < ns; is++) {
00056             // string and index are unused here
00057             int idx = 0;
00058             if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP_REACTANTS: " + (is +
     1) + "/" + ns);
00059             long tempPtrReactantsHandle = amns.ImasAmnsCcSetupReactantsNumber(idx, 4);
00060             if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
     tempPtrReactantsHandle);
00061
00062             ptrReactantsHandle[is] = tempPtrReactantsHandle;
00063
00064             // setup first product
00065             AmnsReactantType species = new AmnsReactantType();
00066             species.ZN = 6;
00067             species.ZA = is;
00068             species.MI = 12;
00069             species.LR = 0;
00070
00071             amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle[is], 1, species);
00072
00073             // setup second product
00074             species.ZN = 1;
00075             species.ZA = 0;
00076             species.MI = 2;
00077             species.LR = 0;
00078
00079             amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle[is], 2, species);
00080
00081             // setup third product
00082             species.ZN = 6;
00083             species.ZA = is - 1;
00084             species.MI = 12;
00085             species.LR = 1;
00086
00087             amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle[is], 3, species);
00088
00089             // setup fourth product
00090             species.ZN = 1;
00091             species.ZA = 1;
00092             species.MI = 2;
00093             species.LR = 1;
00094
00095             amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle[is], 4, species);
```

```
00096
00097             // sanity check for the GET routine
00098             species.ZN = 0;
00099             species.ZA = 0;
00100             species.MI = 0;
00101
00102             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00103
00104             System.out.println( "[JVM] Values from C: \n"
00105                                 + " species.ZN = " + species.ZN
00106                                 + " species.ZA = " + species.ZA
00107                                 + " species.MI = " + species.MI);
00108         }
00109
00110
00111         for (ix=0; ix<nx; ix++) {
00112             for (iy=0; iy<ny; iy++) {
00113                 te[iy][ix] = java.lang.Math.pow( 10.0, (double)ix / ((double)(nx - 1) / 5.0) ) * 0.1;
00114             }
00115         }
00116
00117         for (iy=0; iy<ny; iy++) {
00118             for (ix=0; ix<nx; ix++) {
00119                 ne[iy][ix] = java.lang.Math.pow( 10.0, (double)iy / (((double)ny - 1) / 10.0) ) *
      1.0e15;
00120             }
00121         }
00122
00123         AmnsSetType set = new AmnsSetType();
00124         AmnsErrorType error = new AmnsErrorType();
00125         set.string = "backend=mdsplus";
00126
00127         amns.ImasAmnsCCSet(ptrAmnsHandle, set, error);
00128
00129         AmnsQueryType query = new AmnsQueryType();
00130         query.string = "version";
00131
00132         AmnsAnswerType answer = new AmnsAnswerType();
00133
00134         amns.ImasAmnsCCQuery(ptrAmnsHandle, query, answer, error);
00135         System.out.println("[JVM] AMNS data base version = '" + answer.string + "'," + answer.number);
00136
00137         AmnsReactionType xx_rx = new AmnsReactionType();
00138         xx_rx.string = "CX";
00139
00140         // set up tables for charge-exchange
00141         for (is=0; is < ns; is++) {
00142
00143             AmnsReactantType species = new AmnsReactantType();
00144
00145             // get the species
00146             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00147             if (species.ZA == 0) continue;
00148
00149             long ptrCXHandle = 0;
00150
00151             if (amns_debug) System.err.println("[JVM] Setting up table: " + is);
00152             ptrCXHandle = amns.ImasAmnsCCSetupTable(
00153                                                     ptrAmnsHandle
00154                                                     , xx_rx
00155                                                     , ptrReactantsHandle[is]
00156                                                     , error_stat);
00157             ptrAmnsCxHandle[is] = ptrCXHandle;
00158             if (amns_debug) System.err.println("[JVM] Pointer to table: " + ptrCXHandle);
00159             Amns.printErrorCode( error_stat, "ImasAmnsCCSetupTable");
00160
00161             // we will use query and answer objects declared above
00162             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00163             Amns.printErrorCode( error_stat, "ImasAmnsCCQueryTable");
00164
00165             System.out.println("[JVM] AMNS table version = " + answer.number + " for " + answer.string
      + ", IS = " + is);
00166         }
00167
00168         // query the tables (for one given process)
00169         for (is=0; is < ns; is++) {
00170
00171             AmnsReactantType species = new AmnsReactantType();
00172
00173             // get the species
00174             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00175
00176             if (species.ZA == 0) continue;
00177
00178             // we will reuse previous query and answer objects
00179             query.string = "no_of_reactants";
00180             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
```

```
00181             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00182             System.out.println("[JVM] is, " + query.string + " = " + is + ", " + answer.string);
00183 
00184             query.string = "source";
00185             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00186             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00187             System.out.println("[JVM] is, " + query.string + " = " + is + ", " + answer.string);
00188 
00189             query.string = "filled";
00190             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00191             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00192             System.out.println("[JVM] is, " + query.string + " = " + is + ", " + answer.string);
00193 
00194             query.string = "reaction_type";
00195             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00196             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00197             System.out.println("[JVM] is, " + query.string + " = " + is + ", " + answer.string);
00198 
00199             query.string = "reactants";
00200             amns.ImasAmnsCCQueryTable(ptrAmnsCxHandle[is], query, answer, error_stat);
00201             Amns.printErrorCode( error_stat, "after call to ImasAmnsCCQueryTable with query='" +
      query.string + "'");
00202             if (amns_debug) System.err.println("[JVM] is, " + query.string + " = " + is + ", " +
      answer.string);
00203 
00204         }
00205 
00206         // interpolate charge-exchange data, and output the results
00207         set.string = "nowarn";
00208 
00209         for (is=0; is < ns; is++) {
00210 
00211             AmnsReactantType species = new AmnsReactantType();
00212 
00213             // get the species
00214             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00215             if (species.ZA == 0) continue;
00216 
00217             if (amns_debug) System.err.println("[JVM] Interpolate charge-exchange. Loop: " + is);
00218             if (amns_debug) System.err.println("[JVM] Calling ImasAmnsCCSetTable: " +
      ptrAmnsCxHandle[is]);
00219             amns.ImasAmnsCCSetTable(ptrAmnsCxHandle[is], set, error_stat);
00220             Amns.printErrorCode( error_stat, " after ImasAmnsCCSetTable");
00221 
00222             // we have to reshape table to 1D from 2D
00223             // number of elements in 1D array will be nx * ny
00224             // we have to simply copy row by row an create 1D array here
00225             // NOTE! the result of this call is 1D array of rates. We have to reshape it to proper
      shape
00226             double [] te1D = Amns.reshape2DTo1D(te, ny, nx);
00227             double [] ne1D = Amns.reshape2DTo1D(ne, ny, nx);
00228 
00229             if (amns_debug) System.err.println("[JVM] before calling ImasAmnsCCRX1B: " +
      ptrAmnsCxHandle[is]);
00230             double [] rates1D = amns.ImasAmnsCCRX1B(ptrAmnsCxHandle[is], nx * ny, te1D, ne1D,
      error_stat);
00231 
00232             double [][] rates2D = Amns.reshape1DTo2D( rates1D, ny, nx);
00233             System.out.println("Some CX rates for species is=" + is + "\n");
00234             for (ix=0; ix<10; ix++) {
00235                 System.out.println("is=" + is + " rate(is,0," + ix + ")=" + rates2D[0][ix] + "\n");
00236             }
00237         }
00238 
00239         // finalize the tables
00240         for ( is=0; is < ns; is++) {
00241             // get the species
00242             AmnsReactantType species = new AmnsReactantType();
00243 
00244             // get the species
00245             amns.ImasAmnsCCGetReactant(ptrReactantsHandle[is], species);
00246             if (species.ZA == 0) continue;
00247 
00248             amns.ImasAmnsCCFinishTable(ptrAmnsCxHandle[is], error_stat);
00249             Amns.printErrorCode( error_stat, " after ImasAmnsCCFinishTable");
00250         }
00251 
00252         // Clean up reactants data structures
00253         for (is=0; is < ns; is++) {
00254             amns.ImasAmnsCCFinishReactants(ptrReactantsHandle[is]);
00255         }
00256 
00257         amns.ImasAmnsCCFinish(ptrAmnsHandle, error);
```

```
00258     }
00259 }
00260
```

16.22 examples/py/amns_dump_index.py File Reference

Namespaces

- amns_dump_index

Functions

- def amns_dump_index.str2bool (v)

Variables

- amns_dump_index.parser
- amns_dump_index.type
- amns_dump_index.str
- amns_dump_index.help
- amns_dump_index.default
- amns_dump_index.str2bool
- amns_dump_index.nargs
- amns_dump_index.const
- amns_dump_index.True
- amns_dump_index.False
- amns_dump_index.args = parser.parse_args()
- int amns_dump_index.shot = 0

  Define the Database entry.
- int amns_dump_index.run = 1
- amns_dump_index.pulse_index = imas.ids(shot, run)

  Open the database.
- amns_dump_index.AMNS_index = pulse_index.amns_data

  Get the data from the index block.
- amns_dump_index.pulse = imas.ids(e.shot, e.run)

  Print information about the number of available releases.
- amns_dump_index.AMNS = pulse.amns_data

  Acquiring the amns_data from this new shot/run.

16.23 amns_dump_index.py

```
00001 #! /usr/bin/env python
00002 """
00003 This python program dumps the index block for the AMNS system stored
00004 in the amns_data IDS under the user specified by "--user" (defaulting
00005 to the user running the program) in shot 1, run 0 stored in the "amns"
00006 device.
00007
00008 The program reads the shot 1, run 0 data from the "amns" device and
00009 then loops over the "release_entry" entries, printing out a subset of
00010 the data stored "release_entry", before opening and extracting a
00011 subset of data stored in the "amns" IDS pointed to by each entry
00012 stored in "data_entry" under "release_entry".
00013 """
00014 import imas
00015 import argparse
00016 import os
00017
00018 def str2bool(v):
00019     if isinstance(v, bool):
00020        return v
00021     if v.lower() in ('yes', 'true', 't', 'y', '1'):
00022         return True
```

```
00023     elif v.lower() in ('no', 'false', 'f', 'n', '0'):
00024         return False
00025     else:
00026         raise argparse.ArgumentTypeError('Boolean value expected.')
00027
00028 parser = argparse.ArgumentParser(description="Explore AMNS data", epilog
00029 ="""
00030 """, formatter_class=argparse.RawTextHelpFormatter)
00031 parser.add_argument("--user", "-u", type=str, help="Name of the database owner; use 'public' for the
       public database  (default '%s')" % (os.getenv("USER")),default=os.getenv("USER"))
00032 parser.add_argument("--fast", "-f", type=str2bool, nargs='?', const=True, default=False,
       help="Activate fast mode where only high level data is given")
00033 args=parser.parse_args()
00034
00035 print(args)
00036
00037
00038 shot = 0
00039 run = 1
00040
00041 pulse_index = imas.ids(shot, run)
00042 pulse_index.open_env(args.user, 'amns', '3')
00043
00044 AMNS_index = pulse_index.amns_data
00045 AMNS_index.get()
00046
00047 print("%i releases available" % (len(AMNS_index.release)))
00048
00049 for r in AMNS_index.release:
00050
00051   print("  %s, %s, %i entries" % (r.date, r.description, len(r.data_entry)))
00052
00053   for e in r.data_entry:
00054
00055     print("    ", 'Shot/Run = %s/%s' % (e.shot, e.run),)
00056
00057     if not args.fast:
00058       pulse = imas.ids(e.shot, e.run)
00059       pulse.open_env(args.user, 'amns', '3')
00060
00061       AMNS = pulse.amns_data
00062       AMNS.get()
00063       print("      ", 'ids_properties')
00064       print("        ", 'comment:       %s' % (AMNS.ids_properties.comment))
00065       print("        ", 'source:        %s' % (AMNS.ids_properties.source))
00066       print("        ", 'provider:      %s' % (AMNS.ids_properties.provider))
00067       print("        ", 'creation_date: %s' % (AMNS.ids_properties.creation_date))
00068       print("      ", 'code')
00069       print("        ", 'name: %s' % (AMNS.code.name))
00070       print("        ", 'commit: %s' % (AMNS.code.commit))
00071       print("        ", 'version: %s' % (AMNS.code.version))
00072       print("        ", 'repository: %s' % (AMNS.code.repository))
00073
00074       print("      ", 'process')
00075       print("        ", 'Zn=%s, Am=%s, Processes=%s' % (AMNS.z_n, AMNS.a, [ str(p.label) for p in
       AMNS.process ]))
00076
00077       pulse.close()
00078 pulse_index.close()
```

16.24 examples/py/amns_nuclear.py File Reference

Namespaces

- module amns_nuclear

Functions

- def amns_nuclear.nuclear_HB_tt (r1, r2, p1, p2, reac)
- def amns_nuclear.Energy (R)

Variables

- dictionary amns_nuclear.masses
- amns_nuclear.amnsdb = amns.Amns()
- def amns_nuclear.D_D_p_T = nuclear_HB_tt((1,0,2), (1,0,2), (1,0,1), (1,0,3), b'NUC_TT')
- def amns_nuclear.D_D_n_He = nuclear_HB_tt((1,0,2), (1,0,2), (0,0,1), (2,0,3), b'NUC_TT')

- def amns_nuclear.D_T_n_He = nuclear_HB_tt((1,0,2), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
- def amns_nuclear.D_He_p_He = nuclear_HB_tt((1,0,2), (2,0,3), (1,0,1), (2,0,4), b'NUC_TT')
- def amns_nuclear.T_T_n_He = nuclear_HB_tt((1,0,3), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
- amns_nuclear.E = np.array((10**(np.arange(101)/((101-1)/5.0))*10),order='F')
- amns_nuclear.figsize
- def amns_nuclear.Rxn = Energy(Rxn['R']) ; plt.loglog(E, Rxn['RX'].data(E), label="%s\n$E_T=%5.2f,\;E_↩ 1=%5.2f,\;E_2=%5.2f$ MeV" % (Rxn['RX'].result_label, E_T/1e6, E_1/1e6, E_2/1e6))
- amns_nuclear.loc
- amns_nuclear.fontsize
- amns_nuclear.label

16.25 amns_nuclear.py

```
00001 #! /usr/bin/env python
00002 import amns
00003 import numpy as np
00004 import matplotlib.pyplot as plt
00005 import re
00006 import scipy.constants as const
00007
00008 # masses[protons][atomic_mass_number]
00009 masses = { 0:{ 1:const.m_n },
00010            1:{ 1:const.m_p,
00011                2:const.value('deuteron mass'),
00012                3:const.value('triton mass'),
00013              },
00014            2:{ 3:const.value('helion mass'),
00015                4:const.value('alpha particle mass')
00016              },
00017          }
00018
00019 def nuclear_HB_tt(r1, r2, p1, p2, reac):
00020
00021     r = amns.Reactants()
00022     r.add(r1[0],r1[1],r1[2])
00023     r.add(r2[0],r2[1],r2[2])
00024     r.add(p1[0],p1[1],p1[2],lr=1)
00025     r.add(p2[0],p2[1],p2[2],lr=1)
00026     return dict(RX=amnsdb.get_table(reac, r, isotope_resolved=1), R=r)
00027
00028 def Energy(R):
00029     mass = []
00030     mass_excess = 0.0; r = 0; p = 0;
00031     for S in R.value():
00032         zn = int(round(S["ZN"]))
00033         mi = int(round(S["MI"]))
00034         try:
00035             m = masses[zn][mi]
00036             mass.append(m)
00037         except KeyError as exc:
00038             raise NotImplementedError("unknown mass for particle with atomic mass number %s and %s
      protons" %(mi, zn)) from None
00039         if S["LR"] == 0:
00040             r = r + 1
00041             mass_excess = mass_excess + m
00042         else:
00043             p = p + 1
00044             mass_excess = mass_excess - m
00045     assert r == 2, 'coded for two reactants and two products'
00046     assert p == 2, 'coded for two reactants and two products'
00047     E_t =  mass_excess * const.c**2 / const.eV
00048     E_1 = E_t / (mass[2]/mass[3] + 1)
00049     E_2 = E_t / (mass[3]/mass[2] + 1)
00050     return E_t, E_1, E_2
00051
00052 amnsdb = amns.Amns()
00053 amnsdb.set(b'nodebug')
00054
00055 D_D_p_T   = nuclear_HB_tt( (1,0,2), (1,0,2), (1,0,1), (1,0,3), b'NUC_TT') # D(D,p)T
00056 D_D_n_He  = nuclear_HB_tt( (1,0,2), (1,0,2), (0,0,1), (2,0,3), b'NUC_TT') # D(D,n)^3He
00057 D_T_n_He  = nuclear_HB_tt( (1,0,2), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT') # D(T,n)^4He
00058 D_He_p_He = nuclear_HB_tt( (1,0,2), (2,0,3), (1,0,1), (2,0,4), b'NUC_TT') # D(^3He,p)^4He
00059 T_T_n_He  = nuclear_HB_tt( (1,0,3), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT') # T(T,2n)^4He
00060
00061 E=np.array((10**(np.arange(101)/((101-1)/5.0))*10),order='F')
00062
00063 plt.figure(figsize=(28.0/2.54, 20.0/2.54))
00064 plt.clf()
00065
```

```
00066 plt.subplot(2,2,1)
00067 Rxn = D_D_p_T ; E_T, E_1, E_2 = Energy(Rxn['R'])   ; plt.loglog(E, Rxn['RX'].data(E),
       label="%s\n$E_T=%5.2f,\;E_1=%5.2f,\;E_2=%5.2f$ MeV" % (Rxn['RX'].result_label, E_T/1e6, E_1/1e6,
       E_2/1e6) )
00068 print("%s:\n\tE_T=%5.15f,\n\tE_1=%5.15f,\n\tE_2=%5.15f MeV" % (Rxn['RX'].result_label, E_T/1e6,
       E_1/1e6, E_2/1e6))
00069 Rxn = D_D_n_He ; E_T, E_1, E_2 = Energy(Rxn['R'])  ; plt.loglog(E, Rxn['RX'].data(E),
       label="%s\n$E_T=%5.2f,\;E_1=%5.2f,\;E_2=%5.2f$ MeV" % (Rxn['RX'].result_label, E_T/1e6, E_1/1e6,
       E_2/1e6) )
00070 print("%s:\n\tE_T=%5.15f,\n\tE_1=%5.15f,\n\tE_2=%5.15f MeV" % (Rxn['RX'].result_label, E_T/1e6,
       E_1/1e6, E_2/1e6))
00071 Rxn = D_T_n_He ; E_T, E_1, E_2 = Energy(Rxn['R'])  ; plt.loglog(E, Rxn['RX'].data(E),
       label="%s\n$E_T=%5.2f,\;E_1=%5.2f,\;E_2=%5.2f$ MeV" % (Rxn['RX'].result_label, E_T/1e6, E_1/1e6,
       E_2/1e6) )
00072 print("%s:\n\tE_T=%5.15f,\n\tE_1=%5.15f,\n\tE_2=%5.15f MeV" % (Rxn['RX'].result_label, E_T/1e6,
       E_1/1e6, E_2/1e6))
00073 Rxn = D_He_p_He ; E_T, E_1, E_2 = Energy(Rxn['R']) ; plt.loglog(E, Rxn['RX'].data(E),
       label="%s\n$E_T=%5.2f,\;E_1=%5.2f,\;E_2=%5.2f$ MeV" % (Rxn['RX'].result_label, E_T/1e6, E_1/1e6,
       E_2/1e6) )
00074 print("%s:\n\tE_T=%5.15f,\n\tE_1=%5.15f,\n\tE_2=%5.15f MeV" % (Rxn['RX'].result_label, E_T/1e6,
       E_1/1e6, E_2/1e6))
00075 Rxn = T_T_n_He ; E_T, E_1, E_2 = Energy(Rxn['R'])  ; plt.loglog(E, Rxn['RX'].data(E),
       label="%s\n$E_T=%5.2f$ MeV" % (Rxn['RX'].result_label, E_T/1e6) )
00076 print("%s:\n\tE_T=%5.15f MeV" % (Rxn['RX'].result_label, E_T/1e6))
00077
00078 plt.legend(loc='lower right', fontsize=6)
00079 plt.ylabel("$%s$" % D_D_p_T['RX'].result_unit)
00080 plt.xlim(1e2,1e6)
00081 plt.ylim(1e-30,1e-20)
00082
00083 plt.subplot(2,2,2)
00084 plt.loglog(E, D_T_n_He['RX'].data(E)/(D_D_p_T['RX'].data(E)+D_D_n_He['RX'].data(E)), label='DT/DD')
00085 plt.loglog(E, D_T_n_He['RX'].data(E)/T_T_n_He['RX'].data(E), label='DT/TT')
00086 plt.legend(loc='lower right', fontsize='small')
00087 plt.ylabel('ratio')
00088 plt.xlabel('Ion temperature')
00089
00090 plt.subplot(2,2,3)
00091 plt.plot(E, D_T_n_He['RX'].data(E)/(D_D_p_T['RX'].data(E)+D_D_n_He['RX'].data(E)))
00092 plt.xlim([0,5e4])
00093 plt.ylabel('DT/DD')
00094 plt.xlabel('Ion temperature')
00095
00096 plt.subplot(2,2,4)
00097 plt.semilogx(E, D_T_n_He['RX'].data(E)/(D_D_p_T['RX'].data(E)+D_D_n_He['RX'].data(E)))
00098 plt.xlim([5e3,5e5])
00099 plt.ylabel('DT/DD')
00100 plt.xlabel('Ion temperature')
00101
00102 plt.savefig('amns_nuclear.pdf')
00103 plt.savefig('amns_nuclear.png')
00104
00105 amnsdb.finalize()
```

16.26 examples/py/amns_nuclear_densities.py File Reference

Namespaces

- amns_nuclear_densities

Functions

- def amns_nuclear_densities.nuclear_HB_tt (r1, r2, p1, p2, reac)
- def amns_nuclear_densities.advance_densities (N, M, dt, Nt)

Variables

- amns_nuclear_densities.Species = np.array([[1,0,1], [1,0,2], [1,0,3], [2,0,3], [2,0,4], [0,0,1]])
- amns_nuclear_densities.amnsdb = amns.Amns()
- def amns_nuclear_densities.D_D_p_T = nuclear_HB_tt((1,0,2), (1,0,2), (1,0,1), (1,0,3), b'NUC_TT')
- def amns_nuclear_densities.D_D_n_He = nuclear_HB_tt((1,0,2), (1,0,2), (0,0,1), (2,0,3), b'NUC_TT')
- def amns_nuclear_densities.D_T_n_He = nuclear_HB_tt((1,0,2), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
- def amns_nuclear_densities.D_He_p_He = nuclear_HB_tt((1,0,2), (2,0,3), (1,0,1), (2,0,4), b'NUC_TT')
- def amns_nuclear_densities.T_T_n_He = nuclear_HB_tt((1,0,3), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT')
- amns_nuclear_densities.E = np.array((10**(np.arange(101)/((101-1)/5.0))*10),order='F')

- amns_nuclear_densities.M = np.zeros([6,6,6,len(E)])
- amns_nuclear_densities.N = np.outer(np.array([0.0, 5e19, 5e19, 0.0, 0.0, 0.0]), np.ones(len(E)))
- amns_nuclear_densities.Times
- amns_nuclear_densities.N_Times
- amns_nuclear_densities.loc
- amns_nuclear_densities.le
- amns_nuclear_densities.d = pdf.infodict()

16.27 amns_nuclear_densities.py

```
00001 #! /usr/bin/env python
00002 """
00003 Calculate the 0D densities of H, D, T, He-3, He-4, n produced and/or consumed in thermonuclear
       reactions.
00004 The results are plotted at the end-time as a function of temperature, and at around 20 keV as a
       function of time.
00005 Three starting scenarios are explored: 50:50 D:T, 100% D and 100% T.
00006 """
00007 import amns
00008 import numpy as np
00009 from matplotlib.backends.backend_pdf import PdfPages
00010 import matplotlib.pyplot as plt
00011 import re
00012 import datetime
00013
00014 """
00015 H, D, T, He-3, He-4, n
00016 """
00017 Species = np.array([[1,0,1], [1,0,2], [1,0,3], [2,0,3], [2,0,4], [0,0,1]])
00018
00019 def nuclear_HB_tt(r1, r2, p1, p2, reac):
00020     """
00021     Return a dictionary containing the rate coefficient table, reactants/products, transition matrix,
       reactant1, reactant2, scaling factor for identical reactants
00022     """
00023     i1 = np.array([(r == np.array(r1)).all() for r in Species]).argmax()
00024     i2 = np.array([(r == np.array(r2)).all() for r in Species]).argmax()
00025     i3 = np.array([(r == np.array(p1)).all() for r in Species]).argmax()
00026     i4 = np.array([(r == np.array(p2)).all() for r in Species]).argmax()
00027
00028     V=np.zeros([6])
00029
00030     r = amns.Reactants()
00031     r.add(r1[0],r1[1],r1[2])
00032     r.add(r2[0],r2[1],r2[2])
00033     r.add(p1[0],p1[1],p1[2],lr=1)
00034     r.add(p2[0],p2[1],p2[2],lr=1)
00035
00036     V[i1] = V[i1] - 1
00037     V[i2] = V[i2] - 1
00038     V[i3] = V[i3] + 1
00039     V[i4] = V[i4] + 1
00040
00041     if i1 == i2:
00042         f = 0.5
00043     else:
00044         f = 1.0
00045
00046     # check mass and charge
00047     M = 0; Z = 0;
00048     for r_iter in r.value():
00049         if r_iter['LR'] == 0:
00050             M+=r_iter['MI']
00051             Z+=r_iter['ZN']
00052         else:
00053             M+=-r_iter['MI']
00054             Z+=-r_iter['ZN']
00055     print('Before correction: M = %s, Z = %s' % (M, Z))
00056     # try to see if doubling one of the products solves the mass or charge discrepancy
00057     # we assume two products are specified, one of which might have multiplocity two
00058     if (M != 0) or (Z != 0):
00059         for i, r_iter in enumerate(r.value()):
00060             if r_iter['LR'] == 1:
00061                 if ((M - r_iter['MI']) == 0) and ((Z - r_iter['ZN']) == 0):
00062                     if i+1 == 3:
00063                         V[i3] += 1
00064                     else:
00065                         V[i4] += 1
00066                     M+=-r_iter['MI']
00067                     Z+=-r_iter['ZN']
00068                     break
```

```
00069     print('After correction: M = %s, Z = %s' % (M, Z))
00070
00071     return dict(RX=amnsdb.get_table(reac, r, isotope_resolved=1), R=r, V=V, I1=i1, I2=i2, F=f)
00072
00073 def advance_densities(N, M, dt, Nt):
00074     """
00075     Given the transition matrix M, advance the densities N by Nt steps of dt
00076     """
00077     assert N.shape[0] == M.shape[0]
00078     assert M.shape[0] == M.shape[1]
00079     assert M.shape[0] == M.shape[2]
00080     Times = np.arange(Nt+1)*dt
00081     N_Times = np.zeros([len(Times),N.shape[0],N.shape[1]])
00082     N_Times[0] = N
00083     for t,T in enumerate(Times[1:]):
00084         N = N + np.einsum('ijkl,il,jl->kl', M, N, N) * dt
00085         N_Times[t+1] = N
00086     return Times, N_Times
00087
00088 amnsdb = amns.Amns()
00089 """
00090 Consider the following reactions
00091 """
00092 D_D_p_T   = nuclear_HB_tt( (1,0,2), (1,0,2), (1,0,1), (1,0,3), b'NUC_TT') # D(D,p)T
00093 D_D_n_He  = nuclear_HB_tt( (1,0,2), (1,0,2), (0,0,1), (2,0,3), b'NUC_TT') # D(D,n)^3He
00094 D_T_n_He  = nuclear_HB_tt( (1,0,2), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT') # D(T,n)^4He
00095 D_He_p_He = nuclear_HB_tt( (1,0,2), (2,0,3), (1,0,1), (2,0,4), b'NUC_TT') # D(^3He,p)^4He
00096 T_T_n_He  = nuclear_HB_tt( (1,0,3), (1,0,3), (0,0,1), (2,0,4), b'NUC_TT') # T(T,2n)^4He
00097
00098 """
00099 Temperature axis
00100 """
00101 E=np.array((10**(np.arange(101)/((101-1)/5.0))*10),order='F')
00102 E=np.logspace(4,5,101)
00103
00104 """
00105 Calculate the transition matrix
00106 """
00107 M = np.zeros([6,6,6,len(E)])
00108 M[D_D_p_T['I1'], D_D_p_T['I2']]     = M[D_D_p_T['I1'], D_D_p_T['I2']]     + np.outer(D_D_p_T['V'],
       D_D_p_T['RX'].data(E)*D_D_p_T['F'])
00109 M[D_D_n_He['I1'], D_D_n_He['I2']]   = M[D_D_n_He['I1'], D_D_n_He['I2']]   + np.outer(D_D_n_He['V'],
       D_D_n_He['RX'].data(E)*D_D_n_He['F'])
00110 M[D_T_n_He['I1'] ,D_T_n_He['I2']]   = M[D_T_n_He['I1'], D_T_n_He['I2']]   + np.outer(D_T_n_He['V'],
       D_T_n_He['RX'].data(E)*D_T_n_He['F'])
00111 M[D_He_p_He['I1'], D_He_p_He['I2']] = M[D_He_p_He['I1'], D_He_p_He['I2']] + np.outer(D_He_p_He['V'],
       D_He_p_He['RX'].data(E)*D_He_p_He['F'])
00112 M[T_T_n_He['I1'], T_T_n_He['I2']]   = M[T_T_n_He['I1'], T_T_n_He['I2']]   + np.outer(T_T_n_He['V'],
       T_T_n_He['RX'].data(E)*T_T_n_He['F'])
00113
00114 with PdfPages('amns_nuclear_densities.pdf') as pdf:
00115
00116     """
00117 Start with a D-T 50:50 mixture
00118 """
00119     N = np.outer(np.array([0.0, 5e19, 5e19, 0.0, 0.0, 0.0]), np.ones(len(E)))
00120     # by comparison with dt=0.01 and 0.001, dt=0.1 should be better than 1% accurate
00121     Times, N_Times = advance_densities(N, M, 0.1, 10000)
00122
00123     plt.figure()
00124     plt.semilogy(E,N_Times[-1].T)
00125     plt.ylim(1e16,1.1e20)
00126     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00127     plt.xlabel('Temperature [eV]')
00128     plt.ylabel('Density [$m^{-3}$]')
00129     plt.title('Densities as a function of temperature at T = %s seconds\nStarting with 50:50 D-T' %
       (Times[-1],))
00130     pdf.savefig() ; plt.close()
00131
00132     plt.figure() ; Ie = (len(E)-1)//3
00133     plt.semilogy(Times,N_Times[:,:,Ie])
00134     plt.ylim(1e16,1.1e20)
00135     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00136     plt.xlabel('Time [s]') ;
00137     plt.ylabel('Density [$m^{-3}$]')
00138     plt.title('Densities as a function of time for temperature = %.1f [eV]\nStarting with 50:50 D-T' %
       (E[Ie],))
00139     pdf.savefig() ; plt.close()
00140
00141     """
00142 Start with a pure D plasma
00143 """
00144     N = np.outer(np.array([0.0, 1e20, 0.0, 0.0, 0.0, 0.0]), np.ones(len(E)))
00145     Times, N_Times = advance_densities(N, M, 0.1, 10000)
00146
00147     plt.figure()
00148     plt.semilogy(E,N_Times[-1].T)
```

```
00149     plt.ylim(1e16,1.1e20)
00150     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00151     plt.xlabel('Temperature [eV]')
00152     plt.ylabel('Density [$m^{-3}$]')
00153     plt.title('Densities as a function of temperature at T = %s seconds\nStarting with a pure D
      plasma' % (Times[-1],))
00154     pdf.savefig() ; plt.close()
00155
00156     plt.figure() ; Ie = (len(E)-1)//3
00157     plt.semilogy(Times,N_Times[:,:,Ie])
00158     plt.ylim(1e16,1.1e20)
00159     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00160     plt.xlabel('Time [s]') ;
00161     plt.ylabel('Density [$m^{-3}$]')
00162     plt.title('Densities as a function of time for temperature = %.1f [eV]\nStarting with a pure D
      plasma' % (E[Ie],))
00163     pdf.savefig() ; plt.close()
00164
00165     """
00166 Start with a pure T plasma
00167 """
00168     N = np.outer(np.array([0.0, 0.0, 1e20, 0.0, 0.0, 0.0]), np.ones(len(E)))
00169     Times, N_Times = advance_densities(N, M, 0.1, 10000)
00170
00171     plt.figure()
00172     plt.semilogy(E,N_Times[-1].T)
00173     plt.ylim(1e16,1.1e20)
00174     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00175     plt.xlabel('Temperature [eV]')
00176     plt.ylabel('Density [$m^{-3}$]')
00177     plt.title('Densities as a function of temperature at T = %s seconds\nStarting with a pure T
      plasma' % (Times[-1],))
00178     pdf.savefig() ; plt.close()
00179
00180     plt.figure() ; Ie = (len(E)-1)//3
00181     plt.semilogy(Times,N_Times[:,:,Ie])
00182     plt.ylim(1e16,1.1e20)
00183     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00184     plt.xlabel('Time [s]') ;
00185     plt.ylabel('Density [$m^{-3}$]')
00186     plt.title('Densities as a function of time for temperature = %.1f [eV]\nStarting with a pure T
      plasma' % (E[Ie],))
00187     pdf.savefig() ; plt.close()
00188
00189     """
00190 Start with a pure 99:1 D:T plasma
00191 """
00192     N = np.outer(np.array([0.0, 0.99e20, 0.01e20, 0.0, 0.0, 0.0]), np.ones(len(E)))
00193     Times, N_Times = advance_densities(N, M, 0.1, 10000)
00194
00195     plt.figure()
00196     plt.semilogy(E,N_Times[-1].T)
00197     plt.ylim(1e16,1.1e20)
00198     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00199     plt.xlabel('Temperature [eV]')
00200     plt.ylabel('Density [$m^{-3}$]')
00201     plt.title('Densities as a function of temperature at T = %s seconds\nStarting with a 99:1 D:T
      plasma' % (Times[-1],))
00202     pdf.savefig() ; plt.close()
00203
00204     plt.figure() ; Ie = (len(E)-1)//3
00205     plt.semilogy(Times,N_Times[:,:,Ie])
00206     plt.ylim(1e16,1.1e20)
00207     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00208     plt.xlabel('Time [s]') ;
00209     plt.ylabel('Density [$m^{-3}$]')
00210     plt.title('Densities as a function of time for temperature = %.1f [eV]\nStarting with a 99:1 D:T
      plasma' % (E[Ie],))
00211     pdf.savefig() ; plt.close()
00212
00213     """
00214 Start with a pure 1:99 D:T plasma
00215 """
00216     N = np.outer(np.array([0.0, 0.01e20, 0.99e20, 0.0, 0.0, 0.0]), np.ones(len(E)))
00217     Times, N_Times = advance_densities(N, M, 0.1, 10000)
00218
00219     plt.figure()
00220     plt.semilogy(E,N_Times[-1].T)
00221     plt.ylim(1e16,1.1e20)
00222     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00223     plt.xlabel('Temperature [eV]')
00224     plt.ylabel('Density [$m^{-3}$]')
00225     plt.title('Densities as a function of temperature at T = %s seconds\nStarting with a 1:99 D:T
      plasma' % (Times[-1],))
00226     pdf.savefig() ; plt.close()
00227
00228     plt.figure() ; Ie = (len(E)-1)//3
```

```
00229     plt.semilogy(Times,N_Times[:,:,Ie])
00230     plt.ylim(1e16,1.1e20)
00231     plt.legend(['H','D','T','3He','4He','n'],loc=0)
00232     plt.xlabel('Time [s]') ;
00233     plt.ylabel('Density [$m^{-3}$]')
00234     plt.title('Densities as a function of time for temperature = %.1f [eV]\nStarting with a 1:99 D:T
      plasma' % (E[Ie],))
00235     pdf.savefig() ; plt.close()
00236
00237     # We can also set the file's metadata via the PdfPages object:
00238     d = pdf.infodict()
00239     d['Title'] = '0D calculations of density evolution arising from thermonuclear fusion'
00240     d['Author'] = 'David Coster'
00241     d['Subject'] = 'Using the AMNS library to follow the densities of thermonuclearly reacting species
      in a 0D approximation'
00242     d['Keywords'] = 'AMNS, thermonuclear, 0D'
00243     d['CreationDate'] = datetime.datetime(2019, 8, 1)
00244     d['ModDate'] = datetime.datetime.today()
00245
00246 amnsdb.finalize()
00247
```

16.28 examples/py/amns_scan.py File Reference

Namespaces

- amns_scan

Functions

- def amns_scan.summarize_data (shot, run, USER, IDS, DATAVERSION, file=None)
- def amns_scan.summarize (USER, IDS, DATAVERSION, file=None)

Variables

- amns_scan.parser
- amns_scan.type
- amns_scan.str
- amns_scan.help
- amns_scan.default
- amns_scan.args = parser.parse_args()
- def amns_scan.f = open('amns_scan_%s_%s.tex' % (os.getenv("IMAS_VERSION"),args.user), 'w') ; summarize(args.user, 'amns', os.getenv("IMAS_VERSION"), f) ; f.close()

16.29 amns_scan.py

```
00001 #! /usr/bin/env python
00002 """
00003 This python program scans through the amns IDS's and then outputs a
00004 LaTeX file containing information about the available reactions.
00005
00006 pdflatex can then be used to produce a PDF document..
00007 """
00008 import imas
00009 import time
00010 import re
00011 import os
00012 import pickle
00013 import argparse
00014
00015 def summarize_data(shot, run, USER, IDS, DATAVERSION, file=None):
00016     """
00017     Produce LaTeX summarizing the data contained withi a particular shot/run
00018     """
00019     amns = imas.ids(shot,run)
00020     amns.open_env(USER, IDS, DATAVERSION)
00021     amns.amns_data.get()
00022     AMNS = amns.amns_data
00023
00024     print("ids\_properties/comment: \\spverb|%s|" % (AMNS.ids_properties.comment,), file=file)
00025     print(", file=file)
00026     print("ids\_properties/source: \\spverb|%s|" % (AMNS.ids_properties.source,), file=file)
```

```
00027     print(”, file=file)
00028     print("ids\_properties/provider: \\spverb|%s|" % (AMNS.ids_properties.provider,), file=file)
00029     print(”, file=file)
00030     print("ids\_properties/creation\_date: \\spverb|%s|" % (AMNS.ids_properties.creation_date,),
      file=file)
00031     print(”, file=file)
00032     print("code/name: \\spverb|%s|" % (AMNS.code.name,), file=file)
00033     print(”, file=file)
00034     print("code/commit: \\spverb|%s|" % (AMNS.code.commit,), file=file)
00035     print(”, file=file)
00036     print("code/version: \\spverb|%s|" % (AMNS.code.version,), file=file)
00037     print(”, file=file)
00038     print("code/repository: \\spverb|%s|" % (AMNS.code.repository,), file=file)
00039     print(”, file=file)
00040     print(”, file=file)
00041     print(’Charge and mass: ZN=%s, AMN=%s’ % (AMNS.z_n, AMNS.a), file=file)
00042     print(”, file=file)
00043
00044     columns=[’0.07’, ’0.10’, ’0.07’, ’0.06’, ’0.20’, ’0.20’, ’0.15’]
00045     print(’\\begin{longtable}[]{@{}|l|l|l|l|l|l|l|l|l|l|@{}}’, file=file)
00046     print(’  \\toprule’, file=file)
00047     print(’  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item IND \\item
      PROC \\end{compactitem} \\end{minipage} &’ % (columns[0],), file=file)
00048     print(’  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright LABEL\\strut \\end{minipage} &’ %
      (columns[1],), file=file)
00049     print(’  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item NDIM
      \\item UNITS \\item COORD \\end{compactitem} \\end{minipage} &’ % (columns[2],), file=file)
00050     print(’  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright INTERP FUN\\strut \\end{minipage} &’ %
      (columns[3],), file=file)
00051     print(’  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item SOURCE
      \\item PROVIDER \\item CITATION \\end{compactitem} \\end{minipage} &’ % (columns[4],), file=file)
00052     print(’  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright COORD LABELS\\strut \\end{minipage} &’
      % (columns[5],), file=file)
00053     print(’  \\begin{minipage}[b]{%s\\columnwidth}\\raggedright REACTION\\strut
      \\end{minipage}\\tabularnewline’ % (columns[6],), file=file)
00054     print(’  \\midrule’, file=file)
00055     print(’  \\endhead’, file=file)
00056
00057     lines=[]
00058     nproc = len(AMNS.process)
00059     for k in range(nproc) :
00060         process=AMNS.process[k]
00061         if k>0:
00062             print(’  \midrule’, file=file)
00063         line = []
00064         line.append(k+1)
00065         line.append(process.label)
00066         print(’  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item %s
      \\item \\spverb|%s| \\end{compactitem} \\end{minipage} &’ % (columns[0], k+1, process.label),
      file=file)
00067         line.append(process.result_label)
00068         print(’  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright \\spverb|%s|\\strut
      \\end{minipage} &’ % (columns[1], process.result_label), file=file)
00069         line.append(process.table_dimension)
00070         line.append(process.result_units)
00071         line.append(process.coordinate_index)
00072         print(’  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item %s
      \\item $%s$ \\item %s \\end{compactitem} \\end{minipage} &’ % (columns[2], process.table_dimension,
      process.result_units, process.coordinate_index), file=file)
00073         line.append(process.result_transformation)
00074         print(’  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright %s\\strut \\end{minipage} &’ %
      (columns[3], process.result_transformation), file=file)
00075         line.append(’%s\n%s\n%s’ % (process.source.replace(’/’,’/ ’), process.provider,
      process.citation))
00076         print(’  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright \\begin{compactitem} \\item
      \\spverb|%s| \\item \\spverb|%s| \\item \\spverb|%s| \\end{compactitem} \\end{minipage} &’ %
      (columns[4], process.source.replace(’/’,’/ ’), process.provider, process.citation), file=file)
00077         print(’  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright ’ % (columns[5],) , end=”,
      file=file)
00078         entry=""
00079         l=process.coordinate_index
00080         if l > 0 :
00081             for m in range(len(AMNS.coordinate_system[l-1].coordinate)) :
00082                 if m>0:
00083                     print(’\\\\’, end=”, file=file)
00084                 print(’%s: %s ($%s$)’ % (m+1, AMNS.coordinate_system[l-1].coordinate[m].label,
      AMNS.coordinate_system[l-1].coordinate[m].units), end=”, file=file)
00085                 entry=entry+’ %s: %s ($%s$)\n’ % (m+1,
      AMNS.coordinate_system[l-1].coordinate[m].label, AMNS.coordinate_system[l-1].coordinate[m].units)
00086         line.append(entry)
00087         print(’\\strut \\end{minipage} &’, file=file)
00088         print(’  \\begin{minipage}[t]{%s\\columnwidth}\\raggedright ’ % (columns[6],), end=”,
      file=file)
00089         if len(process.reactants)+len(process.products) > 0:
00090             print(’$’, end=’ ’, file=file)
00091             for m in range(len(process.reactants)) :
00092                 if m != 0:
```

```
00093                     print(' + ', end=' ', file=file)
00094                 if process.reactants[m].element[0].multiplicity > 1:
00095                     print('%i ' % (process.reactants[m].element[0].multiplicity), end=' ', file=file)
00096                 print('\\textrm{%s}^{' % (process.reactants[m].label.replace('|','{\\textbar}')),
    end=' ', file=file)
00097                 if process.reactants[m].relative_charge == 1:
00098                     print('z', end=' ', file=file)
00099                 if process.reactants[m].relative_charge == -1:
00100                     print('}', end=' ', file=file)
00101                 else:
00102                     print('%+i}' % (int(process.reactants[m].charge)), end=' ', file=file)
00103             print(' $ \\\\ \\center $ \\rightarrow $ \\\\ \\raggedleft $', end=' ', file=file)
00104             for m in range(len(process.products)) :
00105                 if m != 0:
00106                     print(' + ', end=' ', file=file)
00107                 if process.products[m].element[0].multiplicity > 1:
00108                     print('%i ' % (process.products[m].element[0].multiplicity), end=' ', file=file)
00109                 print('\\textrm{%s}^{' % (process.products[m].label.replace('|','{\\textbar}')), end='
    ', file=file)
00110                 if process.products[m].relative_charge == 1:
00111                     print('z', end=' ', file=file)
00112                 if process.products[m].relative_charge == -1:
00113                     print('}', end=' ', file=file)
00114                 else:
00115                     print('%+i}' % (int(process.products[m].charge)), end=' ', file=file)
00116             print('$', end='', file=file)
00117         print('\\strut \\end{minipage}\\tabularnewline', file=file)
00118         line.append('')
00119         lines.append(line)
00120 
00121     print('  \\bottomrule', file=file)
00122     print('\\end{longtable}', file=file)
00123     output = open('amns_scan.pckl', 'wb')
00124     pickle.dump(lines,output)
00125 
00126     return
00127 
00128 
00129 def summarize (USER, IDS, DATAVERSION, file=None):
00130     """
00131     Produce LaTeX describing the AMNS data stored by a particular user
00132     """
00133     periodic = re.compile("[A-Z][a-z]*").findall
    ("""HHeLiBeBCNOFNeNaMgAlSiPSClArKCaScTiVCrMnFeCoNiCuZnGaGeAsSeBrKrRbSrYZrNbMoTcRuRhPdAgCdInSnSbTeIXeCsBaLaCePrNdPmSmEu
00134 
00135     amns_index = imas.ids(0,1)
00136     amns_index.open_env(USER, IDS, DATAVERSION)
00137     amns_index.amns_data.get()
00138     AMNS = amns_index.amns_data
00139 
00140     nversions = len(AMNS.release)
00141 
00142     print("'\\documentclass[10pt]{article}
00143 
00144 \\usepackage[a4paper,margin=1in,landscape]{geometry}
00145 
00146 \\setlength{\\pdfpagewidth}{\\paperwidth}
00147 \\setlength{\\pdfpageheight}{\\paperheight}
00148 
00149 \\usepackage{longtable,booktabs,paralist,parskip,spverbatim,hyperref}
00150 
00151 \\setdefaultleftmargin{0.5em}{0.5em}{0.5em}{0.5em}{0.5em}{0.5em}
00152 
00153 \\begin{document}
00154 
00155 \\tableofcontents"', file=file)
00156     print('', file=file)
00157     print('\\newpage\\section{AMNS reactions %s (user %s)}' % (DATAVERSION, USER), file=file)
00158     print('', file=file)
00159     print('Based on data from USER "%s", using the IDS "%s" and DATAVERSION "%s".' % (USER, IDS,
    DATAVERSION), file=file)
00160     print('', file=file)
00161     print('Prepared at %s' % (time.strftime("%Y-%m-%d %H:%M:%S UTC",time.gmtime(time.time())),),
    file=file)
00162     print('', file=file)
00163 
00164     for i in range(nversions) :
00165         default=''
00166         if i+1 == nversions:
00167             default = '[DEFAULT]'
00168         release=AMNS.release[i]
00169         print('\\newpage\\subsection{Release %s (%s) %s}' % (i+1, release.date, default), file=file)
00170         print('', file=file)
00171         print('Description: \\spverb|%s|' % (release.description,), file=file)
00172         print('', file=file)
00173         print('Date: \\spverb|%s|' % (release.date,), file=file)
00174         print('', file=file)
```

```
00175
00176         nrelease =  len(release.data_entry)
00177         for j in range(nrelease) :
00178             data=release.data_entry[j]
00179             if data.shot >= 1000 :
00180                 print('\\newpage\\subsubsection{Data for %s-%s (Release %s on %s)}' %
      (int(data.shot/1000), periodic[data.shot%1000-1], i+1, release.date), file=file)
00181             else :
00182                 print('\\newpage\\subsubsection{Data for %s (Release %s on %s)}' %
      (periodic[data.shot-1], i+1, release.date), file=file)
00183             print('', file=file)
00184             print('The data is stored in SHOT=%s RUN=%s' % (data.shot, data.run), file=file)
00185             print('', file=file)
00186             print('Description: \\spverb|%s|' % (data.description,), file=file)
00187             print('', file=file)
00188             summarize_data(data.shot, data.run, USER, IDS, DATAVERSION, file)
00189             print('', file=file)
00190
00191     print('\end{document}', file=file)
00192
00193     return
00194
00195 parser = argparse.ArgumentParser(description="Catalogue the AMNS data available on the current
      system", epilog
00196 ="""
00197 """, formatter_class=argparse.RawTextHelpFormatter)
00198 parser.add_argument("--user", "-u", type=str, help="Name of the database owner; use 'public' for the
      public database  (default '%s')" % (os.getenv("USER")),default=os.getenv("USER"))
00199 args=parser.parse_args()
00200
00201 try:
00202     print('Scanning for data stored by %s' %(args.user))
00203     f=open('amns_scan_%s_%s.tex' % (os.getenv("IMAS_VERSION"),args.user), 'w') ; summarize(args.user,
      'amns', os.getenv("IMAS_VERSION"), f) ; f.close()
00204     print('Please run\n  pdflatex amns_scan_%s_%s.tex\ntwice' % (os.getenv("IMAS_VERSION"),args.user))
00205 except:
00206     print('For an unknown reason no AMNS data can be found for %s' % (args.user))
00207     print('Please try "imasdbs -t amns -u %s" to see if any data has been installed' % (args.user))
```

16.30 examples/py/amns_test.py File Reference

Namespaces

- amns_test

Variables

- list amns_test.nx = [101, 101]
- amns_test.te = np.empty(nx, dtype=np.float64, order='F')
- amns_test.ne = np.empty(nx, dtype=np.float64, order='F')
- amns_test.amnsdb = amns.Amns()
- amns_test.r = amns.Reactants()
- amns_test.lr
- amns_test.table = amnsdb.get_table(b"CX", r)
- amns_test.res = table.data(te, ne)
- amns_test.f = open('amns_test_cx_te_c.out', 'w')
- amns_test.iy = int(nx[1]/2)
- int amns_test.height = 210/25.4
- int amns_test.width = 297/25.4
- amns_test.figsize
- amns_test.res_max = m.ceil(m.log10(res.max()))
- amns_test.res_min = m.floor(m.log10(res.min()))
- float amns_test.v = 10.0**np.arange(res_min,res_max+1)
- amns_test.norm
- amns_test.ticks
- amns_test.format

16.31 amns_test.py

```
00001 #! /usr/bin/env python
00002
00003 import matplotlib.pyplot as plt
00004 from matplotlib import colors, ticker
00005 import amns
00006 import numpy as np
00007 import math as m
00008
00009 nx = [101, 101]
00010 te = np.empty(nx, dtype=np.float64, order='F')
00011 ne = np.empty(nx, dtype=np.float64, order='F')
00012
00013
00014 for ix in range(nx[0]):
00015     for iy in range(nx[1]):
00016         te[ix, iy] = 10.0 ** ( ix / ((nx[0] - 1) / 5.0) ) * 0.1
00017         ne[ix, iy] = 10.0 ** ( iy / ((nx[1] - 1) / 10.0) ) * 1.0e15
00018
00019 amnsdb = amns.Amns()
00020
00021 r = amns.Reactants()
00022 r.add(6,1,12)
00023 r.add(1,0,2)
00024 r.add(6,0,12,lr=1)
00025 r.add(1,1,2,lr=1)
00026 print("Reactants:", r)
00027
00028 table = amnsdb.get_table(b"CX", r)
00029 print("table.no_of_reactants")
00030 print("table.no_of_reactants", table.no_of_reactants)
00031
00032 res = table.data(te, ne)
00033
00034 #print res
00035
00036 f = open('amns_test_cx_te_c.out', 'w')
00037 f.write('# te ne CX_rate\n')
00038 for ix in range(nx[0]):
00039     iy = int(nx[1]/2)
00040     f.write(str(te[ix, iy]) + " " + str(ne[ix, iy]) + " " + str(res[ix, iy]*1.0e6) + "\n")
00041
00042 f.close()
00043
00044 r = amns.Reactants()
00045 r.add(6,1,12)
00046 r.add(0,-1,0)
00047 r.add(6,2,12,lr=1)
00048 r.add(0,-1,0,lr=1)
00049 print("Reactants:", r)
00050
00051 table = amnsdb.get_table(b"EI", r)
00052 #print table.no_of_reactants
00053 res = table.data(te, ne)
00054
00055 height=210/25.4
00056 width=297/25.4
00057 plt.figure(1, figsize=(width,height))
00058 plt.ion()
00059 plt.clf()
00060 #plt.contourf(te, ne, res, locator=ticker.LogLocator())
00061 res_max=m.ceil(m.log10(res.max()))
00062 res_min=m.floor(m.log10(res.min()))
00063 res_min=max(res_max-20,res_min)
00064 v=10.0**np.arange(res_min,res_max+1)
00065 plt.contourf(te, ne, res, v, norm=colors.LogNorm())
00066 plt.xscale('log')
00067 plt.xlabel('Te')
00068 plt.yscale('log')
00069 plt.ylabel('ne')
00070 plt.title('EI')
00071 plt.colorbar(ticks=v,format=ticker.LogFormatter(10))
00072 #plt.show()
00073 plt.savefig('amns_test.pdf')
00074
00075 amnsdb.finalize()
```

16.32 examples/py/amns_test_adf11_versions.py File Reference

Namespaces

- amns_test_adf11_versions

Variables

- amns_test_adf11_versions.parser
- amns_test_adf11_versions.type
- amns_test_adf11_versions.str
- amns_test_adf11_versions.help
- amns_test_adf11_versions.default
- amns_test_adf11_versions.int
- amns_test_adf11_versions.args = parser.parse_args()
- amns_test_adf11_versions.periodic = re.compile("[A-Z][a-z]∗").findall ()
- amns_test_adf11_versions.te = np.logspace(-1,5,121)
- int amns_test_adf11_versions.ne = te∗0+5e19
- amns_test_adf11_versions.amnsdb_df = amns.Amns(version_user = args.user)
- list amns_test_adf11_versions.amnsdb_v = [ ]
- amns_test_adf11_versions.r = amns.Reactants()
- amns_test_adf11_versions.lr
- amns_test_adf11_versions.T = A.get_table("LR", r)
- amns_test_adf11_versions.label
- amns_test_adf11_versions.loc
- amns_test_adf11_versions.fontsize
- amns_test_adf11_versions.framealpha

16.33 amns_test_adf11_versions.py

```
00001 #! /usr/bin/env python
00002 from matplotlib.backends.backend_pdf import PdfPages
00003 import matplotlib.pyplot as plt
00004 from matplotlib import colors, ticker
00005 import amns
00006 import numpy as np
00007 import argparse
00008 import os
00009 import re
00010
00011 parser = argparse.ArgumentParser(description="Compare different versions of the AMNS adf11 data",
       epilog
00012 ="""
00013 """, formatter_class=argparse.RawTextHelpFormatter)
00014 parser.add_argument("--user", "-u", type=str, help="Name of the database owner; use 'public' for the
       public database  (default '%s')" % (os.getenv("USER")),default=os.getenv("USER"))
00015 parser.add_argument("--zn", type=int, default=10, help="Nuclear charge of the species to be examined")
00016 args=parser.parse_args()
00017
00018 periodic = re.compile("[A-Z][a-z]*").findall
       ("""HHeLiBeBCNOFNeNaMgAlSiPSClArKCaScTiVCrMnFeCoNiCuZnGaGeAsSeBrKrRbSrYZrNbMoTcRuRhPdAgCdInSnSbTeIXeCsBaLaCePrNdPmSmEu
00019
00020 te=np.logspace(-1,5,121)
00021 ne=te*0+5e19
00022
00023 amnsdb_df = amns.Amns(version_user = args.user)
00024 print(amnsdb_df.version)
00025
00026 amnsdb_v = []
00027 for v in range(amnsdb_df.version[0]):
00028   amnsdb_v.append(amns.Amns(version_user = args.user, version_number = v+1))
00029
00030 with PdfPages('amns_%s_versions_%s.pdf' % (periodic[args.zn-1], args.user)) as pdf:
00031   for ic in range(args.zn):
00032     plt.clf()
00033     r = amns.Reactants()
00034     r.add(args.zn,ic,0)
00035     r.add(args.zn,ic,0,lr=1)
00036     for A in amnsdb_v:
00037       try:
00038         T = A.get_table("LR", r)
00039         plt.loglog(te, T.data(te,ne), label='%s/%s/%s)' % (A.version[0],A.version[1],T.prop_creation))
00040       except:
00041         print('No LR data found for %s/%s' % (args.zn, ic))
00042     plt.xlabel('Te')
00043     plt.ylabel('Line radiation')
00044     plt.legend(loc=0, fontsize=7, framealpha=0.5)
00045     plt.title('%s$^{%s+}$   n$_e$=%s' % (periodic[args.zn-1], ic, ne[0]))
00046     pdf.savefig() ; plt.close()
```

```
00047   for ic in range(args.zn):
00048     plt.clf()
00049     r = amns.Reactants()
00050     r.add(args.zn,ic,0)
00051     r.add(0,-1,0)
00052     r.add(args.zn,ic+1,0,lr=1)
00053     r.add(0,-1,0,lr=1)
00054     for A in amnsdb_v:
00055       try:
00056         T = A.get_table("EI", r)
00057         plt.loglog(te, T.data(te,ne), label='%s/%s/%s)' % (A.version[0],A.version[1],T.prop_creation))
00058       except:
00059         print('No EI data found for %s/%s' % (args.zn, ic))
00060     plt.xlabel('Te')
00061     plt.ylabel('Electron impact ionization rate')
00062     plt.legend(loc=0, fontsize=7, framealpha=0.5)
00063     plt.title('%s$^{%s+}$   n$_e$=%s' % (periodic[args.zn-1], ic, ne[0]))
00064     pdf.savefig() ; plt.close()
00065
00066 for A in amnsdb_v:
00067   A.finalize()
00068 amnsdb_df.finalize()
```

16.34 examples/py/amns_test_bms.py File Reference

Namespaces

- amns_test_bms

Functions

- def amns_test_bms.bms_calc (n)
- def amns_test_bms.plot (eng, dens, tion, te, res, coordinates, results, reactants)

16.35 amns_test_bms.py

```
00001 #! /usr/bin/env python
00002
00003 import matplotlib.pyplot as plt
00004 from matplotlib import colors, ticker
00005 import amns
00006 import numpy as np
00007 import math as m
00008 import itertools
00009 import time
00010
00011 def bms_calc(n):
00012     assert n>1, "n should be an integer greater than 1"
00013     nx = [n, n, n, n]
00014
00015     ENG  = np.logspace(np.log10(2.000E+04), np.log10(1.500E+05), nx[0])
00016     DENS = np.logspace(np.log10(1.000E+17), np.log10(1.000E+21), nx[1])
00017     TION = np.logspace(np.log10(1.000E+02), np.log10(2.000E+04), nx[2])
00018     TE   = np.logspace(np.log10(1.000E+02), np.log10(1.000E+04), nx[3])
00019
00020     start = time.time()
00021 #    eng, dens, tion, te = np.array([[w,x,y,z] for z in TE for y in TION for x in DENS for w in
      ENG]).T
00022     te, tion, dens, eng = np.array(list(itertools.product(TE,TION,DENS,ENG))).T
00023     end = time.time()
00024     print('\nSetting up the coordinates (%s elements) took %0.3f seconds' % (np.product(nx),
      end-start))
00025
00026     eng=eng.reshape(nx[::-1])
00027     dens=dens.reshape(nx[::-1])
00028     tion=tion.reshape(nx[::-1])
00029     te=te.reshape(nx[::-1])
00030
00031     amnsdb = amns.Amns()
00032
00033     r = amns.Reactants()
00034     r.add(1,0,1)
00035     r.add(1,1,1)
00036     r.add(1,1,1,lr=1)
00037     r.add(1,0,1,lr=1)
00038     print("Reactants:", r)
00039
00040     start = time.time()
```

```
00041     table = amnsdb.get_table(b"BMS", r)
00042     end = time.time()
00043     print('Setting up the bms table took %0.3f seconds' % (end-start))
00044
00045     print("table.no_of_reactants")
00046     print("table.no_of_reactants", table.no_of_reactants)
00047
00048     start = time.time()
00049     res = table.data(eng.ravel(), dens.ravel(), tion.ravel(), te.ravel()).reshape(nx[::-1])
00050     end = time.time()
00051     print('Calculating the values for %s elements took %0.3f seconds' % (res.size, end-start))
00052
00053     coordinates=table.coordinates.split(" ")
00054     results=table.result_label+"/"+table.result_unit
00055     reactants=table.reactants
00056
00057     start = time.time()
00058     table.finalize()
00059     amnsdb.finalize()
00060     end = time.time()
00061     print('Finishing took %0.3f seconds\n' % (end-start))
00062
00063     return eng, dens, tion, te, res, coordinates, results, reactants
00064
00065 def plot(eng, dens, tion, te, res, coordinates, results, reactants):
00066
00067     x1=np.array([20000.0000, 23100.0000, 26670.0000, 30800.0000, 35570.0000, 41070.0000, 47430.0000,
      54770.0000, 63250.0000, 73040.0000, 84350.0000, 97400.0000, 112500.000, 129900.000, 150000.000])
00068     x2=np.array([9.99999984E+16, 2.78299999E+17, 7.74300003E+17, 2.15400004E+18, 5.99499974E+18,
      1.66800004E+19, 4.64199999E+19, 1.29199996E+20, 3.59400022E+20, 1.00000002E+21])
00069     x3=np.array([100.000000, 180.199997, 324.600006, 584.799988, 1054.00000, 1898.00000, 3420.00000,
      6162.00000, 11100.0000, 20000.0000])
00070     x4=np.array([100.000000, 166.800003, 278.299988, 464.200012, 774.299988, 1292.00000, 2154.00000,
      3594.00000, 5995.00000, 10000.0000])
00071
00072     D1=np.array([1.34100006E-07, 1.30900006E-07, 1.28400004E-07, 1.25499994E-07, 1.20600006E-07,
      1.15600002E-07, 1.09900000E-07, 1.03799998E-07, 9.80599992E-08, 9.36700033E-08, 9.02900013E-08,
      8.77199966E-08, 8.49500026E-08, 8.25599997E-08, 8.06399996E-08])
00073     D2=np.array([1.34100006E-07, 1.34999993E-07, 1.36400004E-07, 1.38399997E-07, 1.41200005E-07,
      1.45300007E-07, 1.51099997E-07, 1.59999999E-07, 1.71099998E-07, 1.80300006E-07])
00074     D3=np.array([1.34100006E-07, 1.33900002E-07, 1.33599997E-07, 1.33200004E-07, 1.32500006E-07,
      1.31299998E-07, 1.29000000E-07, 1.25400007E-07, 1.20199999E-07, 1.13500001E-07])
00075     D4=np.array([1.34100006E-07, 1.34299995E-07, 1.32599993E-07, 1.29699998E-07, 1.26200007E-07,
      1.22599999E-07, 1.19100001E-07, 1.16099997E-07, 1.13399999E-07, 1.11100000E-07])
00076
00077     e1, v1 = np.loadtxt('../fortran/eng_cut.dat').T
00078     e2, v2 = np.loadtxt('../fortran/dens_cut.dat').T
00079     e3, v3 = np.loadtxt('../fortran/tion_cut.dat').T
00080     e4, v4 = np.loadtxt('../fortran/te_cut.dat').T
00081
00082     plt.figure(figsize=(12,8))
00083     plt.subplot(2,2,1)
00084     plt.semilogx(eng [0, 0, 0, :], res[0, 0, 0, :], label='python full')
00085     #plt.semilogx(eng [0, 0, 0, :], table.data(eng[0, 0, 0, :].copy(), dens[0, 0, 0, :].copy(),
      tion[0, 0, 0, :].copy(), te[0, 0, 0, :].copy()), label='slice')
00086     plt.semilogx(x1, D1, 'o', label='ADAS values');
00087     plt.semilogx(e1, v1, 'o', label='fortran full')
00088     plt.xlabel('$%s$' % coordinates[0])
00089     plt.ylabel('$%s$' % results)
00090     plt.legend(loc=0)
00091
00092     plt.subplot(2,2,2)
00093     plt.semilogx(dens[0, 0, :, 0], res[0, 0, :, 0], label='python full')
00094     #plt.semilogx(dens[0, 0, :, 0], table.data(eng[0, 0, :, 0].copy(), dens[0, 0, :, 0].copy(),
      tion[0, 0, :, 0].copy(), te[0, 0, :, 0].copy()), label='slice')
00095     plt.semilogx(x2, D2, 'o', label='ADAS values')
00096     plt.semilogx(e2, v2, 'o', label='fortran full')
00097     plt.xlabel('$%s$' % coordinates[1])
00098     plt.ylabel('$%s$' % results)
00099     plt.legend(loc=0)
00100
00101     plt.subplot(2,2,3)
00102     plt.semilogx(tion[0, :, 0, 0], res[0, :, 0, 0], label='python full')
00103     #plt.semilogx(tion[0, :, 0, 0], table.data(eng[0, :, 0, 0].copy(), dens[0, :, 0, 0].copy(),
      tion[0, :, 0, 0].copy(), te[0, :, 0, 0].copy()), label='slice')
00104     plt.semilogx(x3, D3, 'o', label='ADAS values')
00105     plt.semilogx(e3, v3, 'o', label='fortran full')
00106     plt.xlabel('$%s$' % coordinates[2])
00107     plt.ylabel('$%s$' % results)
00108     plt.legend(loc=0)
00109
00110     plt.subplot(2,2,4)
00111     plt.semilogx(te  [:, 0, 0, 0], res[:, 0, 0, 0], label='python full')
00112     #plt.semilogx(te  [:, 0, 0, 0], table.data(eng[:, 0, 0, 0].copy(), dens[:, 0, 0, 0].copy(),
      tion[:, 0, 0, 0].copy(), te[:, 0, 0, 0].copy()), label='slice')
00113     plt.semilogx(x4, D4, 'o', label='ADAS values')
00114     plt.semilogx(e4, v4, 'o', label='fortran full')
```

```
00115     plt.xlabel('$%s$' % coordinates[3])
00116     plt.ylabel('$%s$' % results)
00117     plt.legend(loc=0)
00118
00119     plt.suptitle(reactants)
00120
00121     plt.subplots_adjust(left=0.07, right=0.98, bottom=0.07, top=0.90, wspace=0.20, hspace=0.25)
00122
00123
00124 plt.ion()
00125 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(2)
00126 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00127 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(5)
00128 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00129 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(11)
00130 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00131 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(21)
00132 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00133 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(51)
00134 plot(eng, dens, tion, te, res, coordinates, results, reactants)
00135 eng, dens, tion, te, res, coordinates, results, reactants = bms_calc(101)
00136 plot(eng, dens, tion, te, res, coordinates, results, reactants)
```

16.36 examples/py/amns_test_bms_interpolation_options.py File Reference

Namespaces

- amns_test_bms_interpolation_options

Variables

- list amns_test_bms_interpolation_options.nx = [1001, 2, 2, 2]
- amns_test_bms_interpolation_options.ENG = np.logspace(np.log10(2.000E+04), np.log10(1.500E+05), nx[0])
- amns_test_bms_interpolation_options.DENS = np.logspace(np.log10(1.000E+17), np.log10(1.000E+21), nx[1])
- amns_test_bms_interpolation_options.TION = np.logspace(np.log10(1.000E+02), np.log10(2.000E+04), nx[2])
- amns_test_bms_interpolation_options.TE = np.logspace(np.log10(1.000E+02), np.log10(1.000E+04), nx[3])
- amns_test_bms_interpolation_options.start = time.time()
- amns_test_bms_interpolation_options.te = te.reshape(nx[::-1])
- amns_test_bms_interpolation_options.tion = tion.reshape(nx[::-1])
- amns_test_bms_interpolation_options.dens = dens.reshape(nx[::-1])
- amns_test_bms_interpolation_options.eng = eng.reshape(nx[::-1])
- amns_test_bms_interpolation_options.end = time.time()
- amns_test_bms_interpolation_options.amnsdb = amns.Amns()
- amns_test_bms_interpolation_options.r = amns.Reactants()
- amns_test_bms_interpolation_options.lr
- amns_test_bms_interpolation_options.table = amnsdb.get_table(b"BMS", r)
- amns_test_bms_interpolation_options.res = table.data(eng.ravel(), dens.ravel(), tion.ravel(), te.↩ ravel()).reshape(nx[::-1])
- amns_test_bms_interpolation_options.coordinates = table.coordinates.split(" ")
- string amns_test_bms_interpolation_options.results = table.result_label+"/"+table.result_unit
- amns_test_bms_interpolation_options.reactants = table.reactants
- amns_test_bms_interpolation_options.x1 = np.array([20000.0000, 23100.0000, 26670.0000, 30800.0000, 35570.0000, 41070.0000, 47430.0000, 54770.0000, 63250.0000, 73040.0000, 84350.0000, 97400.0000, 112500.000, 129900.000, 150000.000])
- amns_test_bms_interpolation_options.x2 = np.array([9.99999984E+16, 2.78299999E+17, 7.74300003E+17, 2.15400004E+18, 5.99499974E+18, 1.66800004E+19, 4.64199999E+19, 1.29199996E+20, 3.↩ 59400022E+20, 1.00000002E+21])
- amns_test_bms_interpolation_options.x3 = np.array([100.000000, 180.199997, 324.600006, 584.799988, 1054.00000, 1898.00000, 3420.00000, 6162.00000, 11100.0000, 20000.0000])

- amns_test_bms_interpolation_options.x4 = np.array([100.000000, 166.800003, 278.299988, 464.200012, 774.299988, 1292.00000, 2154.00000, 3594.00000, 5995.00000, 10000.0000])
- amns_test_bms_interpolation_options.D1 = np.array([1.34100006E-07, 1.30900006E-07, 1.28400004E-07, 1.25499994E-07, 1.20600006E-07, 1.15600002E-07, 1.09900000E-07, 1.03799998E-07, 9.80599992E-08, 9.36700033E-08, 9.02900013E-08, 8.77199966E-08, 8.49500026E-08, 8.25599997E-08, 8.06399996E-08])
- amns_test_bms_interpolation_options.D2 = np.array([1.34100006E-07, 1.34999993E-07, 1.36400004E-07, 1.38399997E-07, 1.41200005E-07, 1.45300007E-07, 1.51099997E-07, 1.59999999E-07, 1.71099998E-07, 1.80300006E-07])
- amns_test_bms_interpolation_options.D3 = np.array([1.34100006E-07, 1.33900002E-07, 1.33599997E-07, 1.33200004E-07, 1.32500006E-07, 1.31299998E-07, 1.29000000E-07, 1.25400007E-07, 1.20199999E-07, 1.13500001E-07])
- amns_test_bms_interpolation_options.D4 = np.array([1.34100006E-07, 1.34299995E-07, 1.32599993E-07, 1.29699998E-07, 1.26200007E-07, 1.22599999E-07, 1.19100001E-07, 1.16099997E-07, 1.13399999E-07, 1.11100000E-07])
- amns_test_bms_interpolation_options.e1
- amns_test_bms_interpolation_options.v1
- amns_test_bms_interpolation_options.e2
- amns_test_bms_interpolation_options.v2
- amns_test_bms_interpolation_options.e3
- amns_test_bms_interpolation_options.v3
- amns_test_bms_interpolation_options.e4
- amns_test_bms_interpolation_options.v4
- amns_test_bms_interpolation_options.label
- amns_test_bms_interpolation_options.loc

16.37 amns_test_bms_interpolation_options.py

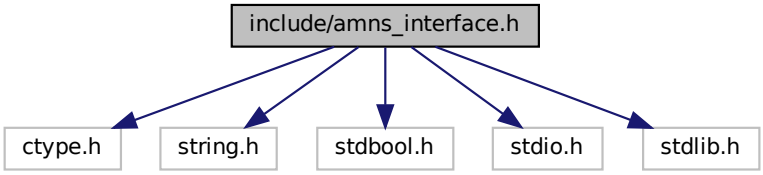
```
00001 #! /usr/bin/env python
00002
00003 import matplotlib.pyplot as plt
00004 from matplotlib import colors, ticker
00005 import amns
00006 import numpy as np
00007 import math as m
00008 import itertools
00009 import time
00010
00011 nx = [1001, 2, 2, 2]
00012
00013 ENG  = np.logspace(np.log10(2.000E+04), np.log10(1.500E+05), nx[0])
00014 DENS = np.logspace(np.log10(1.000E+17), np.log10(1.000E+21), nx[1])
00015 TION = np.logspace(np.log10(1.000E+02), np.log10(2.000E+04), nx[2])
00016 TE   = np.logspace(np.log10(1.000E+02), np.log10(1.000E+04), nx[3])
00017
00018 start = time.time()
00019 #    eng, dens, tion, te = np.array([[w,x,y,z] for z in TE for y in TION for x in DENS for w in
      ENG]).T
00020 te, tion, dens, eng = np.array(list(itertools.product(TE,TION,DENS,ENG))).T
00021 end = time.time()
00022 print('Setting up the coordinates took %0.3f seconds' % (end-start))
00023
00024 eng=eng.reshape(nx[::-1])
00025 dens=dens.reshape(nx[::-1])
00026 tion=tion.reshape(nx[::-1])
00027 te=te.reshape(nx[::-1])
00028
00029 amnsdb = amns.Amns()
00030
00031 print('Data created %s' % (amnsdb.prop_creation))
00032
00033 r = amns.Reactants()
00034 r.add(1,0,1)
00035 r.add(1,1,1)
00036 r.add(1,1,1,lr=1)
00037 r.add(1,0,1,lr=1)
00038 print("Reactants:", r)
00039
00040 start = time.time()
00041 table = amnsdb.get_table(b"BMS", r)
00042 end = time.time()
00043 print('Setting up the bms table took %0.3f seconds' % (end-start))
00044
```

```
00045 print("table.no_of_reactants")
00046 print("table.no_of_reactants", table.no_of_reactants)
00047
00048 start = time.time()
00049 res = table.data(eng.ravel(), dens.ravel(), tion.ravel(), te.ravel()).reshape(nx[::-1])
00050 end = time.time()
00051 print('Calculating the values for %s elements took %0.3f seconds' % (res.size, end-start))
00052
00053 coordinates=table.coordinates.split(" ")
00054 results=table.result_label+"/"+table.result_unit
00055 reactants=table.reactants
00056
00057 amnsdb.finalize()
00058
00059 x1=np.array([20000.0000, 23100.0000, 26670.0000, 30800.0000, 35570.0000, 41070.0000, 47430.0000,
       54770.0000, 63250.0000, 73040.0000, 84350.0000, 97400.0000, 112500.000, 129900.000, 150000.000])
00060 x2=np.array([9.99999984E+16, 2.78299999E+17, 7.74300003E+17, 2.15400004E+18, 5.99499974E+18,
       1.66800004E+19, 4.64199999E+19, 1.29199996E+20, 3.59400022E+20, 1.00000002E+21])
00061 x3=np.array([100.000000, 180.199997, 324.600006, 584.799988, 1054.00000, 1898.00000, 3420.00000,
       6162.00000, 11100.0000, 20000.0000])
00062 x4=np.array([100.000000, 166.800003, 278.299988, 464.200012, 774.299988, 1292.00000, 2154.00000,
       3594.00000, 5995.00000, 10000.0000])
00063
00064 D1=np.array([1.34100006E-07, 1.30900006E-07, 1.28400004E-07, 1.25499994E-07, 1.20600006E-07,
       1.15600002E-07, 1.09900000E-07, 1.03799998E-07, 9.80599992E-08, 9.36700033E-08, 9.02900013E-08,
       8.77199966E-08, 8.49500026E-08, 8.25599997E-08, 8.06399996E-08])
00065 D2=np.array([1.34100006E-07, 1.34999993E-07, 1.36400004E-07, 1.38399997E-07, 1.41200005E-07,
       1.45300007E-07, 1.51099997E-07, 1.59999999E-07, 1.71099998E-07, 1.80300006E-07])
00066 D3=np.array([1.34100006E-07, 1.33900002E-07, 1.33599997E-07, 1.33200004E-07, 1.32500006E-07,
       1.31299998E-07, 1.29000000E-07, 1.25400007E-07, 1.20199999E-07, 1.13500001E-07])
00067 D4=np.array([1.34100006E-07, 1.34299995E-07, 1.32599993E-07, 1.29699998E-07, 1.26200007E-07,
       1.22599999E-07, 1.19100001E-07, 1.16099997E-07, 1.13399999E-07, 1.11100000E-07])
00068
00069 e1, v1 = np.loadtxt('../fortran/eng_cut.dat').T
00070 e2, v2 = np.loadtxt('../fortran/dens_cut.dat').T
00071 e3, v3 = np.loadtxt('../fortran/tion_cut.dat').T
00072 e4, v4 = np.loadtxt('../fortran/te_cut.dat').T
00073
00074 plt.clf()
00075 plt.semilogx(eng [0, 0, 0, :], res[0, 0, 0, :], label='python full')
00076 #plt.semilogx(eng [0, 0, 0, :], table.data(eng[0, 0, 0, :].copy(), dens[0, 0, 0, :].copy(), tion[0, 0,
       0, :].copy(), te[0, 0, 0, :].copy()), label='slice')
00077 plt.semilogx(x1, D1, 'o', label='ADAS values');
00078 plt.semilogx(e1, v1, 'o', label='fortran full')
00079 plt.xlabel('$%s$' % coordinates[0])
00080 plt.ylabel('$%s$' % results)
00081 plt.legend(loc=0)
```

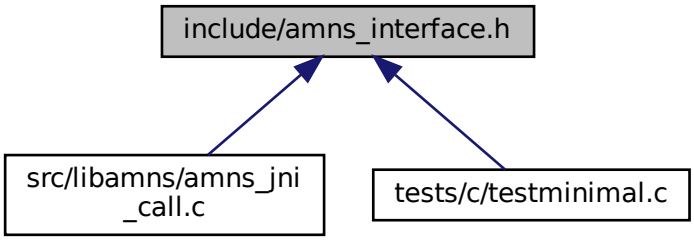
16.38 include/amns_interface.h File Reference

```
#include <ctype.h>
#include <string.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for amns_interface.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct amns_version_type

  Type for specifying the AMNS version ("interoperable" version)
- struct amns_c_version_type

  Type for specifying the AMNS version ("C" version)
- struct amns_reactant_type

  Type for indicating a single reactant or product when using the AMNS interface.
- struct amns_error_type

  Type for error returns from the AMNS interface ("interoperable" version)
- struct amns_c_error_type

  Type for error returns from the AMNS interface ("C" version)
- struct amns_reaction_type

  Type used for specifying reactions when using the AMNS interface ("interoperable" version)
- struct amns_c_reaction_type

  Type used for specifying reactions when using the AMNS interface ("C" version)
- struct amns_set_type

  Type for setting parameters in the AMNS package ("interoperable" version)
- struct amns_c_set_type

  Type for setting parameters in the AMNS package ("C" version)
- struct amns_query_type

  Type for querying parameters in the AMNS package ("interoperable" version)
- struct amns_c_query_type

  Type for querying parameters in the AMNS package ("C" version)
- struct amns_answer_type

  Type for answers from queries in the AMNS package ("interoperable" version)
- struct amns_c_answer_type

  Type for answers from queries in the AMNS package ("C" version)

Macros

- #define version_length 32
- #define set_length 32
- #define reaction_length 16
- #define query_length 16
- #define answer_length 128

- #define amns_max_length 128
- #define IMAS_INVALID_FLOAT -9.0E40
- #define IMAS_INVALID_INT -999999999

Typedefs

- typedef amns_reactant_type amns_c_reactant_type

Functions

- char * strcpy_f2c (char *fsrc, int flen, char **cdest)
- char * strcpy_c2f (char *csrc, char *fdest, int flen)
- amns_c_version_type get_default_amns_c_version_type (void)

  prototype
- void amns_version_type_f2c (amns_version_type ftype, amns_c_version_type *ctype)
- void amns_version_type_c2f (amns_c_version_type ctype, amns_version_type *ftype)
- amns_c_reactant_type get_default_amns_c_reactant_type (void)

  prototype
- amns_c_error_type get_default_amns_c_error_type (void)

  prototype
- void amns_error_type_f2c (amns_error_type ftype, amns_c_error_type *ctype)
- void amns_error_type_c2f (amns_c_error_type ctype, amns_error_type *ftype)
- amns_c_reaction_type get_default_amns_c_reaction_type (void)

  prototype
- void amns_reaction_type_f2c (amns_reaction_type ftype, amns_c_reaction_type *ctype)
- void amns_reaction_type_c2f (amns_c_reaction_type ctype, amns_reaction_type *ftype)
- void amns_set_type_f2c (amns_set_type ftype, amns_c_set_type *ctype)
- void amns_set_type_c2f (amns_c_set_type ctype, amns_set_type *ftype)
- void amns_query_type_f2c (amns_query_type ftype, amns_c_query_type *ctype)
- void amns_query_type_c2f (amns_c_query_type ctype, amns_query_type *ftype)
- void amns_answer_type_f2c (amns_answer_type ftype, amns_c_answer_type *ctype)
- void amns_answer_type_c2f (amns_c_answer_type ctype, amns_answer_type *ftype)
- void IMAS_AMNS_C_SETUP (void **handle_out, amns_error_type *error_status)
- void IMAS_AMNS_C_SETUP_VERSION (void **handle_in, amns_version_type *version, amns_error_type *error_status)
- void IMAS_AMNS_C_FINISH (void **handle_inout, amns_error_type *error_status)
- void IMAS_AMNS_C_FINISH_TABLE (void **handle_rx_inout, amns_error_type *error_status)
- void IMAS_AMNS_C_SET (void *handle_in, amns_set_type *set, amns_error_type *error_status)
- void IMAS_AMNS_C_QUERY (void *handle_in, amns_query_type *query, amns_answer_type *answer, amns_error_type *error_status)
- void IMAS_AMNS_C_SETUP_TABLE (void *handle_in, amns_reaction_type *reaction_type, void *reactant_handle_in, void **handle_rx_out, amns_error_type *error_status)
- void IMAS_AMNS_C_QUERY_TABLE (void *handle_rx_in, amns_query_type *query, amns_answer_type *answer, amns_error_type *error_status)
- void IMAS_AMNS_C_SET_TABLE (void *handle_rx_in, amns_set_type *set, amns_error_type *error_↩ status)
- void IMAS_AMNS_C_RX_0_A (void *handle_rx_in, double *out, double arg1, amns_error_type *error_↩ status)
- void IMAS_AMNS_C_RX_0_B (void *handle_rx_in, double *out, double arg1, double arg2, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_0_C (void *handle_rx_in, double *out, double arg1, double arg2, double arg3, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_0_D (void *handle_rx_in, double *out, double arg1, double arg2, double arg3, double arg4, amns_error_type *error_status)

- void IMAS_AMNS_C_RX_1_A (void *handle_rx_in, int nx, double *out, double *arg1, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_1_B (void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_1_C (void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, double *arg3, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_1_D (void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, double *arg3, double *arg4, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_2_A (void *handle_rx_in, int nx, int ny, double *out, double *arg1, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_2_B (void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_2_C (void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2, double *arg3, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_2_D (void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2, double *arg3, double *arg4, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_3_A (void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_3_B (void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double *arg2, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_3_C (void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double *arg2, double *arg3, amns_error_type *error_status)
- void IMAS_AMNS_C_RX_3_D (void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double *arg2, double *arg3, double *arg4, amns_error_type *error_status)
- void IMAS_AMNS_C_SETUP_REACTANTS (void **reactants_handle_out, char string_in[reaction_length], int index_in, int n_reactants)
- void IMAS_AMNS_C_SET_REACTANT (void *reactants_handle_in, int reactant_index, amns_reactant_type *reactant_in)
- void IMAS_AMNS_C_GET_REACTANT (void *reactants_handle_in, int reactant_index, amns_reactant_type *reactant_out)
- void IMAS_AMNS_C_FINISH_REACTANTS (void **reactants_handle_inout)
- int fstrlen (char *fstr, int flen)
- void IMAS_AMNS_CC_SETUP (void **handle_out, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_SETUP_VERSION (void **handle_in, amns_c_version_type *version, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_FINISH (void **handle_inout, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_FINISH_TABLE (void **handle_rx_inout, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_SET (void *handle_in, amns_c_set_type *set, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_QUERY (void *handle_in, amns_c_query_type *query, amns_c_answer_type *answer, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_SETUP_TABLE (void *handle_in, amns_c_reaction_type *reaction_type, void *reactant_handle_in, void **handle_rx_out, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_QUERY_TABLE (void *handle_rx_in, amns_c_query_type *query, amns_c_answer_type *answer, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_SET_TABLE (void *handle_rx_in, amns_c_set_type *set, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_0_A (void *handle_rx_in, double *out, double arg1, amns_c_error_type *error↩_status)
- void IMAS_AMNS_CC_RX_0_B (void *handle_rx_in, double *out, double arg1, double arg2, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_0_C (void *handle_rx_in, double *out, double arg1, double arg2, double arg3, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_0_D (void *handle_rx_in, double *out, double arg1, double arg2, double arg3, double arg4, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_1_A (void *handle_rx_in, int nx, double *out, double *arg1, amns_c_error_type *error_status)

- void IMAS_AMNS_CC_RX_1_B (void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_1_C (void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, double *arg3, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_1_D (void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, double *arg3, double *arg4, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_2_A (void *handle_rx_in, int nx, int ny, double *out, double *arg1, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_2_B (void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_2_C (void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2, double *arg3, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_2_D (void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2, double *arg3, double *arg4, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_3_A (void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_3_B (void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double *arg2, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_3_C (void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double *arg2, double *arg3, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_RX_3_D (void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double *arg2, double *arg3, double *arg4, amns_c_error_type *error_status)
- void IMAS_AMNS_CC_SETUP_REACTANTS (void **reactants_handle_out, char *string_in, int index_in, int n_reactants)
- void IMAS_AMNS_CC_SET_REACTANT (void *reactants_handle_in, int reactant_index, amns_c_reactant_type *reactant_in)
- void IMAS_AMNS_CC_GET_REACTANT (void *reactants_handle_in, int reactant_index, amns_c_reactant_type *reactant_out)
- void IMAS_AMNS_CC_FINISH_REACTANTS (void **reactants_handle_inout)

Variables

- const amns_version_type DEFAULT_AMNS_VERSION_TYPE = { "", 0, "", "" }
- const amns_c_version_type DEFAULT_AMNS_C_VERSION_TYPE = { "", 0, "", "" }
- const amns_reactant_type DEFAULT_AMNS_REACTANT_TYPE = { 0.0, 0.0, 0.0, 0, IMAS_INVALID_FLOAT, IMAS_INVALID_INT }
- const amns_c_reactant_type DEFAULT_AMNS_C_REACTANT_TYPE = { 0.0, 0.0, 0.0, 0, IMAS_INVALID_FLOAT, IMAS_INVALID_INT }
- const amns_error_type DEFAULT_AMNS_ERROR_TYPE = { false, "" }
- const amns_c_error_type DEFAULT_AMNS_C_ERROR_TYPE = { false, "" }
- const amns_reaction_type DEFAULT_AMNS_REACTION_TYPE = { "", 0 }
- const amns_c_reaction_type DEFAULT_AMNS_C_REACTION_TYPE = { "", 0 }
- const amns_set_type DEFAULT_AMNS_SET_TYPE = { "" }
- const amns_c_set_type DEFAULT_AMNS_C_SET_TYPE = { "" }
- const amns_query_type DEFAULT_AMNS_QUERY_TYPE = { "" }
- const amns_c_query_type DEFAULT_AMNS_C_QUERY_TYPE = { "" }
- const amns_answer_type DEFAULT_AMNS_ANSWER_TYPE = { "", 0 }
- const amns_c_answer_type DEFAULT_AMNS_C_ANSWER_TYPE = { "", 0 }

16.38.1 Macro Definition Documentation

16.38.1.1 amns_max_length

```
#define amns_max_length 128
```

Definition at line 45 of file amns_interface.h.

16.38.1.2 answer_length

```
#define answer_length 128
```
Definition at line 44 of file amns_interface.h.

16.38.1.3 IMAS_INVALID_FLOAT

```
#define IMAS_INVALID_FLOAT -9.0E40
```
Definition at line 110 of file amns_interface.h.

16.38.1.4 IMAS_INVALID_INT

```
#define IMAS_INVALID_INT -999999999
```
Definition at line 111 of file amns_interface.h.

16.38.1.5 query_length

```
#define query_length 16
```
Definition at line 43 of file amns_interface.h.

16.38.1.6 reaction_length

```
#define reaction_length 16
```
Definition at line 42 of file amns_interface.h.

16.38.1.7 set_length

```
#define set_length 32
```
Definition at line 41 of file amns_interface.h.

16.38.1.8 version_length

```
#define version_length 32
```
Definition at line 40 of file amns_interface.h.

16.38.2 Typedef Documentation

16.38.2.1 amns_c_reactant_type

```
typedef amns_reactant_type amns_c_reactant_type
```
Definition at line 108 of file amns_interface.h.

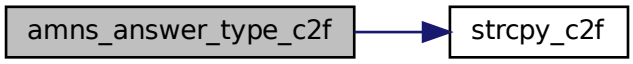
16.38.3 Function Documentation

16.38.3.1 amns_answer_type_c2f()

```
void amns_answer_type_c2f (
            amns_c_answer_type ctype,
            amns_answer_type * ftype )
```
Definition at line 276 of file amns_interface.h.

```
00276                                                                                {
00277     strcpy_c2f(ctype.string, ftype->string, answer_length);
00278     ftype->number = ctype.number;
00279 }
```

Here is the call graph for this function:



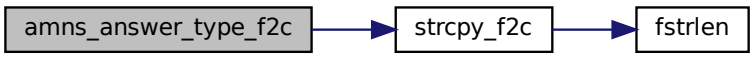
16.38.3.2 amns_answer_type_f2c()

```
void amns_answer_type_f2c (
            amns_answer_type ftype,
            amns_c_answer_type * ctype )
```

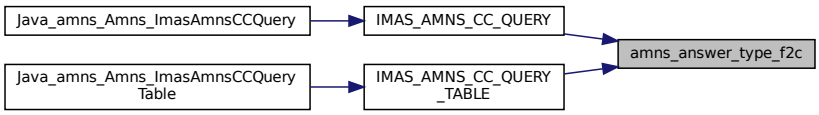
Definition at line 271 of file amns_interface.h.

```
00271                                                                                {
00272     strcpy_f2c(ftype.string, answer_length, &(ctype->string));
00273     ctype->number = ftype.number;
00274 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.38.3.3 amns_error_type_c2f()

```
void amns_error_type_c2f (
            amns_c_error_type ctype,
            amns_error_type * ftype )
```

Definition at line 154 of file amns_interface.h.

```
00154                                                                                {
00155     strcpy_c2f(ctype.string, ftype->string, answer_length);
00156     ftype->flag = ctype.flag;
```

```
00157 }
```

Here is the call graph for this function:

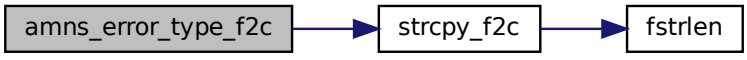
16.38.3.4 amns_error_type_f2c()

```
void amns_error_type_f2c (
            amns_error_type ftype,
            amns_c_error_type * ctype )
```

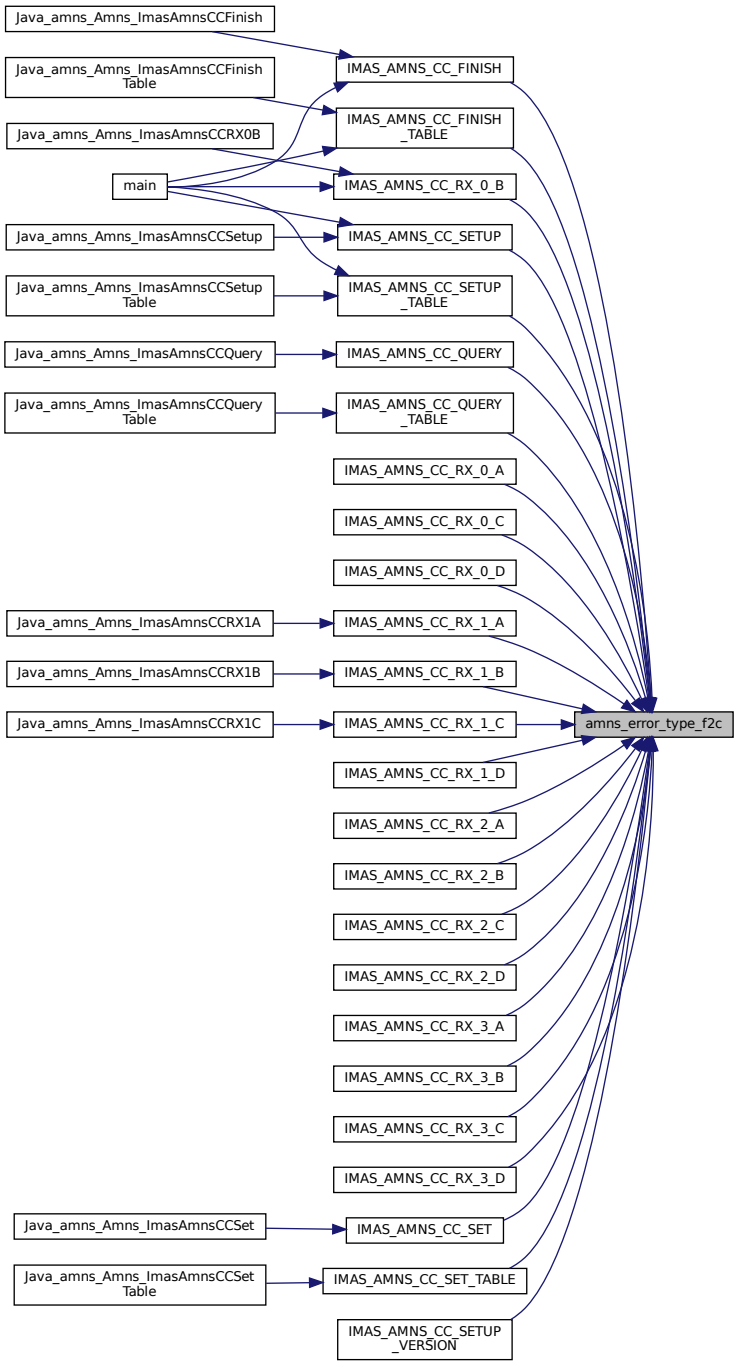
Definition at line 149 of file amns_interface.h.

```
00149                                                                   {
00150     strcpy_f2c(ftype.string, answer_length, &(ctype->string));
00151     ctype->flag = ftype.flag;
00152 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.38.3.5 amns_query_type_c2f()

```
void amns_query_type_c2f (
            amns_c_query_type ctype,
            amns_query_type * ftype )
```

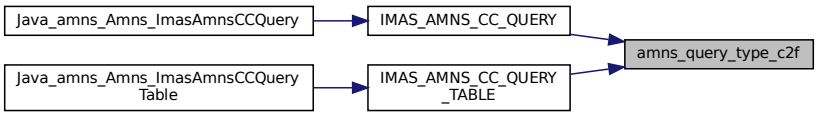
Definition at line 246 of file amns_interface.h.

```
00246                                                                   {
```

```
00247     strcpy_c2f(ctype.string, ftype->string, query_length);
00248 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



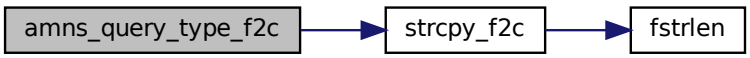
16.38.3.6 amns_query_type_f2c()

```
void amns_query_type_f2c (
            amns_query_type ftype,
            amns_c_query_type * ctype )
```

Definition at line 242 of file amns_interface.h.

```
00242                                                                    {
00243     strcpy_f2c(ftype.string, query_length, &(ctype->string));
00244 }
```

Here is the call graph for this function:



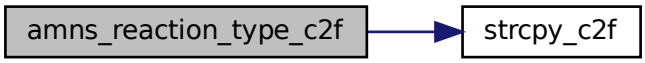
16.38.3.7 amns_reaction_type_c2f()

```
void amns_reaction_type_c2f (
            amns_c_reaction_type ctype,
            amns_reaction_type * ftype )
```

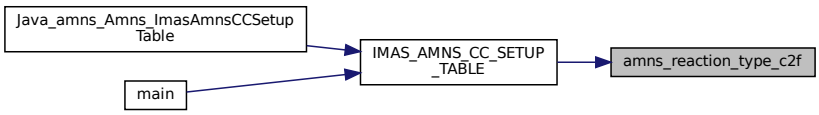
Definition at line 191 of file amns_interface.h.

```
00191                                                                    {
00192     strcpy_c2f(ctype.string, ftype->string, reaction_length);
00193     ftype->isotope_resolved = ctype.isotope_resolved;
00194 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



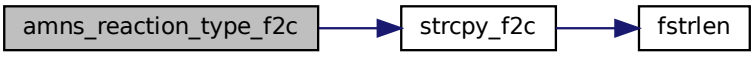
16.38.3.8 amns_reaction_type_f2c()

```
void amns_reaction_type_f2c (
            amns_reaction_type ftype,
            amns_c_reaction_type * ctype )
```

Definition at line 186 of file amns_interface.h.

```
00186                                                                    {
00187     strcpy_f2c(ftype.string, reaction_length, &(ctype->string));
00188     ctype->isotope_resolved = ftype.isotope_resolved;
00189 }
```

Here is the call graph for this function:



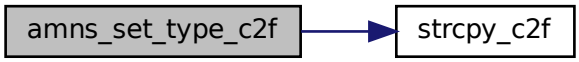
16.38.3.9 amns_set_type_c2f()

```
void amns_set_type_c2f (
            amns_c_set_type ctype,
            amns_set_type * ftype )
```

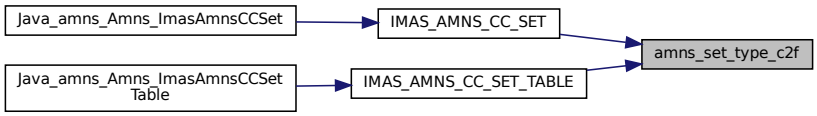
Definition at line 219 of file amns_interface.h.

```
00219                                                                {
00220     strcpy_c2f(ctype.string, ftype->string, set_length);
00221 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.38.3.10 amns_set_type_f2c()

```
void amns_set_type_f2c (
            amns_set_type ftype,
            amns_c_set_type * ctype )
```

Definition at line 215 of file amns_interface.h.

```
00215                                                                   {
00216     strcpy_f2c(ftype.string, set_length, &(ctype->string));
00217 }
```

Here is the call graph for this function:

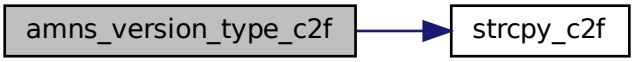
16.38.3.11 amns_version_type_c2f()

```
void amns_version_type_c2f (
            amns_c_version_type ctype,
            amns_version_type * ftype )
```

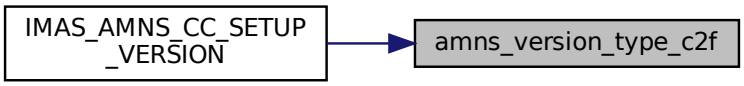
Definition at line 88 of file amns_interface.h.

```
00088                                                                   {
00089     strcpy_c2f(ctype.string, ftype->string, version_length);
00090     ftype->number = ctype.number;
00091     strcpy_c2f(ctype.backend, ftype->backend, version_length);
00092     strcpy_c2f(ctype.user, ftype->user, version_length);
00093 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



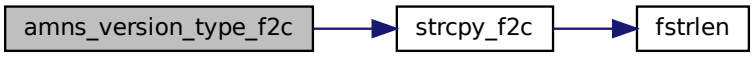
16.38.3.12 amns_version_type_f2c()

```
void amns_version_type_f2c (
            amns_version_type ftype,
            amns_c_version_type * ctype )
```

Definition at line 81 of file amns_interface.h.

```
00081                                                                                {
00082     strcpy_f2c(ftype.string, version_length, &((*ctype).string));
00083     ctype->number = ftype.number;
00084     strcpy_f2c(ftype.backend, version_length, &(ctype->backend));
00085     strcpy_f2c(ftype.user, version_length, &(ctype->user));
00086 }
```

Here is the call graph for this function:



16.38.3.13 fstrlen()

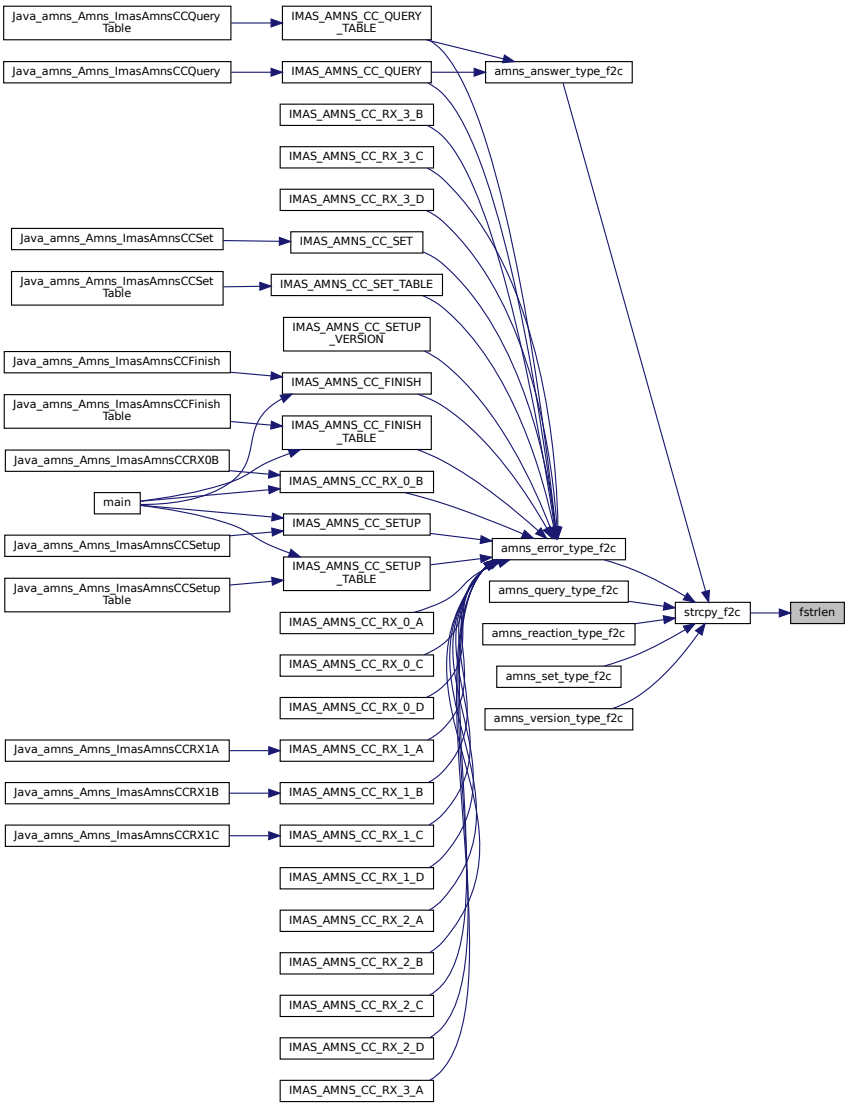
```
int fstrlen (
            char * fstr,
            int flen )
```

Definition at line 330 of file amns_interface.h.

```
00330                                {
00331     char *end;
00332     end = fstr + flen - 1;
00333     while(end >= fstr && isspace(*end)) end--;
```

```
00334     return end - fstr + 1;
00335 }
```

Here is the caller graph for this function:



16.38.3.14 get_default_amns_c_error_type()

```
amns_c_error_type get_default_amns_c_error_type (
            void  )
```

prototype

Definition at line 145 of file amns_interface.h.

```
00145                                          {
00146     return DEFAULT_AMNS_C_ERROR_TYPE;
00147 }
```

16.38.3.15 get_default_amns_c_reactant_type()

```
amns_c_reactant_type get_default_amns_c_reactant_type (
            void  )
```

prototype
Definition at line 118 of file amns_interface.h.

```
00118                                                    {
00119   return DEFAULT_AMNS_C_REACTANT_TYPE;
00120 }
```

16.38.3.16 get_default_amns_c_reaction_type()

```
amns_c_reaction_type get_default_amns_c_reaction_type (
            void  )
```

prototype
Definition at line 182 of file amns_interface.h.

```
00182                                                    {
00183     return DEFAULT_AMNS_C_REACTION_TYPE;
00184 }
```

16.38.3.17 get_default_amns_c_version_type()

```
amns_c_version_type get_default_amns_c_version_type (
            void  )
```

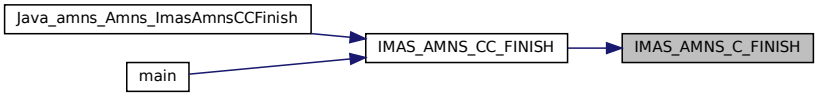
prototype
Definition at line 77 of file amns_interface.h.

```
00077                                                   {
00078   return DEFAULT_AMNS_C_VERSION_TYPE;
00079 }
```

16.38.3.18 IMAS_AMNS_C_FINISH()

```
void IMAS_AMNS_C_FINISH (
            void ** handle_inout,
            amns_error_type * error_status )
```

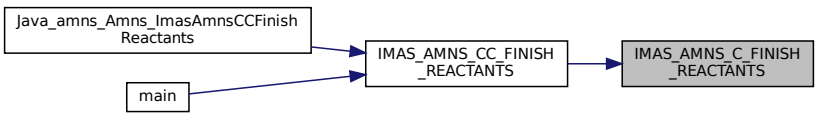
Here is the caller graph for this function:



16.38.3.19 IMAS_AMNS_C_FINISH_REACTANTS()

```
void IMAS_AMNS_C_FINISH_REACTANTS (
            void ** reactants_handle_inout )
```

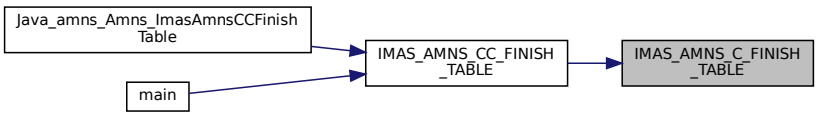
Here is the caller graph for this function:



16.38.3.20 IMAS_AMNS_C_FINISH_TABLE()

```
void IMAS_AMNS_C_FINISH_TABLE (
            void ** handle_rx_inout,
            amns_error_type * error_status )
```

Here is the caller graph for this function:



16.38.3.21 IMAS_AMNS_C_GET_REACTANT()

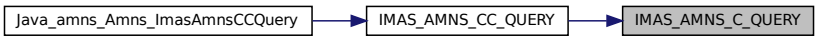
```
void IMAS_AMNS_C_GET_REACTANT (
            void * reactants_handle_in,
            int reactant_index,
            amns_reactant_type * reactant_out )
```

Here is the caller graph for this function:

16.38.3.22 IMAS_AMNS_C_QUERY()

```
void IMAS_AMNS_C_QUERY (
            void * handle_in,
            amns_query_type * query,
            amns_answer_type * answer,
            amns_error_type * error_status )
```

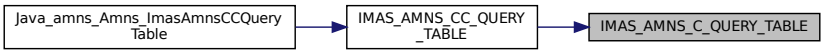
Here is the caller graph for this function:



16.38.3.23 IMAS_AMNS_C_QUERY_TABLE()

```
void IMAS_AMNS_C_QUERY_TABLE (
            void * handle_rx_in,
            amns_query_type * query,
            amns_answer_type * answer,
            amns_error_type * error_status )
```

Here is the caller graph for this function:



16.38.3.24 IMAS_AMNS_C_RX_0_A()

```
void IMAS_AMNS_C_RX_0_A (
            void * handle_rx_in,
            double * out,
            double arg1,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.25 IMAS_AMNS_C_RX_0_B()

```
void IMAS_AMNS_C_RX_0_B (
            void * handle_rx_in,
            double * out,
            double arg1,
            double arg2,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.26 IMAS_AMNS_C_RX_0_C()

```
void IMAS_AMNS_C_RX_0_C (
            void * handle_rx_in,
            double * out,
            double arg1,
            double arg2,
```

```
            double arg3,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.27 IMAS_AMNS_C_RX_0_D()

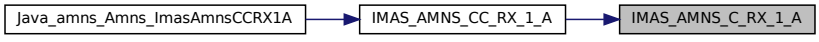
```
void IMAS_AMNS_C_RX_0_D (
            void * handle_rx_in,
            double * out,
            double arg1,
            double arg2,
            double arg3,
            double arg4,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.28 IMAS_AMNS_C_RX_1_A()

```
void IMAS_AMNS_C_RX_1_A (
            void * handle_rx_in,
            int nx,
            double * out,
            double * arg1,
            amns_error_type * error_status )
```

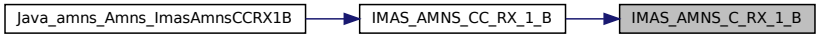
Here is the caller graph for this function:



16.38.3.29 IMAS_AMNS_C_RX_1_B()

```
void IMAS_AMNS_C_RX_1_B (
            void * handle_rx_in,
            int nx,
            double * out,
            double * arg1,
            double * arg2,
            amns_error_type * error_status )
```

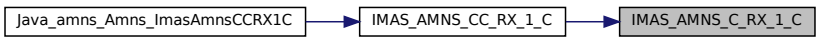
Here is the caller graph for this function:



16.38.3.30 IMAS_AMNS_C_RX_1_C()

```
void IMAS_AMNS_C_RX_1_C (
            void * handle_rx_in,
            int nx,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            amns_error_type * error_status )
```

Here is the caller graph for this function:



16.38.3.31 IMAS_AMNS_C_RX_1_D()

```
void IMAS_AMNS_C_RX_1_D (
            void * handle_rx_in,
            int nx,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            double * arg4,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.32 IMAS_AMNS_C_RX_2_A()

```
void IMAS_AMNS_C_RX_2_A (
            void * handle_rx_in,
            int nx,
            int ny,
            double * out,
            double * arg1,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.33 IMAS_AMNS_C_RX_2_B()

```
void IMAS_AMNS_C_RX_2_B (
            void * handle_rx_in,
            int nx,
            int ny,
            double * out,
            double * arg1,
            double * arg2,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.34 IMAS_AMNS_C_RX_2_C()

```
void IMAS_AMNS_C_RX_2_C (
            void * handle_rx_in,
            int nx,
            int ny,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.35 IMAS_AMNS_C_RX_2_D()

```
void IMAS_AMNS_C_RX_2_D (
            void * handle_rx_in,
            int nx,
            int ny,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            double * arg4,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.36 IMAS_AMNS_C_RX_3_A()

```
void IMAS_AMNS_C_RX_3_A (
            void * handle_rx_in,
            int nx,
            int ny,
            int nz,
            double * out,
```

```
            double * arg1,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.37 IMAS_AMNS_C_RX_3_B()

```
void IMAS_AMNS_C_RX_3_B (
            void * handle_rx_in,
            int nx,
            int ny,
            int nz,
            double * out,
            double * arg1,
            double * arg2,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.38 IMAS_AMNS_C_RX_3_C()

```
void IMAS_AMNS_C_RX_3_C (
            void * handle_rx_in,
            int nx,
            int ny,
            int nz,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.39 IMAS_AMNS_C_RX_3_D()

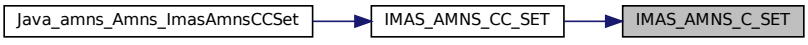
```
void IMAS_AMNS_C_RX_3_D (
            void * handle_rx_in,
            int nx,
            int ny,
            int nz,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            double * arg4,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.40 IMAS_AMNS_C_SET()

```
void IMAS_AMNS_C_SET (
            void * handle_in,
            amns_set_type * set,
            amns_error_type * error_status )
```

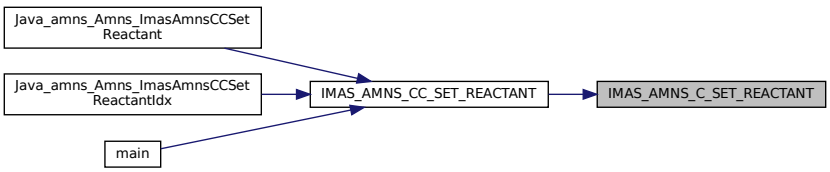
Here is the caller graph for this function:



16.38.3.41 IMAS_AMNS_C_SET_REACTANT()

```
void IMAS_AMNS_C_SET_REACTANT (
            void * reactants_handle_in,
            int reactant_index,
            amns_reactant_type * reactant_in )
```

Here is the caller graph for this function:



16.38.3.42 IMAS_AMNS_C_SET_TABLE()

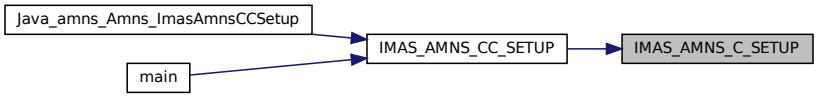
```
void IMAS_AMNS_C_SET_TABLE (
            void * handle_rx_in,
            amns_set_type * set,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.43 IMAS_AMNS_C_SETUP()

```
void IMAS_AMNS_C_SETUP (
            void ** handle_out,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

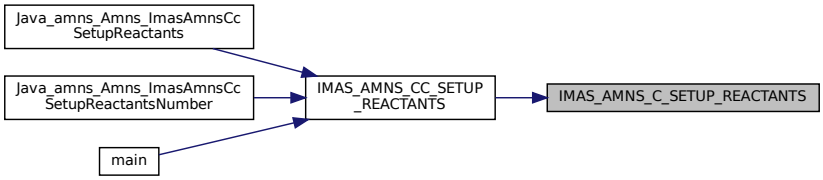


16.38.3.44 IMAS_AMNS_C_SETUP_REACTANTS()

```
void IMAS_AMNS_C_SETUP_REACTANTS (
            void ** reactants_handle_out,
```

```
            char string_in[reaction_length],
            int index_in,
            int n_reactants )
```

Here is the caller graph for this function:



16.38.3.45 IMAS_AMNS_C_SETUP_TABLE()

```
void IMAS_AMNS_C_SETUP_TABLE (
            void * handle_in,
            amns_reaction_type * reaction_type,
            void * reactant_handle_in,
            void ** handle_rx_out,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

16.38.3.46 IMAS_AMNS_C_SETUP_VERSION()

```
void IMAS_AMNS_C_SETUP_VERSION (
            void ** handle_in,
            amns_version_type * version,
            amns_error_type * error_status )
```

Here is the caller graph for this function:

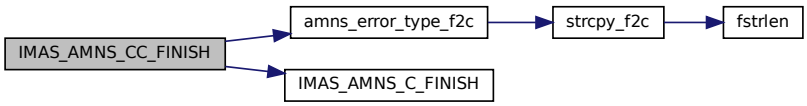
16.38.3.47 IMAS_AMNS_CC_FINISH()

```
void IMAS_AMNS_CC_FINISH (
            void ** handle_inout,
            amns_c_error_type * error_status )
```

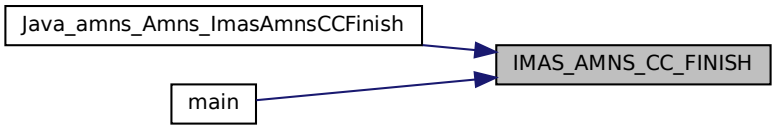
Definition at line 400 of file amns_interface.h.

```
00400                                                                                {
00401     amns_error_type f_error_status;
00402     //amns_error_type_c2f(*error_status, &f_error_status);
00403     IMAS_AMNS_C_FINISH(handle_inout, &f_error_status);
00404     amns_error_type_f2c(f_error_status, error_status);
00405 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



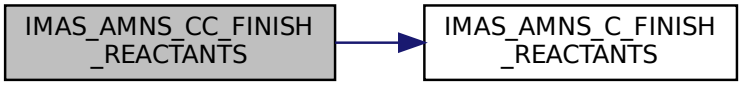
16.38.3.48 IMAS_AMNS_CC_FINISH_REACTANTS()

```
void IMAS_AMNS_CC_FINISH_REACTANTS (
            void ** reactants_handle_inout )
```

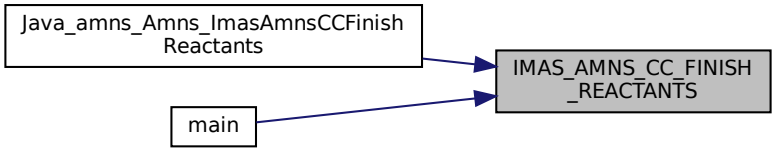
Definition at line 598 of file amns_interface.h.

```
00598                                                                                {
00599     IMAS_AMNS_C_FINISH_REACTANTS(reactants_handle_inout);
00600 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



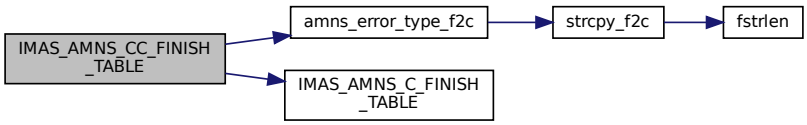
16.38.3.49 IMAS_AMNS_CC_FINISH_TABLE()

```
void IMAS_AMNS_CC_FINISH_TABLE (
            void ** handle_rx_inout,
            amns_c_error_type * error_status )
```

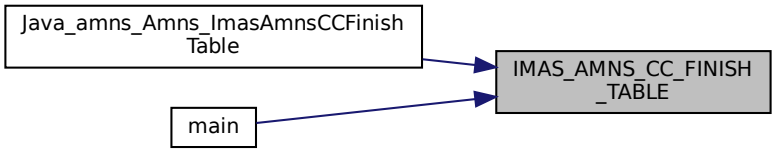
Definition at line 407 of file amns_interface.h.

```
00407                                                                                {
00408     amns_error_type f_error_status;
00409     //amns_error_type_c2f(*error_status, &f_error_status);
00410     IMAS_AMNS_C_FINISH_TABLE(handle_rx_inout, &f_error_status);
00411     amns_error_type_f2c(f_error_status, error_status);
00412 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.38.3.50 IMAS_AMNS_CC_GET_REACTANT()

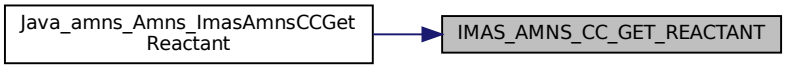
```
void IMAS_AMNS_CC_GET_REACTANT (
            void * reactants_handle_in,
            int reactant_index,
            amns_c_reactant_type * reactant_out )
```

Definition at line 594 of file amns_interface.h.

```
00594
                {
00595     IMAS_AMNS_C_GET_REACTANT(reactants_handle_in, reactant_index, (amns_reactant_type*)reactant_out);
00596 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



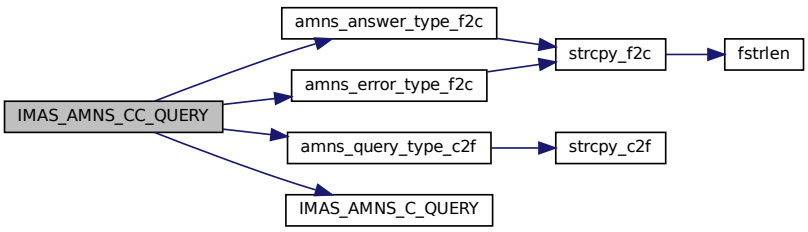
16.38.3.51 IMAS_AMNS_CC_QUERY()

```
void IMAS_AMNS_CC_QUERY (
            void * handle_in,
            amns_c_query_type * query,
            amns_c_answer_type * answer,
            amns_c_error_type * error_status )
```

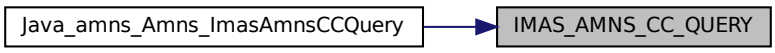
Definition at line 424 of file amns_interface.h.

```
00424
                            {
00425     amns_error_type f_error_status;
00426     amns_answer_type f_answer;
00427     amns_query_type f_query;
00428     //amns_error_type_c2f(*error_status, &f_error_status);
00429     //amns_answer_type_c2f(*answer, &f_answer);
00430     amns_query_type_c2f(*query, &f_query);
00431     IMAS_AMNS_C_QUERY(handle_in, &f_query, &f_answer, &f_error_status);
00432     amns_error_type_f2c(f_error_status, error_status);
00433     amns_answer_type_f2c(f_answer, answer);
00434     //amns_query_type_f2c(f_query, query);
00435 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



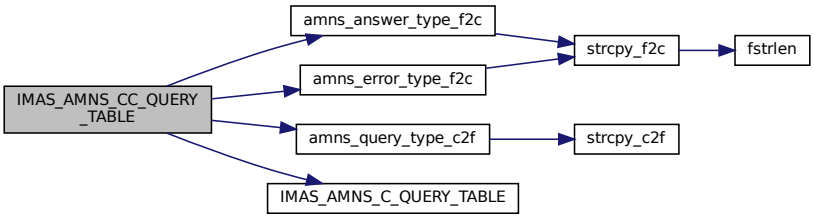
16.38.3.52 IMAS_AMNS_CC_QUERY_TABLE()

```
void IMAS_AMNS_CC_QUERY_TABLE (
            void * handle_rx_in,
            amns_c_query_type * query,
            amns_c_answer_type * answer,
            amns_c_error_type * error_status )
```

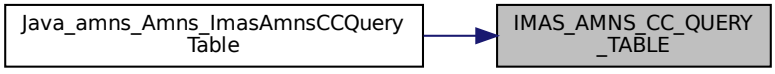
Definition at line 447 of file amns_interface.h.

```
00447
                                    {
00448     amns_error_type f_error_status;
00449     amns_query_type f_query;
00450     amns_answer_type f_answer;
00451     //amns_error_type_c2f(*error_status, &f_error_status);
00452     amns_query_type_c2f(*query, &f_query);
00453     //amns_answer_type_c2f(*answer, &f_answer);
00454     IMAS_AMNS_C_QUERY_TABLE(handle_rx_in, &f_query, &f_answer, &f_error_status);
00455     amns_error_type_f2c(f_error_status, error_status);
00456     //amns_query_type_f2c(f_query, query);
00457     amns_answer_type_f2c(f_answer, answer);
00458 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.38.3.53 IMAS_AMNS_CC_RX_0_A()

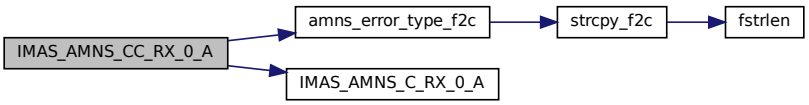
```
void IMAS_AMNS_CC_RX_0_A (
```

```
            void * handle_rx_in,
            double * out,
            double arg1,
            amns_c_error_type * error_status )
```

Definition at line 470 of file amns_interface.h.

```
00470
      {
00471     amns_error_type f_error_status;
00472     //amns_error_type_c2f(*error_status, &f_error_status);
00473     IMAS_AMNS_C_RX_0_A(handle_rx_in, out, arg1, &f_error_status);
00474     amns_error_type_f2c(f_error_status, error_status);
00475 }
```

Here is the call graph for this function:



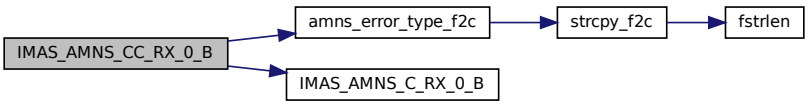
16.38.3.54 IMAS_AMNS_CC_RX_0_B()

```
void IMAS_AMNS_CC_RX_0_B (
            void * handle_rx_in,
            double * out,
            double arg1,
            double arg2,
            amns_c_error_type * error_status )
```

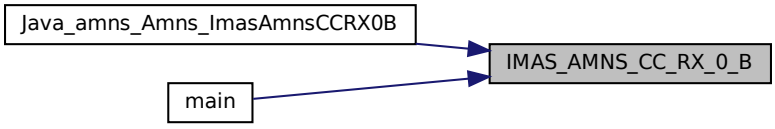
Definition at line 477 of file amns_interface.h.

```
00477
                {
00478     amns_error_type f_error_status;
00479     //amns_error_type_c2f(*error_status, &f_error_status);
00480     IMAS_AMNS_C_RX_0_B(handle_rx_in, out, arg1, arg2, &f_error_status);
00481     amns_error_type_f2c(f_error_status, error_status);
00482 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



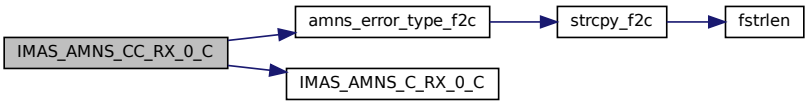
16.38.3.55 IMAS_AMNS_CC_RX_0_C()

```
void IMAS_AMNS_CC_RX_0_C (
            void * handle_rx_in,
            double * out,
            double arg1,
            double arg2,
            double arg3,
            amns_c_error_type * error_status )
```

Definition at line 484 of file amns_interface.h.

```
00484
                                    {
00485     amns_error_type f_error_status;
00486     //amns_error_type_c2f(*error_status, &f_error_status);
00487     IMAS_AMNS_C_RX_0_C(handle_rx_in, out, arg1, arg2, arg3, &f_error_status);
00488     amns_error_type_f2c(f_error_status, error_status);
00489 }
```

Here is the call graph for this function:



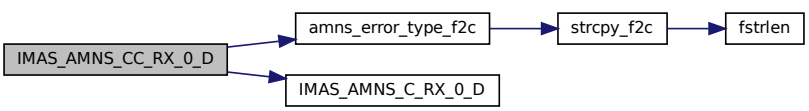
16.38.3.56 IMAS_AMNS_CC_RX_0_D()

```
void IMAS_AMNS_CC_RX_0_D (
            void * handle_rx_in,
            double * out,
            double arg1,
            double arg2,
            double arg3,
            double arg4,
            amns_c_error_type * error_status )
```

Definition at line 491 of file amns_interface.h.

```
00491
                                            {
00492     amns_error_type f_error_status;
00493     //amns_error_type_c2f(*error_status, &f_error_status);
00494     IMAS_AMNS_C_RX_0_D(handle_rx_in, out, arg1, arg2, arg3, arg4, &f_error_status);
00495     amns_error_type_f2c(f_error_status, error_status);
00496 }
```

Here is the call graph for this function:



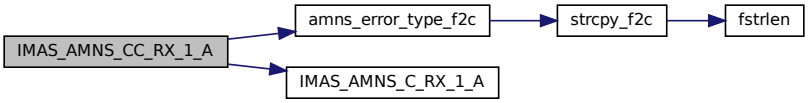
16.38.3.57 IMAS_AMNS_CC_RX_1_A()

```
void IMAS_AMNS_CC_RX_1_A (
            void * handle_rx_in,
            int nx,
            double * out,
            double * arg1,
            amns_c_error_type * error_status )
```

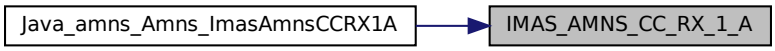
Definition at line 498 of file amns_interface.h.

```
00498
              {
00499     amns_error_type f_error_status;
00500     //amns_error_type_c2f(*error_status, &f_error_status);
00501     IMAS_AMNS_C_RX_1_A(handle_rx_in, nx, out, arg1, &f_error_status);
00502     amns_error_type_f2c(f_error_status, error_status);
00503 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



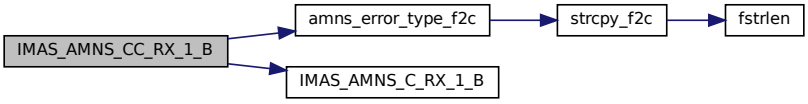
16.38.3.58 IMAS_AMNS_CC_RX_1_B()

```
void IMAS_AMNS_CC_RX_1_B (
            void * handle_rx_in,
            int nx,
            double * out,
            double * arg1,
            double * arg2,
            amns_c_error_type * error_status )
```

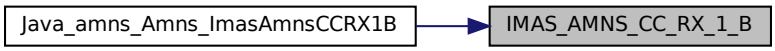
Definition at line 505 of file amns_interface.h.

```
00505
                          {
00506     amns_error_type f_error_status;
00507     //amns_error_type_c2f(*error_status, &f_error_status);
00508     IMAS_AMNS_C_RX_1_B(handle_rx_in, nx, out, arg1, arg2, &f_error_status);
00509     amns_error_type_f2c(f_error_status, error_status);
00510 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



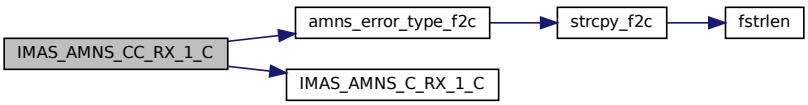
16.38.3.59 IMAS_AMNS_CC_RX_1_C()

```
void IMAS_AMNS_CC_RX_1_C (
            void * handle_rx_in,
            int nx,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            amns_c_error_type * error_status )
```

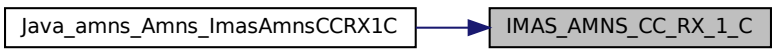
Definition at line 512 of file amns_interface.h.

```
00512                                                {
00513     amns_error_type f_error_status;
00514     //amns_error_type_c2f(*error_status, &f_error_status);
00515     IMAS_AMNS_C_RX_1_C(handle_rx_in, nx, out, arg1, arg2, arg3, &f_error_status);
00516     amns_error_type_f2c(f_error_status, error_status);
00517 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



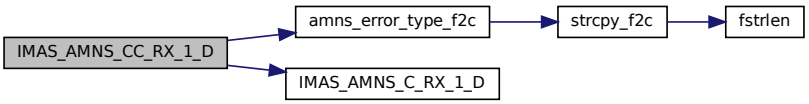
16.38.3.60 IMAS_AMNS_CC_RX_1_D()

```
void IMAS_AMNS_CC_RX_1_D (
            void * handle_rx_in,
            int nx,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            double * arg4,
            amns_c_error_type * error_status )
```

Definition at line 519 of file amns_interface.h.

```
00519
                                                                {
00520     amns_error_type f_error_status;
00521     //amns_error_type_c2f(*error_status, &f_error_status);
00522     IMAS_AMNS_C_RX_1_D(handle_rx_in, nx, out, arg1, arg2, arg3, arg4, &f_error_status);
00523     amns_error_type_f2c(f_error_status, error_status);
00524 }
```

Here is the call graph for this function:



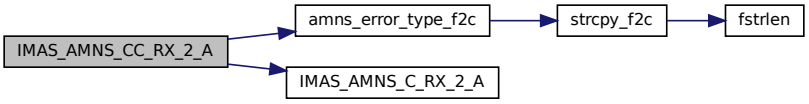
16.38.3.61 IMAS_AMNS_CC_RX_2_A()

```
void IMAS_AMNS_CC_RX_2_A (
            void * handle_rx_in,
            int nx,
            int ny,
            double * out,
            double * arg1,
            amns_c_error_type * error_status )
```

Definition at line 526 of file amns_interface.h.

```
00526
                      {
00527     amns_error_type f_error_status;
00528     //amns_error_type_c2f(*error_status, &f_error_status);
00529     IMAS_AMNS_C_RX_2_A(handle_rx_in, nx, ny, out, arg1, &f_error_status);
00530     amns_error_type_f2c(f_error_status, error_status);
00531 }
```

Here is the call graph for this function:



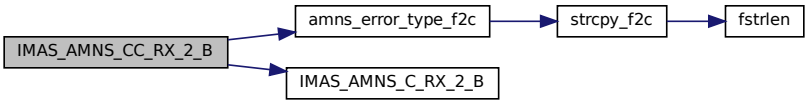
16.38.3.62 IMAS_AMNS_CC_RX_2_B()

```
void IMAS_AMNS_CC_RX_2_B (
            void * handle_rx_in,
            int nx,
            int ny,
            double * out,
            double * arg1,
            double * arg2,
            amns_c_error_type * error_status )
```

Definition at line 533 of file amns_interface.h.

```
00533                                       {
00534     amns_error_type f_error_status;
00535     //amns_error_type_c2f(*error_status, &f_error_status);
00536     IMAS_AMNS_C_RX_2_B(handle_rx_in, nx, ny, out, arg1, arg2, &f_error_status);
00537     amns_error_type_f2c(f_error_status, error_status);
00538 }
```

Here is the call graph for this function:



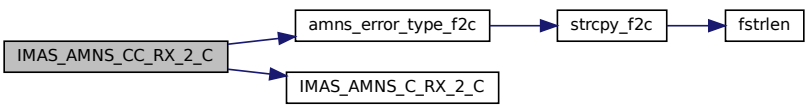
16.38.3.63 IMAS_AMNS_CC_RX_2_C()

```
void IMAS_AMNS_CC_RX_2_C (
            void * handle_rx_in,
            int nx,
            int ny,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            amns_c_error_type * error_status )
```

Definition at line 540 of file amns_interface.h.

```
00540                                                {
00541     amns_error_type f_error_status;
00542     //amns_error_type_c2f(*error_status, &f_error_status);
00543     IMAS_AMNS_C_RX_2_C(handle_rx_in, nx, ny, out, arg1, arg2, arg3, &f_error_status);
00544     amns_error_type_f2c(f_error_status, error_status);
00545 }
```

Here is the call graph for this function:



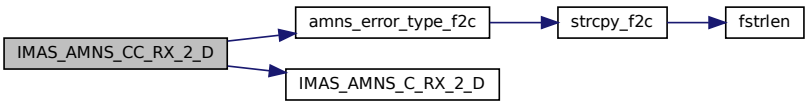
16.38.3.64 IMAS_AMNS_CC_RX_2_D()

```
void IMAS_AMNS_CC_RX_2_D (
            void * handle_rx_in,
            int nx,
            int ny,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            double * arg4,
            amns_c_error_type * error_status )
```

Definition at line 547 of file amns_interface.h.

```
00547
                                                                {
00548     amns_error_type f_error_status;
00549     //amns_error_type_c2f(*error_status, &f_error_status);
00550     IMAS_AMNS_C_RX_2_D(handle_rx_in, nx, ny, out, arg1, arg2, arg3, arg4, &f_error_status);
00551     amns_error_type_f2c(f_error_status, error_status);
00552 }
```

Here is the call graph for this function:



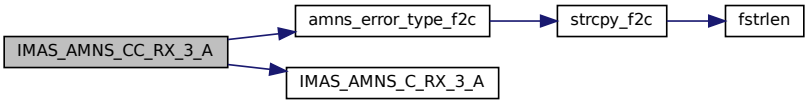
16.38.3.65 IMAS_AMNS_CC_RX_3_A()

```
void IMAS_AMNS_CC_RX_3_A (
            void * handle_rx_in,
            int nx,
            int ny,
            int nz,
            double * out,
            double * arg1,
            amns_c_error_type * error_status )
```

Definition at line 554 of file amns_interface.h.

```
00554
                                {
00555     amns_error_type f_error_status;
00556     //amns_error_type_c2f(*error_status, &f_error_status);
00557     IMAS_AMNS_C_RX_3_A(handle_rx_in, nx, ny, nz, out, arg1, &f_error_status);
00558     amns_error_type_f2c(f_error_status, error_status);
00559 }
```

Here is the call graph for this function:



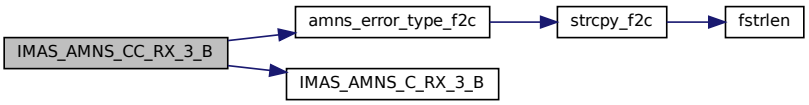
16.38.3.66 IMAS_AMNS_CC_RX_3_B()

```
void IMAS_AMNS_CC_RX_3_B (
            void * handle_rx_in,
            int nx,
            int ny,
            int nz,
            double * out,
            double * arg1,
            double * arg2,
            amns_c_error_type * error_status )
```

Definition at line 561 of file amns_interface.h.

```
00561                                                {
00562     amns_error_type f_error_status;
00563     //amns_error_type_c2f(*error_status, &f_error_status);
00564     IMAS_AMNS_C_RX_3_B(handle_rx_in, nx, ny, nz, out, arg1, arg2, &f_error_status);
00565     amns_error_type_f2c(f_error_status, error_status);
00566 }
```

Here is the call graph for this function:



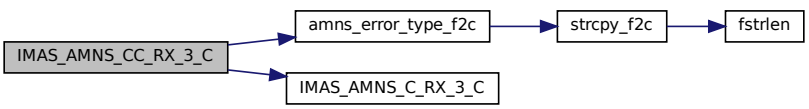
16.38.3.67 IMAS_AMNS_CC_RX_3_C()

```
void IMAS_AMNS_CC_RX_3_C (
            void * handle_rx_in,
            int nx,
            int ny,
            int nz,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            amns_c_error_type * error_status )
```

Definition at line 568 of file amns_interface.h.

```
00568                                                          {
00569     amns_error_type f_error_status;
00570     //amns_error_type_c2f(*error_status, &f_error_status);
00571     IMAS_AMNS_C_RX_3_C(handle_rx_in, nx, ny, nz, out, arg1, arg2, arg3, &f_error_status);
00572     amns_error_type_f2c(f_error_status, error_status);
00573 }
```

Here is the call graph for this function:



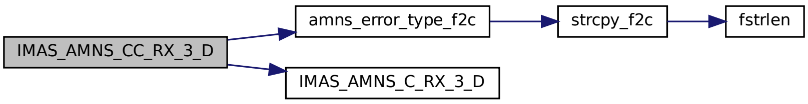
16.38.3.68 IMAS_AMNS_CC_RX_3_D()

```
void IMAS_AMNS_CC_RX_3_D (
            void * handle_rx_in,
            int nx,
            int ny,
            int nz,
            double * out,
            double * arg1,
            double * arg2,
            double * arg3,
            double * arg4,
            amns_c_error_type * error_status )
```

Definition at line 575 of file amns_interface.h.

```
00575                                                                                           {
00576     amns_error_type f_error_status;
00577     //amns_error_type_c2f(*error_status, &f_error_status);
00578     IMAS_AMNS_C_RX_3_D(handle_rx_in, nx, ny, nz, out, arg1, arg2, arg3, arg4, &f_error_status);
00579     amns_error_type_f2c(f_error_status, error_status);
00580 }
```

Here is the call graph for this function:



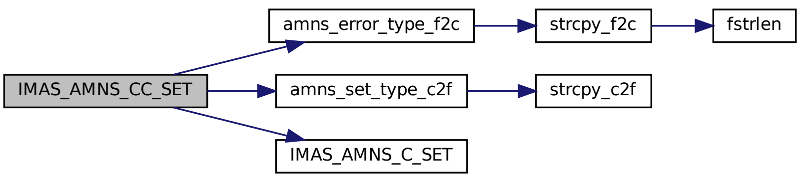
16.38.3.69 IMAS_AMNS_CC_SET()

```
void IMAS_AMNS_CC_SET (
            void * handle_in,
            amns_c_set_type * set,
            amns_c_error_type * error_status )
```

Definition at line 414 of file amns_interface.h.

```
00414                                                                                           {
00415     amns_error_type f_error_status;
00416     amns_set_type f_set;
00417     //amns_error_type_c2f(*error_status, &f_error_status);
00418     amns_set_type_c2f(*set, &f_set);
00419     IMAS_AMNS_C_SET(handle_in, &f_set, &f_error_status);
00420     amns_error_type_f2c(f_error_status, error_status);
00421     //amns_set_type_f2c(f_set, set);
00422 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

16.38.3.70 IMAS_AMNS_CC_SET_REACTANT()

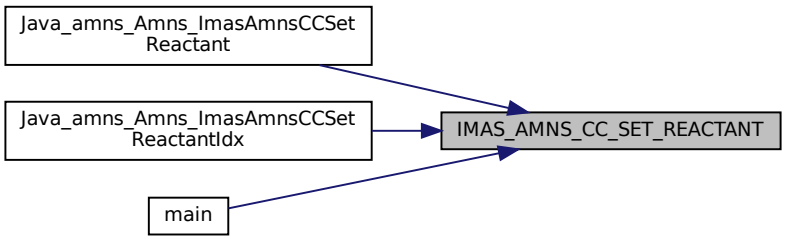
```
void IMAS_AMNS_CC_SET_REACTANT (
            void * reactants_handle_in,
            int reactant_index,
            amns_c_reactant_type * reactant_in )
```

Definition at line 589 of file amns_interface.h.

```
00589
              {
00590     // amns_c_reactant_type and amns_reactant_type are the same
00591     IMAS_AMNS_C_SET_REACTANT(reactants_handle_in, reactant_index, (amns_reactant_type*)reactant_in);
00592 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



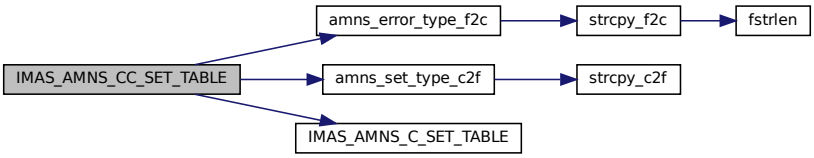
16.38.3.71 IMAS_AMNS_CC_SET_TABLE()

```
void IMAS_AMNS_CC_SET_TABLE (
            void * handle_rx_in,
            amns_c_set_type * set,
            amns_c_error_type * error_status )
```

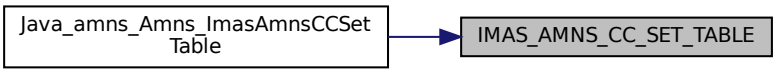
Definition at line 460 of file amns_interface.h.

```
00460
      {
00461     amns_error_type f_error_status;
00462     amns_set_type f_set;
00463     //amns_error_type_c2f(*error_status, &f_error_status);
00464     amns_set_type_c2f(*set, &f_set);
00465     IMAS_AMNS_C_SET_TABLE(handle_rx_in, &f_set, &f_error_status);
00466     amns_error_type_f2c(f_error_status, error_status);
00467     //amns_set_type_f2c(f_set, set);
00468 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



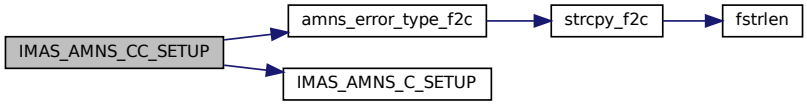
16.38.3.72 IMAS_AMNS_CC_SETUP()

```
void IMAS_AMNS_CC_SETUP (
            void ** handle_out,
            amns_c_error_type * error_status )
```

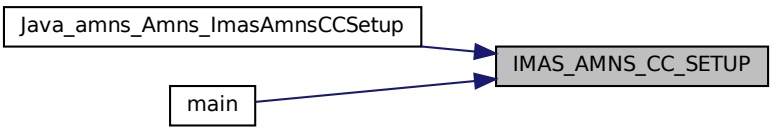
Definition at line 383 of file amns_interface.h.

```
00383                                                                    {
00384     amns_error_type f_error_status;
00385     //amns_error_type_c2f(*error_status, &f_error_status);
00386     IMAS_AMNS_C_SETUP(handle_out, &f_error_status);
00387     amns_error_type_f2c(f_error_status, error_status);
00388 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



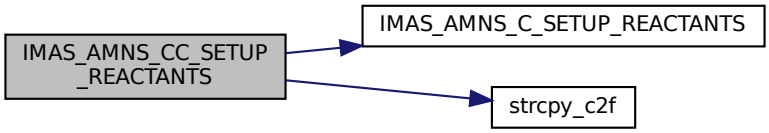
16.38.3.73 IMAS_AMNS_CC_SETUP_REACTANTS()

```
void IMAS_AMNS_CC_SETUP_REACTANTS (
            void ** reactants_handle_out,
            char * string_in,
            int index_in,
            int n_reactants )
```

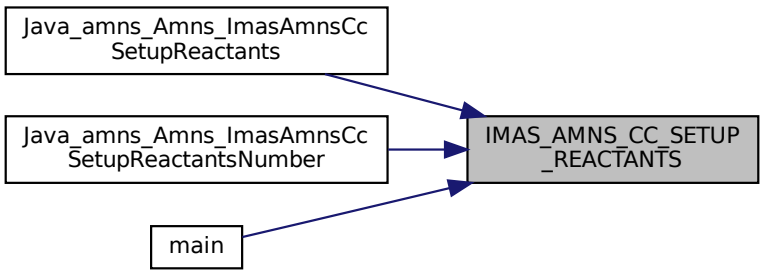
Definition at line 582 of file amns_interface.h.

```
00582
            {
00583     char f_string_in[reaction_length];
00584     strcpy_c2f(string_in, f_string_in, reaction_length);
00585     IMAS_AMNS_C_SETUP_REACTANTS(reactants_handle_out, f_string_in, index_in, n_reactants);
00586     //strcpy_f2c(f_string_inchar, reaction_length, string_in) {
00587 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



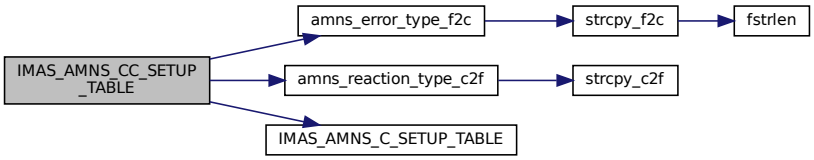
16.38.3.74 IMAS_AMNS_CC_SETUP_TABLE()

```
void IMAS_AMNS_CC_SETUP_TABLE (
            void * handle_in,
            amns_c_reaction_type * reaction_type,
            void * reactant_handle_in,
            void ** handle_rx_out,
            amns_c_error_type * error_status )
```

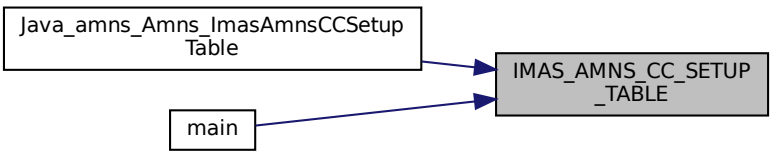
Definition at line 437 of file amns_interface.h.

```
00437
                                                                    {
00438     amns_error_type f_error_status;
00439     amns_reaction_type f_reaction;
00440     //amns_error_type_c2f(*error_status, &f_error_status);
00441     amns_reaction_type_c2f(*reaction_type, &f_reaction);
00442     IMAS_AMNS_C_SETUP_TABLE(handle_in, &f_reaction, reactant_handle_in, handle_rx_out,
      &f_error_status);
00443     amns_error_type_f2c(f_error_status, error_status);
00444     //amns_reaction_type_f2c(f_reaction, reaction_type);
00445 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.38.3.75 IMAS_AMNS_CC_SETUP_VERSION()

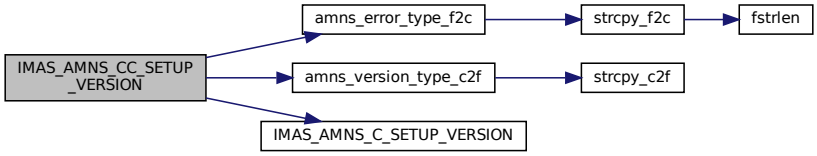
```
void IMAS_AMNS_CC_SETUP_VERSION (
            void ** handle_in,
            amns_c_version_type * version,
            amns_c_error_type * error_status )
```

Definition at line 390 of file amns_interface.h.

```
00390
              {
00391     amns_error_type f_error_status;
00392     amns_version_type f_version;
00393     //amns_error_type_c2f(*error_status, &f_error_status);
00394     amns_version_type_c2f(*version, &f_version);
00395     IMAS_AMNS_C_SETUP_VERSION(handle_in, &f_version, &f_error_status);
00396     amns_error_type_f2c(f_error_status, error_status);
```

```
00397     //amns_version_type_f2c(f_version, version);
00398 }
```

Here is the call graph for this function:



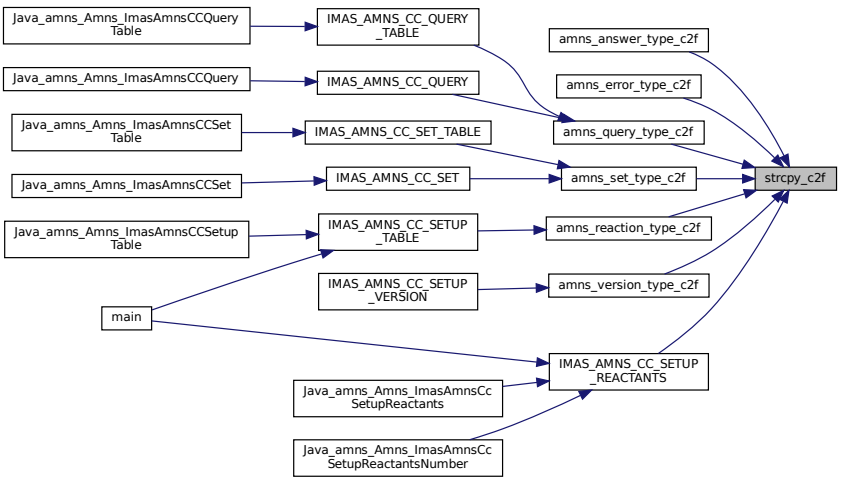
16.38.3.76 strcpy_c2f()

```
char * strcpy_c2f (
            char * csrc,
            char * fdest,
            int flen )
```

Definition at line 355 of file amns_interface.h.

```
00355                                                   {
00356     memset(fdest, ' ', flen);
00357     if (csrc != NULL) {
00358         int clen = strlen(csrc);
00359         if (clen > flen) {
00360             printf("strcpy_c2f: WARNING: C string exceeds size of target Fortran string.
      Truncating.");
00361             clen = flen;
00362         }
00363         memcpy(fdest, csrc, clen);
00364     }
00365     return fdest;
00366 }
```

Here is the caller graph for this function:



16.38.3.77 strcpy_f2c()

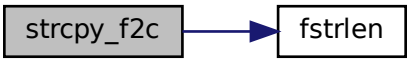
```
char * strcpy_f2c (
            char * fsrc,
```

```
            int flen,
            char ** cdest )
```

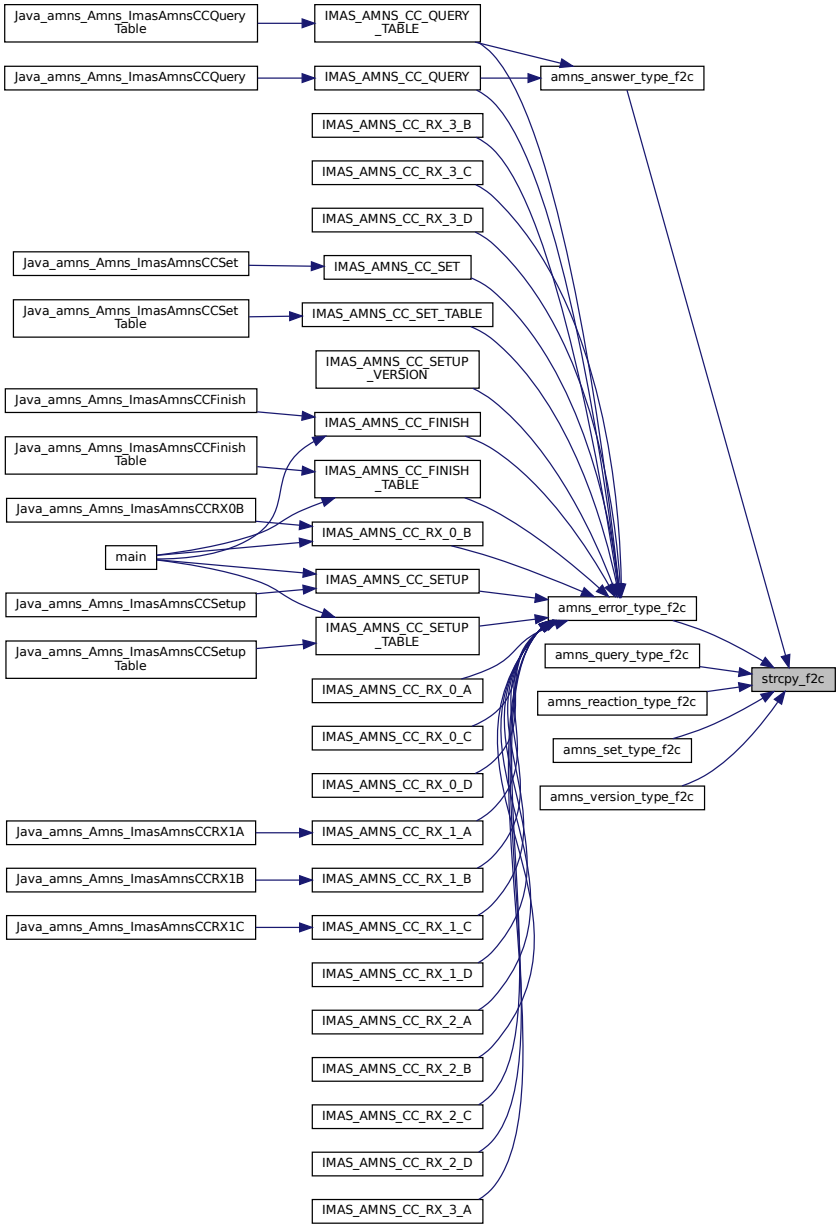
Definition at line 341 of file amns_interface.h.

```
00341                                                                {
00342     int trimlen = fstrlen(fsrc, flen);
00343     if (((*cdest) = malloc(trimlen+1)) != NULL) {
00344         memcpy(*cdest, fsrc, trimlen);
00345         *(*cdest + trimlen) = 0;
00346     }
00347     return *cdest;
00348 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.38.4 Variable Documentation

16.38.4.1 DEFAULT_AMNS_ANSWER_TYPE

```
const amns_answer_type DEFAULT_AMNS_ANSWER_TYPE = { "", 0 }
```

Definition at line 268 of file amns_interface.h.

16.38.4.2 DEFAULT_AMNS_C_ANSWER_TYPE

```
const amns_c_answer_type DEFAULT_AMNS_C_ANSWER_TYPE = { "", 0 }
```

Definition at line 269 of file amns_interface.h.

16.38.4.3 DEFAULT_AMNS_C_ERROR_TYPE

```
const amns_c_error_type DEFAULT_AMNS_C_ERROR_TYPE = { false, "" }
```
Definition at line 141 of file amns_interface.h.

16.38.4.4 DEFAULT_AMNS_C_QUERY_TYPE

```
const amns_c_query_type DEFAULT_AMNS_C_QUERY_TYPE = { "" }
```
Definition at line 240 of file amns_interface.h.

16.38.4.5 DEFAULT_AMNS_C_REACTANT_TYPE

```
const amns_c_reactant_type DEFAULT_AMNS_C_REACTANT_TYPE = { 0.0, 0.0, 0.0, 0, IMAS_INVALID_FLOAT,
IMAS_INVALID_INT }
```
Definition at line 114 of file amns_interface.h.

16.38.4.6 DEFAULT_AMNS_C_REACTION_TYPE

```
const amns_c_reaction_type DEFAULT_AMNS_C_REACTION_TYPE = { "", 0 }
```
Definition at line 178 of file amns_interface.h.

16.38.4.7 DEFAULT_AMNS_C_SET_TYPE

```
const amns_c_set_type DEFAULT_AMNS_C_SET_TYPE = { "" }
```
Definition at line 213 of file amns_interface.h.

16.38.4.8 DEFAULT_AMNS_C_VERSION_TYPE

```
const amns_c_version_type DEFAULT_AMNS_C_VERSION_TYPE = { "", 0, "", "" }
```
Definition at line 73 of file amns_interface.h.

16.38.4.9 DEFAULT_AMNS_ERROR_TYPE

```
const amns_error_type DEFAULT_AMNS_ERROR_TYPE = { false, "" }
```
Definition at line 140 of file amns_interface.h.

16.38.4.10 DEFAULT_AMNS_QUERY_TYPE

```
const amns_query_type DEFAULT_AMNS_QUERY_TYPE = { "" }
```
Definition at line 239 of file amns_interface.h.

16.38.4.11 DEFAULT_AMNS_REACTANT_TYPE

```
const amns_reactant_type DEFAULT_AMNS_REACTANT_TYPE = { 0.0, 0.0, 0.0, 0, IMAS_INVALID_FLOAT,
IMAS_INVALID_INT }
```
Definition at line 113 of file amns_interface.h.

16.38.4.12 DEFAULT_AMNS_REACTION_TYPE

```
const amns_reaction_type DEFAULT_AMNS_REACTION_TYPE = { "", 0 }
```

Definition at line 177 of file amns_interface.h.

16.38.4.13 DEFAULT_AMNS_SET_TYPE

```
const amns_set_type DEFAULT_AMNS_SET_TYPE = { "" }
```

Definition at line 212 of file amns_interface.h.

16.38.4.14 DEFAULT_AMNS_VERSION_TYPE

```
const amns_version_type DEFAULT_AMNS_VERSION_TYPE = { "", 0, "", "" }
```

Definition at line 72 of file amns_interface.h.

16.39 amns_interface.h

```
00001
00029 #include <ctype.h>
00030 #include <string.h>
00031 #include <stdbool.h>
00032 #include <stdio.h>
00033 #include <stdlib.h>
00034
00035 #ifndef AMNS_INTERFACE_H
00036 #define AMNS_INTERFACE_H
00037
00038
00039 // TODO: these should be uppercase...
00040 #define version_length 32
00041 #define set_length 32
00042 #define reaction_length 16
00043 #define query_length 16
00044 #define answer_length 128
00045 #define amns_max_length 128
00046
00047 char *strcpy_f2c(char *fsrc, int flen, char **cdest);
00048 char *strcpy_c2f(char *csrc, char *fdest, int flen);
00049
00054 typedef struct {
00055     char string[version_length];
00056     int number;
00057     char backend[version_length];
00058     char user[version_length];
00059 } amns_version_type;
00060
00065 typedef struct {
00066     char *string;
00067     int number;
00068     char *backend;
00069     char *user;
00070 } amns_c_version_type;
00071
00072 const amns_version_type DEFAULT_AMNS_VERSION_TYPE = { "", 0, "", "" };
00073 const amns_c_version_type DEFAULT_AMNS_C_VERSION_TYPE = { "", 0, "", "" };
00074
00075 amns_c_version_type get_default_amns_c_version_type(void);
00077 amns_c_version_type get_default_amns_c_version_type() {
00078   return DEFAULT_AMNS_C_VERSION_TYPE;
00079 }
00080
00081 void amns_version_type_f2c(amns_version_type ftype, amns_c_version_type *ctype) {
00082     strcpy_f2c(ftype.string, version_length, &((*ctype).string));
00083     ctype->number = ftype.number;
00084     strcpy_f2c(ftype.backend, version_length, &(ctype->backend));
00085     strcpy_f2c(ftype.user, version_length, &(ctype->user));
00086 }
00087
00088 void amns_version_type_c2f(amns_c_version_type ctype, amns_version_type *ftype) {
00089     strcpy_c2f(ctype.string, ftype->string, version_length);
00090     ftype->number = ctype.number;
00091     strcpy_c2f(ctype.backend, ftype->backend, version_length);
00092     strcpy_c2f(ctype.user, ftype->user, version_length);
00093 }
00094
00099 typedef struct {
```

```
00100   double ZN;
00101   double ZA;
00102   double MI;
00103   int LR;
00104   double real_specifier;
00105   int int_specifier;
00106 } amns_reactant_type;
00107
00108 typedef amns_reactant_type amns_c_reactant_type;
00109
00110 #define IMAS_INVALID_FLOAT -9.0E40
00111 #define IMAS_INVALID_INT   -999999999
00112
00113 const amns_reactant_type DEFAULT_AMNS_REACTANT_TYPE = { 0.0, 0.0, 0.0, 0, IMAS_INVALID_FLOAT,
       IMAS_INVALID_INT };
00114 const amns_c_reactant_type DEFAULT_AMNS_C_REACTANT_TYPE = { 0.0, 0.0, 0.0, 0, IMAS_INVALID_FLOAT,
       IMAS_INVALID_INT };
00115
00116 amns_c_reactant_type get_default_amns_c_reactant_type(void);
00118 amns_c_reactant_type get_default_amns_c_reactant_type() {
00119   return DEFAULT_AMNS_C_REACTANT_TYPE;
00120 }
00121
00126 typedef struct {
00127     bool flag;
00128     char string[answer_length];
00129 } amns_error_type;
00130
00135 typedef struct {
00136     bool flag;
00137     char *string;
00138 } amns_c_error_type;
00139
00140 const amns_error_type DEFAULT_AMNS_ERROR_TYPE = { false, "" };
00141 const amns_c_error_type DEFAULT_AMNS_C_ERROR_TYPE = { false, "" };
00142
00143 amns_c_error_type get_default_amns_c_error_type(void);
00145 amns_c_error_type get_default_amns_c_error_type() {
00146     return DEFAULT_AMNS_C_ERROR_TYPE;
00147 }
00148
00149 void amns_error_type_f2c(amns_error_type ftype, amns_c_error_type *ctype) {
00150     strcpy_f2c(ftype.string, answer_length, &(ctype->string));
00151     ctype->flag = ftype.flag;
00152 }
00153
00154 void amns_error_type_c2f(amns_c_error_type ctype, amns_error_type *ftype) {
00155     strcpy_c2f(ctype.string, ftype->string, answer_length);
00156     ftype->flag = ctype.flag;
00157 }
00158
00163 typedef struct {
00164     char string[reaction_length];
00165     int isotope_resolved;
00166 } amns_reaction_type;
00167
00172 typedef struct {
00173     char *string;
00174     int isotope_resolved;
00175 } amns_c_reaction_type;
00176
00177 const amns_reaction_type DEFAULT_AMNS_REACTION_TYPE = { "", 0 };
00178 const amns_c_reaction_type DEFAULT_AMNS_C_REACTION_TYPE = { "", 0 };
00179
00180 amns_c_reaction_type get_default_amns_c_reaction_type(void);
00182 amns_c_reaction_type get_default_amns_c_reaction_type() {
00183     return DEFAULT_AMNS_C_REACTION_TYPE;
00184 }
00185
00186 void amns_reaction_type_f2c(amns_reaction_type ftype, amns_c_reaction_type *ctype) {
00187     strcpy_f2c(ftype.string, reaction_length, &(ctype->string));
00188     ctype->isotope_resolved = ftype.isotope_resolved;
00189 }
00190
00191 void amns_reaction_type_c2f(amns_c_reaction_type ctype, amns_reaction_type *ftype) {
00192     strcpy_c2f(ctype.string, ftype->string, reaction_length);
00193     ftype->isotope_resolved = ctype.isotope_resolved;
00194 }
00195
00200 typedef struct {
00201     char string[set_length];
00202 } amns_set_type;
00203
00208 typedef struct {
00209     char *string;
00210 } amns_c_set_type;
00211
```

```
00212 const amns_set_type DEFAULT_AMNS_SET_TYPE = { "" };
00213 const amns_c_set_type DEFAULT_AMNS_C_SET_TYPE = { "" };
00214
00215 void amns_set_type_f2c(amns_set_type ftype, amns_c_set_type *ctype) {
00216     strcpy_f2c(ftype.string, set_length, &(ctype->string));
00217 }
00218
00219 void amns_set_type_c2f(amns_c_set_type ctype, amns_set_type *ftype) {
00220     strcpy_c2f(ctype.string, ftype->string, set_length);
00221 }
00222
00227 typedef struct {
00228     char string[query_length];
00229 } amns_query_type;
00230
00235 typedef struct {
00236     char *string;
00237 } amns_c_query_type;
00238
00239 const amns_query_type DEFAULT_AMNS_QUERY_TYPE = { "" };
00240 const amns_c_query_type DEFAULT_AMNS_C_QUERY_TYPE = { "" };
00241
00242 void amns_query_type_f2c(amns_query_type ftype, amns_c_query_type *ctype) {
00243     strcpy_f2c(ftype.string, query_length, &(ctype->string));
00244 }
00245
00246 void amns_query_type_c2f(amns_c_query_type ctype, amns_query_type *ftype) {
00247     strcpy_c2f(ctype.string, ftype->string, query_length);
00248 }
00249
00254 typedef struct {
00255     char string[answer_length];
00256     int number;
00257 } amns_answer_type;
00258
00263 typedef struct {
00264     char *string;
00265     int number;
00266 } amns_c_answer_type;
00267
00268 const amns_answer_type DEFAULT_AMNS_ANSWER_TYPE = { "", 0 };
00269 const amns_c_answer_type DEFAULT_AMNS_C_ANSWER_TYPE = { "", 0 };
00270
00271 void amns_answer_type_f2c(amns_answer_type ftype, amns_c_answer_type *ctype) {
00272     strcpy_f2c(ftype.string, answer_length, &(ctype->string));
00273     ctype->number = ftype.number;
00274 }
00275
00276 void amns_answer_type_c2f(amns_c_answer_type ctype, amns_answer_type *ftype) {
00277     strcpy_c2f(ctype.string, ftype->string, answer_length);
00278     ftype->number = ctype.number;
00279 }
00280
00281 // Function prototypes for the Fortran functions
00282
00283 #ifdef __cplusplus
00284 extern "C" {
00285 #endif
00286 void IMAS_AMNS_C_SETUP(void **handle_out, amns_error_type *error_status);
00287 void IMAS_AMNS_C_SETUP_VERSION(void **handle_in, amns_version_type *version, amns_error_type
       *error_status);
00288 void IMAS_AMNS_C_FINISH(void **handle_inout, amns_error_type *error_status);
00289 void IMAS_AMNS_C_FINISH_TABLE(void **handle_rx_inout, amns_error_type *error_status);
00290 void IMAS_AMNS_C_SET(void *handle_in, amns_set_type *set, amns_error_type *error_status);
00291 void IMAS_AMNS_C_QUERY(void *handle_in, amns_query_type *query, amns_answer_type *answer,
       amns_error_type *error_status);
00292
00293 void IMAS_AMNS_C_SETUP_TABLE(void *handle_in, amns_reaction_type *reaction_type, void
       *reactant_handle_in, void **handle_rx_out, amns_error_type *error_status);
00294 void IMAS_AMNS_C_QUERY_TABLE(void *handle_rx_in, amns_query_type *query, amns_answer_type *answer,
       amns_error_type *error_status);
00295 void IMAS_AMNS_C_SET_TABLE(void *handle_rx_in, amns_set_type *set, amns_error_type *error_status);
00296
00297 void IMAS_AMNS_C_RX_0_A(void *handle_rx_in, double *out, double arg1, amns_error_type *error_status);
00298 void IMAS_AMNS_C_RX_0_B(void *handle_rx_in, double *out, double arg1, double arg2, amns_error_type
       *error_status);
00299 void IMAS_AMNS_C_RX_0_C(void *handle_rx_in, double *out, double arg1, double arg2, double arg3,
       amns_error_type *error_status);
00300 void IMAS_AMNS_C_RX_0_D(void *handle_rx_in, double *out, double arg1, double arg2, double arg3, double
       arg4, amns_error_type *error_status);
00301
00302 void IMAS_AMNS_C_RX_1_A(void *handle_rx_in, int nx, double *out, double *arg1, amns_error_type
       *error_status);
00303 void IMAS_AMNS_C_RX_1_B(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2,
       amns_error_type *error_status);
00304 void IMAS_AMNS_C_RX_1_C(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, double
       *arg3, amns_error_type *error_status);
```

```
00305 void IMAS_AMNS_C_RX_1_D(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, double
       *arg3, double *arg4, amns_error_type *error_status);
00306
00307 void IMAS_AMNS_C_RX_2_A(void *handle_rx_in, int nx, int ny, double *out, double *arg1, amns_error_type
       *error_status);
00308 void IMAS_AMNS_C_RX_2_B(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2,
       amns_error_type *error_status);
00309 void IMAS_AMNS_C_RX_2_C(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2,
       double *arg3, amns_error_type *error_status);
00310 void IMAS_AMNS_C_RX_2_D(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2,
       double *arg3, double *arg4, amns_error_type *error_status);
00311
00312 void IMAS_AMNS_C_RX_3_A(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1,
       amns_error_type *error_status);
00313 void IMAS_AMNS_C_RX_3_B(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double
       *arg2, amns_error_type *error_status);
00314 void IMAS_AMNS_C_RX_3_C(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double
       *arg2, double *arg3, amns_error_type *error_status);
00315 void IMAS_AMNS_C_RX_3_D(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double
       *arg2, double *arg3, double *arg4, amns_error_type *error_status);
00316
00317 void IMAS_AMNS_C_SETUP_REACTANTS(void **reactants_handle_out, char string_in[reaction_length], int
       index_in, int n_reactants);
00318 void IMAS_AMNS_C_SET_REACTANT(void *reactants_handle_in, int reactant_index, amns_reactant_type
       *reactant_in);
00319 void IMAS_AMNS_C_GET_REACTANT(void *reactants_handle_in, int reactant_index, amns_reactant_type
       *reactant_out);
00320 void IMAS_AMNS_C_FINISH_REACTANTS(void **reactants_handle_inout);
00321
00322 #ifdef __cplusplus
00323 }
00324 #endif
00325
00326
00327 // Routines for C / Fortran string conversion
00328
00329 /* Return the length of a Fortran string, not counting trailing whitespace */
00330 int fstrlen(char *fstr, int flen) {
00331     char *end;
00332     end = fstr + flen - 1;
00333     while(end >= fstr && isspace(*end)) end--;
00334     return end - fstr + 1;
00335 }
00336
00337 /* Copy fixed-length Fortran string to C, omitting trailing whitespace
00338    and allocating memory as needed. If the Fortran string contains only
00339    whitespace (i.e., is empty), the C string is allocated but of zero length
00340    (i.e. only contains the \0 termination character).*/
00341 char *strcpy_f2c(char *fsrc, int flen, char **cdest) {
00342     int trimlen = fstrlen(fsrc, flen);
00343     if (((*cdest) = malloc(trimlen+1)) != NULL) {
00344         memcpy(*cdest, fsrc, trimlen);
00345         *(*cdest + trimlen) = 0;
00346     }
00347     return *cdest;
00348 }
00349
00350 /* Copy zero-terminated C string into fixed-length Fortran string, ensuring
00351    that the allowed string length is not exceeded. The Fortran string
00352    fdest is assumed to be allocated to the length flen. If the C string
00353    is unassociated (NULL), the Fortran string is returned empty (i.e., filled with
00354    whitespace/spaces) */
00355 char *strcpy_c2f(char *csrc, char *fdest, int flen) {
00356     memset(fdest, ' ', flen);
00357     if (csrc != NULL) {
00358         int clen = strlen(csrc);
00359         if (clen > flen) {
00360             printf("strcpy_c2f: WARNING: C string exceeds size of target Fortran string.
       Truncating.");
00361             clen = flen;
00362         }
00363         memcpy(fdest, csrc, clen);
00364     }
00365     return fdest;
00366 }
00367
00368
00369 // Convenience versions of the IMAS_AMNS_C_* routines using the amns_c_* variants
00370 // of the AMNS structures. These routines transparently handle conversion between
00371 // zero-terminated C strings and fixed length Fortran strings
00372 //
00373 // Notation: CC = "C Convenience"
00374 //
00375 // Note: the routines below contain calls to copy routines in both directions
00376 //       for all arguments of C convenience type. In the Fortran code, the
00377 //       arguments are specified to be intent in, out or inout. Thus some
00378 //       copy calls are unnecessary, and commented out in the below code.
```

```
00379 //
00380 // TODO: for some routines, the convenience routines are missing (commented below)
00381
00382
00383 void IMAS_AMNS_CC_SETUP(void **handle_out, amns_c_error_type *error_status) {
00384     amns_error_type f_error_status;
00385     //amns_error_type_c2f(*error_status, &f_error_status);
00386     IMAS_AMNS_C_SETUP(handle_out, &f_error_status);
00387     amns_error_type_f2c(f_error_status, error_status);
00388 }
00389
00390 void IMAS_AMNS_CC_SETUP_VERSION(void **handle_in, amns_c_version_type *version, amns_c_error_type
      *error_status) {
00391     amns_error_type f_error_status;
00392     amns_version_type f_version;
00393     //amns_error_type_c2f(*error_status, &f_error_status);
00394     amns_version_type_c2f(*version, &f_version);
00395     IMAS_AMNS_C_SETUP_VERSION(handle_in, &f_version, &f_error_status);
00396     amns_error_type_f2c(f_error_status, error_status);
00397     //amns_version_type_f2c(f_version, version);
00398 }
00399
00400 void IMAS_AMNS_CC_FINISH(void **handle_inout, amns_c_error_type *error_status) {
00401     amns_error_type f_error_status;
00402     //amns_error_type_c2f(*error_status, &f_error_status);
00403     IMAS_AMNS_C_FINISH(handle_inout, &f_error_status);
00404     amns_error_type_f2c(f_error_status, error_status);
00405 }
00406
00407 void IMAS_AMNS_CC_FINISH_TABLE(void **handle_rx_inout, amns_c_error_type *error_status) {
00408     amns_error_type f_error_status;
00409     //amns_error_type_c2f(*error_status, &f_error_status);
00410     IMAS_AMNS_C_FINISH_TABLE(handle_rx_inout, &f_error_status);
00411     amns_error_type_f2c(f_error_status, error_status);
00412 }
00413
00414 void IMAS_AMNS_CC_SET(void *handle_in, amns_c_set_type *set, amns_c_error_type *error_status) {
00415     amns_error_type f_error_status;
00416     amns_set_type f_set;
00417     //amns_error_type_c2f(*error_status, &f_error_status);
00418     amns_set_type_c2f(*set, &f_set);
00419     IMAS_AMNS_C_SET(handle_in, &f_set, &f_error_status);
00420     amns_error_type_f2c(f_error_status, error_status);
00421     //amns_set_type_f2c(f_set, set);
00422 }
00423
00424 void IMAS_AMNS_CC_QUERY(void *handle_in, amns_c_query_type *query, amns_c_answer_type *answer,
      amns_c_error_type *error_status) {
00425     amns_error_type f_error_status;
00426     amns_answer_type f_answer;
00427     amns_query_type f_query;
00428     //amns_error_type_c2f(*error_status, &f_error_status);
00429     //amns_answer_type_c2f(*answer, &f_answer);
00430     amns_query_type_c2f(*query, &f_query);
00431     IMAS_AMNS_C_QUERY(handle_in, &f_query, &f_answer, &f_error_status);
00432     amns_error_type_f2c(f_error_status, error_status);
00433     amns_answer_type_f2c(f_answer, answer);
00434     //amns_query_type_f2c(f_query, query);
00435 }
00436
00437 void IMAS_AMNS_CC_SETUP_TABLE(void *handle_in, amns_c_reaction_type *reaction_type, void
      *reactant_handle_in, void **handle_rx_out, amns_c_error_type *error_status) {
00438     amns_error_type f_error_status;
00439     amns_reaction_type f_reaction;
00440     //amns_error_type_c2f(*error_status, &f_error_status);
00441     amns_reaction_type_c2f(*reaction_type, &f_reaction);
00442     IMAS_AMNS_C_SETUP_TABLE(handle_in, &f_reaction, reactant_handle_in, handle_rx_out,
      &f_error_status);
00443     amns_error_type_f2c(f_error_status, error_status);
00444     //amns_reaction_type_f2c(f_reaction, reaction_type);
00445 }
00446
00447 void IMAS_AMNS_CC_QUERY_TABLE(void *handle_rx_in, amns_c_query_type *query, amns_c_answer_type
      *answer, amns_c_error_type *error_status) {
00448     amns_error_type f_error_status;
00449     amns_query_type f_query;
00450     amns_answer_type f_answer;
00451     //amns_error_type_c2f(*error_status, &f_error_status);
00452     amns_query_type_c2f(*query, &f_query);
00453     //amns_answer_type_c2f(*answer, &f_answer);
00454     IMAS_AMNS_C_QUERY_TABLE(handle_rx_in, &f_query, &f_answer, &f_error_status);
00455     amns_error_type_f2c(f_error_status, error_status);
00456     //amns_query_type_f2c(f_query, query);
00457     amns_answer_type_f2c(f_answer, answer);
00458 }
00459
00460 void IMAS_AMNS_CC_SET_TABLE(void *handle_rx_in, amns_c_set_type *set, amns_c_error_type *error_status)
```

```
      {
00461     amns_error_type f_error_status;
00462     amns_set_type f_set;
00463     //amns_error_type_c2f(*error_status, &f_error_status);
00464     amns_set_type_c2f(*set, &f_set);
00465     IMAS_AMNS_C_SET_TABLE(handle_rx_in, &f_set, &f_error_status);
00466     amns_error_type_f2c(f_error_status, error_status);
00467     //amns_set_type_f2c(f_set, set);
00468 }
00469
00470 void IMAS_AMNS_CC_RX_0_A(void *handle_rx_in, double *out, double arg1, amns_c_error_type
       *error_status) {
00471     amns_error_type f_error_status;
00472     //amns_error_type_c2f(*error_status, &f_error_status);
00473     IMAS_AMNS_C_RX_0_A(handle_rx_in, out, arg1, &f_error_status);
00474     amns_error_type_f2c(f_error_status, error_status);
00475 }
00476
00477 void IMAS_AMNS_CC_RX_0_B(void *handle_rx_in, double *out, double arg1, double arg2, amns_c_error_type
       *error_status) {
00478     amns_error_type f_error_status;
00479     //amns_error_type_c2f(*error_status, &f_error_status);
00480     IMAS_AMNS_C_RX_0_B(handle_rx_in, out, arg1, arg2, &f_error_status);
00481     amns_error_type_f2c(f_error_status, error_status);
00482 }
00483
00484 void IMAS_AMNS_CC_RX_0_C(void *handle_rx_in, double *out, double arg1, double arg2, double arg3,
       amns_c_error_type *error_status) {
00485     amns_error_type f_error_status;
00486     //amns_error_type_c2f(*error_status, &f_error_status);
00487     IMAS_AMNS_C_RX_0_C(handle_rx_in, out, arg1, arg2, arg3, &f_error_status);
00488     amns_error_type_f2c(f_error_status, error_status);
00489 }
00490
00491 void IMAS_AMNS_CC_RX_0_D(void *handle_rx_in, double *out, double arg1, double arg2, double arg3,
       double arg4, amns_c_error_type *error_status) {
00492     amns_error_type f_error_status;
00493     //amns_error_type_c2f(*error_status, &f_error_status);
00494     IMAS_AMNS_C_RX_0_D(handle_rx_in, out, arg1, arg2, arg3, arg4, &f_error_status);
00495     amns_error_type_f2c(f_error_status, error_status);
00496 }
00497
00498 void IMAS_AMNS_CC_RX_1_A(void *handle_rx_in, int nx, double *out, double *arg1, amns_c_error_type
       *error_status) {
00499     amns_error_type f_error_status;
00500     //amns_error_type_c2f(*error_status, &f_error_status);
00501     IMAS_AMNS_C_RX_1_A(handle_rx_in, nx, out, arg1, &f_error_status);
00502     amns_error_type_f2c(f_error_status, error_status);
00503 }
00504
00505 void IMAS_AMNS_CC_RX_1_B(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2,
       amns_c_error_type *error_status) {
00506     amns_error_type f_error_status;
00507     //amns_error_type_c2f(*error_status, &f_error_status);
00508     IMAS_AMNS_C_RX_1_B(handle_rx_in, nx, out, arg1, arg2, &f_error_status);
00509     amns_error_type_f2c(f_error_status, error_status);
00510 }
00511
00512 void IMAS_AMNS_CC_RX_1_C(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, double
       *arg3, amns_c_error_type *error_status) {
00513     amns_error_type f_error_status;
00514     //amns_error_type_c2f(*error_status, &f_error_status);
00515     IMAS_AMNS_C_RX_1_C(handle_rx_in, nx, out, arg1, arg2, arg3, &f_error_status);
00516     amns_error_type_f2c(f_error_status, error_status);
00517 }
00518
00519 void IMAS_AMNS_CC_RX_1_D(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2, double
       *arg3, double *arg4, amns_c_error_type *error_status) {
00520     amns_error_type f_error_status;
00521     //amns_error_type_c2f(*error_status, &f_error_status);
00522     IMAS_AMNS_C_RX_1_D(handle_rx_in, nx, out, arg1, arg2, arg3, arg4, &f_error_status);
00523     amns_error_type_f2c(f_error_status, error_status);
00524 }
00525
00526 void IMAS_AMNS_CC_RX_2_A(void *handle_rx_in, int nx, int ny, double *out, double *arg1,
       amns_c_error_type *error_status) {
00527     amns_error_type f_error_status;
00528     //amns_error_type_c2f(*error_status, &f_error_status);
00529     IMAS_AMNS_C_RX_2_A(handle_rx_in, nx, ny, out, arg1, &f_error_status);
00530     amns_error_type_f2c(f_error_status, error_status);
00531 }
00532
00533 void IMAS_AMNS_CC_RX_2_B(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2,
       amns_c_error_type *error_status) {
00534     amns_error_type f_error_status;
00535     //amns_error_type_c2f(*error_status, &f_error_status);
00536     IMAS_AMNS_C_RX_2_B(handle_rx_in, nx, ny, out, arg1, arg2, &f_error_status);
```

```
00537     amns_error_type_f2c(f_error_status, error_status);
00538 }
00539
00540 void IMAS_AMNS_CC_RX_2_C(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2,
      double *arg3, amns_c_error_type *error_status) {
00541     amns_error_type f_error_status;
00542     //amns_error_type_c2f(*error_status, &f_error_status);
00543     IMAS_AMNS_C_RX_2_C(handle_rx_in, nx, ny, out, arg1, arg2, arg3, &f_error_status);
00544     amns_error_type_f2c(f_error_status, error_status);
00545 }
00546
00547 void IMAS_AMNS_CC_RX_2_D(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double *arg2,
      double *arg3, double *arg4, amns_c_error_type *error_status) {
00548     amns_error_type f_error_status;
00549     //amns_error_type_c2f(*error_status, &f_error_status);
00550     IMAS_AMNS_C_RX_2_D(handle_rx_in, nx, ny, out, arg1, arg2, arg3, arg4, &f_error_status);
00551     amns_error_type_f2c(f_error_status, error_status);
00552 }
00553
00554 void IMAS_AMNS_CC_RX_3_A(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1,
      amns_c_error_type *error_status) {
00555     amns_error_type f_error_status;
00556     //amns_error_type_c2f(*error_status, &f_error_status);
00557     IMAS_AMNS_C_RX_3_A(handle_rx_in, nx, ny, nz, out, arg1, &f_error_status);
00558     amns_error_type_f2c(f_error_status, error_status);
00559 }
00560
00561 void IMAS_AMNS_CC_RX_3_B(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double
      *arg2, amns_c_error_type *error_status) {
00562     amns_error_type f_error_status;
00563     //amns_error_type_c2f(*error_status, &f_error_status);
00564     IMAS_AMNS_C_RX_3_B(handle_rx_in, nx, ny, nz, out, arg1, arg2, &f_error_status);
00565     amns_error_type_f2c(f_error_status, error_status);
00566 }
00567
00568 void IMAS_AMNS_CC_RX_3_C(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double
      *arg2, double *arg3, amns_c_error_type *error_status) {
00569     amns_error_type f_error_status;
00570     //amns_error_type_c2f(*error_status, &f_error_status);
00571     IMAS_AMNS_C_RX_3_C(handle_rx_in, nx, ny, nz, out, arg1, arg2, arg3, &f_error_status);
00572     amns_error_type_f2c(f_error_status, error_status);
00573 }
00574
00575 void IMAS_AMNS_CC_RX_3_D(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1, double
      *arg2, double *arg3, double *arg4, amns_c_error_type *error_status) {
00576     amns_error_type f_error_status;
00577     //amns_error_type_c2f(*error_status, &f_error_status);
00578     IMAS_AMNS_C_RX_3_D(handle_rx_in, nx, ny, nz, out, arg1, arg2, arg3, arg4, &f_error_status);
00579     amns_error_type_f2c(f_error_status, error_status);
00580 }
00581
00582 void IMAS_AMNS_CC_SETUP_REACTANTS(void **reactants_handle_out, char *string_in, int index_in, int
      n_reactants) {
00583     char f_string_in[reaction_length];
00584     strcpy_c2f(string_in, f_string_in, reaction_length);
00585     IMAS_AMNS_C_SETUP_REACTANTS(reactants_handle_out, f_string_in, index_in, n_reactants);
00586     //strcpy_f2c(f_string_inchar, reaction_length, string_in) {
00587 }
00588
00589 void IMAS_AMNS_CC_SET_REACTANT(void *reactants_handle_in, int reactant_index, amns_c_reactant_type
      *reactant_in) {
00590     // amns_c_reactant_type and amns_reactant_type are the same
00591     IMAS_AMNS_C_SET_REACTANT(reactants_handle_in, reactant_index, (amns_reactant_type*)reactant_in);
00592 }
00593
00594 void IMAS_AMNS_CC_GET_REACTANT(void *reactants_handle_in, int reactant_index, amns_c_reactant_type
      *reactant_out) {
00595     IMAS_AMNS_C_GET_REACTANT(reactants_handle_in, reactant_index, (amns_reactant_type*)reactant_out);
00596 }
00597
00598 void IMAS_AMNS_CC_FINISH_REACTANTS(void **reactants_handle_inout) {
00599     IMAS_AMNS_C_FINISH_REACTANTS(reactants_handle_inout);
00600 }
00601
00602 #endif
00603
00604
```

16.40 src/amns_driver/amns_adas.f90 File Reference

Modules

- module amns_adas

Functions/Subroutines

- subroutine amns_adas::adas_amns (amns, root_in, zn_in, amn_in, nreac_in, sp_in, lu_read)

  produce the AMNS database from ADAS data
- integer function handle_coordinates (ncoord, dtev, ddens)
- character(len=len(string)) function upcase (string)
- subroutine allocate_process (process, nr, np)
- subroutine assign_reactantproduct (reactantproduct, label, zn, amn, multiplicity, relative, za)

16.40.1 Function/Subroutine Documentation

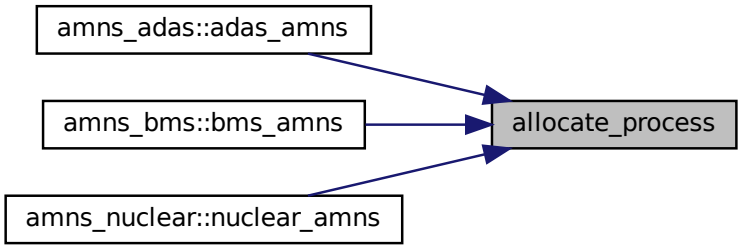
16.40.1.1 allocate_process()

```
subroutine adas_amns::allocate_process (
              type (ids_amns_data_process) process,
              integer nr,
              integer np )
```

Definition at line 390 of file amns_adas.f90.

```
00391       use ids_schemas   ! IGNORE
00392       implicit none
00393       type (ids_amns_data_process) :: process
00394       integer :: nr, np
00395       integer :: irp
00396
00397       allocate(process%reactants(nr), process%products(np))
00398       do irp = 1, size(process%reactants)
00399          allocate(process%reactants(irp)%label(1))
00400          allocate(process%reactants(irp)%element(1))
00401 !         allocate(process%reactants(irp)%element(1)%label(1))
00402       enddo
00403       do irp = 1, size(process%products)
00404          allocate(process%products(irp)%label(1))
00405          allocate(process%products(irp)%element(1))
00406 !         allocate(process%products(irp)%element(1)%label(1))
00407       enddo
00408
```

Here is the caller graph for this function:



16.40.1.2 assign_reactantproduct()

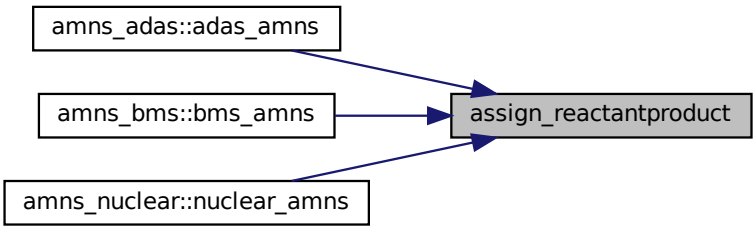
```
subroutine adas_amns::assign_reactantproduct (
              type (ids_amns_data_process_reactant) reactantproduct,
              character(len=*) label,
              integer zn,
              real (ids_real) amn,
```

```
            integer multiplicity,
            integer relative,
            real (ids_real) za )
```

Definition at line 411 of file amns_adas.f90.

```
00412       use ids_schemas   ! IGNORE
00413       implicit none
00414       type (ids_amns_data_process_reactant) :: reactantproduct
00415       character(len=*) :: label
00416       integer :: zn, multiplicity, relative
00417       real (ids_real) :: amn, za
00418
00419       reactantproduct%label(1)=label
00420       reactantproduct%multiplicity=multiplicity
00421       reactantproduct%element(1)%atoms_n=multiplicity
00422 !      reactantproduct%element(1)%label(1)=label
00423       reactantproduct%element(1)%z_n=zn
00424       reactantproduct%element(1)%a=nint(amn)
00425       reactantproduct%mass=amn
00426       reactantproduct%relative_charge=relative
00427       reactantproduct%charge=za
00428
```

Here is the caller graph for this function:



16.40.1.3 handle_coordinates()

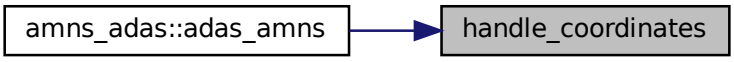
```
integer function adas_amns::handle_coordinates (
            integer ncoord,
            real (ids_real), dimension(:)  dtev,
            real (ids_real), dimension(:)  ddens )
```

Definition at line 311 of file amns_adas.f90.

```
00312
00313 ! amns%process%id(1, jproc) gives the dimension of the data table,
00314 ! i.e. the number of dependent variables (in the example shown here
00315 ! the variables are always density and temperature and the table is
00316 ! therefore 2d).
00317 ! amns%process%id(2, jproc) gives the position in the table, i.e. the last
00318 ! argument in the table. For instance for a 2D table, and process jproc the
00319 ! data are given by amns%process%tables_2d%table(:,;,jproc, ipos) where
00320 ! ipos =  amns%process%id(2, jproc).
00321 ! Dependent variables (electron density and temperature in this case)
00322
00323       implicit none
00324
00325       real (ids_real), dimension(:):: ddens
00326       real (ids_real), dimension(:):: dtev
00327       integer ncoord, icoord, itmax, idmax
00328
00329       itmax=size(dtev)
00330       idmax=size(ddens)
00331 ! check whether we already have the coordinate
00332       do icoord=1, ncoord
00333          if(itmax.ne.size(amns%coordinate_system(icoord)%coordinate(1)%values)) cycle
00334          if(idmax.ne.size(amns%coordinate_system(icoord)%coordinate(2)%values)) cycle
00335          write(*,*) maxval(abs(dtev-amns%coordinate_system(icoord)%coordinate(1)%values))
00336          if(maxval(abs(dtev-amns%coordinate_system(icoord)%coordinate(1)%values)).gt.0.0) cycle
00337          write(*,*) maxval(abs(6.0 + ddens-amns%coordinate_system(icoord)%coordinate(2)%values))
00338          if(maxval(abs(6.0 + ddens-amns%coordinate_system(icoord)%coordinate(2)%values)).gt.0.0) cycle
```

```
00339 ! we have a match
00340          handle_coordinates=icoord
00341          return
00342       enddo
00343 ! we have no match
00344       ncoord=ncoord+1
00345       allocate(amns%coordinate_system(ncoord)%coordinate(2))
00346       do idim=1, 2
00347          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%extrapolation_type(2))
00348          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%label(1))
00349          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%units(1))
00350       enddo
00351       allocate(amns%coordinate_system(ncoord)%coordinate(1)%values(itmax))
00352       allocate(amns%coordinate_system(ncoord)%coordinate(2)%values(idmax))
00353
00354       amns%coordinate_system(ncoord)%coordinate(1)%label(1) = 'Electron Temperature'
00355       amns%coordinate_system(ncoord)%coordinate(1)%extrapolation_type = (/ 2, 2 /)
00356       amns%coordinate_system(ncoord)%coordinate(1)%interpolation_type = 1
00357       amns%coordinate_system(ncoord)%coordinate(1)%units = 'eV'
00358       amns%coordinate_system(ncoord)%coordinate(1)%transformation = 1
00359       amns%coordinate_system(ncoord)%coordinate(1)%spacing = 0
00360       amns%coordinate_system(ncoord)%coordinate(1)%values = dtev(1:itmax)
00361
00362       amns%coordinate_system(ncoord)%coordinate(2)%label(1) = 'Electron Density'
00363       amns%coordinate_system(ncoord)%coordinate(2)%extrapolation_type = (/ 2, 2 /)
00364       amns%coordinate_system(ncoord)%coordinate(2)%interpolation_type = 1
00365       amns%coordinate_system(ncoord)%coordinate(2)%units = 'm^{-3}'
00366       amns%coordinate_system(ncoord)%coordinate(2)%transformation = 1
00367       amns%coordinate_system(ncoord)%coordinate(2)%spacing = 0
00368       amns%coordinate_system(ncoord)%coordinate(2)%values = 6.0 + ddens(1:idmax)
00369
00370       handle_coordinates=ncoord
00371       return
00372
```

Here is the caller graph for this function:



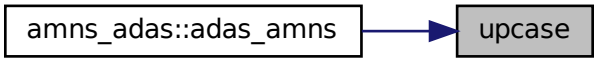
16.40.1.4 upcase()

```
character(len=len(string)) function adas_amns::upcase (
            character(len=*), intent(in) string )
```

Definition at line 375 of file amns_adas.f90.

```
00376       character(len=*), intent(in) :: string
00377       character(len=len(string)) :: upper
00378       integer :: j
00379       do j = 1,len(string)
00380          if(string(j:j) >= "a" .and. string(j:j) <= "z") then
00381             upper(j:j) = achar(iachar(string(j:j)) - 32)
00382          else
00383             upper(j:j) = string(j:j)
00384          end if
00385       end do
00386
```

Here is the caller graph for this function:



16.41 amns_adas.f90

```
00001 module amns_adas
00002
00020
00021 contains
00022
00023   subroutine adas_amns(amns, root_in, zn_in, amn_in, nreac_in, sp_in, lu_read)
00024
00025     !-----------------------------------------------------------------------
00026     !
00027     !  SUBROUTINE : adas_amns
00028     !
00029     !  PURPOSE    : to read ADAS adf11 data for carbon and to output into
00030     !               AMNS datastructure format.
00031     !
00032     !  NOTES      : (1) Demonstration project
00033     !
00034     !  ROUTINES:
00035     !          routine    source    brief description
00036     !          ------------------------------------------------------------
00037     !          xxdata_11  adas      read contents of adf11 dataset
00038     !          i4unit     adas      fetch unit number for output of messages
00039     !          i4fctn     adas      convert string to integer form
00040     !          xfelem     adas      return element name given nuclear charge
00041     !          xxword     adas      extract position of number in buffer
00042     !          xxslen     adas      find string less front and tail blanks
00043     !          xxcase     adas      convert a string to upper or lower case
00044     !          xxrptn     adas      analyse an adf11 file partition block
00045     !
00046     !
00047     !
00048     !
00049     ! VERSION : 1.0
00050     ! DATE    : 24-03-2010
00051     ! MODIFIED: Martin O'Mullane
00052     !           - First version.
00053     !
00054     ! DATE    : 07-07-2010
00055     ! MODIFIED: Lars-Goran Eriksson
00056     !               - Extended to eight processes for Carbon and more complete
00057     !                 entries in the data structure.
00058     !
00059     ! DATE    : 2010-12-14
00060     ! MODIFIED: David Coster
00061     !               - Generalized
00062     !
00063     !-----------------------------------------------------------------------
00064
00065     use ids_schemas   ! IGNORE
00066     use ids_types       ! IGNORE
00067     use codata, only: &
00068          & e_mass_in_amu => electron_mass_in_u, &
00069          & d_mass_in_amu => deuteron_mass_in_u
00070
00071     implicit none
00072     !-----------------------------------------------------------------------
00073
00074     ! AMNS
00075
00076     type (ids_amns_data)                :: amns
00077
00078     real (ids_real), parameter :: zero = 0.0
00079     real (ids_real), parameter :: log_zero = -300.0
00080
00081     ! xxdata_11 variables
00082
```

```
00083     integer, parameter               :: ISDIMD = 200, izdimd = 92, itdimd = 50 , &
00084          iddimd = 40,  ndptn  = 128, ndptnl = 4 , &
00085          ndptnc = 256, ndcnct = 100
00086
00087     integer                          :: IZ0       , IS1MIN    , IS1MAX,  IBLMX , &
00088          ISMAX     , ITMAX     , IDMAX ,  NPTNL , &
00089          NCNCT     , iclass
00090     integer                          :: idmax1, idmax2, idmax3,  idmax4
00091     integer                          :: idmax5, idmax6, idmax7,  idmax8
00092     integer                          :: itmax1, itmax2, itmax3, itmax4
00093     integer                          :: itmax5, itmax6, itmax7, itmax8
00094     real (ids_real)                   :: DNR_AMS
00095     character(len=12)                :: DNR_ELE
00096     logical                          :: LRES      , LSTAN     , LPTN
00097     integer, dimension(NDPTNL)       :: nptn      , IPTNLA
00098     integer, dimension(NDPTNL,NDPTN) :: NPTNC     , IPTNA
00099     integer, dimension(NDPTNL,NDPTN,NDPTNC) :: IPTNCA
00100     integer, dimension(NDCNCT)       :: ICNCTV
00101     integer, dimension(ISDIMD)       :: ISPPR     , ISPBR     , ISSTGR
00102     real (ids_real), dimension(IDDIMD):: ddens
00103     real (ids_real), dimension(ITDIMD):: dtev
00104     real (ids_real), dimension(ISDIMD,ITDIMD,IDDIMD):: drcof
00105
00106     ! program variables
00107
00108     character (len=256)              :: root_in
00109     character (len=256)              :: file
00110     integer                          :: zn_in
00111     integer                          :: nreac_in
00112     character (len=10)               :: sp_in
00113     character (len=11)               :: sp_loc, sp_up
00114     real (ids_real)                   :: amn_in
00115     integer                          :: lu_read
00116
00117     integer                          :: npoints, iz, iunit, iostat, nz, iproc, nprocs, idim
00118     integer                          :: is, ireac
00119     integer                          :: ncoord
00120
00121     ncoord = 0
00122
00123 !-----------------------------------------------------------------------
00124     allocate (amns%time(1))
00125     amns%time=0.0
00126 !    allocate (amns%version(1))
00127 !    allocate (amns%source(1))
00128 !-----------------------------------------------------------------------
00129
00130     sp_loc = sp_in
00131     if(index(sp_loc,'_').gt.0) sp_loc=trim(sp_loc) // '_'
00132     sp_up = upcase(sp_loc)
00133
00134 !    amns%version = 'v0'
00135 !    amns%source  = 'ADAS'
00136
00137     ! Element information
00138
00139     amns%z_n      = real(zn_in)
00140     amns%a        = amn_in
00141
00142
00143 !-----------------------
00144 ! Allocate the structure
00145 !-----------------------
00146
00147 ! Number of processes
00148     nprocs = nreac_in
00149
00150 ! Number of charge states
00151     nz    = zn_in+1
00152
00153     allocate(amns%process(nprocs))
00154 !    allocate(amns%process(nprocs))
00155     allocate(amns%coordinate_system(nprocs))
00156     do iproc=1, nprocs
00157        allocate(amns%process(iproc)%label(1))
00158        allocate(amns%process(iproc)%charge_state(nz))
00159 !       allocate(amns%process(iproc)%zmax(nz))
00160        do is=1, nz
00161           allocate(amns%process(iproc)%charge_state(is)%label(1))
00162        enddo
00163        allocate(amns%process(iproc)%result_label(1))
00164        allocate(amns%process(iproc)%result_units(1))
00165        allocate(amns%process(iproc)%source(1))
00166        allocate(amns%process(iproc)%provider(1))
00167        allocate(amns%process(iproc)%citation(1))
00168 !       allocate(amns%process(iproc)%table(nz))
00169     enddo
```

```
00170
00171 !-----------------------------
00172 ! Fill the structure with data
00173 !-----------------------------
00174
00175     do iproc=1, nprocs
00176        amns%process(iproc)%charge_state(1)%z_min    = 0
00177        amns%process(iproc)%charge_state(1)%z_max    = 0
00178        amns%process(iproc)%charge_state(1)%label(1) = trim(sp_loc) // '0'
00179        do is=1, nz-1
00180           amns%process(iproc)%charge_state(is+1)%z_min = is
00181           amns%process(iproc)%charge_state(is+1)%z_max = is
00182           if(is.lt.10) then
00183              write(amns%process(iproc)%charge_state(is+1)%label(1),'(a,i1,''+'')') trim(sp_loc), is
00184           else
00185              write(amns%process(iproc)%charge_state(is+1)%label(1),'(a,i2,''+'')') trim(sp_loc), is
00186           endif
00187        enddo
00188     enddo
00189
00190 !-----------------------------------
00191 ! Read data for the various processes
00192 !-----------------------------------
00193
00194     iunit = 67
00195
00196     do ireac = 1, nreac_in
00197
00198        read(lu_read, *) amns%process(ireac)%label(1), iclass, file, &
00199             amns%process(ireac)%result_label(1), amns%process(ireac)%result_units(1), &
00200             amns%process(ireac)%result_transformation, amns%process(ireac)%table_dimension
00201
00202        file = trim(root_in) // trim(file)
00203
00204        write(*,*) 'Opening ', trim(file)
00205        open(unit=iunit, file=trim(adjustl(file)), action='read', status='old', iostat=iostat)
00206
00207        call xxdata_11(iunit , iclass ,            &
00208             isdimd , iddimd , itdimd ,            &
00209             ndptnl , ndptn  , ndptnc , ndcnct ,   &
00210             iz0    , is1min , is1max ,            &
00211             nptnl  , nptn   , nptnc  ,            &
00212             iptnla , iptna  , iptnca ,            &
00213             ncnct  , icnctv ,                     &
00214             iblmx  , ismax  , dnr_ele, dnr_ams,   &
00215             isppr  , ispbr  , isstgr ,            &
00216             idmax  , itmax  ,                     &
00217             ddens  , dtev   , drcof ,             &
00218             lres   , lstan  , lptn                )
00219
00220        close(unit=iunit)
00221
00222        write(*,*) 'Closed ', trim(file)
00223
00224        amns%process(ireac)%source   = file    ! Add again later
00225        amns%process(ireac)%provider = 'David.Coster@ipp.mpg.de on behalf of AMNS Team'
00226        amns%process(ireac)%citation = 'http://www.adas.ac.uk/'
00227        amns%process(ireac)%coordinate_index  = handle_coordinates(ncoord, dtev(1:itmax),
      ddens(1:idmax))
00228
00229        select case (amns%process(ireac)%result_transformation)
00230        case (0)
00231           do iz=1, nz
00232              allocate(amns%process(ireac)%charge_state(iz)%table_2d(itmax,idmax))
00233              amns%process(ireac)%charge_state(iz)%table_2d = zero
00234           enddo
00235        case (1)
00236           do iz=1, nz
00237              allocate(amns%process(ireac)%charge_state(iz)%table_2d(itmax,idmax))
00238              amns%process(ireac)%charge_state(iz)%table_2d = log_zero
00239           enddo
00240        case default
00241           write(*,*) 'Case result_trans', amns%process(ireac)%result_transformation, ' not coded'
00242           stop 'Not coded'
00243        end select
00244
00245        select case (iclass)
00246        case (1, 3, 4)
00247           do iz = 1, nz-1
00248              amns%process(ireac)%charge_state(iz+1)%table_2d(:,:) = -6.0 + drcof(iz,1:itmax,1:idmax)
00249           end do
00250        case (2, 8)
00251           do iz = 1, nz-1
00252              amns%process(ireac)%charge_state(iz  )%table_2d(:,:) = -6.0 + drcof(iz,1:itmax,1:idmax)
00253           end do
00254        case (10, 11, 12)
00255           do iz = 1, nz
```

```
00256             amns%process(ireac)%charge_state(iz  )%table_2d(:,:) =         +
      drcof(iz,1:itmax,1:idmax)
00257          end do
00258       case default
00259          write(*,*) 'Case iclass ', iclass, ' not coded'
00260          stop 'Not Coded'
00261       end select
00262
00263
00264 ! Reactant and product are 4.10b extensions of the ITM, they are not present in IMAS yet
00265       select case (iclass)
00266       case (1) ! RC / acd
00267          call allocate_process(amns%process(ireac), 2, 2)
00268          call assign_reactantproduct(amns%process(ireac)%reactants(1), &
00269               trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00270          call assign_reactantproduct(amns%process(ireac)%reactants(2), &
00271               'e', 0, e_mass_in_amu, 2, 0, -1.0d+0)
00272          call assign_reactantproduct(amns%process(ireac)%products(1), &
00273               trim(sp_loc), zn_in, amn_in, 1, 1, -1.0d+0)
00274          call assign_reactantproduct(amns%process(ireac)%products(2), &
00275               'e', 0, e_mass_in_amu, 1, 0, -1.0d+0)
00276       case (2) ! EI / scd
00277          call allocate_process(amns%process(ireac), 2, 2)
00278          call assign_reactantproduct(amns%process(ireac)%reactants(1), &
00279               trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00280          call assign_reactantproduct(amns%process(ireac)%reactants(2), &
00281               'e', 0, e_mass_in_amu, 1, 0, -1.0d+0)
00282          call assign_reactantproduct(amns%process(ireac)%products(1), &
00283               trim(sp_loc), zn_in, amn_in, 1, 1, +1.0d+0)
00284          call assign_reactantproduct(amns%process(ireac)%products(2), &
00285               'e', 0, e_mass_in_amu, 2, 0, -1.0d+0)
00286       case (3) ! CX / ccd
00287          call allocate_process(amns%process(ireac), 2, 2)
00288          call assign_reactantproduct(amns%process(ireac)%reactants(1), &
00289               trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00290          call assign_reactantproduct(amns%process(ireac)%reactants(2), &
00291               'H|D|T', 1, d_mass_in_amu, 1, 0, 0.0d+0)
00292          call assign_reactantproduct(amns%process(ireac)%products(1), &
00293               trim(sp_loc), zn_in, amn_in, 1, 1, -1.0d+0)
00294          call assign_reactantproduct(amns%process(ireac)%products(2), &
00295               'H|D|T', 1, d_mass_in_amu, 1, 0, 1.0d+0)
00296       case (4, 8, 10, 11, 12) ! BR/prb, LR/plt, ZE/zcd, ZE2/ycd, EIP/ecd
00297          call allocate_process(amns%process(ireac), 1, 1)
00298          call assign_reactantproduct(amns%process(ireac)%reactants(1), &
00299               trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00300          call assign_reactantproduct(amns%process(ireac)%products(1), &
00301               trim(sp_loc), zn_in, amn_in, 1, 1, 0.0d+0)
00302       end select
00303
00304    end do
00305    write(*,*) 'Number of different coordinates = ', ncoord
00306
00307    return
00308
00309  contains
00310
00311    integer function handle_coordinates(ncoord, dtev, ddens)
00312
00313 ! amns%process%id(1, jproc) gives the dimension of the data table,
00314 ! i.e. the number of dependent variables (in the example shown here
00315 ! the variables are always density and temperature and the table is
00316 ! therefore 2d).
00317 ! amns%process%id(2, jproc) gives the position in the table, i.e. the last
00318 ! argument in the table. For instance for a 2D table, and process jproc the
00319 ! data are given by amns%process%tables_2d%table(:,;,jproc, ipos) where
00320 ! ipos =  amns%process%id(2, jproc).
00321 ! Dependent variables (electron density and temperature in this case)
00322
00323       implicit none
00324
00325       real (ids_real), dimension(:):: ddens
00326       real (ids_real), dimension(:):: dtev
00327       integer ncoord, icoord, itmax, idmax
00328
00329       itmax=size(dtev)
00330       idmax=size(ddens)
00331 ! check whether we already have the coordinate
00332       do icoord=1, ncoord
00333          if(itmax.ne.size(amns%coordinate_system(icoord)%coordinate(1)%values)) cycle
00334          if(idmax.ne.size(amns%coordinate_system(icoord)%coordinate(2)%values)) cycle
00335          write(*,*) maxval(abs(dtev-amns%coordinate_system(icoord)%coordinate(1)%values))
00336          if(maxval(abs(dtev-amns%coordinate_system(icoord)%coordinate(1)%values)).gt.0.0) cycle
00337          write(*,*) maxval(abs(6.0 + ddens-amns%coordinate_system(icoord)%coordinate(2)%values))
00338          if(maxval(abs(6.0 + ddens-amns%coordinate_system(icoord)%coordinate(2)%values)).gt.0.0) cycle
00339 ! we have a match
00340          handle_coordinates=icoord
00341          return
```

```
00342       enddo
00343 ! we have no match
00344       ncoord=ncoord+1
00345       allocate(amns%coordinate_system(ncoord)%coordinate(2))
00346       do idim=1, 2
00347          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%extrapolation_type(2))
00348          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%label(1))
00349          allocate(amns%coordinate_system(ncoord)%coordinate(idim)%units(1))
00350       enddo
00351       allocate(amns%coordinate_system(ncoord)%coordinate(1)%values(itmax))
00352       allocate(amns%coordinate_system(ncoord)%coordinate(2)%values(idmax))
00353
00354       amns%coordinate_system(ncoord)%coordinate(1)%label(1) = 'Electron Temperature'
00355       amns%coordinate_system(ncoord)%coordinate(1)%extrapolation_type = (/ 2, 2 /)
00356       amns%coordinate_system(ncoord)%coordinate(1)%interpolation_type = 1
00357       amns%coordinate_system(ncoord)%coordinate(1)%units = 'eV'
00358       amns%coordinate_system(ncoord)%coordinate(1)%transformation = 1
00359       amns%coordinate_system(ncoord)%coordinate(1)%spacing = 0
00360       amns%coordinate_system(ncoord)%coordinate(1)%values = dtev(1:itmax)
00361
00362       amns%coordinate_system(ncoord)%coordinate(2)%label(1) = 'Electron Density'
00363       amns%coordinate_system(ncoord)%coordinate(2)%extrapolation_type = (/ 2, 2 /)
00364       amns%coordinate_system(ncoord)%coordinate(2)%interpolation_type = 1
00365       amns%coordinate_system(ncoord)%coordinate(2)%units = 'm^{-3}'
00366       amns%coordinate_system(ncoord)%coordinate(2)%transformation = 1
00367       amns%coordinate_system(ncoord)%coordinate(2)%spacing = 0
00368       amns%coordinate_system(ncoord)%coordinate(2)%values = 6.0 + ddens(1:idmax)
00369
00370       handle_coordinates=ncoord
00371       return
00372
00373     end function handle_coordinates
00374
00375     function upcase(string) result(upper)
00376       character(len=*), intent(in) :: string
00377       character(len=len(string)) :: upper
00378       integer :: j
00379       do j = 1,len(string)
00380          if(string(j:j) >= "a" .and. string(j:j) <= "z") then
00381             upper(j:j) = achar(iachar(string(j:j)) - 32)
00382          else
00383             upper(j:j) = string(j:j)
00384          end if
00385       end do
00386
00387     end function upcase
00388
00389 ! 4.10b extension, not in IMAS yet
00390     subroutine allocate_process(process, nr, np)
00391       use ids_schemas   ! IGNORE
00392       implicit none
00393       type (ids_amns_data_process) :: process
00394       integer :: nr, np
00395       integer :: irp
00396
00397       allocate(process%reactants(nr), process%products(np))
00398       do irp = 1, size(process%reactants)
00399          allocate(process%reactants(irp)%label(1))
00400          allocate(process%reactants(irp)%element(1))
00401 !         allocate(process%reactants(irp)%element(1)%label(1))
00402       enddo
00403       do irp = 1, size(process%products)
00404          allocate(process%products(irp)%label(1))
00405          allocate(process%products(irp)%element(1))
00406 !         allocate(process%products(irp)%element(1)%label(1))
00407       enddo
00408
00409     end subroutine allocate_process
00410
00411     subroutine assign_reactantproduct(reactantproduct, label, zn, amn, multiplicity, relative, za)
00412       use ids_schemas   ! IGNORE
00413       implicit none
00414       type (ids_amns_data_process_reactant) :: reactantproduct
00415       character(len=*) :: label
00416       integer :: zn, multiplicity, relative
00417       real (ids_real) :: amn, za
00418
00419       reactantproduct%label(1)=label
00420       reactantproduct%multiplicity=multiplicity
00421       reactantproduct%element(1)%atoms_n=multiplicity
00422 !      reactantproduct%element(1)%label(1)=label
00423       reactantproduct%element(1)%z_n=zn
00424       reactantproduct%element(1)%a=nint(amn)
00425       reactantproduct%mass=amn
00426       reactantproduct%relative_charge=relative
00427       reactantproduct%charge=za
00428
```

```
00429     end subroutine assign_reactantproduct
00430
00431   end subroutine adas_amns
00432
00433 end module amns_adas
```

16.42 src/amns_driver/amns_bms.f90 File Reference

Modules

- module amns_bms

Functions/Subroutines

- subroutine amns_bms::bms_amns (amns, zn, am)
- subroutine amns_bms::allocate_process (process, nr, np)
- subroutine amns_bms::assign_reactantproduct (reactantproduct, label, zn, amn, multiplicity, relative, za)

Variables

- integer, parameter amns_bms::r8 = SELECTED_REAL_KIND (15, 300)

16.43 amns_bms.f90

```
00001 module amns_bms
00002
00003   INTEGER,  PARAMETER :: r8 = selected_real_kind (15, 300) ! Real*8
00004
00005 contains
00006
00007   subroutine bms_amns(amns, zn, am)
00008     use ids_schemas   ! IGNORE
00009     use codata, only: h_mass_in_amu => proton_mass_in_u
00010     use bms
00011
00012     implicit none
00013
00014     type (ids_amns_data) :: amns
00015     integer :: zn, am
00016     integer :: nprocs, iproc
00017     integer :: ncoords, icoord
00018     integer :: npoints(4)
00019     real, allocatable :: y(:,:,:,:), x1(:), x2(:), x3(:), x4(:)
00020     integer :: i
00021
00022     nprocs = 1
00023     ncoords = 1
00024     allocate(amns%coordinate_system(ncoords))
00025     allocate(amns%process(nprocs))
00026     iproc=1
00027     allocate(amns%process(iproc)%label(1))
00028     amns%process(iproc)%label = 'BMS'
00029     call allocate_process(amns%process(iproc), 2, 2)
00030     call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00031          'H', 1, h_mass_in_amu, 1,  0, 0.0_r8)
00032     call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00033          'H', 1, h_mass_in_amu, 1,  1, 0.0_r8)
00034     call assign_reactantproduct(amns%process(iproc)%products(1), &
00035          'H', 1, h_mass_in_amu, 1,  1, 0.0_r8)
00036     call assign_reactantproduct(amns%process(iproc)%products(2), &
00037          'H', 1, h_mass_in_amu, 1,  0, 0.0_r8)
00038     allocate(amns%process(iproc)%result_label(1))
00039     amns%process(iproc)%result_label = 'BMS'
00040     allocate(amns%process(iproc)%result_units(1))
00041     amns%process(iproc)%result_units = '??'
00042     allocate(amns%process(iproc)%charge_state(1))
00043
00044     allocate(amns%process(iproc)%source(1))
00045     amns%process(iproc)%source = 'amns_driver/bms.f90'
00046     allocate(amns%process(iproc)%provider(1))
00047     amns%process(iproc)%provider = 'David.Coster@ipp.mpg.de on behalf of AMNS Team'
00048     allocate(amns%process(iproc)%citation(1))
00049     amns%process(iproc)%citation = 'ToDo'
00050     amns%process(iproc)%label(1)='BMS'
00051
```

```
00052     call read_bms('../../test-data/bms#h_h1_value.dat', y, x1, x2, x3, x4)
00053     write(*,*) 'x1: ', x1
00054     write(*,*) 'x2: ', x2
00055     write(*,*) 'x3: ', x3
00056     write(*,*) 'x4: ', x4
00057     write(*,*) minval(y), maxval(y)
00058     npoints=(/ size(x1), size(x2), size(x3), size(x4) /)
00059
00060     allocate(amns%process(iproc)%charge_state(1)%table_4d(size(x1), size(x2), size(x3), size(x4)))
00061     amns%process(iproc)%charge_state(1)%table_4d = y
00062     amns%process(iproc)%table_dimension = 4
00063     amns%process(iproc)%result_transformation = 0
00064
00065     icoord = 1
00066     amns%process(iproc)%coordinate_index = icoord
00067     allocate(amns%coordinate_system(icoord)%coordinate(4))
00068     do i=1,size(amns%coordinate_system(icoord)%coordinate)
00069        allocate(amns%coordinate_system(icoord)%coordinate(i)%values(npoints(i)),&
00070             &   amns%coordinate_system(icoord)%coordinate(i)%label(1),&
00071             &   amns%coordinate_system(icoord)%coordinate(i)%units(1),&
00072             &   amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(2))
00073        amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(1:2)=1 ! 0= No extrapolation
00074        amns%coordinate_system(icoord)%coordinate(i)%interpolation_type=1      ! 1= Linear
      interpolation
00075        amns%coordinate_system(icoord)%coordinate(i)%transformation=0
00076        amns%coordinate_system(icoord)%coordinate(i)%spacing=0
00077     end do
00078     amns%coordinate_system(icoord)%coordinate(1)%values = x1
00079     amns%coordinate_system(icoord)%coordinate(1)%label = 'NENG'
00080     amns%coordinate_system(icoord)%coordinate(1)%units = 'eV'
00081     amns%coordinate_system(icoord)%coordinate(2)%values = x2
00082     amns%coordinate_system(icoord)%coordinate(2)%label = 'NDENS'
00083     amns%coordinate_system(icoord)%coordinate(2)%units = 'm^{-3}'
00084     amns%coordinate_system(icoord)%coordinate(3)%values = x3
00085     amns%coordinate_system(icoord)%coordinate(3)%label = 'NTION'
00086     amns%coordinate_system(icoord)%coordinate(3)%units = 'eV'
00087     amns%coordinate_system(icoord)%coordinate(4)%values = x4
00088     amns%coordinate_system(icoord)%coordinate(4)%label = 'NTE'
00089     amns%coordinate_system(icoord)%coordinate(4)%units = 'eV'
00090
00091 !! test DPC   try interpolating the log10 of the data with log10 of the coordinates ...
00092     amns%process(iproc)%result_transformation = 1    ! interpolating using log10
00093     amns%process(iproc)%charge_state(1)%table_4d = log10(amns%process(iproc)%charge_state(1)%table_4d)
00094     do i=1,size(amns%coordinate_system(icoord)%coordinate)
00095        amns%coordinate_system(icoord)%coordinate(i)%transformation=1          ! interpolating using
      log10
00096        amns%coordinate_system(icoord)%coordinate(i)%values =
      log10(amns%coordinate_system(icoord)%coordinate(i)%values)
00097     enddo
00098 !! end of test
00099
00100   end subroutine bms_amns
00101
00102   subroutine allocate_process(process, nr, np)
00103     use ids_schemas   ! IGNORE
00104     implicit none
00105     type (ids_amns_data_process) :: process
00106     integer :: nr, np
00107     integer :: irp
00108
00109     allocate(process%reactants(nr), process%products(np))
00110     do irp = 1, size(process%reactants)
00111        allocate(process%reactants(irp)%label(1))
00112        allocate(process%reactants(irp)%element(1))
00113        !         allocate(process%reactants(irp)%element(1)%label(1))
00114     enddo
00115     do irp = 1, size(process%products)
00116        allocate(process%products(irp)%label(1))
00117        allocate(process%products(irp)%element(1))
00118        !         allocate(process%products(irp)%element(1)%label(1))
00119     enddo
00120
00121   end subroutine allocate_process
00122
00123   subroutine assign_reactantproduct(reactantproduct, label, zn, amn, multiplicity, relative, za)
00124     use ids_schemas   ! IGNORE
00125     implicit none
00126     type (ids_amns_data_process_reactant) :: reactantproduct
00127     character(len=*) :: label
00128     integer :: zn, multiplicity, relative
00129     real (ids_real) :: amn, za
00130
00131     reactantproduct%label(1)=label
00132     reactantproduct%multiplicity=multiplicity
00133     reactantproduct%element(1)%atoms_n=multiplicity
00134     !      reactantproduct%element(1)%label(1)=label
00135     reactantproduct%element(1)%z_n=zn
```

```
00136     reactantproduct%element(1)%a=nint(amn)
00137     reactantproduct%mass=amn
00138     reactantproduct%relative_charge=relative
00139     reactantproduct%charge=za
00140
00141   end subroutine assign_reactantproduct
00142
00143 end module amns_bms
```

16.44 src/amns_driver/amns_driver.f90 File Reference

Data Types

- type type_list_data_release

Functions/Subroutines

- program amns_driver

  driver program to produce the AMNS database

16.44.1 Function/Subroutine Documentation

16.44.1.1 amns_driver()

`program amns_driver`
driver program to produce the AMNS database
This sets data enties for the species specified in amns_driver.data.

Author

Coster

Definition at line 7 of file amns_driver.f90.

16.45 amns_driver.f90

```
00001
00006
00007 program amns_driver
00008
00009   use ids_schemas   ! IGNORE
00010   use ids_routines  ! IGNORE
00011   use amns_provider_types
00012   use amns_adas
00013 !  use amns_surface
00014   use amns_nuclear
00015   use amns_bms
00016 !  use amns_elastic
00017 !  use amns_rct
00018 !  use copy_structures          ! IGNORE
00019 !  use write_structures         ! IGNORE
00020 !  use read_structures          ! IGNORE
00021 !  use deallocate_structures    ! IGNORE
00022 !  use itm_elements
00023 !  use amns_version
00024
00025   implicit none
00026
00027 !NLL  integer :: imas_open_env, imas_open_hdf5, imas_create_env, imas_create_hdf5
00028   integer :: ids_status
00029
00030   type (ids_amns_data)     :: amns00, amns
00031   type (amns_ids_list), pointer :: first => null()
00032   type (amns_ids_list), pointer :: last => null(), work => null()
00033
00034   integer                         :: idx               !index (internal)
00035   integer                         :: shot, refshot     !shot number
00036   integer                         :: run,  refrun      !run number
00037   integer                         :: nprocs, ncoord, nz
00038   integer                         :: iprocs, icoord
```

```
00039   integer                          :: jprocs, jcoord
00040   integer                          :: sprocs, scoord
00041   character(len=5)                 :: treename
00042   character(len=11)                :: amnspath
00043
00044 ! program variables
00045
00046   character (len=32)               :: ds_version='3'    ! default: overriden by environment variable
       IDS_VERSION
00047   integer                          :: backend_id
00048   character (len=32)               :: backend='mdsplus', backend_options=''
00049   character (len=256)              :: user
00050   character (len=256)              :: root_in
00051   character (len=256)              :: file, filename
00052   character (len=256)              :: label
00053   character (len=256)              :: amns_description, amns_environment
00054   character (len=256)              :: imas_version, al_version
00055   integer                          :: zn_in, spec_zn, spec_am
00056   integer                          :: nreac_in
00057   character (len=10)               :: sp_in
00058   real(ids_real)                   :: amn_in
00059   logical                          :: dirty
00060   character (len=256)              :: arg
00061   integer                          :: iargc
00062   character (STRMAXLEN)            :: uri
00063
00064   type (ids_amns_data_release),pointer :: release(:) => null()
00065   integer :: nversion, iversion
00066   character*8 date
00067   character*10 time
00068   character*5 zone
00069   character*64 date_time_zone
00070   type type_list_data_release
00071      type (ids_amns_data_data_entry), pointer :: data_entry => null()
00072      type (type_list_data_release), pointer :: next => null()
00073   end type type_list_data_release
00074   type (type_list_data_release), pointer :: list_data_release_head => null(), list_data_release_next
      => null()
00075   integer :: nrelease, irelease, new_data_release_count, icount
00076   logical :: exist
00077
00078   integer :: n1, n2, i1
00079   real (ids_real) :: s1, a1
00080
00081    if(iargc().ge.1) then
00082      call getarg(1,arg)
00083      if(arg.eq.'mdsplus') then
00084         backend='mdsplus'
00085         backend_id=mdsplus_backend
00086         backend_options=''
00087      elseif(arg.eq.'hdf5') then
00088         backend='hdf5'
00089         backend_id=hdf5_backend
00090         backend_options=''
00091      elseif(arg.eq.'ascii') then
00092         backend='ascii'
00093         backend_id=ascii_backend
00094         backend_options='-prefix IDS/'
00095      else
00096         write(0,*) 'Argument not recognized: ', trim(arg)
00097      endif
00098   else   ! default to MDSplus
00099      backend='mdsplus'
00100      backend_id=mdsplus_backend
00101      backend_options=''
00102   endif
00103
00104 ! Set source and version  information
00105
00106   treename    = 'ids'
00107   refshot     = 0
00108   refrun      = 0
00109   amnspath    = 'amns_data'     !IDS name/occurence ! Caution, declare it with the right size !
00110   call getenv('USER', user)
00111   call getenv('IMAS_VERSION', imas_version)
00112   call getenv('AL_VERSION', al_version)
00113   write(*,*) 'Using IMAS_VERSION ', trim(imas_version), ' and AL_VERSION ', trim(al_version)
00114   amns_description = 'AMNS data created by version ' // trim(git_version_amns) // ' of the amns_driver
      system'
00115   write(*,*) 'AMNS Description ', trim(amns_description)
00116   amns_environment = 'IMAS_VERSION = ' // trim(imas_version) // '   AL_VERSION = ' // trim(al_version)
00117   write(*,*) 'AMNS Environment ', trim(amns_environment)
00118
00119 ! new
00120
00121   shot = 0
00122   run = 1
```

```
00123   nversion = 0
00124   !call ual_begin_pulse_action(backend_id, shot, run, &
00125   !     trim(USER), 'amns', trim(ds_version), idx)
00126   !call ual_open_pulse(idx, OPEN_PULSE, backend_options, ids_status)
00127   call al_build_uri_from_legacy_parameters(backend_id, shot, run, &
00128        trim(user), 'amns', trim(ds_version), backend_options, uri, ids_status)
00129   call al_begin_dataentry_action(uri, open_pulse, idx, ids_status)
00130   write(*,*) 'IDX = ', idx
00131   write(*,*) 'ids_STATUS = ', ids_status
00132   if(ids_status.eq.0) then
00133      call ids_get(idx, amnspath, amns00, ids_status)
00134      write(*,*) 'ids_STATUS = ', ids_status
00135      call imas_close(idx)
00136      if(ids_status.eq.0) nversion = -1
00137   endif
00138
00139   if(nversion.eq.0) then  ! this is the case where there was no amns/0/1
00140      nversion=1
00141      allocate(amns00%time(1))
00142      amns00%time = 0.0
00143      amns00%ids_properties%homogeneous_time=1
00144      allocate(amns00%ids_properties%source(1))
00145      amns00%ids_properties%source(1)=amns_description
00146      allocate(amns00%ids_properties%comment(1))
00147      amns00%ids_properties%comment(1)=amns_environment
00148      allocate(amns00%ids_properties%provider(1))
00149      amns00%ids_properties%provider(1)=trim(user) // ' on behalf of contributors to the AMNS task of
       EFDA TF-ITM and EUROfusion WPCD'
00150      allocate(amns00%ids_properties%creation_date(1))
00151      amns00%ids_properties%creation_date(1)=date_time_zone
00152      allocate(amns00%code%name(1))
00153      amns00%code%name(1)='amns_driver'
00154      allocate(amns00%code%commit(1))
00155      amns00%code%commit(1)=trim(git_version_amns)
00156      allocate(amns00%code%version(1))
00157      amns00%code%version(1)=trim(git_version_amns)
00158      allocate(amns00%code%repository(1))
00159      amns00%code%repository(1)='ssh://git@git.iter.org/imex/amns.git'
00160   else  ! this is the case where there was an amns/0/1
00161      if(associated(amns00%release)) then
00162         nversion=size(amns00%release)+1
00163      else
00164         nversion=1
00165      endif
00166   endif
00167   allocate(release(nversion))
00168   do iversion=1, nversion-1
00169      call ids_copy(amns00%release(iversion),release(iversion))
00170   enddo
00171   write(*,*) 'nversion = ', nversion
00172   allocate(release(nversion)%description(1))
00173   release(nversion)%description(1)=amns_description
00174   write(*,*) trim(release(nversion)%description(1))
00175   allocate(release(nversion)%date(1))
00176   call date_and_time(date,time,zone)
00177   date_time_zone=  &
00178        date(1:4)//'-'//date(5:6)//'-'//date(7:8)//' '//  &
00179        time(1:2)//':'//time(3:4)//':'//time(5:10)//' '//zone
00180   release(nversion)%date(1)=date_time_zone
00181   write(*,*) trim(release(nversion)%date(1))
00182
00183   allocate (amns00%time(1))
00184   amns00%time = 0.0
00185   amns00%ids_properties%homogeneous_time=1
00186   allocate(amns00%ids_properties%source(1))
00187   amns00%ids_properties%source(1)=amns_description
00188   allocate(amns00%ids_properties%comment(1))
00189   amns00%ids_properties%comment(1)=amns_environment
00190   allocate(amns00%ids_properties%provider(1))
00191   amns00%ids_properties%provider(1)=trim(user) // ' on behalf of contributors to the AMNS task of EFDA
       TF-ITM and EUROfusion WPCD'
00192   allocate(amns00%ids_properties%creation_date(1))
00193   amns00%ids_properties%creation_date(1)=date_time_zone
00194   allocate(amns00%code%name(1))
00195   amns00%code%name(1)='amns_driver'
00196   allocate(amns00%code%commit(1))
00197   amns00%code%commit(1)=trim(git_version_amns)
00198   allocate(amns00%code%version(1))
00199   amns00%code%version(1)=trim(git_version_amns)
00200   allocate(amns00%code%repository(1))
00201   amns00%code%repository(1)='ssh://git@git.iter.org/imex/amns.git'
00202 !  allocate (amns00%version(1))
00203 !  amns00%version = version
00204 !  allocate (amns00%source(1))
00205 !  amns00%source  = 'Contributors to the AMNS task of the EFDA Task Force on Integerated Tokamak
       Modelling'
00206
```

```
00207 ! DRIVER
00208   open(51,file='amns_driver.data')
00209 !   read(51,*) svn_version
00210 !   write(*,*) 'SVN version of the driver file "amns_driver.data" is ',trim(svn_version)
00211   read(51,*) root_in      ! This line is the path to the original ADAS data
00212 10 continue
00213      read(51, *, end=20) label
00214      label = adjustl(label)
00215      select case (label)
00216
00217      case ("SPECIES")
00218         read(51, *, err=30) spec_zn, spec_am
00219         write(*,*) 'Processing ZN = ',spec_zn,' AM = ', spec_am
00220
00221 ! create the head of the list
00222         nprocs = 0
00223         ncoord = 0
00224         allocate(first)
00225         work => first
00226         last => first
00227         dirty = .false.
00228
00229         goto 10
00230
00231      case ("ADF11")
00232
00233         read(51, *, err=30) zn_in, amn_in, nreac_in, sp_in
00234         if(zn_in .ne. spec_zn) then
00235            write(0,*) 'mismatched block found: SPECIES/ADAS mismatch ',spec_zn, zn_in
00236            stop 1
00237         endif
00238         write(*,*) 'Process ', zn_in, amn_in, nreac_in, sp_in
00239 !        write(*,*) 'Element = ', element(zn_in, nint(amn_in))
00240
00241 ! call the adas subroutine
00242         if(dirty) then
00243            last => work
00244            allocate(last%next)
00245            work => last%next
00246            work%prev => last
00247            dirty = .false.
00248         endif
00249         call adas_amns(work%amns_ids, root_in, zn_in, amn_in, nreac_in, sp_in, 51)
00250         nprocs = nprocs + size(work%amns_ids%process)
00251         ncoord = ncoord + size(work%amns_ids%coordinate_system)
00252         dirty = .true.
00253
00254         write(*,*) 'Finished reading ADF11 type file'
00255
00256         goto 10
00257
00258      case ("SURFACESPUTTER")
00259         read(51,*) filename
00260 !         if(dirty) then
00261 !            last => work
00262 !            allocate(last%next)
00263 !            work => last%next
00264 !            work%prev => last
00265 !            dirty = .false.
00266 !         endif
00267 !         call surface_sputter_amns(work%amns_ids, filename)
00268 !         nprocs = nprocs + size(work%amns_ids%process)
00269 !         ncoord = ncoord + size(work%amns_ids%coordinate_system)
00270 !         dirty = .true.
00271
00272         goto 10
00273
00274      case ("SURFACEREFLECTION")
00275         read(51,*) filename
00276 !         if(dirty) then
00277 !            last => work
00278 !            allocate(last%next)
00279 !            work => last%next
00280 !            work%prev => last
00281 !            dirty = .false.
00282 !         endif
00283 !         call surface_partrefl_amns(work%amns_ids, filename)
00284 !         nprocs = nprocs + size(work%amns_ids%process)
00285 !         ncoord = ncoord + size(work%amns_ids%coordinate_system)
00286 !         dirty = .true.
00287 !
00288         goto 10
00289
00290 !     case ("ELASTIC_TOTAL")
00291 !        read(51,*) filename, n1
00292 !        if(dirty) then
00293 !           last => work
```

```
00294 !           allocate(last%next)
00295 !           work => last%next
00296 !           work%prev => last
00297 !           dirty = .false.
00298 !         endif
00299 !         call elastic_total(work%amns_ids, filename, n1)
00300 !         nprocs = nprocs + size(work%amns_ids%process)
00301 !         ncoord = ncoord + size(work%amns_ids%coordinate_system)
00302 !         dirty = .true.
00303 !
00304 !         goto 10
00305
00306 !      case ("ELASTIC_DIFFERENTIAL")
00307 !         read(51,*) filename, n1, n2
00308 !         if(dirty) then
00309 !            last => work
00310 !            allocate(last%next)
00311 !            work => last%next
00312 !            work%prev => last
00313 !            dirty = .false.
00314 !         endif
00315 !         call elastic_differential(work%amns_ids, filename, n1, n2)
00316 !         nprocs = nprocs + size(work%amns_ids%process)
00317 !         ncoord = ncoord + size(work%amns_ids%coordinate_system)
00318 !         dirty = .true.
00319
00320 !         goto 10
00321
00322       case ("NUCLEAR")
00323          if(dirty) then
00324             last => work
00325             allocate(last%next)
00326             work => last%next
00327             work%prev => last
00328             dirty = .false.
00329          endif
00330          call nuclear_amns(work%amns_ids, spec_zn, spec_am)
00331          nprocs = nprocs + size(work%amns_ids%process)
00332          ncoord = ncoord + size(work%amns_ids%coordinate_system)
00333          dirty = .true.
00334
00335          goto 10
00336
00337 !      case ("RCT")  ! resonant charge transfer, non H/D/T
00338 !         read(51,*) s1, a1
00339 !         if(dirty) then
00340 !            last => work
00341 !            allocate(last%next)
00342 !            work => last%next
00343 !            work%prev => last
00344 !            dirty = .false.
00345 !         endif
00346 !         call rct_amns(work%amns_ids, s1, a1)
00347 !         nprocs = nprocs + size(work%amns_ids%process)
00348 !         ncoord = ncoord + size(work%amns_ids%coordinate_system)
00349 !         dirty = .true.
00350
00351 !         goto 10
00352
00353       case ("BMS")
00354          if(dirty) then
00355             last => work
00356             allocate(last%next)
00357             work => last%next
00358             work%prev => last
00359             dirty = .false.
00360          endif
00361          call bms_amns(work%amns_ids, spec_zn, spec_am)
00362          nprocs = nprocs + size(work%amns_ids%process)
00363          ncoord = ncoord + size(work%amns_ids%coordinate_system)
00364          dirty = .true.
00365
00366          goto 10
00367
00368       case ("END")
00369 ! now combine the data from the pieces
00370          allocate (amns%time(1))
00371          amns%time=0.0
00372          amns%ids_properties%homogeneous_time = 1
00373          amns%ids_properties%homogeneous_time=1
00374          allocate(amns%ids_properties%source(1))
00375          amns%ids_properties%source(1)=amns_description ! 'Contributors to the AMNS task of EFDA TF-ITM
       and EUROfusion WPCD'
00376          allocate(amns%ids_properties%comment(1))
00377          amns%ids_properties%comment(1)=amns_environment
00378          allocate(amns%ids_properties%provider(1))
00379          amns%ids_properties%provider(1)=trim(user) // ' on behalf of contributors to the AMNS task of
```

```
      EFDA TF-ITM and EUROfusion WPCD'
00380         allocate(amns%ids_properties%creation_date(1))
00381         amns%ids_properties%creation_date(1)=date_time_zone
00382         allocate(amns%code%name(1))
00383         amns%code%name(1)='amns_driver'
00384         allocate(amns%code%commit(1))
00385         amns%code%commit(1)=trim(git_version_amns)
00386         allocate(amns%code%version(1))
00387         amns%code%version(1)=trim(git_version_amns)
00388         allocate(amns%code%repository(1))
00389         amns%code%repository(1)='ssh://git@git.iter.org/imex/amns.git'
00390  !       allocate (amns%version(1))
00391  !       amns%version = version
00392  !       allocate (amns%source(1))
00393  !       amns%source  = 'Contributors to the AMNS task of the EFDA Task Force on Integerated Tokamak
      Modelling'
00394         sprocs=0
00395         scoord=0
00396         if(nprocs.gt.0) allocate(amns%process(nprocs))
00397         !if(nprocs.gt.0) allocate(amns%tables(nprocs))
00398         if(ncoord.gt.0) allocate(amns%coordinate_system(ncoord))
00399         work => first
00400  ! copy the data from the adas block (here we assume that adas data exists)
00401         amns%z_n = work%amns_ids%z_n
00402         amns%a = work%amns_ids%a
00403  ! other
00404         do while(associated(work))
00405  ! process information
00406            if(associated(work%amns_ids%process)) then
00407               jprocs = size(work%amns_ids%process)
00408               do iprocs=1, jprocs
00409                  call ids_copy(work%amns_ids%process(iprocs), amns%process(iprocs+sprocs))
00410               enddo
00411            else
00412               jprocs=0
00413            endif
00414  ! now the tables
00415            if(associated(work%amns_ids%process)) then
00416               do iprocs=1, jprocs
00417                  call ids_copy(work%amns_ids%process(iprocs), amns%process(iprocs+sprocs))
00418                  if(amns%process(iprocs+sprocs)%coordinate_index.gt.0) then
00419                     amns%process(iprocs+sprocs)%coordinate_index =
      amns%process(iprocs+sprocs)%coordinate_index + scoord
00420                  endif
00421               enddo
00422            endif
00423  ! and the coordinates
00424            if(associated(work%amns_ids%coordinate_system)) then
00425               jcoord = size(work%amns_ids%coordinate_system)
00426               do icoord=1, jcoord
00427                  call ids_copy(work%amns_ids%coordinate_system(icoord),
      amns%coordinate_system(icoord+scoord))
00428               enddo
00429            else
00430               jcoord=0
00431            endif
00432            sprocs = sprocs + jprocs
00433            scoord = scoord + jcoord
00434            work => work%next
00435         enddo
00436
00437  !find local version number for shot
00438  ! pseudo AMNS DB, to be done later ....
00439         shot = spec_zn + 1000*spec_am
00440         run = 1
00441         if(.not.associated(list_data_release_head)) then
00442            allocate(list_data_release_head)
00443            list_data_release_next => list_data_release_head
00444            new_data_release_count = 1
00445         else
00446            allocate(list_data_release_next%next)
00447            list_data_release_next => list_data_release_next%next
00448            new_data_release_count = new_data_release_count + 1
00449         endif
00450         allocate(list_data_release_next%data_entry)
00451         list_data_release_next%data_entry%shot = shot
00452         run = 0
00453         do iversion=1, nversion-1   ! loop over versions
00454            if(associated(amns00%release(iversion)%data_entry)) then
00455               nrelease=size(amns00%release(iversion)%data_entry)
00456               do irelease=1, nrelease   ! loop over data in a version
00457                  if(amns00%release(iversion)%data_entry(irelease)%shot .eq. shot) then
00458                     run=max(run, amns00%release(iversion)%data_entry(irelease)%run)
00459                  endif
00460               enddo
00461            endif
00462         enddo
```

```
00463         run = run + 1
00464         list_data_release_next%data_entry%run = run
00465         allocate(list_data_release_next%data_entry%description(1))
00466         list_data_release_next%data_entry%description=amns_description
00467
00468         write(*,'(a,a,a,i0,1x,i0,1x,i0,1x,i0,1x,a,a,a)') &
00469              trim(backend), ' backend ', treename, &
00470              shot, run, refshot, refrun, &
00471              trim(user), ' amns ', trim(ds_version)
00472         !call ual_begin_pulse_action(backend_id, shot, run, &
00473         !     trim(USER), 'amns', trim(ds_version), idx)
00474         !call ual_open_pulse(idx, FORCE_CREATE_PULSE, backend_options, ids_status)
00475         call al_build_uri_from_legacy_parameters(backend_id, shot, run, &
00476              trim(user), 'amns', trim(ds_version), backend_options, uri, ids_status)
00477         call al_begin_dataentry_action(uri, force_create_pulse, idx, ids_status)
00478         write(*,*) 'IDX = ', idx
00479         write(*,*) 'ids_STATUS = ', ids_status
00480         if(ids_status.ne.0) then
00481            write(0,*) 'Failure opening IDS!'
00482            stop 1
00483         endif
00484
00485         write(*,*) 'Calling ids_put'
00486         call ids_put(idx, amnspath, amns, ids_status)
00487         write(*,*) 'ids_STATUS = ', ids_status
00488         if(ids_status.ne.0) then
00489            write(0,*) 'Failure writing IDS!'
00490            stop 1
00491         endif
00492         write(*,*) 'ids_put done'
00493         call imas_close  (idx)
00494 !
00495         call ids_deallocate(amns)
00496         work => first
00497         do while(associated(work))
00498            last => work
00499            work => last%next
00500            deallocate(last)
00501         enddo
00502         nullify(first)
00503
00504         goto 10
00505
00506      case ("EOF")
00507         goto 20
00508
00509      case default
00510         write(0,*) 'Label ',trim(label),' not yet coded'
00511         stop 1
00512
00513      end select
00514
00515 20 continue
00516   write(*,*) 'Completed amns_data IDS writing, program successfully completed'
00517   close(51)
00518
00519   if(nversion.gt.1) then
00520 !!!DPC-TMP     call ids_deallocate(amns00%release)
00521   endif
00522   allocate(amns00%release(nversion))
00523   do iversion=1, nversion
00524      call ids_copy(release(iversion),amns00%release(iversion))
00525   enddo
00526   allocate(amns00%release(nversion)%data_entry(new_data_release_count))
00527   list_data_release_next => list_data_release_head
00528   icount = 0
00529   do while(associated(list_data_release_next))
00530      icount=icount+1
00531      call ids_copy(list_data_release_next%data_entry,amns00%release(nversion)%data_entry(icount))
00532 !NLL     call ids_deallocate(list_data_release_next%data_entry)
00533      call ids_deallocate_struct(list_data_release_next%data_entry, .false.)
00534      list_data_release_next => list_data_release_next%next
00535      deallocate(list_data_release_head)
00536      list_data_release_head => list_data_release_next
00537   enddo
00538   if(icount.ne.new_data_release_count) then
00539      write(0,*) 'length mismatch ', new_data_release_count, icount
00540      stop 1
00541   endif
00542
00543   write(*,*) 'Finishing ... update index'
00544
00545   shot = 0
00546   run = 1
00547   write(*,'(a,a,a,i0,1x,i0,1x,i0,1x,i0,1x,a,a,a)') &
00548        trim(backend), ' backend ', treename, &
00549        shot, run, refshot, refrun, &
```

```
00550         trim(user), ' amns ', trim(ds_version)
00551   !call ual_begin_pulse_action(backend_id, shot, run, &
00552   !     trim(USER), 'amns', trim(ds_version), idx)
00553   !call ual_open_pulse(idx, FORCE_CREATE_PULSE, backend_options, ids_status)
00554   call al_build_uri_from_legacy_parameters(backend_id, shot, run, &
00555        trim(user), 'amns', trim(ds_version), backend_options, uri, ids_status)
00556   call al_begin_dataentry_action(uri, force_create_pulse, idx, ids_status)
00557   write(*,*) 'IDX = ', idx
00558   write(*,*) 'ids_STATUS = ', ids_status
00559   if(ids_status.eq.-1) then
00560      write(0,*) 'Failure opening IDS!'
00561      stop 1
00562   endif
00563
00564   write(*,*) 'Calling ids_put'
00565   call ids_put(idx, amnspath, amns00, ids_status)
00566   write(*,*) 'ids_STATUS = ', ids_status
00567   if(ids_status.ne.0) then
00568      write(0,*) 'Failure writing IDS!'
00569      stop 1
00570   endif
00571   write(*,*) 'ids_put done'
00572   call imas_close  (idx)
00573
00574 ! the deallocation seems to cause occasional problems --- comment it out!
00575 !  do i1 = 1, size(amns00%release)
00576 !     call ids_deallocate_struct(amns00%release(i1), .false.)
00577 !  enddo
00578 !  deallocate(amns00%release)
00579 !  call ids_deallocate(amns00)
00580
00581   stop
00582 30 continue
00583   write(0,*) 'error'
00584   close(51)
00585   stop 1
00586
00587 end program amns_driver
00588
```

16.46 src/amns_driver/amns_nuclear.f90 File Reference

Modules

- module amns_nuclear

Functions/Subroutines

- subroutine amns_nuclear::nuclear_amns (amns, zn, am)
- subroutine add_beam_target (iproc, icoord, btreaction)
- subroutine amns_nuclear::allocate_process (process, nr, np)
- subroutine amns_nuclear::assign_reactantproduct (reactantproduct, label, zn, amn, multiplicity, relative, za)

Variables

- integer, parameter amns_nuclear::r8 = SELECTED_REAL_KIND (15, 300)

16.46.1 Function/Subroutine Documentation

16.46.1.1 add_beam_target()

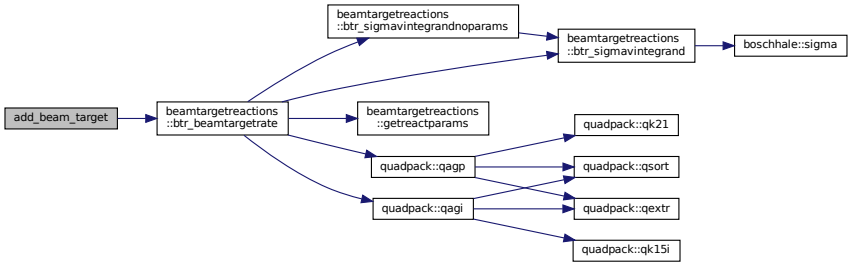
```
subroutine nuclear_amns::add_beam_target (
              integer, intent(in) iproc,
              integer, intent(in) icoord,
              integer, intent(in) btreaction )
```

Definition at line 460 of file amns_nuclear.f90.

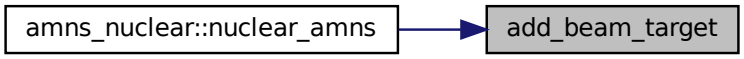
```
00461       use beamtargetreactions , only : btr_beamtargetrate
00462       implicit none
00463       integer, intent(in) :: iproc,icoord,btreaction
00464
```

```
00465       allocate(amns%process(iproc)%result_label(1))
00466       allocate(amns%process(iproc)%result_units(1))
00467
00468       amns%process(iproc)%result_label = 'Reaction rate for ' // trim(amns%process(iproc)%label(1))
00469       amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00470       amns%process(iproc)%result_transformation = 1 !0=no transformation, 1=10^(), 2=exp()
00471       amns%process(iproc)%table_dimension = 2
00472
00473       ! Set up the coordinates for the table
00474       amns%process(iproc)%coordinate_index = icoord
00475       npoints=(/79,199/)
00476
00477
00478       tmin=2.0_r8 !10^() eV   ! was 2.0_R8 (DPC: 2012-06-28)
00479       tmax=9.0_r8 !10^() eV
00480       emin=1.0_r8 !10^() eV
00481       emax=9.0_r8 !10^() eV   ! was 2.0_R8 (DPC: 2012-06-28)
00482
00483       if(lastwrittencoordinate<icoord) then
00484          allocate(amns%coordinate_system(icoord)%coordinate(2))
00485          do i=1,size(amns%coordinate_system(icoord)%coordinate)
00486             allocate(amns%coordinate_system(icoord)%coordinate(i)%values(npoints(i)),&
00487                  &   amns%coordinate_system(icoord)%coordinate(i)%label(1),&
00488                  &   amns%coordinate_system(icoord)%coordinate(i)%units(1),&
00489                  &   amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(2))
00490             amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(1:2)=1 ! 0= No
    extrapolation
00491             amns%coordinate_system(icoord)%coordinate(i)%interpolation_type=1      ! 1= Linear
    interpolation
00492          end do
00493
00494          amns%coordinate_system(icoord)%coordinate(1)%label(1)='Temperature x kB'
00495          amns%coordinate_system(icoord)%coordinate(2)%label(1)='Particle energy'
00496          amns%coordinate_system(icoord)%coordinate(1)%units(1)='eV'
00497          amns%coordinate_system(icoord)%coordinate(2)%units(1)='eV'
00498          amns%coordinate_system(icoord)%coordinate(1)%transformation = 1
00499          amns%coordinate_system(icoord)%coordinate(2)%transformation = 1
00500          amns%coordinate_system(icoord)%coordinate(1)%spacing = 0
00501          amns%coordinate_system(icoord)%coordinate(2)%spacing = 0
00502
00503          do i=1,npoints(1)
00504             amns%coordinate_system(icoord)%coordinate(1)%values(i) = &
00505                  & tmin + (tmax-tmin)/real(npoints(1)-1,kind=r8)*real(i-1,kind=r8)
00506          end do
00507          do i=1,npoints(2)
00508             amns%coordinate_system(icoord)%coordinate(2)%values(i) = &
00509                  & emin + (emax-emin)/real(npoints(2)-1,kind=r8)*real(i-1,kind=r8)
00510          end do
00511       end if
00512
00513       lastwrittencoordinate=icoord
00514
00515       ! Fill the table itself
00516       allocate(amns%process(iproc)%charge_state(1)%table_2d(npoints(1),npoints(2)))
00517
00518       write(*,*) 'Integrating beam-target fusion reaction "'//trim(amns%process(iproc)%label(1))//'".'
00519       do itemp=1,npoints(1)
00520          do iene=1,npoints(2)
00521             call btr_beamtargetrate(btreaction,&
00522                  & 10.0_r8 ** amns%coordinate_system(icoord)%coordinate(2)%values(iene ),&
00523                  & 10.0_r8 ** amns%coordinate_system(icoord)%coordinate(1)%values(itemp),&
00524                  & amns%process(iproc)%charge_state(1)%table_2d(itemp,iene),err)
00525             if(err /= 0) then
00526                write(*,*) 'error: reaction',btreaction,&
00527                     &'Temperature',10.0_r8 **
    amns%coordinate_system(icoord)%coordinate(1)%values(itemp),&
00528                     &'Energy',     10.0_r8 ** amns%coordinate_system(icoord)%coordinate(2)%values(iene
    ),&
00529                     &'err',err
00530                write(*,*) 'FILE: "'//__file__//'" line:',__line__
00531                stop 1
00532             end if
00533             if(amns%process(iproc)%charge_state(1)%table_2d(itemp,iene).ge.1e-300_r8) then
00534
    amns%process(iproc)%charge_state(1)%table_2d(itemp,iene)=log10(amns%process(iproc)%charge_state(1)%table_2d(itemp,iene)
00535             else
00536                amns%process(iproc)%charge_state(1)%table_2d(itemp,iene)=-300.0_r8
00537             endif
00538          end do
00539       end do
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.47 amns_nuclear.f90

```
00001 module amns_nuclear
00002
00003   INTEGER,  PARAMETER :: r8 = selected_real_kind (15, 300) ! Real*8
00004
00005 contains
00006
00007   subroutine nuclear_amns(amns, zn, am)
00008     use ids_schemas   ! IGNORE
00009     use codata, only: &
00010          & e_mass_in_amu => electron_mass_in_u, &
00011          & n_mass_in_amu => neutron_mass_in_u,  &
00012          & h_mass_in_amu => proton_mass_in_u,   &
00013          & d_mass_in_amu => deuteron_mass_in_u, &
00014          & t_mass_in_amu => triton_mass_in_u,   &
00015          & he3_mass_in_amu => helion_mass_in_u, &
00016          & he4_mass_in_amu => alpha_particle_mass_in_u
00017
00018     use beamtargetreactions , only : &
00019          & btr_reaction_dtn4he,  &
00020          & btr_reaction_tdn4he,  &
00021          & btr_reaction_ddpt,    &
00022          & btr_reaction_ddn3he,  &
00023          & btr_reaction_d3hep4he,&
00024          & btr_reaction_3hedp4he
00025
00026     implicit none
00027
00028     type (ids_amns_data) :: amns
00029     integer :: zn, am
00030     integer :: nprocs, iproc
00031     integer :: ncoords,icoord
00032     integer :: npoints(2)
00033     integer :: i,iene,itemp,btreaction,err
00034     real(kind=r8) :: emin,emax,tmin,tmax
00035     integer :: lastWrittenCoordinate !This is used to keep track of when the coords have been already
       written
00036     ! when calling add_beam_target multiple times
00037
00038     lastwrittencoordinate=0
00039
00040     nprocs=0
00041     if(zn.eq.1) then           ! H, D or T
00042        if(am.eq.2) then        ! D (shot = 2001)
00043           nprocs=6
```

```
00044           ncoords=1
00045           allocate(amns%coordinate_system(ncoords))
00046           allocate(amns%process(nprocs))
00047           do iproc=1, nprocs
00048              allocate(amns%process(iproc)%label(1))
00049              allocate(amns%process(iproc)%result_label(1))
00050              allocate(amns%process(iproc)%result_units(1))
00051              allocate(amns%process(iproc)%charge_state(1))
00052              allocate(amns%process(iproc)%source(1))
00053              amns%process(iproc)%source = 'amns_driver/amns_nuclear.f90'
00054              allocate(amns%process(iproc)%provider(1))
00055              amns%process(iproc)%provider = 'David.Coster@ipp.mpg.de on behalf of AMNS Team'
00056              allocate(amns%process(iproc)%citation(1))
00057              amns%process(iproc)%citation = 'ToDo'
00058           enddo
00059
00060           iproc=1
00061           amns%process(iproc)%label(1)='NUC_BB'                 ! D(D,p)T
00062           call allocate_process(amns%process(iproc), 2, 2)
00063           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00064                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00065           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00066                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00067           call assign_reactantproduct(amns%process(iproc)%products(1), &
00068                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00069           call assign_reactantproduct(amns%process(iproc)%products(2), &
00070                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00071           amns%process(iproc)%result_label = 'cross section for D(D,p)T'
00072           amns%process(iproc)%result_units = 'm^{2}'
00073           amns%process(iproc)%result_transformation = 1001
00074           amns%process(iproc)%table_dimension = 1
00075           amns%process(iproc)%coordinate_index = -1
00076           allocate(amns%process(iproc)%charge_state(1)%table_2d(12,1))
00077           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00078                (/ 31.3970_r8,  &
00079                   5.5576e4_r8, 2.1054e2_r8, -3.2638e-2_r8, 1.4987e-6_r8, 1.8181e-10_r8, &
00080                   0.0_r8, 0.0_r8, 0.0_r8, 0.0_r8, &
00081                   0.5_r8, 5000.0_r8 /), &
00082                (/ 12, 1 /) )
00083
00084           iproc=2
00085           amns%process(iproc)%label(1)='NUC_BB'                 ! D(D,n)^3He
00086           call allocate_process(amns%process(iproc), 2, 2)
00087           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00088                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00089           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00090                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00091           call assign_reactantproduct(amns%process(iproc)%products(1), &
00092                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00093           call assign_reactantproduct(amns%process(iproc)%products(2), &
00094                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00095           amns%process(iproc)%result_label = 'cross section for D(D,n)^3He'
00096           amns%process(iproc)%result_units = 'm^{2}'
00097           amns%process(iproc)%result_transformation = 1001
00098           amns%process(iproc)%table_dimension = 1
00099           amns%process(iproc)%coordinate_index = -1
00100           allocate(amns%process(iproc)%charge_state(1)%table_2d(12,1))
00101           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00102                (/ 31.3970_r8, &
00103                   5.3701e4_r8, 3.3027e2_r8, -1.2706e-1_r8, 2.9327e-5_r8, -2.5151e-9_r8, &
00104                   0.0_r8, 0.0_r8, 0.0_r8, 0.0_r8, &
00105                   0.5_r8, 4900.0_r8 /), &
00106                (/ 12, 1 /) )
00107
00108           iproc=3
00109           amns%process(iproc)%label(1)='NUC_TT'                 ! tt D(D,p)T
00110           call allocate_process(amns%process(iproc), 2, 2)
00111           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00112                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00113           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00114                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00115           call assign_reactantproduct(amns%process(iproc)%products(1), &
00116                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00117           call assign_reactantproduct(amns%process(iproc)%products(2), &
00118                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00119           amns%process(iproc)%result_label = 'reactivity for tt D(D,p)T'
00120           amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00121           amns%process(iproc)%result_transformation = 1002
00122           amns%process(iproc)%table_dimension = 1
00123           amns%process(iproc)%coordinate_index = -1
00124           allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00125           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00126                (/ 31.3970_r8,    937814.0_r8,&
00127                & 5.65718e-12_r8,   3.41267e-3_r8,   1.99167e-3_r8,  0.0_r8, &
00128                & 1.05060e-5_r8,  0.0_r8,          0.0_r8,  &
00129                   0.2_r8, 100.0_r8 /), &
00130                (/ 11, 1 /) )
```

```
00131
00132           iproc=4
00133           amns%process(iproc)%label(1)='NUC_TT'                 ! tt D(D,n)^3He
00134           call allocate_process(amns%process(iproc), 2, 2)
00135           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00136                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00137           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00138                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00139           call assign_reactantproduct(amns%process(iproc)%products(1), &
00140                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00141           call assign_reactantproduct(amns%process(iproc)%products(2), &
00142                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00143           amns%process(iproc)%result_label = 'reactivity for tt D(D,n)^3He'
00144           amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00145           amns%process(iproc)%result_transformation = 1002
00146           amns%process(iproc)%table_dimension = 1
00147           amns%process(iproc)%coordinate_index = -1
00148           allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00149           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00150                (/ 31.3970_r8,       937814.0_r8,&
00151                &  5.43360e-12_r8,    5.85778e-3_r8,   7.68222e-3_r8,  0.0_r8, &
00152                &  -2.96400e-6_r8,  0.0_r8,         0.0_r8,  &
00153                &  0.2_r8, 100.0_r8 /), &
00154                (/ 11, 1 /) )
00155
00156           iproc=5
00157           icoord=1
00158           amns%process(iproc)%label(1)='bt D(D,p)T'
00159           call allocate_process(amns%process(iproc), 2, 2)
00160           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00161                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00162           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00163                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00164           call assign_reactantproduct(amns%process(iproc)%products(1), &
00165                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00166           call assign_reactantproduct(amns%process(iproc)%products(2), &
00167                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00168           btreaction= btr_reaction_ddpt ! these are defined in beamTargetReactions.F90
00169           call add_beam_target(iproc,icoord,btreaction)
00170           amns%process(iproc)%label(1)='NUC_BT'                 ! bt D(D,p)T
00171
00172           iproc=6
00173           icoord=1
00174           amns%process(6)%label(1)='bt D(D,n)^3He'
00175           call allocate_process(amns%process(6), 2, 2)
00176           call assign_reactantproduct(amns%process(6)%reactants(1), &
00177                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00178           call assign_reactantproduct(amns%process(6)%reactants(2), &
00179                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00180           call assign_reactantproduct(amns%process(6)%products(1), &
00181                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00182           call assign_reactantproduct(amns%process(6)%products(2), &
00183                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00184           btreaction= btr_reaction_ddn3he ! these are defined in beamTargetReactions.F90
00185           call add_beam_target(iproc,icoord,btreaction)
00186           amns%process(6)%label(1)='NUC_BT'                 ! bt D(D,n)^3He
00187
00188        elseif(am.eq.3) then    ! T (shot = 3001)
00189           nprocs=6
00190           ncoords=1
00191           allocate(amns%coordinate_system(ncoords))
00192           allocate(amns%process(nprocs))
00193           do iproc=1, nprocs
00194              allocate(amns%process(iproc)%label(1))
00195              allocate(amns%process(iproc)%result_label(1))
00196              allocate(amns%process(iproc)%result_units(1))
00197              allocate(amns%process(iproc)%charge_state(1))
00198           enddo
00199
00200           iproc=1
00201           amns%process(iproc)%label(1)='NUC_BB'                 ! D(T,n)^4He
00202           call allocate_process(amns%process(iproc), 2, 2)
00203           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00204                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00205           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00206                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00207           call assign_reactantproduct(amns%process(iproc)%products(1), &
00208                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00209           call assign_reactantproduct(amns%process(iproc)%products(2), &
00210                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00211           amns%process(iproc)%result_label = 'cross section for D(T,n)^4He'
00212           amns%process(iproc)%result_units = 'm^{2}'
00213           amns%process(iproc)%result_transformation = 1001
00214           amns%process(iproc)%table_dimension = 1
00215           amns%process(iproc)%coordinate_index = -1
00216           allocate(amns%process(iproc)%charge_state(1)%table_2d(12,2))
00217           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
```

```
00218                (/ 34.3827_r8,  &
00219                   6.927e4_r8, 7.454e8_r8, 2.050e6_r8, 5.2002e4_r8, 0.0_r8, &
00220                   6.38e1_r8, -9.95e-1_r8, 6.981e-5_r8, 1.728e-4_r8, &
00221                   0.5_r8, 530.0_r8, &
00222                   34.3827_r8, &
00223                   -1.4714e6_r8, 0.0_r8, 0.0_r8, 0.0_r8, 0.0_r8, &
00224                   -8.4127e-3_r8, 4.7983e-6_r8, -1.0748e-9_r8, 8.5184e-14_r8, &
00225                   530.0_r8, 4700.0_r8 /), &
00226                (/ 12, 2 /) )
00227
00228           iproc=2
00229           amns%process(iproc)%label(1)='NUC_TT'                ! tt D(T,n)^4He
00230           call allocate_process(amns%process(iproc), 2, 2)
00231           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00232                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00233           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00234                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00235           call assign_reactantproduct(amns%process(iproc)%products(1), &
00236                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00237           call assign_reactantproduct(amns%process(iproc)%products(2), &
00238                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00239           amns%process(iproc)%result_label = 'reactivity for tt D(T,n)^4He'
00240           amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00241           amns%process(iproc)%result_transformation = 1002
00242           amns%process(iproc)%table_dimension = 1
00243           amns%process(iproc)%coordinate_index = -1
00244           allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00245           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00246                (/ 34.3827_r8,  1124656.0_r8,&
00247                &  1.17302e-9_r8,  1.51361e-2_r8, 7.51886e-2_r8, 4.60643e-3_r8, &
00248                   1.35000e-2_r8, -1.06750e-4_r8, 1.36600e-5_r8, &
00249                   0.2_r8, 100.0_r8 /), &
00250                (/ 11, 1 /) )
00251
00252           iproc=3
00253           icoord=1
00254           amns%process(iproc)%label(1)='bt D(T,n)^4He'
00255           call allocate_process(amns%process(iproc), 2, 2)
00256           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00257                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00258           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00259                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00260           call assign_reactantproduct(amns%process(iproc)%products(1), &
00261                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00262           call assign_reactantproduct(amns%process(iproc)%products(2), &
00263                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00264           btreaction= btr_reaction_dtn4he ! these are defined in beamTargetReactions.F90
00265           call add_beam_target(iproc,icoord,btreaction)
00266           amns%process(iproc)%label(1)='NUC_BT'                ! bt D(T,n)^4He
00267
00268           iproc=4
00269           icoord=1
00270           amns%process(iproc)%label(1)='bt T(D,n)^4He'
00271           call allocate_process(amns%process(iproc), 2, 2)
00272           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00273                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00274           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00275                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00276           call assign_reactantproduct(amns%process(iproc)%products(1), &
00277                'n', 0, n_mass_in_amu, 1, -1, 0.0_r8)
00278           call assign_reactantproduct(amns%process(iproc)%products(2), &
00279                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00280           btreaction= btr_reaction_tdn4he ! these are defined in beamTargetReactions.F90
00281           call add_beam_target(iproc,icoord,btreaction)
00282           amns%process(iproc)%label(1)='NUC_BT'                ! bt T(D,n)^4He
00283
00284           iproc=5
00285           amns%process(iproc)%label(1)='NUC_BB'                ! T(T,2n)^4He
00286           call allocate_process(amns%process(iproc), 2, 2)
00287           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00288                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00289           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00290                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00291           call assign_reactantproduct(amns%process(iproc)%products(1), &
00292                'n', 0, n_mass_in_amu, 2, -1, 0.0_r8)
00293           call assign_reactantproduct(amns%process(iproc)%products(2), &
00294                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00295           amns%process(iproc)%result_label = 'cross section for T(T,2n)^4He'
00296           amns%process(iproc)%result_units = 'm^{2}'
00297           amns%process(iproc)%result_transformation = 1006
00298           amns%process(iproc)%table_dimension = 1
00299           amns%process(iproc)%coordinate_index = -1
00300           allocate(amns%process(iproc)%charge_state(1)%table_2d(14,2))
00301 !
00302 ! Data taken from
00303 !
00304 !  https://gforge6.eufus.eu/svn/amnsproto/curation/nuclear/work_T(t,2n)4He.dat
```

```
00305 !
00306 ! revision 640
00307 !
00308           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00309                (/ 38.4224306526_r8, &
00310                   1.83514616e+05_r8, -3.17779782e+02_r8, 3.76709752e+00_r8, &
00311                   -2.60896847e-03_r8, 1.21563195e-06_r8, 6.78016367e-11_r8, &
00312                   3.36105299e-05_r8, 1.86611385e-05_r8, -2.88648569e-08_r8, &
00313                   1.45561737e-11_r8, -5.27692243e-16_r8, &
00314                   0.25_r8, 10.0e03_r8 /), &
00315                (/ 14, 1 /) )
00316
00317           iproc=6
00318           amns%process(iproc)%label(1)='NUC_TT'                 ! tt T(T,2n)^4He
00319           call allocate_process(amns%process(iproc), 2, 2)
00320           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00321                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00322           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00323                'T', 1, t_mass_in_amu, 1, -1, 0.0_r8)
00324           call assign_reactantproduct(amns%process(iproc)%products(1), &
00325                'n', 0, n_mass_in_amu, 2, -1, 0.0_r8)
00326           call assign_reactantproduct(amns%process(iproc)%products(2), &
00327                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00328           amns%process(iproc)%result_label = 'reactivity for tt T(T,2n)^4He'
00329           amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00330           amns%process(iproc)%result_transformation = 1002
00331           amns%process(iproc)%table_dimension = 1
00332           amns%process(iproc)%coordinate_index = -1
00333           allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00334 !
00335 ! Data taken from
00336 !
00337 ! https://gforge6.eufus.eu/svn/amnsproto/curation/nuclear/work_thermonuclear_T(t,2n)4He.dat
00338 !
00339 ! Revision: 641
00340 !
00341           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00342                (/ 38.4224306526_r8, 1404460.50227_r8, &
00343                   2.86867049e-11_r8, -1.26581396e-02_r8, -6.68347948e-01_r8, 6.31775307e-03_r8, &
00344                   1.44900887e-01_r8, -3.93623787e-03_r8, 1.74153264e-02_r8, &
00345                   0.2_r8, 100.0_r8 /), &
00346                (/ 11, 1 /) )
00347
00348        endif
00349
00350
00351      elseif(zn.eq.2) then       ! 3-He, 4-He
00352         if(am.eq.3) then        ! 3-He (shot = 3002)
00353           nprocs=4
00354           ncoords=1
00355           allocate(amns%coordinate_system(ncoords))
00356           allocate(amns%process(nprocs))
00357           do iproc=1, nprocs
00358              allocate(amns%process(iproc)%label(1))
00359              allocate(amns%process(iproc)%result_label(1))
00360              allocate(amns%process(iproc)%result_units(1))
00361              allocate(amns%process(iproc)%charge_state(1))
00362           enddo
00363
00364           iproc=1
00365           amns%process(iproc)%label(1)='NUC_BB'                 ! D(^3He,p)^4He
00366           call allocate_process(amns%process(iproc), 2, 2)
00367           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00368                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00369           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00370                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00371           call assign_reactantproduct(amns%process(iproc)%products(1), &
00372                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00373           call assign_reactantproduct(amns%process(iproc)%products(2), &
00374                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00375           amns%process(iproc)%result_label = 'cross section for D(^3He,p)^4He'
00376           amns%process(iproc)%result_units = 'm^{2}'
00377           amns%process(iproc)%result_transformation = 1001
00378           amns%process(iproc)%table_dimension = 1
00379           amns%process(iproc)%coordinate_index = -1
00380           allocate(amns%process(iproc)%charge_state(1)%table_2d(12,2))
00381           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00382                (/ 68.7508_r8,  &
00383                   5.7501e6_r8, 2.5226e3_r8, 4.5566e1_r8, 0.0_r8, 0.0_r8, &
00384                   -3.1995e-3_r8, -8.5530e-6_r8, 5.9014e-8_r8, 0.0_r8, &
00385                   0.3_r8, 900.0_r8, &
00386                   68.7508_r8, &
00387                   -8.3993e5_r8, 0.0_r8, 0.0_r8, 0.0_r8, 0.0_r8, &
00388                   -2.6830e-3_r8, 1.1633e-6_r8, -2.1332e-10_r8, 1.4250e-14_r8, &
00389                   900.0_r8, 4800.0_r8 /), &
00390                (/ 12, 2 /) )
00391
```

```
00392           iproc=2
00393           amns%process(iproc)%label(1)='NUC_TT'                ! tt D(^3He,p)^4He
00394           call allocate_process(amns%process(iproc), 2, 2)
00395           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00396                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00397           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00398                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00399           call assign_reactantproduct(amns%process(iproc)%products(1), &
00400                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00401           call assign_reactantproduct(amns%process(iproc)%products(2), &
00402                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00403           amns%process(iproc)%result_label = 'reactivity for tt D(^3He,p)^4He'
00404           amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00405           amns%process(iproc)%result_transformation = 1002
00406           amns%process(iproc)%table_dimension = 1
00407           amns%process(iproc)%coordinate_index = -1
00408           allocate(amns%process(iproc)%charge_state(1)%table_2d(11,1))
00409           amns%process(iproc)%charge_state(1)%table_2d = reshape( &
00410                (/ 68.7508_r8, 1124572.0_r8, &
00411                &  5.51036e-10_r8, 6.41918e-3_r8, -2.02896e-3_r8, -1.91080e-5_r8, &
00412                   1.35776e-4_r8,  0.0_r8,         0.0_r8, &
00413                   0.5_r8, 190.0_r8 /), &
00414                (/ 11, 1 /) )
00415
00416           iproc=3
00417           icoord=1
00418           amns%process(iproc)%label(1)='bt ^3He(D,p)^4He'
00419           call allocate_process(amns%process(iproc), 2, 2)
00420           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00421                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00422           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00423                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00424           call assign_reactantproduct(amns%process(iproc)%products(1), &
00425                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00426           call assign_reactantproduct(amns%process(iproc)%products(2), &
00427                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00428           btreaction= btr_reaction_3hedp4he ! these are defined in beamTargetReactions.F90
00429           call add_beam_target(iproc,icoord,btreaction)
00430           amns%process(iproc)%label(1)='NUC_BT'                ! bt ^3He(D,p)^4He
00431
00432           iproc=4
00433           icoord=1
00434           amns%process(iproc)%label(1)='bt D(^3He,p)^4He'
00435           call allocate_process(amns%process(iproc), 2, 2)
00436           call assign_reactantproduct(amns%process(iproc)%reactants(1), &
00437                'D', 1, d_mass_in_amu, 1, -1, 0.0_r8)
00438           call assign_reactantproduct(amns%process(iproc)%reactants(2), &
00439                'He', 2, he3_mass_in_amu, 1, -1, 0.0_r8)
00440           call assign_reactantproduct(amns%process(iproc)%products(1), &
00441                'H', 1, h_mass_in_amu, 1, -1, 0.0_r8)
00442           call assign_reactantproduct(amns%process(iproc)%products(2), &
00443                'He', 2, he4_mass_in_amu, 1, -1, 0.0_r8)
00444           btreaction= btr_reaction_d3hep4he ! these are defined in beamTargetReactions.F90
00445           call add_beam_target(iproc,icoord,btreaction)
00446           amns%process(iproc)%label(1)='NUC_BT'                ! bt D(^3He,p)^4He
00447
00448        endif
00449
00450     endif
00451     if(nprocs.gt.0) then
00452 !       allocate(amns%version(1), amns%source(1))
00453 !       amns%version = 'v0'
00454 !       amns%source  = 'NUCLEAR'
00455        amns%z_n     = zn
00456        amns%a       = am
00457     endif
00458
00459   contains
00460     subroutine add_beam_target(iproc,icoord,btreaction)
00461       use beamtargetreactions , only : btr_beamtargetrate
00462       implicit none
00463       integer, intent(in) :: iproc,icoord,btreaction
00464
00465       allocate(amns%process(iproc)%result_label(1))
00466       allocate(amns%process(iproc)%result_units(1))
00467
00468       amns%process(iproc)%result_label = 'Reaction rate for ' // trim(amns%process(iproc)%label(1))
00469       amns%process(iproc)%result_units = 'm^{3} s^{-1}'
00470       amns%process(iproc)%result_transformation = 1 !0=no transformation, 1=10^(), 2=exp()
00471       amns%process(iproc)%table_dimension = 2
00472
00473       ! Set up the coordinates for the table
00474       amns%process(iproc)%coordinate_index = icoord
00475       npoints=(/79,199/)
00476
00477
00478       tmin=2.0_r8 !10^() eV   ! was 2.0_R8 (DPC: 2012-06-28)
```

```
00479       tmax=9.0_r8 !10^() eV
00480       emin=1.0_r8 !10^() eV
00481       emax=9.0_r8 !10^() eV   ! was 2.0_R8 (DPC: 2012-06-28)
00482
00483       if(lastwrittencoordinate<icoord) then
00484          allocate(amns%coordinate_system(icoord)%coordinate(2))
00485          do i=1,size(amns%coordinate_system(icoord)%coordinate)
00486             allocate(amns%coordinate_system(icoord)%coordinate(i)%values(npoints(i)),&
00487                  &   amns%coordinate_system(icoord)%coordinate(i)%label(1),&
00488                  &   amns%coordinate_system(icoord)%coordinate(i)%units(1),&
00489                  &   amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(2))
00490             amns%coordinate_system(icoord)%coordinate(i)%extrapolation_type(1:2)=1 ! 0= No
      extrapolation
00491             amns%coordinate_system(icoord)%coordinate(i)%interpolation_type=1      ! 1= Linear
      interpolation
00492          end do
00493
00494          amns%coordinate_system(icoord)%coordinate(1)%label(1)='Temperature x kB'
00495          amns%coordinate_system(icoord)%coordinate(2)%label(1)='Particle energy'
00496          amns%coordinate_system(icoord)%coordinate(1)%units(1)='eV'
00497          amns%coordinate_system(icoord)%coordinate(2)%units(1)='eV'
00498          amns%coordinate_system(icoord)%coordinate(1)%transformation = 1
00499          amns%coordinate_system(icoord)%coordinate(2)%transformation = 1
00500          amns%coordinate_system(icoord)%coordinate(1)%spacing = 0
00501          amns%coordinate_system(icoord)%coordinate(2)%spacing = 0
00502
00503          do i=1,npoints(1)
00504             amns%coordinate_system(icoord)%coordinate(1)%values(i) = &
00505                  & tmin + (tmax-tmin)/real(npoints(1)-1,kind=r8)*real(i-1,kind=r8)
00506          end do
00507          do i=1,npoints(2)
00508             amns%coordinate_system(icoord)%coordinate(2)%values(i) = &
00509                  & emin + (emax-emin)/real(npoints(2)-1,kind=r8)*real(i-1,kind=r8)
00510          end do
00511       end if
00512
00513       lastwrittencoordinate=icoord
00514
00515       ! Fill the table itself
00516       allocate(amns%process(iproc)%charge_state(1)%table_2d(npoints(1),npoints(2)))
00517
00518       write(*,*) 'Integrating beam-target fusion reaction "'//trim(amns%process(iproc)%label(1))//'".'
00519       do itemp=1,npoints(1)
00520          do iene=1,npoints(2)
00521             call btr_beamtargetrate(btreaction,&
00522                  & 10.0_r8 ** amns%coordinate_system(icoord)%coordinate(2)%values(iene ),&
00523                  & 10.0_r8 ** amns%coordinate_system(icoord)%coordinate(1)%values(itemp),&
00524                  & amns%process(iproc)%charge_state(1)%table_2d(itemp,iene),err)
00525             if(err /= 0) then
00526                write(*,*) 'error: reaction',btreaction,&
00527                     &'Temperature',10.0_r8 **
      amns%coordinate_system(icoord)%coordinate(1)%values(itemp),&
00528                     &'Energy',     10.0_r8 ** amns%coordinate_system(icoord)%coordinate(2)%values(iene
      ),&
00529                     &'err',err
00530                write(*,*) 'FILE: "'//__file__//'" line:',__line__
00531                stop 1
00532             end if
00533             if(amns%process(iproc)%charge_state(1)%table_2d(itemp,iene).ge.1e-300_r8) then
00534
      amns%process(iproc)%charge_state(1)%table_2d(itemp,iene)=log10(amns%process(iproc)%charge_state(1)%table_2d(itemp,iene)
00535             else
00536                amns%process(iproc)%charge_state(1)%table_2d(itemp,iene)=-300.0_r8
00537             endif
00538          end do
00539       end do
00540    end subroutine add_beam_target
00541
00542
00543  end subroutine nuclear_amns
00544
00545  subroutine allocate_process(process, nr, np)
00546    use ids_schemas   ! IGNORE
00547    implicit none
00548    type (ids_amns_data_process) :: process
00549    integer :: nr, np
00550    integer :: irp
00551
00552    allocate(process%reactants(nr), process%products(np))
00553    do irp = 1, size(process%reactants)
00554       allocate(process%reactants(irp)%label(1))
00555       allocate(process%reactants(irp)%element(1))
00556 !       allocate(process%reactants(irp)%element(1)%label(1))
00557    enddo
00558    do irp = 1, size(process%products)
00559       allocate(process%products(irp)%label(1))
00560       allocate(process%products(irp)%element(1))
```

```
00561 !       allocate(process%products(irp)%element(1)%label(1))
00562     enddo
00563
00564   end subroutine allocate_process
00565
00566   subroutine assign_reactantproduct(reactantproduct, label, zn, amn, multiplicity, relative, za)
00567     use ids_schemas   ! IGNORE
00568 !    use imas_types       ! IGNORE
00569     implicit none
00570     type (ids_amns_data_process_reactant) :: reactantproduct
00571     character(len=* ):: label
00572     integer :: zn, multiplicity, relative
00573     real (kind=r8) :: amn, za
00574
00575     reactantproduct%label(1)=label
00576 !    reactantproduct%element(1)%label(1)=label
00577     reactantproduct%element(1)%z_n=zn
00578     reactantproduct%element(1)%a=nint(amn)
00579     reactantproduct%element(1)%multiplicity=multiplicity
00580     reactantproduct%mass=amn
00581     reactantproduct%relative_charge=relative
00582     reactantproduct%charge=za
00583
00584   end subroutine assign_reactantproduct
00585
00586 end module amns_nuclear
```

16.48 src/amns_driver/amns_provider_types.f90 File Reference

Data Types

- type amns_provider_types::amns_ids_list

  type for linked list of amns cpos

Modules

- module amns_provider_types

  Data types used for the ITM AMNS provider routines.

16.49 amns_provider_types.f90

```
00001
00004
00005 module amns_provider_types
00006
00007   !use ids_types       ! IGNORE
00008   use ids_schemas   ! IGNORE
00009
00010   include '../libamns/git_version_AMNS.h'
00011
00013   type amns_ids_list
00014      type (ids_amns_data) ::     amns_ids
00015      type (amns_ids_list), pointer :: prev, next => null()
00016   end type amns_ids_list
00017
00018 end module amns_provider_types
```

16.50 src/amns_driver/beamTargetReactions.f90 File Reference

Modules

- module beamtargetreactions

Functions/Subroutines

- subroutine, public beamtargetreactions::btr_error (err)

  a routine for printing the error messages

- subroutine, public beamtargetreactions::btr_beamtargetrate (reaction, E, T, rate, err)

this subroutine calculates the fusion reaction rate in units of [m^3/s] for the beam particle having energy E [eV] colliding with Maxwellian target at temperature T [eV] The actual reaction rate is then obtained by mulitplying the result with the target particle density.

- integer function beamtargetreactions::getreactparams (reaction, erange, mb, mt, sigmaMax, beamCorrection)
- real(kind=params_wp) function beamtargetreactions::btr_sigmavintegrandnoparams (u)

  This function is the one called by the numerical integrator.
- real(kind=params_wp) function beamtargetreactions::btr_sigmavintegrand (u, reaction, energyBeam, temperatureTarget)

  This function calculates the integrand for the beam-target fusion reaction rate.
- subroutine, public beamtargetreactions::btr_test_beamtargetrate ()

  This routine test the calculation of the beam-target fusion rate.
- subroutine beamtargetreactions::btr_test_integrand ()

  a test routine to print out the beam-target reaction rate integrand
- subroutine beamtargetreactions::btr_test_integrandmatrix ()

  a test routine to print out the beam-target reaction rate integrand at some beam energy but at several temperatures

Variables

- real(kind=params_wp), parameter beamtargetreactions::consts_pi = 3.14159265358979323846264338327950288419↩71_params_wp
- real(kind=params_wp), parameter beamtargetreactions::consts_twopi = 2.0_params_wp ∗ consts_pi
- real(kind=params_wp), parameter beamtargetreactions::consts_mtriton = 5.00735588e-27_params_wp
- real(kind=params_wp), parameter beamtargetreactions::consts_amu = 1.660538782e-27_params_wp
- real(kind=params_wp), parameter beamtargetreactions::consts_e = 1.602176487e-19_params_wp
- real(kind=params_wp), parameter beamtargetreactions::consts_mdeuteron = 3.34358320e-27_params_wp
- real(kind=params_wp), parameter beamtargetreactions::consts_mhe3 = 3.0160293_params_wp ∗ consts↩_amu
- integer, parameter beamtargetreactions::btr_success = 0
- integer, parameter beamtargetreactions::btr_unsupportedreaction = 1
- integer, parameter beamtargetreactions::btr_reaction_dtn4he = 1

  These flags define the different reactions inside this module.
- integer, parameter beamtargetreactions::btr_reaction_tdn4he = 2

  Tritium beam collides with deuterium plasma, yielding neutron and helium-4.
- integer, parameter beamtargetreactions::btr_reaction_ddpt = 3

  Deuterium beam collides with deuterium plasma, yielding proton and tritium.
- integer, parameter beamtargetreactions::btr_reaction_ddn3he = 4

  Deuterium beam collides with deuterium plasma, yielding neutron and helium-3.
- integer, parameter beamtargetreactions::btr_reaction_d3hep4he = 5

  Deuterium beam collides with helium-3, yielding proton and helium-4.
- integer, parameter beamtargetreactions::btr_reaction_3hedp4he = 6

  Helium-3 beam collides with deuterium, yielding proton and helium-4.

16.51 beamTargetReactions.f90

```
00001 !
00002 ! math in https://solps-mdsplus.aug.ipp.mpg.de/repos/ASCOT/branches/oasunta/docs/fusion_reactivity.tex
00003 !
00004 module beamtargetreactions
00005   use boschhale
00006   use quadpack
00007
00008   implicit none
00009
00010   integer, parameter,private :: params_wp =kind(1.0d0)
00011
00012   real(kind=params_wp), parameter :: consts_pi =
      3.14159265358979323846264338327950288419_params_wp!acos(-1.0_params_wp)
00013   real(kind=params_wp), parameter :: consts_twopi = 2.0_params_wp * consts_pi
00014   real(kind=params_wp), parameter :: consts_mtriton   = 5.00735588e-27_params_wp
```

```
00015   real(kind=params_wp), parameter :: consts_amu = 1.660538782e-27_params_wp ! kg
00016   real(kind=params_wp), parameter :: consts_e = 1.602176487e-19_params_wp   ! Coulomb
00017   real(kind=params_wp), parameter :: consts_mdeuteron = 3.34358320e-27_params_wp
00018   real(kind=params_wp), parameter :: consts_mhe3      = 3.0160293_params_wp * consts_amu !wikipedia!!!
00019
00020   integer, parameter :: btr_success = 0
00021   integer, parameter :: btr_unsupportedreaction = 1
00022
00025   integer, parameter :: btr_reaction_dtn4he = 1
00026   integer, parameter :: btr_reaction_tdn4he = 2
00027   integer, parameter :: btr_reaction_ddpt   = 3
00028   integer, parameter :: btr_reaction_ddn3he = 4
00029   integer, parameter :: btr_reaction_d3hep4he = 5
00030   integer, parameter :: btr_reaction_3hedp4he = 6
00031
00032
00033   public :: btr_error
00034   public :: btr_beamtargetrate
00035   public :: btr_test_beamtargetrate
00036
00037 contains
00038
00041   subroutine btr_error(err)
00042     implicit none
00043     integer, intent(in) :: err
00044     select case(err)
00045     case(btr_success)
00046     case(btr_unsupportedreaction)
00047        write(*,*) '------------------------------'
00048        write(*,*) 'btr_error:',btr_unsupportedreaction
00049        write(*,*) 'unsupported reaction. Aborting'
00050        write(*,*) '------------------------------'
00051     case default
00052        write(*,*) '----------------------------------------'
00053        write(*,*) 'btr_error:'
00054        write(*,*) 'Unknown error identifier.'
00055        write(*,*) '----------------------------------------'
00056     end select
00057   end subroutine btr_error
00058
00059
00060 !!$  !> This routine returns the fusion cross section
00061 !!$  subroutine btr_sigma(ene,reaction,crossSection,err)
00062 !!$    IMPLICIT NONE
00063 !!$    REAL(KIND=params_wp), INTENT(IN) :: ENE !< kinetic energy (keV) in CM-FRAME
00064 !!$    REAL(KIND=params_wp), intent(out) :: crossSection !<FUSION CROSS-SECTION (m^2)
00065 !!$
00066 !!$    !> Variables for passing information to the internal function INTEG:
00067 !!$    !! react_reaction - REACTION FLAG:
00068 !!$    !!                                1 = D(T,n)^4He
00069 !!$    !!                                2 = T(D,n)^4He
00070 !!$    !!                                3 = D(D,p)T
00071 !!$    !!                                4 = D(D,n)^3He
00072 !!$    !!                                5 = D(^3He,p)^4He
00073 !!$    !!                                6 = ^3He(D,p)^4He
00074 !!$    INTEGER, intent(in) :: Reaction
00075 !!$    integer, intent(out) :: err
00076 !!$
00077 !!$    err=btr_SUCCESS
00078 !!$
00079 !!$    crossSection=sigma(ene,reaction)
00080 !!$
00081 !!$  end subroutine btr_sigma
00082
00088   subroutine btr_beamtargetrate(reaction,E,T,rate,err)
00089     implicit none
00090     ! I/O variables
00091     integer, intent(in) :: reaction
00092     real(kind=params_wp), intent(in) :: e
00093     real(kind=params_wp), intent(in) :: t
00094     real(kind=params_wp), intent(out) :: rate
00095     integer, intent(out) :: err
00096     ! internal variables
00097     integer, parameter :: nthermal = 15
00098     real(kind=params_wp) :: erange(2)
00099     real(kind=params_wp) :: urange(2)
00100     real(kind=params_wp) :: sigmamax
00101     real(kind=params_wp) :: epsrel
00102     real(kind=params_wp) :: epsabs
00103     real(kind=params_wp) :: abserr
00104     real(kind=params_wp) :: tmp
00105     real(kind=params_wp) :: upperbound,beamcorrection
00106     real(kind=params_wp) :: vthermal,vbeam
00107     real(kind=params_wp) :: mt,mb
00108     real(kind=params_wp) :: u( 2*(2*nthermal +1) +2 )
00109     integer :: nu,iu,inf
00110     integer :: neval
```

```
00111
00112
00113     err=btr_success
00114
00115
00116
00117     err = getreactparams(reaction,erange,mb,mt,sigmamax,beamcorrection)
00118     if(err/=0) return
00119
00120     urange    =  sqrt(2.0_params_wp * consts_e * erange  *(mb+mt)/(mb*mt))
00121     sigmamax =  sqrt(2.0_params_wp * consts_e * sigmamax *(mb+mt)/(mb*mt))
00122     vthermal =  sqrt(2.0_params_wp * consts_e * t       /mt) ! *(mb+mt)/(mb*mt))
00123     vbeam     =  sqrt(2.0_params_wp * consts_e * e       /mb) ! *(mb+mt)/(mb*mt))
00124
00125     ! set the parameters for the integrator
00126     epsabs=1e-60_params_wp
00127     epsrel=1e-20_params_wp
00128
00129     ! The list of "special points" in u:
00130     ! vbeam + n*vthermal (n=-5:5)
00131     ! equivalent velocity for erange min, max
00132     ! Maxima of the sigma
00133     ! the last bound...infty starts from max of the others +3*vthermal
00134
00135     nu=0
00136
00137     ! Beam velocity \pm Thermal velocity
00138     do iu=1,(nthermal *2 +1)
00139        nu=nu+1
00140        u(nu) = vbeam*beamcorrection + (iu-nthermal-1) * vthermal
00141        if(u(nu)<= 0.0_params_wp) then
00142           nu = nu - 1
00143        end if
00144     end do
00145
00146     if(.false.) then
00147        ! sigma peak velocity \pm Thermal velocity
00148        do iu=1,(nthermal *2 +1)
00149           nu=nu+1
00150           u(nu) = sigmamax + (iu-nthermal-1) * vthermal
00151           if(u(nu)<= 0.0_params_wp) then
00152              nu = nu - 1
00153           end if
00154        end do
00155     end if
00156
00157     u( (nu+1) : (nu+2) ) = urange
00158     nu=nu+2
00159
00160     if(minval(u(1:nu)) .le. 0.0_params_wp) then
00161        write(*,*) 'lt zero velocity'
00162        write(*,*) u(1:nu)
00163        stop 10
00164     end if
00165
00166     upperbound = 3.0_params_wp*vthermal + maxval(u(1:nu))
00167
00168
00169     ! initialize the integrand function
00170     tmp=btr_sigmavintegrand(upperbound,reaction,e,t)
00171
00172 #ifdef VERBOSE
00173     write(*,*) 'calculating the finite integral'
00174 #endif
00175 !    call qags(btr_sigmavIntegrandNoParams,u(iu-1),u(iu),epsabs,epsrel,tmp,abserr,neval,err)
00176     call qagp (btr_sigmavintegrandnoparams, 0.0_params_wp, upperbound, &
00177          &     nu, u(1:nu), &
00178          &     epsabs, epsrel,&
00179          &     tmp, abserr, &
00180          &     neval, err )
00181
00182 #ifdef VERBOSE
00183     write(*,*) '     err:',err,' neval:',neval
00184     write(*,*) '     rate',tmp,'+-',abserr
00185 #endif
00186     select case(err)
00187     case(0)
00188     case(2)
00189 #ifdef VERBOSE
00190        write(*,*) '    the occurrence of roundoff error is'
00191        write(*,*) '           detected, which prevents the requested'
00192        write(*,*) '           tolerance from being achieved.'
00193        write(*,*) '           the error may be under-estimated.'
00194 #endif
00195     case(3)
00196 #ifdef VERBOSE
00197        write(*,*) '    extremely bad integrand behavior occurs'
```

```
00198        write(*,*) '           at some points of the integration '
00199        write(*,*) '           interval.'
00200 #endif
00201     case(4)
00202 #ifdef VERBOSE
00203       write(*,*) 'the algorithm does not converge.  Roundoff error is detected in the'
00204       write(*,*) 'extrapolation table.  It is assumed that the requested tolerance'
00205       write(*,*) 'cannot be achieved, and that the returned result is the best which '
00206       write(*,*) 'can be obtained.'
00207 #endif
00208     case(5)
00209 #ifdef VERBOSE
00210        write(*,*) '  the integral is probably divergent, or slowly convergent.'
00211 #endif
00212     case default
00213        stop 1
00214     end select
00215
00216     rate = tmp
00217
00218 #ifdef VERBOSE
00219     write(*,*) 'calculating the infinite integral'
00220 #endif
00221     inf = 1
00222     call qagi ( btr_sigmavintegrandnoparams, upperbound, inf, epsabs, epsrel, tmp, abserr, neval, err
     )
00223
00224 #ifdef VERBOSE
00225     write(*,*) '     err:',err,' neval:',neval
00226     write(*,*) '     rate',tmp,'+-',abserr
00227     write(*,*)
00228 #endif
00229
00230     select case(err)
00231     case(0)
00232     case(2)
00233 #ifdef VERBOSE
00234        write(*,*) '    the occurrence of roundoff error is'
00235        write(*,*) '           detected, which prevents the requested'
00236        write(*,*) '           tolerance from being achieved.'
00237        write(*,*) '           the error may be under-estimated.'
00238 #endif
00239     case(3)
00240 #ifdef VERBOSE
00241        write(*,*) '    extremely bad integrand behavior occurs'
00242        write(*,*) '           at some points of the integration '
00243        write(*,*) '           interval.'
00244 #endif
00245     case(4)
00246 #ifdef VERBOSE
00247       write(*,*) 'the algorithm does not converge.  Roundoff error is detected in the'
00248       write(*,*) 'extrapolation table.  It is assumed that the requested tolerance'
00249       write(*,*) 'cannot be achieved, and that the returned result is the best which '
00250       write(*,*) 'can be obtained.'
00251 #endif
00252     case(5)
00253 #ifdef VERBOSE
00254        write(*,*) '  the integral is probably divergent, or slowly convergent.'
00255 #endif
00256     case default
00257        stop 1
00258     end select
00259
00260     rate = rate + tmp
00261     err = 0
00262
00263
00264   end subroutine btr_beamtargetrate
00265
00266
00267   function getreactparams(reaction,erange,mb,mt,sigmaMax,beamCorrection) result(err)
00268     implicit none
00269     integer ::  err
00270     integer, intent(in) :: reaction
00271     real(kind=params_wp),intent(out) :: erange(2)
00272     real(kind=params_wp),intent(out) :: sigmamax
00273     real(kind=params_wp),intent(out) :: beamcorrection
00274     real(kind=params_wp),intent(out) :: mt,mb
00275
00276     err = 0
00277
00278     select case(reaction)
00279     case(btr_reaction_dtn4he)
00280        erange = (/ 0.5_params_wp, 4700.0_params_wp /)*1e3_params_wp
00281        mb=consts_mdeuteron
00282        mt=consts_mtriton
00283        sigmamax = 65.0e3_params_wp
```

```
00284     case(btr_reaction_tdn4he)
00285        erange = (/ 0.5_params_wp, 4700.0_params_wp /)*1e3_params_wp
00286        mb=consts_mtriton
00287        mt=consts_mdeuteron
00288     case(btr_reaction_ddpt)
00289        erange = (/ 0.5_params_wp, 5000.0_params_wp /)*1e3_params_wp
00290        mb=consts_mdeuteron
00291        mt=consts_mdeuteron
00292        sigmamax = 1580.0e3_params_wp
00293     case(btr_reaction_ddn3he)
00294        erange = (/ 0.5_params_wp, 4900.0_params_wp /)*1e3_params_wp
00295        mb=consts_mdeuteron
00296        mt=consts_mdeuteron
00297        sigmamax = 1100.0e3_params_wp
00298     case(btr_reaction_d3hep4he)
00299        erange = (/ 0.3_params_wp, 4800.0_params_wp /)*1e3_params_wp
00300        mb=consts_mdeuteron
00301        mt=consts_mhe3
00302        sigmamax = 265.0e3_params_wp
00303     case(btr_reaction_3hedp4he)
00304        erange = (/ 0.3_params_wp, 4800.0_params_wp /)*1e3_params_wp
00305        mb=consts_mhe3
00306        mt=consts_mdeuteron
00307        sigmamax = 265.0e3_params_wp
00308     case default
00309        err=btr_unsupportedreaction
00310        return
00311     end select
00312
00313
00314     beamcorrection  = 1.0_params_wp !0.81_params_wp
00315
00316   end function getreactparams
00317
00320   function btr_sigmavintegrandnoparams(u) result(integrand)
00321     implicit none
00322     real(kind=params_wp), intent(in) :: u
00323     real(kind=params_wp) :: integrand
00324
00325     integrand=btr_sigmavintegrand(u)
00326
00327   end function btr_sigmavintegrandnoparams
00328
00331   function btr_sigmavintegrand(u,reaction,energyBeam,temperatureTarget) result(integrand)
00332     implicit none
00333     ! I/O variables
00334     real(kind=params_wp), intent(in) :: u
00335     integer, intent(in), optional :: reaction
00336     real(kind=params_wp), intent(in), optional :: energybeam
00337     real(kind=params_wp), intent(in), optional :: temperaturetarget
00338     real(kind=params_wp) :: integrand
00339     ! Internal variables, some of them are saved to allow noParams version of the function for
    numerical integration
00340     real(kind=params_wp), save :: vb
00341     real(kind=params_wp), save :: tt
00342     real(kind=params_wp), save :: mt
00343     real(kind=params_wp), save :: mb
00344     integer, save :: reactionsave
00345     real(kind=params_wp) :: ecom
00346     real(kind=params_wp) :: crosssection
00347     real(kind=params_wp) :: firstexponent
00348     real(kind=params_wp) :: secondexponent
00349     real(kind=params_wp) :: hyperbolicsinarg
00350
00351     if(present(reaction)) then ! save the parameters
00352        select case(reaction)
00353        case(btr_reaction_dtn4he)
00354           mb=consts_mdeuteron
00355           mt=consts_mtriton
00356        case(btr_reaction_tdn4he)
00357           mb=consts_mtriton
00358           mt=consts_mdeuteron
00359        case(btr_reaction_ddpt)
00360           mb=consts_mdeuteron
00361           mt=consts_mdeuteron
00362        case(btr_reaction_ddn3he)
00363           mb=consts_mdeuteron
00364           mt=consts_mdeuteron
00365        case(btr_reaction_d3hep4he)
00366           mb=consts_mdeuteron
00367           mt=consts_mhe3
00368        case(btr_reaction_3hedp4he)
00369           mb=consts_mhe3
00370           mt=consts_mdeuteron
00371        case default
00372           write(*,*) 'error'
00373           stop 10
```

```
00374        end select
00375        vb=sqrt(2*consts_e*energybeam/mb) ! calculate the beam velocity [m/s] using classical model
00376        tt=temperaturetarget
00377        reactionsave=reaction
00378        integrand=0._params_wp
00379     else ! evaluate the integrand
00380        ecom=0.5_params_wp*mt*mb/(mt+mb)*u**2 ! center of mass frame energy for the reaction in Joules
00381        crosssection=sigma(ecom/(consts_e*1e3),reactionsave) ! the cross section needs the energy in
     keV
00382        ! We use different forms depending on u, vb and Tt to prevent numerical difficulties
00383        firstexponent=-mt*vb**2/(2*consts_e*tt)
00384        secondexponent=-mt*u**2/(2*consts_e*tt)
00385        hyperbolicsinarg=mt*u*vb/(consts_e*tt)
00386        if(hyperbolicsinarg<=10) then
00387           integrand=2*sqrt(mt/(consts_twopi*consts_e*tt))/vb*exp(firstexponent+secondexponent)&
00388                &*crosssection*u**2*sinh(hyperbolicsinarg)
00389        else ! for x>=10 the relative difference between sinh(x) and 1/2*exp(x) is less than 2e-9
00390
     integrand=sqrt(mt/(consts_twopi*consts_e*tt))/vb*exp(firstexponent+secondexponent+hyperbolicsinarg)*crosssection*u**2
00391        end if
00392     end if
00393   end function btr_sigmavintegrand
00394
00397   subroutine btr_test_beamtargetrate()
00398     implicit none
00399     real(kind=params_wp) :: e1,e2,u1
00400     real(kind=params_wp) :: t1,t2,u2
00401     real(kind=params_wp) :: erange(2),mb,mt,sigmamax,beamcorrection
00402     integer :: ne, nt, ie, it, nu, iu
00403     real(kind=params_wp), allocatable :: rate(:,:), integrand(:,:,:)
00404     real(kind=params_wp), allocatable :: e(:)
00405     real(kind=params_wp), allocatable :: t(:)
00406     real(kind=params_wp), allocatable :: u(:)
00407     real(kind=params_wp), allocatable :: ulo(:,:),uhi(:,:)
00408     integer :: reaction
00409     integer :: err
00410     integer :: chn
00411
00412     err=btr_success
00413
00414     write(*,*) 'Give the reaction'
00415     write(*,*) '1 == D(T,n)^4He'
00416     write(*,*) '2 == T(D,n)^4He'
00417     write(*,*) '3 == D(D,p)T'
00418     write(*,*) '4 == D(D,n)^3He'
00419     write(*,*) '5 == D(^3He,p)^4He'
00420     write(*,*) '6 == ^3He(D,p)^4He'
00421     read(*,*) reaction
00422     if(reaction<1 .or. reaction>6) then
00423        write(*,*) 'Invalid reaction. Aborting...'
00424        return
00425     end if
00426
00427     write(*,*) 'Give the beam energy range (E1 E2) [keV], and the number of energies (min 2)'
00428     read(*,*) e1, e2, ne
00429     write(*,*) 'Give the plasma temperature range (T1 T2) [keV], and the nmbuer of temperatures (min
     2)'
00430     read(*,*) t1, t2, nt
00431     e1=e1*1e3
00432     e2=e2*1e3
00433     t1=t1*1e3
00434     t2=t2*1e3
00435     u1=1.0_params_wp ! m/s
00436     u2=3.0e8
00437     nu=200
00438
00439     if(ne<2 .or. nt<2) then
00440        write(*,*) 'Too few energies or temperatures'
00441        return
00442     end if
00443
00444     err=  getreactparams(reaction,erange,mb,mt,sigmamax,beamcorrection)
00445     if(err /= 0 ) return
00446
00447
00448     allocate(rate(ne,nt),e(ne),t(nt),u(nu),integrand(ne,nt,nu))
00449     allocate(ulo(ne,nt),uhi(ne,nt))
00450     e1=log10(e1)
00451     e2=log10(e2)
00452     t1=log10(t1)
00453     t2=log10(t2)
00454     u1=log10(u1)
00455     u2=log10(u2)
00456     e=e1+(/ ( (ie-1)*( (e2-e1)/(ne-1) ), ie=1,ne) /)
00457     t=t1+(/ ( (it-1)*( (t2-t1)/(nt-1) ), it=1,nt) /)
00458     u=u1+(/ ( (iu-1)*( (u2-u1)/(nu-1) ), iu=1,nu) /)
00459     e = 10.0_params_wp ** e
```

```
00460     t = 10.0_params_wp ** t
00461     u = 10.0_params_wp ** u
00462
00463
00464     chn=26
00465     open(unit=chn,file="btr_E.test")
00466     write(chn,*) e
00467     close(chn)
00468     write(*,*) 'wrote the E-vector into file "btr_E.test"'
00469     open(unit=chn,file="btr_T.test")
00470     write(chn,*) t
00471     close(chn)
00472     write(*,*) 'wrote the T-vector into file "btr_T.test"'
00473     open(unit=chn,file="btr_u.test")
00474     write(chn,*) u
00475     close(chn)
00476     write(*,*) 'wrote the u-vector into file "btr_u.test"'
00477
00478     do ie=1,ne
00479        do it=1,nt
00480           call btr_beamtargetrate(reaction,e(ie),t(it),rate(ie,it),err)
00481           if(err/=btr_success) then
00482              call btr_error(err)
00483              stop 2
00484           end if
00485           ulo(ie,it) =  sqrt(2.0_params_wp * consts_e * e(ie) /mb)*beamcorrection - &
00486                &   sqrt(2.0_params_wp * consts_e * t(it)      /mt)
00487           uhi(ie,it) =  sqrt(2.0_params_wp * consts_e * e(ie) /mb)*beamcorrection + &
00488                &   sqrt(2.0_params_wp * consts_e * t(it)      /mt)
00489           integrand(ie,it,1) = btr_sigmavintegrand(10.0_params_wp,reaction,e(ie),t(it))
00490           do iu=1,nu
00491              integrand(ie,it,iu)=btr_sigmavintegrandnoparams(u(iu))
00492           end do
00493        end do
00494     end do
00495
00496     open(unit=chn,file="btr_rate.test")
00497     do ie=1,ne
00498        write(chn,*) rate(ie,:)
00499     end do
00500     close(chn)
00501     write(*,*) 'wrote the resulting rates into file "btr_rate.test"'
00502
00503     open(unit=chn,file="btr_ulo.test")
00504     do ie=1,ne
00505        write(chn,*) ulo(ie,:)
00506     end do
00507     close(chn)
00508     write(*,*) 'wrote the resulting lowever limits into file "btr_ulo.test"'
00509     open(unit=chn,file="btr_uhi.test")
00510     do ie=1,ne
00511        write(chn,*) uhi(ie,:)
00512     end do
00513     close(chn)
00514     write(*,*) 'wrote the resulting lowever limits into file "btr_uhi.test"'
00515
00516     chn=25
00517     open(unit=chn,file="btr_integrand.test")
00518     do ie=1,ne
00519        do it=1,nt
00520           write(chn,*) integrand(ie,it,:)
00521        end do
00522     end do
00523     close(chn)
00524     write(*,*) 'wrote the resulting rates into file "btr_integrand.test"'
00525
00526     deallocate(rate,e,t,u,integrand,ulo,uhi)
00527
00528   end subroutine btr_test_beamtargetrate
00529
00530
00533   subroutine btr_test_integrand()
00534     implicit none
00535     real(kind=params_wp), allocatable :: u(:)
00536     real(kind=params_wp), allocatable :: integrand(:)
00537     integer :: reaction
00538     integer :: nu
00539     integer :: iu
00540     integer :: chn
00541     real(kind=params_wp) :: erange(2)
00542     real(kind=params_wp) :: urange(2)
00543     real(kind=params_wp) :: eb
00544     real(kind=params_wp) :: t
00545
00546     iu=0
00547     nu=10000
00548
```

```
00549     write(*,*) 'Give the reaction for which the integrand will be tabulated'
00550     write(*,*) '1 == D(T,n)^4He'
00551     write(*,*) '2 == T(D,n)^4He'
00552     write(*,*) '3 == D(D,p)T'
00553     write(*,*) '4 == D(D,n)^3He'
00554     write(*,*) '5 == D(^3He,p)^4He'
00555     write(*,*) '6 == ^3He(D,p)^4He'
00556     read(*,*) reaction
00557
00558     select case(reaction) ! the energy range is given in keV!!!
00559     case(btr_reaction_dtn4he,btr_reaction_tdn4he)
00560        erange = (/ 0.5_params_wp, 4700.0_params_wp /)
00561        urange =
      sqrt(2.0_params_wp*consts_e*1e3*erange*(consts_mdeuteron+consts_mtriton)/(consts_mdeuteron*consts_mtriton))
00562     case(btr_reaction_ddpt)
00563        erange = (/ 0.5_params_wp, 5000.0_params_wp /)
00564        urange = sqrt(4*consts_e*1e3*erange/consts_mdeuteron)
00565     case(btr_reaction_ddn3he)
00566        erange = (/ 0.5_params_wp, 4900.0_params_wp /)
00567        urange = sqrt(4*consts_e*1e3*erange/consts_mdeuteron)
00568     case(btr_reaction_d3hep4he,btr_reaction_3hedp4he)
00569        erange = (/ 0.3_params_wp, 4800.0_params_wp /)
00570        urange =
      sqrt(2*consts_e*1e3*erange*(consts_mdeuteron+consts_mhe3)/(consts_mdeuteron*consts_mhe3))
00571     case default
00572        write(*,*) 'Unsupported reaction. Try again'
00573        return
00574     end select
00575
00576     write(*,*) 'Give the beam energy [keV]'
00577     read(*,*) eb
00578     write(*,*) 'Give the plasma temperature [keV]'
00579     read(*,*) t
00580
00581     allocate(u(nu),integrand(nu))
00582     u=urange(1)+(/ ( (iu-1)*( (urange(2)-urange(1))/(nu-1) ), iu=1,nu) /) ! make the u vector
00583     integrand(1)=btr_sigmavintegrand(u(1),reaction,eb*1e3,t*1e3) ! initialize the integrand function
00584     do iu=1,nu
00585        integrand(iu)=btr_sigmavintegrand(u(iu))
00586     end do
00587
00588     chn=27
00589     open(unit=chn,file="btr_integrand.test")
00590     do iu=1,nu
00591        write(chn,*) u(iu),integrand(iu)
00592     end do
00593     close(chn)
00594     write(*,*) 'The integrand is now tabulated into file "btr_integrand.test"'
00595
00596     deallocate(u,integrand)
00597
00598   end subroutine btr_test_integrand
00599
00603   subroutine btr_test_integrandmatrix()
00604     implicit none
00605     real(kind=params_wp), allocatable :: u(:)
00606     real(kind=params_wp), allocatable :: integrand(:,:)
00607     real(kind=params_wp), allocatable :: t(:)
00608     integer :: reaction
00609     integer :: nu
00610     integer :: iu
00611     integer :: chn
00612     real(kind=params_wp) :: erange(2)
00613     real(kind=params_wp) :: urange(2)
00614     real(kind=params_wp) :: eb
00615     real(kind=params_wp) :: t1,t2
00616     integer :: nT,iT
00617
00618     iu=0
00619     nu=10000
00620
00621     write(*,*) 'Give the reaction for which the integrand will be tabulated'
00622     write(*,*) '1 == D(T,n)^4He'
00623     write(*,*) '2 == T(D,n)^4He'
00624     write(*,*) '3 == D(D,p)T'
00625     write(*,*) '4 == D(D,n)^3He'
00626     write(*,*) '5 == D(^3He,p)^4He'
00627     write(*,*) '6 == ^3He(D,p)^4He'
00628     read(*,*) reaction
00629
00630     select case(reaction) ! the energy range is given in keV!!!
00631     case(btr_reaction_dtn4he,btr_reaction_tdn4he)
00632        erange = (/ 0.5_params_wp, 4700.0_params_wp /)
00633        urange =
      sqrt(2*consts_e*1e3*erange*(consts_mdeuteron+consts_mtriton)/(consts_mdeuteron*consts_mtriton))
00634     case(btr_reaction_ddpt)
00635        erange = (/ 0.5_params_wp, 5000.0_params_wp /)
```

```
00636         urange = sqrt(4*consts_e*1e3*erange/consts_mdeuteron)
00637      case(btr_reaction_ddn3he)
00638         erange = (/ 0.5_params_wp, 4900.0_params_wp /)
00639         urange = sqrt(4*consts_e*1e3*erange/consts_mdeuteron)
00640      case(btr_reaction_d3hep4he,btr_reaction_3hedp4he)
00641         erange = (/ 0.3_params_wp, 4800.0_params_wp /)
00642         urange =
      sqrt(2*consts_e*1e3*erange*(consts_mdeuteron+consts_mhe3)/(consts_mdeuteron*consts_mhe3))
00643      case default
00644         write(*,*) 'Unsupported reaction. Try again'
00645         return
00646      end select
00647
00648      write(*,*) 'Give the beam energy [keV]'
00649      read(*,*) eb
00650      write(*,*) 'Give the plasma temperature range (T1 T2) [keV], and the number of temperatures (min
      2)'
00651      read(*,*) t1, t2, nt
00652      eb=eb*1e3
00653      t1=t1*1e3
00654      t2=t2*1e3
00655
00656      allocate(u(nu),integrand(nu,nt),t(nt))
00657      u=urange(1)+(/ ( (iu-1)*( (urange(2)-urange(1))/(nu-1) ), iu=1,nu) /) ! make the u vector
00658      t=t1+(/ ( (it-1)*( (t2-t1)/(nt-1) ), it=1,nt) /)
00659      do it=1,nt
00660         integrand(1,it)=btr_sigmavintegrand(u(1),reaction,eb,t(it)) ! initialize the integrand function
00661         do iu=1,nu
00662            integrand(iu,it)=btr_sigmavintegrand(u(iu))
00663         end do
00664      end do
00665
00666      write(777,*) t
00667      write(888,*) u
00668      chn=29
00669      open(unit=chn,file="btr_integrandMatrix.test")
00670      do iu=1,nu
00671         write(chn,*) integrand(iu,:)
00672      end do
00673      close(chn)
00674      write(*,*) 'The integrand is now tabulated into file "btr_integrandMatrix.test"'
00675
00676      deallocate(u,integrand,t)
00677
00678    end subroutine btr_test_integrandmatrix
00679
00680 !!$  !> a test routine to print out the fusion cross section as a function of center of mass energy
00681 !!$  !<
00682 !!$  subroutine btr_test_sigma()
00683 !!$    implicit none
00684 !!$    real(kind=params_wp), allocatable :: energy(:)
00685 !!$    real(kind=params_wp), allocatable :: sigma(:)
00686 !!$    integer :: reaction
00687 !!$    integer :: nEnergies
00688 !!$    integer :: iE
00689 !!$    integer :: err
00690 !!$    integer :: chn
00691 !!$    real(kind=params_wp) :: erange(2)
00692 !!$
00693 !!$    err=btr_SUCCESS
00694 !!$    iE=0
00695 !!$    nEnergies=10000
00696 !!$
00697 !!$    write(*,*) 'Give the reaction'
00698 !!$    write(*,*) '1 == D(T,n)^4He'
00699 !!$    write(*,*) '2 == T(D,n)^4He'
00700 !!$    write(*,*) '3 == D(D,p)T'
00701 !!$    write(*,*) '4 == D(D,n)^3He'
00702 !!$    write(*,*) '5 == D(^3He,p)^4He'
00703 !!$    write(*,*) '6 == ^3He(D,p)^4He'
00704 !!$    read(*,*) reaction
00705 !!$
00706 !!$    select case(reaction)
00707 !!$    case(1,2)
00708 !!$       erange = (/ 0.5_params_wp, 4700.0_params_wp /)
00709 !!$    case(3)
00710 !!$       erange = (/ 0.5_params_wp, 5000.0_params_wp /)
00711 !!$    case(4)
00712 !!$       ERANGE = (/ 0.5_params_wp, 4900.0_params_wp /)
00713 !!$    case(5,6)
00714 !!$       ERANGE = (/ 0.3_params_wp, 4800.0_params_wp /)
00715 !!$    case default
00716 !!$       write(*,*) 'Unsupported reaction. Try again'
00717 !!$       return
00718 !!$    end select
00719 !!$
00720 !!$    allocate(energy(nEnergies),sigma(nEnergies))
```

```
00721 !!$    energy=erange(1)+(/ ( (iE-1)*( (erange(2)-erange(1))/(nEnergies-1) ), iE=1,nEnergies) /)
00722 !!$    do iE=1,nEnergies
00723 !!$       call btr_sigma(energy(iE),reaction,sigma(iE),err)
00724 !!$       if(err/=btr_SUCCESS) then
00725 !!$          call btr_error(err)
00726 !!$          return
00727 !!$       end if
00728 !!$    end do
00729 !!$
00730 !!$    chn=getFreeLUN()
00731 !!$    open(unit=chn,file="btr_sigma.test")
00732 !!$    do iE=1,nEnergies
00733 !!$       write(chn,*) energy(iE),sigma(iE)
00734 !!$    end do
00735 !!$    close(chn)
00736 !!$    deallocate(energy,sigma)
00737 !!$
00738 !!$  end subroutine btr_test_sigma
00739 !!$
00740 end module beamtargetreactions
```

16.52 src/amns_driver/bms.f90 File Reference

Modules

- module bms

Functions/Subroutines

- subroutine bms::read_bms (filename, y, x1, x2, x3, x4)

16.53 bms.f90

```
00001 ! routine to read 4D beam stopping data
00002 ! this is a temporary routine for testing
00003 ! later it will be replaced by an officially supplied routine (from ADAS)
00004 ! David.Coster@ipp.mpg.de
00005 ! 2020-12-15
00006 module bms
00007   implicit none
00008 contains
00009   subroutine read_bms(filename, y, x1, x2, x3, x4)
00010     real, allocatable :: y(:,:,:,:), x1(:), x2(:), x3(:), x4(:)
00011     integer n1, n2, n3, n4, i1, i2, i3, i4
00012     character (len=*) :: filename
00013
00014     open(unit=10, file=filename)
00015     read(10,*)  ! skip line 1
00016     read(10,*)  ! skip line 2
00017     read(10,*)  n1, n2, n3, n4
00018     read(10,*)  ! skip line 4
00019     allocate(x1(n1), x2(n2), x3(n3), x4(n4), y(n1, n2, n3, n4))
00020     read(10,*) (x1(i1), i1=1, n1)
00021     read(10,*) (x2(i2), i2=1, n2)
00022     read(10,*) (x3(i3), i3=1, n3)
00023     read(10,*) (x4(i4), i4=1, n4)
00024     read(10,*)  ! skip ?
00025     read(10,*) ((((y(i1,i2,i3,i4), i4=1, n4), i3=1, n3), i2=1, n2), i1=1, n1)
00026     x2 = x2 * 1e6
00027     return
00028   end subroutine read_bms
00029 end module bms
```

16.54 src/amns_driver/boschHale.f90 File Reference

Modules

- module boschhale

Functions/Subroutines

- real(kind=params_wp) function boschhale::sigma (ENE, reaction)

16.55 boschHale.f90

```
00001 #define IGNORE_LOW_E_LIM true
00002 module boschhale
00003
00004   INTEGER, PARAMETER, PRIVATE :: params_wp=kind(1.0d0)
00005
00006
00007 contains
00037   FUNCTION sigma(ENE,reaction)
00038
00039
00040     IMPLICIT NONE
00041
00042     !     ERANGE  - RANGE OF VALIDITY FOR THE MODEL
00043     !     S       - PADE POLYNOMIAL
00044     ! ----------------------------------------------------------------------
00045
00046     REAL(kind=params_wp), INTENT(IN) :: ene
00047     real(kind=params_wp) :: atene
00048     REAL(kind=params_wp) :: a(5), b(4), bg
00049     REAL(kind=params_wp), dimension(2) :: erange
00050     REAL(kind=params_wp) :: s
00051     REAL(kind=params_wp) :: sigma
00052     REAL(kind=params_wp), parameter :: tom2 = 1.0e-31_params_wp !(originally mbarn)
00053
00054     INTEGER, SAVE :: reactionsave
00055
00064     INTEGER, intent(in), optional :: reaction
00065
00066 #ifdef IGNORE_LOW_E_LIM
00067     if(ene==0.0_params_wp) then
00068        !         write(*,*) 'ENE==0.0 => SIGMA=0.0 ENE=',ENE
00069        sigma = 0.0_params_wp
00070        RETURN
00071     end if
00072 #endif
00073
00074     if(present(reaction)) then
00075        reactionsave=reaction
00076        !         write(*,*) 'Saving reaction', reaction
00077     end if
00078
00079     ! Initialize vars to 0.
00080     bg=0.
00081     a=(/ 0., 0. ,0. ,0. ,0. /)
00082     b=(/ 0., 0. ,0. ,0. /)
00083     !
00084     !     SET THE CONSTANTS BG, A(), b(), AND THE VALIDITY RANGE ERANGE
00085
00086     SELECT CASE(reactionsave)
00087     CASE(1,2) ! DT
00088        IF(ene <= 530) THEN
00089           bg = 34.3827
00090           a = (/ 6.927e4, 7.454e8, 2.050e6, 5.2002e4, 0.0 /)
00091           b = (/ 6.38e1, -9.95e-1, 6.981e-5, 1.728e-4 /)
00092        ELSE
00093           bg = 34.3827
00094           a = (/ -1.4714e6, 0.0, 0.0, 0.0, 0.0 /)
00095           b = (/ -8.4127e-3, 4.7983e-6, -1.0748e-9, 8.5184e-14 /)
00096        ENDIF
00097        erange = (/ 0.5, 4700.0 /)
00098 #ifdef IGNORE_LOW_E_LIM
00099        erange(1) = 0.0
00100 #endif
00101     CASE(3) ! DD -> T
00102        bg = 31.3970
00103        a = (/ 5.5576e4, 2.1054e2, -3.2638e-2, 1.4987e-6, 1.8181e-10 /)
00104        b = (/ 0.0, 0.0, 0.0, 0.0 /)
00105        erange = (/ 0.5, 5000.0 /)
00106 #ifdef IGNORE_LOW_E_LIM
00107        erange(1) = 0.0
00108 #endif
00109     CASE(4) ! DD -> 3He
00110        bg = 31.3970
00111        a = (/ 5.3701e4, 3.3027e2, -1.2706e-1, 2.9327e-5, -2.5151e-9 /)
00112        b = (/ 0.0, 0.0, 0.0, 0.0 /)
00113        erange = (/ 0.5, 4900.0 /)
00114 #ifdef IGNORE_LOW_E_LIM
00115        erange(1) = 0.0
00116 #endif
00117     CASE(5,6) ! 3HeD
00118        IF(ene <= 900) THEN
00119           bg = 68.7508
00120           a = (/ 5.7501e6, 2.5226e3, 4.5566e1, 0.0, 0.0 /)
00121           b = (/ -3.1995e-3, -8.5530e-6, 5.9014e-8, 0.0 /)
00122        ELSE
```

```
00123           bg = 68.7508
00124           a = (/ -8.3993e5, 0.0, 0.0, 0.0, 0.0 /)
00125           b = (/ -2.6830e-3, 1.1633e-6, -2.1332e-10, 1.4250e-14 /)
00126        ENDIF
00127        erange = (/ 0.3, 4800.0 /)
00128 #ifdef IGNORE_LOW_E_LIM
00129        erange(1) = 0.0
00130 #endif
00131     CASE DEFAULT
00132        write(*,*) 'ASSERTION FAILED SVNROOT'
00133        write(*,*)  'reaction:',reactionsave,'Energy', ene
00134        write(*,*) 'FILE: "'//__file__//'" line:',__line__
00135        write(*,*) "Unknown fusion reaction!"
00136        stop 40
00137     END SELECT
00138
00139     !      IF(ENE < ERANGE(1) .OR. ENE > ERANGE(2)) THEN
00140     !         WRITE(*,*)"Energy", ENE, "outside the validity range (",  &
00141     !              & ERANGE(1),"...", ERANGE(2),") of the used fusion cross-section ", &
00142     !              "model encountered!"
00143     !         write(*,*) 'ASSERTION FAILED SVNROOT'
00144     !         write(*,*) 'FILE: "'//__FILE__//'" line:',__LINE__
00145     !         stop 45
00146     !      ENDIF
00147
00148     ! Clamp the energy for evaluating the Astrophysical S-factor to the maximum.
00149     atene=ene
00150     IF( ene > erange(2)) atene=erange(2)
00151
00152
00153     !     CALCULATE THE VALUE OF S FROM THE PADE POLYNOMIAL
00154     s = ( a(1) + atene*(a(2) + atene*(a(3) &
00155          & + atene*( a(4)+atene*a(5) ) ) ) ) &
00156          & /( 1 + atene*(b(1) + atene*(b(2) &
00157          & + atene*( b(3)+atene*b(4) ) ) ) )
00158
00159     ! There tends to happen numerical underflow, so let's catch it!
00160 #ifdef IGNORE_LOW_E_LIM
00161     if(bg/sqrt(atene)>710.0_params_wp )then
00162        !write(*,*) 'BG/SQRT(atENE)>710.0 => SIGMA=0.0 BG/SQRT(atENE)=',BG/SQRT(atENE)
00163        sigma = 0.0_params_wp
00164        RETURN
00165     end if
00166 #endif
00167
00168     !     CALCULATE SIGMA FROM S, energy, AND BG
00169     !     (using the actual energy, not the clamped one)
00170     !
00171     sigma = s / (ene * exp(bg/sqrt(ene))) * tom2
00172     !      OPEN(123,file='sigma.txt')
00173     !      WRITE(123,*) atENE, SIGMAX
00174     ! CLOSE(123)
00175
00176   END FUNCTION sigma
00177   ! ------------------------------------------------------------------------------
00178   ! ------------------------------------------------------------------------------
00179
00180
00181
00182
00183 end module boschhale
```

16.56 src/amns_driver/fundamental_constants.f90 File Reference

16.57 fundamental_constants.f90

```
00001 include 'codata_2018.f90'      ! Recommended physical constants since 2018
```

16.58 src/amns_driver/i4fctn.f File Reference

Functions/Subroutines

- integer function i4fctn (STR, IABT)

16.58.1 Function/Subroutine Documentation

16.58.1.1 i4fctn()

```
integer function i4fctn (
            character, dimension(*) STR,
            integer IABT )
```

Definition at line 3 of file i4fctn.f.

```
00004       IMPLICIT NONE
00005 C-----------------------------------------------------------------------
00006 C
00007 C  *************** FORTRAN77 INTEGER*4 FUNCTION: I4FCTN ****************
00008 C
00009 C  PURPOSE:  TO CONVERT AN INTEGER NUMBER STORED IN A STRING
00010 C            INTO A INTEGER*4 VARIABLE
00011 C
00012 C  CALLING PROGRAM: GENERAL USE
00013 C
00014 C  FUNCTION:
00015 C
00016 C          (I*4)   I4FCTN  = FUNCTION NAME
00017 C          (C*(*)) STR     = STRING CONTAINING SINGLE FLOATING POINT NO.
00018 C          (I*4)   IABT    = RETURN CODE:
00019 C                                0 => NO ERROR
00020 C                                1 => ERROR (A VALUE 'I4FCTN=0' WILL BE
00021 C                                           RETURNED).
00022 C
00023 C          (C*1)   CH0      = PARAMETER = '0'
00024 C          (C*1)   CH9      = PARAMETER = '9'
00025 C          (C*1)   BLANK    = PARAMETER = ' '
00026 C          (C*1)   CPLUS    = PARAMETER = '+'
00027 C          (C*1)   CMINUS   = PARAMETER = '-'
00028 C
00029 C          (I*4)   ILEN     = LENGTH OF 'STR' STRING IN BYTES
00030 C          (I*4)   ILAST    = POSITION OF LAST BYTE OF IDENTIFIED NUMBER
00031 C          (I*4)   I1       = STARTING BYTE IN 'STR' OF NUMBER
00032 C                             INCLUDING SIGN IF PRESENT
00033 C          (I*4)   IS       = 0 => NUMBER HAS NO SIGN
00034 C                             1 => NUMBER HAS A SIGN
00035 C          (I*4)   ICH0     = ICHAR('0')
00036 C          (I*4)   ICH9     = ICHAR('9')
00037 C          (I*4)   ISTR     = ICHAR(CURRENT BYTE POSITION IN 'STR')
00038 C          (I*4)   I        = GENERAL USE
00039 C
00040 C          (L*4)   LFOUND   = .TRUE.  => ALL OF THE INPUT NUMBER BYTES
00041 C                                        HAVE BEEN ASSESSED.
00042 C                             .FALSE. => INPUT NUMBER BYTES STILL BEING
00043 C                                        ASSESSED.
00044 C          (L*4)   LSTART   = .TRUE.  => THE FIRST DIGIT HAS BEEN FOUND
00045 C                             .FALSE. => THE FIRST DIGIT HAS NOT YET
00046 C                                        BEEN REACHED.
00047 C
00048 C          (C*5)   CFORM5   = FORMAT FOR INTERNAL READING OF INTEGER
00049 C
00050 C
00051 C NOTE:     AN ERROR WILL OCCUR (IABT=1) IF THERE IS MORE THAN ONE
00052 C           NUMBER OCCURING IN THE STRING 'STR()'
00053 C
00054 C
00055 C AUTHOR:   PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00056 C           K1/0/37
00057 C           JET EXT. 2520
00058 C
00059 C DATE:     11/07/90
00060 C
00061 C UPDATE:   11/02/92 - PE BRIDEN: BLANKS NOW ALLOWED BETWEEN SIGN AND
00062 C                                 FIRST DIGIT. LSTART VARIABLE ADDED.
00063 C                                 VARIABLE I2 REMOVED.
00064 C                                 + SOME MINOR RECODING - (IF STRING
00065 C                                 ENTERED IS BLANK IABT IS NOW SET TO 1)
00066 C
00067 C UPDATE:   16/08/93 - PE BRIDEN: CORRECTED BUG TO ALLOW BLANKS BETWEEN
00068 C                                 SIGN AND FIRST DIGIT (SEE ABOVE).
00069 C                                 1) ILAST VARIABLE ADDED.
00070 C                                 2) FORMATTED READ USED INSTEAD OF *
00071 C                                    WHEN CONVERTING IDENTIFIED INTEGER
00072 C                                    USING THE INTERNAL READ. (THIS
00073 C                                    RESTRICTS IDENTIFIED NUMBER TO BE
00074 C                                    < 100 BYTES IN LENGTH!)
00075 C                                 3) EXCLUDE TRAILING BLANKS IN THE
00076 C                                    INTERNAL READING OF THE INTEGER
00077 C                                    I.E. STR(I1:ILAST) INSTEAD OF
00078 C                                         STR(I1:ILEN)
00079 C
00080 C UPDATE:   07/03/95 - PE BRIDEN: INSTEAD OF USING FORMAT SPECIFIER I99
00081 C                                 WHEN INTERNALLY READING THE INTEGER
00082 C                                 CREATE THE APPROPRIATE SPECIFIER
```

```
00083 C                                WITHIN CFORM5 AND USE THIS.
00084 C
00085 C VERSION  : 1.3
00086 C DATE     : 20-12-2001
00087 C MODIFIED : Martin O'Mullane
00088 C               - Removed mainframe listing information beyond column 72.
00089 C
00090 C VERSION  : 1.3
00091 C DATE     : 10-04-2007
00092 C MODIFIED : Allan Whiteford
00093 C               - Modified documentation as part of automated
00094 C                 subroutine documentation preparation.
00095 C
00096 C-----------------------------------------------------------------------
00097 C
00098 C-----------------------------------------------------------------------
00099       CHARACTER  CH0*1   , CH9*1   , BLANK*1   , CPLUS*1   , CMINUS*1
00100 C-----------------------------------------------------------------------
00101       parameter( ch0='0', ch9='9', blank=' ', cplus='+', cminus='-' )
00102 C-----------------------------------------------------------------------
00103       CHARACTER  STR*(*)
00104 C-----------------------------------------------------------------------
00105       INTEGER    I4FCTN  , IABT
00106       INTEGER    I1      , IS      , ILEN      , ILAST     ,
00107      &           ICH0    , ICH9    , ISTR      , I
00108 C-----------------------------------------------------------------------
00109       LOGICAL    LSTART  , LFOUND
00110 C-----------------------------------------------------------------------
00111       CHARACTER  CFORM5*5
00112 C-----------------------------------------------------------------------
00113       DATA       cform5 / '(I??)' /
00114 C-----------------------------------------------------------------------
00115       SAVE       cform5
00116 C-----------------------------------------------------------------------
00117 C
00118 C-----------------------------------------------------------------------
00119 C INITIALIZE VALUES
00120 C-----------------------------------------------------------------------
00121 C
00122       i4fctn = 0
00123       iabt   = 0
00124       i1     = 0
00125       is     = 0
00126       lstart = .false.
00127       lfound = .false.
00128       ich0   = ichar(ch0)
00129       ich9   = ichar(ch9)
00130       ilen   = len(str)
00131       ilast  = ilen
00132 C
00133 C-----------------------------------------------------------------------
00134 C FIND STARTING BYTE OF NUMBER
00135 C-----------------------------------------------------------------------
00136 C
00137          DO 1 i=1,ilen
00138                IF ( str(i:i).NE.blank ) THEN
00139                   i1 = i
00140                   GOTO 2
00141                ENDIF
00142     1    CONTINUE
00143 C
00144 C-----------------------------------------------------------------------
00145 C IDENTIFY IF NUMBER HAS A SIGN
00146 C-----------------------------------------------------------------------
00147 C
00148     2    IF (i1.EQ.0) THEN
00149             iabt = 1
00150             RETURN
00151          ENDIF
00152 C
00153       IF   ( ( str(i1:i1).EQ.cplus  )
00154      &                    .OR.
00155      &       ( str(i1:i1).EQ.cminus ) ) is=1
00156 C
00157 C-----------------------------------------------------------------------
00158 C IDENTIFY IF NUMBER IS OF A VALID FORM
00159 C-----------------------------------------------------------------------
00160 C
00161          DO 3 i=i1+is,ilen
00162
00163                IF (lfound) THEN
00164 C
00165 C-----------------------------------------------------------------------
00166 C INPUT NO. COMPLETELY DEFINED: IDENTIFY IF EXTRA NON-BLANK BYTES EXIST
00167 C-----------------------------------------------------------------------
00168 C
00169                   IF (str(i:i).NE.blank) iabt=1
```

```
00170 C-----------------------------------------------------------------------
00171                ELSEIF (str(i:i).EQ.blank) THEN
00172                   lfound = lstart
00173 C-----------------------------------------------------------------------
00174                ELSE
00175                   lstart = .true.
00176                   ilast  = i
00177                   istr   = ichar(str(i:i))
00178                   IF ( (istr.LT.ich0) .OR. (istr.GT.ich9) ) iabt=1
00179                ENDIF
00180 C
00181 C-----------------------------------------------------------------------
00182 C RETURN ERROR CODE IF ERROR FOUND
00183 C-----------------------------------------------------------------------
00184 C
00185             IF (iabt.NE.0) RETURN
00186     3    CONTINUE
00187 C
00188 C-----------------------------------------------------------------------
00189 C IDENTIFY IF VALID NUMBER FOUND (RECODED: PEB 11/02/92)
00190 C                                (RECODED: PEB 07/03/95 - ADDED CFORM5)
00191 C YES => USE INTERNAL READ TO OBTAIN THE INTEGER NUMBER
00192 C NO  => RETURN ERROR CODE IF ERROR FOUND
00193 C-----------------------------------------------------------------------
00194 C
00195          IF (lstart) THEN
00196             i      = 1 + ilast - i1
00197             i      = min0(i,99)
00198             WRITE(cform5(3:4),'(I2.2)') i
00199             READ(str(i1:ilast),cform5) i4fctn
00200          ELSE
00201             iabt=1
00202          ENDIF
00203 C
00204 C-----------------------------------------------------------------------
00205 C
00206       RETURN
```

16.59 i4fctn.f

```
00001 CX ULTRIX PORT - SCCS info: Module @(#)$Header: /home/adascvs/fortran/adaslib/utility/i4fctn.for,v 1.4
       2007/04/11 13:02:01 allan Exp $ Date $Date: 2007/04/11 13:02:01 $
00002 CX
00003       FUNCTION i4fctn( STR , IABT )
00004       IMPLICIT NONE
00005 C-----------------------------------------------------------------------
00006 C
00007 C  *************** FORTRAN77 INTEGER*4 FUNCTION: I4FCTN ****************
00008 C
00009 C  PURPOSE:  TO CONVERT AN INTEGER NUMBER STORED IN A STRING
00010 C             INTO A INTEGER*4 VARIABLE
00011 C
00012 C  CALLING PROGRAM: GENERAL USE
00013 C
00014 C  FUNCTION:
00015 C
00016 C          (I*4)   I4FCTN  = FUNCTION NAME
00017 C          (C*(*)) STR     = STRING CONTAINING SINGLE FLOATING POINT NO.
00018 C          (I*4)   IABT    = RETURN CODE:
00019 C                                0 => NO ERROR
00020 C                                1 => ERROR (A VALUE 'I4FCTN=0' WILL BE
00021 C                                           RETURNED).
00022 C
00023 C          (C*1)   CH0      = PARAMETER = '0'
00024 C          (C*1)   CH9      = PARAMETER = '9'
00025 C          (C*1)   BLANK    = PARAMETER = ' '
00026 C          (C*1)   CPLUS    = PARAMETER = '+'
00027 C          (C*1)   CMINUS   = PARAMETER = '-'
00028 C
00029 C          (I*4)   ILEN     = LENGTH OF 'STR' STRING IN BYTES
00030 C          (I*4)   ILAST    = POSITION OF LAST BYTE OF IDENTIFIED NUMBER
00031 C          (I*4)   I1       = STARTING BYTE IN 'STR' OF NUMBER
00032 C                             INCLUDING SIGN IF PRESENT
00033 C          (I*4)   IS       = 0 => NUMBER HAS NO SIGN
00034 C                             1 => NUMBER HAS A SIGN
00035 C          (I*4)   ICH0     = ICHAR('0')
00036 C          (I*4)   ICH9     = ICHAR('9')
00037 C          (I*4)   ISTR     = ICHAR(CURRENT BYTE POSITION IN 'STR')
00038 C          (I*4)   I        = GENERAL USE
00039 C
00040 C          (L*4)   LFOUND   = .TRUE.  => ALL OF THE INPUT NUMBER BYTES
00041 C                                        HAVE BEEN ASSESSED.
00042 C                             .FALSE. => INPUT NUMBER BYTES STILL BEING
00043 C                                        ASSESSED.
00044 C          (L*4)   LSTART   = .TRUE.  => THE FIRST DIGIT HAS BEEN FOUND
```

```
00045 C                              .FALSE. => THE FIRST DIGIT HAS NOT YET
00046 C                                         BEEN REACHED.
00047 C
00048 C           (C*5)    CFORM5    = FORMAT FOR INTERNAL READING OF INTEGER
00049 C
00050 C
00051 C NOTE:      AN ERROR WILL OCCUR (IABT=1) IF THERE IS MORE THAN ONE
00052 C            NUMBER OCCURING IN THE STRING 'STR()'
00053 C
00054 C
00055 C AUTHOR:    PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00056 C            K1/0/37
00057 C            JET EXT. 2520
00058 C
00059 C DATE:      11/07/90
00060 C
00061 C UPDATE:    11/02/92 - PE BRIDEN: BLANKS NOW ALLOWED BETWEEN SIGN AND
00062 C                                  FIRST DIGIT. LSTART VARIABLE ADDED.
00063 C                                  VARIABLE I2 REMOVED.
00064 C                                  + SOME MINOR RECODING - (IF STRING
00065 C                                  ENTERED IS BLANK IABT IS NOW SET TO 1)
00066 C
00067 C UPDATE:    16/08/93 - PE BRIDEN: CORRECTED BUG TO ALLOW BLANKS BETWEEN
00068 C                                  SIGN AND FIRST DIGIT (SEE ABOVE).
00069 C                                  1) ILAST VARIABLE ADDED.
00070 C                                  2) FORMATTED READ USED INSTEAD OF *
00071 C                                     WHEN CONVERTING IDENTIFIED INTEGER
00072 C                                     USING THE INTERNAL READ. (THIS
00073 C                                     RESTRICTS IDENTIFIED NUMBER TO BE
00074 C                                     < 100 BYTES IN LENGTH!)
00075 C                                  3) EXCLUDE TRAILING BLANKS IN THE
00076 C                                     INTERNAL READING OF THE INTEGER
00077 C                                     I.E. STR(I1:ILAST) INSTEAD OF
00078 C                                          STR(I1:ILEN)
00079 C
00080 C UPDATE:    07/03/95 - PE BRIDEN: INSTEAD OF USING FORMAT SPECIFIER I99
00081 C                                  WHEN INTERNALLY READING THE INTEGER
00082 C                                  CREATE THE APPROPRIATE SPECIFIER
00083 C                                  WITHIN CFORM5 AND USE THIS.
00084 C
00085 C VERSION  : 1.3
00086 C DATE     : 20-12-2001
00087 C MODIFIED : Martin O'Mullane
00088 C               - Removed mainframe listing information beyond column 72.
00089 C
00090 C VERSION  : 1.3
00091 C DATE     : 10-04-2007
00092 C MODIFIED : Allan Whiteford
00093 C               - Modified documentation as part of automated
00094 C                 subroutine documentation preparation.
00095 C
00096 C-----------------------------------------------------------------------
00097 C
00098 C-----------------------------------------------------------------------
00099       CHARACTER  ch0*1   , ch9*1   , blank*1   , cplus*1   , cminus*1
00100 C-----------------------------------------------------------------------
00101       parameter( ch0='0', ch9='9', blank=' ', cplus='+', cminus='-' )
00102 C-----------------------------------------------------------------------
00103       CHARACTER  str*(*)
00104 C-----------------------------------------------------------------------
00105       INTEGER    i4fctn  , iabt
00106       INTEGER    i1      , is      , ilen      , ilast     ,
00107      &           ich0    , ich9    , istr      , i
00108 C-----------------------------------------------------------------------
00109       LOGICAL    lstart  , lfound
00110 C-----------------------------------------------------------------------
00111       CHARACTER  cform5*5
00112 C-----------------------------------------------------------------------
00113       DATA       cform5 / '(I??)' /
00114 C-----------------------------------------------------------------------
00115       SAVE       cform5
00116 C-----------------------------------------------------------------------
00117 C
00118 C-----------------------------------------------------------------------
00119 C INITIALIZE VALUES
00120 C-----------------------------------------------------------------------
00121 C
00122       i4fctn = 0
00123       iabt   = 0
00124       i1     = 0
00125       is     = 0
00126       lstart = .false.
00127       lfound = .false.
00128       ich0   = ichar(ch0)
00129       ich9   = ichar(ch9)
00130       ilen   = len(str)
00131       ilast  = ilen
```

```
00132 C
00133 C-----------------------------------------------------------------------
00134 C FIND STARTING BYTE OF NUMBER
00135 C-----------------------------------------------------------------------
00136 C
00137           DO 1 i=1,ilen
00138                 IF ( str(i:i).NE.blank ) THEN
00139                    i1 = i
00140                    GOTO 2
00141                 ENDIF
00142     1     CONTINUE
00143 C
00144 C-----------------------------------------------------------------------
00145 C IDENTIFY IF NUMBER HAS A SIGN
00146 C-----------------------------------------------------------------------
00147 C
00148     2     IF (i1.EQ.0) THEN
00149              iabt = 1
00150              RETURN
00151           ENDIF
00152 C
00153        IF    ( ( str(i1:i1).EQ.cplus  )
00154       &                    .OR.
00155       &        ( str(i1:i1).EQ.cminus ) ) is=1
00156 C
00157 C-----------------------------------------------------------------------
00158 C IDENTIFY IF NUMBER IS OF A VALID FORM
00159 C-----------------------------------------------------------------------
00160 C
00161           DO 3 i=i1+is,ilen
00162
00163                 IF (lfound) THEN
00164 C
00165 C-----------------------------------------------------------------------
00166 C INPUT NO. COMPLETELY DEFINED: IDENTIFY IF EXTRA NON-BLANK BYTES EXIST
00167 C-----------------------------------------------------------------------
00168 C
00169                    IF (str(i:i).NE.blank) iabt=1
00170 C-----------------------------------------------------------------------
00171                 ELSEIF (str(i:i).EQ.blank) THEN
00172                    lfound = lstart
00173 C-----------------------------------------------------------------------
00174                 ELSE
00175                    lstart = .true.
00176                    ilast  = i
00177                    istr   = ichar(str(i:i))
00178                    IF ( (istr.LT.ich0) .OR. (istr.GT.ich9) ) iabt=1
00179                 ENDIF
00180 C
00181 C-----------------------------------------------------------------------
00182 C RETURN ERROR CODE IF ERROR FOUND
00183 C-----------------------------------------------------------------------
00184 C
00185              IF (iabt.NE.0) RETURN
00186     3     CONTINUE
00187 C
00188 C-----------------------------------------------------------------------
00189 C IDENTIFY IF VALID NUMBER FOUND (RECODED: PEB 11/02/92)
00190 C                                (RECODED: PEB 07/03/95 - ADDED CFORM5)
00191 C YES => USE INTERNAL READ TO OBTAIN THE INTEGER NUMBER
00192 C NO  => RETURN ERROR CODE IF ERROR FOUND
00193 C-----------------------------------------------------------------------
00194 C
00195           IF (lstart) THEN
00196              i      = 1 + ilast - i1
00197              i      = min0(i,99)
00198              WRITE(cform5(3:4),'(I2.2)') i
00199              READ(str(i1:ilast),cform5) i4fctn
00200           ELSE
00201              iabt=1
00202           ENDIF
00203 C
00204 C-----------------------------------------------------------------------
00205 C
00206        RETURN
00207        END
```

16.60 src/amns_driver/i4unit.f File Reference

Functions/Subroutines

- integer function i4unit (IUNIT)

16.60.1 Function/Subroutine Documentation

16.60.1.1 i4unit()

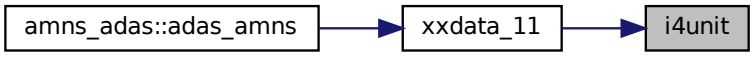
```
integer function i4unit (
            integer IUNIT )
```

Definition at line 3 of file i4unit.f.

```
00004       IMPLICIT NONE
00005 C-----------------------------------------------------------------------
00006 C
00007 C  ************** FORTRAN77 INTEGER*4 FUNCTION: I4UNIT *****************
00008 C
00009 C  PURPOSE: TO RESET OR RETURN A STORED INTEGER*4 VALUE GREATER THAN OR
00010 C           EQUAL TO ZERO.
00011 C           THIS IS USED WITHIN ADAS TO STORE THE STREAM/UNIT NUMBER
00012 C           FOR THE OUTPUT OF ERROR MESSAGES (TO THE SCREEN).
00013 C
00014 C           BY DEFAULT THE STORED VALUE WILL BE 6, AND WILL BE RETURNED
00015 C           BY THE FUNCTION IF IUNIT ON INPUT < 0.
00016 C
00017 C           TO RESET THE STORED VALUE THEN SET IUNIT TO THE REQUIRED
00018 C           POSITIVE INTEGER (INC. ZERO). THIS VALUE WILL ALSO BE
00019 C           RETURNED BY THE FUNCTION.
00020 C
00021 C                 IUNIT VALUE               RETURNED FUNCTION VALUE
00022 C                 -----------               -----------------------
00023 C                 IUNIT <  0            = CURRENT STORED INTEGER VALUE
00024 C                                         (6 BY DEFAULT).
00025 C                 IUNIT >= 0            = IUNIT , AND RESETS THE STORED
00026 C                                           VALUE TO IUNIT.
00027 C
00028 C
00029 C  CALLING PROGRAM: GENERAL USE
00030 C
00031 C  SUBROUTINE:
00032 C
00033 C  O     : (I*4)  I4UNIT   = FUNCTION NAME - (SEE ABOVE)
00034 C
00035 C  I     : (I*4)  IUNIT    = FUNCTION ARGUMENT - (SEE ABOVE)
00036 C
00037 C          (I*4)  IDEFLT   = PARAMETER = DEFAULT STORED INTEGER VALUE
00038 C
00039 C          (I*4)  ICURNT   = CURRENT STORED INTEGER VALUE
00040 C
00041 C
00042 C ROUTINES:
00043 C          ROUTINE    SOURCE    BRIEF DESCRIPTION
00044 C          ------------------------------------------------------------
00045 C
00046 C
00047 C AUTHOR:  PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00048 C          K1/0/37
00049 C          JET EXT. 5023
00050 C
00051 C DATE:    23/04/93
00052 C
00053 C UPDATE:  24/05/93 - PE BRIDEN - ALLOWED 0 TO BE A VALID STORED NUMBER
00054 C
00055 C-----------------------------------------------------------------------
00056       INTEGER    I4UNIT     , IDEFLT
00057 C-----------------------------------------------------------------------
00058       parameter( ideflt = 6 )
00059 C-----------------------------------------------------------------------
00060       INTEGER    IUNIT      , ICURNT
00061 C-----------------------------------------------------------------------
00062       SAVE       icurnt
00063 C-----------------------------------------------------------------------
00064       DATA       icurnt / ideflt /
00065 C-----------------------------------------------------------------------
00066 C
00067 C-----------------------------------------------------------------------
00068 C RETRIEVE OR RESET STORED INTEGER VALUE ACCORDINGLY
00069 C-----------------------------------------------------------------------
00070 C
00071       IF (iunit.LT.0) THEN
00072         i4unit = icurnt
00073       ELSE
00074         icurnt = iunit
00075         i4unit = iunit
00076       ENDIF
00077 C
00078 C-----------------------------------------------------------------------
```

```
00079 C
00080       RETURN
```

Here is the caller graph for this function:



16.61 i4unit.f

```
00001 CX UNIX PORT - SCCS info: Module @(#)$Header: /home/adascvs/fortran/adaslib/system/i4unit.for,v 1.1
       2004/07/06 14:08:05 whitefor Exp $ Date $Date: 2004/07/06 14:08:05 $
00002 CX
00003       FUNCTION i4unit( IUNIT )
00004       IMPLICIT NONE
00005 C-----------------------------------------------------------------------
00006 C
00007 C  ************** FORTRAN77 INTEGER*4 FUNCTION: I4UNIT *****************
00008 C
00009 C  PURPOSE: TO RESET OR RETURN A STORED INTEGER*4 VALUE GREATER THAN OR
00010 C           EQUAL TO ZERO.
00011 C           THIS IS USED WITHIN ADAS TO STORE THE STREAM/UNIT NUMBER
00012 C           FOR THE OUTPUT OF ERROR MESSAGES (TO THE SCREEN).
00013 C
00014 C           BY DEFAULT THE STORED VALUE WILL BE 6, AND WILL BE RETURNED
00015 C           BY THE FUNCTION IF IUNIT ON INPUT < 0.
00016 C
00017 C           TO RESET THE STORED VALUE THEN SET IUNIT TO THE REQUIRED
00018 C           POSITIVE INTEGER (INC. ZERO). THIS VALUE WILL ALSO BE
00019 C           RETURNED BY THE FUNCTION.
00020 C
00021 C                 IUNIT VALUE              RETURNED FUNCTION VALUE
00022 C                 -----------              -----------------------
00023 C                 IUNIT <  0            = CURRENT STORED INTEGER VALUE
00024 C                                         (6 BY DEFAULT).
00025 C                 IUNIT >= 0            = IUNIT , AND RESETS THE STORED
00026 C                                                 VALUE TO IUNIT.
00027 C
00028 C
00029 C  CALLING PROGRAM: GENERAL USE
00030 C
00031 C  SUBROUTINE:
00032 C
00033 C  O     : (I*4)  I4UNIT   = FUNCTION NAME - (SEE ABOVE)
00034 C
00035 C  I     : (I*4)  IUNIT    = FUNCTION ARGUMENT - (SEE ABOVE)
00036 C
00037 C          (I*4)  IDEFLT   = PARAMETER = DEFAULT STORED INTEGER VALUE
00038 C
00039 C          (I*4)  ICURNT   = CURRENT STORED INTEGER VALUE
00040 C
00041 C
00042 C ROUTINES:
00043 C          ROUTINE    SOURCE    BRIEF DESCRIPTION
00044 C          ------------------------------------------------------------
00045 C
00046 C
00047 C AUTHOR:  PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00048 C          K1/0/37
00049 C          JET EXT. 5023
00050 C
00051 C DATE:    23/04/93
00052 C
00053 C UPDATE:  24/05/93 - PE BRIDEN - ALLOWED 0 TO BE A VALID STORED NUMBER
00054 C
00055 C-----------------------------------------------------------------------
00056       INTEGER    i4unit     , ideflt
00057 C-----------------------------------------------------------------------
00058       parameter( ideflt = 6 )
00059 C-----------------------------------------------------------------------
00060       INTEGER    iunit      , icurnt
00061 C-----------------------------------------------------------------------
00062       SAVE       icurnt
00063 C-----------------------------------------------------------------------
```

```
00064       DATA       icurnt / ideflt /
00065 C------------------------------------------------------------------------
00066 C
00067 C------------------------------------------------------------------------
00068 C RETRIEVE OR RESET STORED INTEGER VALUE ACCORDINGLY
00069 C------------------------------------------------------------------------
00070 C
00071       IF (iunit.LT.0) THEN
00072         i4unit = icurnt
00073       ELSE
00074         icurnt = iunit
00075         i4unit = iunit
00076       ENDIF
00077 C
00078 C------------------------------------------------------------------------
00079 C
00080       RETURN
00081       END
```

16.62 src/amns_driver/quadpack.f90 File Reference

Modules

- module quadpack

Functions/Subroutines

- subroutine quadpack::aaaa
- subroutine quadpack::qag (f, a, b, epsabs, epsrel, key, result, abserr, neval, ier)
- subroutine quadpack::qage (f, a, b, epsabs, epsrel, key, limit, result, abserr, neval, ier, alist, blist, rlist, elist, iord, last)
- subroutine quadpack::qagi (f, bound, inf, epsabs, epsrel, result, abserr, neval, ier)
- subroutine quadpack::qagp (f, a, b, npts2, points, epsabs, epsrel, result, abserr, neval, ier)
- subroutine quadpack::qags (f, a, b, epsabs, epsrel, result, abserr, neval, ier)
- subroutine quadpack::qawc (f, a, b, c, epsabs, epsrel, result, abserr, neval, ier)
- subroutine quadpack::qawce (f, a, b, c, epsabs, epsrel, limit, result, abserr, neval, ier, alist, blist, rlist, elist, iord, last)
- subroutine quadpack::qawf (f, a, omega, integr, epsabs, result, abserr, neval, ier)
- subroutine quadpack::qawfe (f, a, omega, integr, epsabs, limlst, limit, maxp1, result, abserr, neval, ier, rslst, erlst, ierlst, lst, alist, blist, rlist, elist, iord, nnlog, chebmo)
- subroutine quadpack::qawo (f, a, b, omega, integr, epsabs, epsrel, result, abserr, neval, ier)
- subroutine quadpack::qaws (f, a, b, alfa, beta, integr, epsabs, epsrel, result, abserr, neval, ier)
- subroutine quadpack::qawse (f, a, b, alfa, beta, integr, epsabs, epsrel, limit, result, abserr, neval, ier, alist, blist, rlist, elist, iord, last)
- subroutine quadpack::qc25c (f, a, b, c, result, abserr, krul, neval)
- subroutine quadpack::qc25o (f, a, b, omega, integr, nrmom, maxp1, ksave, result, abserr, neval, resabs, resasc, momcom, chebmo)
- subroutine quadpack::qc25s (f, a, b, bl, br, alfa, beta, ri, rj, rg, rh, result, abserr, resasc, integr, neval)
- subroutine quadpack::qcheb (x, fval, cheb12, cheb24)
- subroutine quadpack::qextr (n, epstab, result, abserr, res3la, nres)
- subroutine quadpack::qfour (f, a, b, omega, integr, epsabs, epsrel, limit, icall, maxp1, result, abserr, neval, ier, alist, blist, rlist, elist, iord, nnlog, momcom, chebmo)
- subroutine quadpack::qk15 (f, a, b, result, abserr, resabs, resasc)
- subroutine quadpack::qk15i (f, boun, inf, a, b, result, abserr, resabs, resasc)
- subroutine quadpack::qk15w (f, w, p1, p2, p3, p4, kp, a, b, result, abserr, resabs, resasc)
- subroutine quadpack::qk21 (f, a, b, result, abserr, resabs, resasc)
- subroutine quadpack::qk31 (f, a, b, result, abserr, resabs, resasc)
- subroutine quadpack::qk41 (f, a, b, result, abserr, resabs, resasc)
- subroutine quadpack::qk51 (f, a, b, result, abserr, resabs, resasc)
- subroutine quadpack::qk61 (f, a, b, result, abserr, resabs, resasc)
- subroutine quadpack::qmomo (alfa, beta, ri, rj, rg, rh, integr)

- subroutine quadpack::qng (f, a, b, epsabs, epsrel, result, abserr, neval, ier)
- subroutine quadpack::qsort (limit, last, maxerr, ermax, elist, iord, nrmax)
- real(kind=params_wp) function quadpack::qwgtc (x, c, p2, p3, p4, kp)
- real(kind=params_wp) function quadpack::qwgto (x, omega, p2, p3, p4, integr)
- real(kind=params_wp) function quadpack::qwgts (x, a, b, alfa, beta, integr)
- subroutine quadpack::timestamp ()

16.63 quadpack.f90

```
00001 module quadpack
00002   implicit none
00003   public  !:: qags, qwgtc, qwgto, qwgts
00004   integer, parameter,private :: params_wp =kind(1.0d0)
00005 contains
00006 ! This file is from: http://people.sc.fsu.edu/~jburkardt/f_src/quadpack/quadpack.html
00007 ! 13th May 2011
00008 !
00009 ! The licence question:
00010 !
00011 ! Dear Dr. Burkardt,
00012 !
00013 !I'd like to use some existing adaptive quadrature code in the plasma
00014 !physics code I'm working on. I found quadpack, but at least the netlib
00015 !version is in f77, and I'd rather use a f90 version. I found your
00016 !wonderful fortran source code. :-) I tried a bit to find out if you have
00017 !released the stuff to the public domain or not. I know quadpack is in
00018 !the public domain, but what about the version you wrote?
00019 !
00020 !So, may I please use the code found at
00021 !
00022 !http://people.sc.fsu.edu/~jburkardt/f_src/quadpack/quadpack.html
00023 !
00024 !within our code. I will certainly write to the source code files where
00025 !the programs came from, and we can also cite you on the first article
00026 !using your routine.
00027 !
00028 !Thank you very much and best regards,
00029 !--
00030 ! - Simppa Akaslompolo -
00031 ! http://tfy.tkk.fi/personnel/official.php?id=428
00032 !
00033 !----------------------------------------------------------------------
00034 !I don't impose any licensing restrictions on the F90 version of
00035 !QUADPACK.  You are free to use it in any way you like.
00036 !
00037 !John Burkardt
00038 !
00039 !----------------------------------------------------------------------
00040
00041 !   1. Robert Piessens, Elise deDoncker-Kapenga, Christian Ueberhuber, David Kahaner,
00042 !      QUADPACK: A Subroutine Package for Automatic Integration,
00043 !      Springer, 1983,
00044 !      ISBN: 3540125531,
00045 !      LC: QA299.3.Q36.
00046
00047
00048
00049
00050 subroutine aaaa
00051
00052 !*******************************************************************************80
00053 !
00054 !! AAAA is a dummy subroutine with QUADPACK documentation in its comments.
00055 !
00056 ! 1. introduction
00057 !
00058 !    quadpack is a fortran subroutine package for the numerical
00059 !    computation of definite one-dimensional integrals. it originated
00060 !    from a joint project of r. piessens and e. de doncker (appl.
00061 !    math. and progr. div.- k.u.leuven, belgium), c. ueberhuber (inst.
00062 !    fuer math.- techn.u.wien, austria), and d. kahaner (nation. bur.
00063 !    of standards- washington d.c., u.s.a.).
00064 !
00065 ! 2. survey
00066 !
00067 !    - qags : is an integrator based on globally adaptive interval
00068 !             subdivision in connection with extrapolation (de doncker,
00069 !             1978) by the epsilon algorithm (wynn, 1956).
00070 !
00071 !    - qagp : serves the same purposes as qags, but also allows
00072 !             for eventual user-supplied information, i.e. the
00073 !             abscissae of internal singularities, discontinuities
```

```
00074 !             and other difficulties of the integrand function.
00075 !             the algorithm is a modification of that in qags.
00076 !
00077 !    - qagi : handles integration over infinite intervals. the
00078 !             infinite range is mapped onto a finite interval and
00079 !             then the same strategy as in qags is applied.
00080 !
00081 !    - qawo : is a routine for the integration of cos(omega*x)*f(x)
00082 !             or sin(omega*x)*f(x) over a finite interval (a,b).
00083 !             omega is is specified by the user
00084 !             the rule evaluation component is based on the
00085 !             modified clenshaw-curtis technique.
00086 !             an adaptive subdivision scheme is used connected with
00087 !             an extrapolation procedure, which is a modification
00088 !             of that in qags and provides the possibility to deal
00089 !             even with singularities in f.
00090 !
00091 !    - qawf : calculates the fourier cosine or fourier sine
00092 !             transform of f(x), for user-supplied interval (a,
00093 !             infinity), omega, and f. the procedure of qawo is
00094 !             used on successive finite intervals, and convergence
00095 !             acceleration by means of the epsilon algorithm (wynn,
00096 !             1956) is applied to the series of the integral
00097 !             contributions.
00098 !
00099 !    - qaws : integrates w(x)*f(x) over (a,b) with a < b finite,
00100 !             and   w(x) = ((x-a)**alfa)*((b-x)**beta)*v(x)
00101 !             where v(x) = 1 or log(x-a) or log(b-x)
00102 !                            or log(x-a)*log(b-x)
00103 !             and   -1 < alfa, -1 < beta.
00104 !             the user specifies a, b, alfa, beta and the type of
00105 !             the function v.
00106 !             a globally adaptive subdivision strategy is applied,
00107 !             with modified clenshaw-curtis integration on the
00108 !             subintervals which contain a or b.
00109 !
00110 !    - qawc : computes the cauchy principal value of f(x)/(x-c)
00111 !             over a finite interval (a,b) and for
00112 !             user-determined c.
00113 !             the strategy is globally adaptive, and modified
00114 !             clenshaw-curtis integration is used on the subranges
00115 !             which contain the point x = c.
00116 !
00117 !  each of the routines above also has a "more detailed" version
00118 !    with a name ending in e, as qage.  these provide more
00119 !    information and control than the easier versions.
00120 !
00121 !
00122 !   the preceeding routines are all automatic.  that is, the user
00123 !      inputs his problem and an error tolerance.  the routine
00124 !      attempts to perform the integration to within the requested
00125 !      absolute or relative error.
00126 !   there are, in addition, a number of non-automatic integrators.
00127 !      these are most useful when the problem is such that the
00128 !      user knows that a fixed rule will provide the accuracy
00129 !      required.  typically they return an error estimate but make
00130 !      no attempt to satisfy any particular input error request.
00131 !
00132 !      qk15
00133 !      qk21
00134 !      qk31
00135 !      qk41
00136 !      qk51
00137 !      qk61
00138 !           estimate the integral on [a,b] using 15, 21,..., 61
00139 !           point rule and return an error estimate.
00140 !      qk15i 15 point rule for (semi)infinite interval.
00141 !      qk15w 15 point rule for special singular weight functions.
00142 !      qc25c 25 point rule for cauchy principal values
00143 !      qc25o 25 point rule for sin/cos integrand.
00144 !      qmomo integrates k-th degree chebychev polynomial times
00145 !            function with various explicit singularities.
00146 !
00147 ! 3. guidelines for the use of quadpack
00148 !
00149 !    here it is not our purpose to investigate the question when
00150 !    automatic quadrature should be used. we shall rather attempt
00151 !    to help the user who already made the decision to use quadpack,
00152 !    with selecting an appropriate routine or a combination of
00153 !    several routines for handling his problem.
00154 !
00155 !    for both quadrature over finite and over infinite intervals,
00156 !    one of the first questions to be answered by the user is
00157 !    related to the amount of computer time he wants to spend,
00158 !    versus his -own- time which would be needed, for example, for
00159 !    manual subdivision of the interval or other analytic
00160 !    manipulations.
```

```
00161 !
00162 !    (1) the user may not care about computer time, or not be
00163 !        willing to do any analysis of the problem. especially when
00164 !        only one or a few integrals must be calculated, this attitude
00165 !        can be perfectly reasonable. in this case it is clear that
00166 !        either the most sophisticated of the routines for finite
00167 !        intervals, qags, must be used, or its analogue for infinite
00168 !        intervals, qagi. these routines are able to cope with
00169 !        rather difficult, even with improper integrals.
00170 !        this way of proceeding may be expensive. but the integrator
00171 !        is supposed to give you an answer in return, with additional
00172 !        information in the case of a failure, through its error
00173 !        estimate and flag. yet it must be stressed that the programs
00174 !        cannot be totally reliable.
00175 !
00176 !    (2) the user may want to examine the integrand function.
00177 !        if bad local difficulties occur, such as a discontinuity, a
00178 !        singularity, derivative singularity or high peak at one or
00179 !        more points within the interval, the first advice is to
00180 !        split up the interval at these points. the integrand must
00181 !        then be examined over each of the subintervals separately,
00182 !        so that a suitable integrator can be selected for each of
00183 !        them. if this yields problems involving relative accuracies
00184 !        to be imposed on -finite- subintervals, one can make use of
00185 !        qagp, which must be provided with the positions of the local
00186 !        difficulties. however, if strong singularities are present
00187 !        and a high accuracy is requested, application of qags on the
00188 !        subintervals may yield a better result.
00189 !
00190 !        for quadrature over finite intervals we thus dispose of qags
00191 !        and
00192 !        - qng for well-behaved integrands,
00193 !        - qag for functions with an oscillating behavior of a non
00194 !          specific type,
00195 !        - qawo for functions, eventually singular, containing a
00196 !          factor cos(omega*x) or sin(omega*x) where omega is known,
00197 !        - qaws for integrands with algebraico-logarithmic end point
00198 !          singularities of known type,
00199 !        - qawc for cauchy principal values.
00200 !
00201 !        remark
00202 !
00203 !        on return, the work arrays in the argument lists of the
00204 !        adaptive integrators contain information about the interval
00205 !        subdivision process and hence about the integrand behavior:
00206 !        the end points of the subintervals, the local integral
00207 !        contributions and error estimates, and eventually other
00208 !        characteristics. for this reason, and because of its simple
00209 !        globally adaptive nature, the routine qag in particular is
00210 !        well-suited for integrand examination. difficult spots can
00211 !        be located by investigating the error estimates on the
00212 !        subintervals.
00213 !
00214 !        for infinite intervals we provide only one general-purpose
00215 !        routine, qagi. it is based on the qags algorithm applied
00216 !        after a transformation of the original interval into (0,1).
00217 !        yet it may eventuate that another type of transformation is
00218 !        more appropriate, or one might prefer to break up the
00219 !        original interval and use qagi only on the infinite part
00220 !        and so on. these kinds of actions suggest a combined use of
00221 !        different quadpack integrators. note that, when the only
00222 !        difficulty is an integrand singularity at the finite
00223 !        integration limit, it will in general not be necessary to
00224 !        break up the interval, as qagi deals with several types of
00225 !        singularity at the boundary point of the integration range.
00226 !        it also handles slowly convergent improper integrals, on
00227 !        the condition that the integrand does not oscillate over
00228 !        the entire infinite interval. if it does we would advise
00229 !        to sum succeeding positive and negative contributions to
00230 !        the integral -e.g. integrate between the zeros- with one
00231 !        or more of the finite-range integrators, and apply
00232 !        convergence acceleration eventually by means of quadpack
00233 !        subroutine qelg which implements the epsilon algorithm.
00234 !        such quadrature problems include the fourier transform as
00235 !        a special case. yet for the latter we have an automatic
00236 !        integrator available, qawf.
00237 !
00238   return
00239 end subroutine aaaa
00240 subroutine qag ( f, a, b, epsabs, epsrel, key, result, abserr, neval, ier )
00241
00242 !*******************************************************************************80
00243 !
00244 !! QAG approximates an integral over a finite interval.
00245 !
00246 !  Discussion:
00247 !
```

```
00248 !    The routine calculates an approximation RESULT to a definite integral
00249 !      I = integral of F over (A,B),
00250 !    hopefully satisfying
00251 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
00252 !
00253 !    QAG is a simple globally adaptive integrator using the strategy of
00254 !    Aind (Piessens, 1973).  It is possible to choose between 6 pairs of
00255 !    Gauss-Kronrod quadrature formulae for the rule evaluation component.
00256 !    The pairs of high degree of precision are suitable for handling
00257 !    integration difficulties due to a strongly oscillating integrand.
00258 !
00259 !  Author:
00260 !
00261 !    Robert Piessens, Elise de Doncker-Kapenger,
00262 !    Christian Ueberhuber, David Kahaner
00263 !
00264 !  Reference:
00265 !
00266 !    Robert Piessens, Elise de Doncker-Kapenger,
00267 !    Christian Ueberhuber, David Kahaner,
00268 !    QUADPACK, a Subroutine Package for Automatic Integration,
00269 !    Springer Verlag, 1983
00270 !
00271 !  Parameters:
00272 !
00273 !    Input, external real F, the name of the function routine, of the form
00274 !      function f ( x )
00275 !      real f
00276 !      real x
00277 !    which evaluates the integrand function.
00278 !
00279 !    Input, real A, B, the limits of integration.
00280 !
00281 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
00282 !
00283 !    Input, integer KEY, chooses the order of the local integration rule:
00284 !    1,  7 Gauss points, 15 Gauss-Kronrod points,
00285 !    2, 10 Gauss points, 21 Gauss-Kronrod points,
00286 !    3, 15 Gauss points, 31 Gauss-Kronrod points,
00287 !    4, 20 Gauss points, 41 Gauss-Kronrod points,
00288 !    5, 25 Gauss points, 51 Gauss-Kronrod points,
00289 !    6, 30 Gauss points, 61 Gauss-Kronrod points.
00290 !
00291 !    Output, real RESULT, the estimated value of the integral.
00292 !
00293 !    Output, real ABSERR, an estimate of || I - RESULT ||.
00294 !
00295 !    Output, integer NEVAL, the number of times the integral was evaluated.
00296 !
00297 !    Output, integer IER, return code.
00298 !    0, normal and reliable termination of the routine.  It is assumed that the
00299 !      requested accuracy has been achieved.
00300 !    1, maximum number of subdivisions allowed has been achieved.  One can
00301 !      allow more subdivisions by increasing the value of LIMIT in QAG.
00302 !      However, if this yields no improvement it is advised to analyze the
00303 !      integrand to determine the integration difficulties.  If the position
00304 !      of a local difficulty can be determined, such as a singularity or
00305 !      discontinuity within the interval) one will probably gain from
00306 !      splitting up the interval at this point and calling the integrator
00307 !      on the subranges.  If possible, an appropriate special-purpose
00308 !      integrator should be used which is designed for handling the type
00309 !      of difficulty involved.
00310 !    2, the occurrence of roundoff error is detected, which prevents the
00311 !      requested tolerance from being achieved.
00312 !    3, extremely bad integrand behavior occurs at some points of the
00313 !      integration interval.
00314 !    6, the input is invalid, because EPSABS < 0 and EPSREL < 0.
00315 !
00316 !  Local parameters:
00317 !
00318 !    LIMIT is the maximum number of subintervals allowed in
00319 !    the subdivision process of QAGE.
00320 !
00321   implicit none
00322
00323   integer, parameter :: limit = 500
00324
00325   real(kind=params_wp) a
00326   real(kind=params_wp) abserr
00327   real(kind=params_wp) alist(limit)
00328   real(kind=params_wp) b
00329   real(kind=params_wp) blist(limit)
00330   real(kind=params_wp) elist(limit)
00331   real(kind=params_wp) epsabs
00332   real(kind=params_wp) epsrel
00333   real(kind=params_wp), external :: f
00334   integer ier
```

```
00335   integer iord(limit)
00336   integer key
00337   integer last
00338   integer neval
00339   real(kind=params_wp) result
00340   real(kind=params_wp) rlist(limit)
00341
00342   call qage ( f, a, b, epsabs, epsrel, key, limit, result, abserr, neval, &
00343     ier, alist, blist, rlist, elist, iord, last )
00344
00345   return
00346 end subroutine qag
00347 subroutine qage ( f, a, b, epsabs, epsrel, key, limit, result, abserr, neval, &
00348   ier, alist, blist, rlist, elist, iord, last )
00349
00350 !*******************************************************************************80
00351 !
00352 !! QAGE estimates a definite integral.
00353 !
00354 !  Discussion:
00355 !
00356 !    The routine calculates an approximation RESULT to a definite integral
00357 !      I = integral of F over (A,B),
00358 !    hopefully satisfying
00359 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
00360 !
00361 !  Author:
00362 !
00363 !    Robert Piessens, Elise de Doncker-Kapenger,
00364 !    Christian Ueberhuber, David Kahaner
00365 !
00366 !  Reference:
00367 !
00368 !    Robert Piessens, Elise de Doncker-Kapenger,
00369 !    Christian Ueberhuber, David Kahaner,
00370 !    QUADPACK, a Subroutine Package for Automatic Integration,
00371 !    Springer Verlag, 1983
00372 !
00373 !  Parameters:
00374 !
00375 !    Input, external real F, the name of the function routine, of the form
00376 !      function f ( x )
00377 !      real f
00378 !      real x
00379 !    which evaluates the integrand function.
00380 !
00381 !    Input, real A, B, the limits of integration.
00382 !
00383 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
00384 !
00385 !    Input, integer KEY, chooses the order of the local integration rule:
00386 !    1,  7 Gauss points, 15 Gauss-Kronrod points,
00387 !    2, 10 Gauss points, 21 Gauss-Kronrod points,
00388 !    3, 15 Gauss points, 31 Gauss-Kronrod points,
00389 !    4, 20 Gauss points, 41 Gauss-Kronrod points,
00390 !    5, 25 Gauss points, 51 Gauss-Kronrod points,
00391 !    6, 30 Gauss points, 61 Gauss-Kronrod points.
00392 !
00393 !    Input, integer LIMIT, the maximum number of subintervals that
00394 !    can be used.
00395 !
00396 !    Output, real RESULT, the estimated value of the integral.
00397 !
00398 !    Output, real ABSERR, an estimate of || I - RESULT ||.
00399 !
00400 !    Output, integer NEVAL, the number of times the integral was evaluated.
00401 !
00402 !    Output, integer IER, return code.
00403 !    0, normal and reliable termination of the routine.  It is assumed that the
00404 !      requested accuracy has been achieved.
00405 !    1, maximum number of subdivisions allowed has been achieved.  One can
00406 !      allow more subdivisions by increasing the value of LIMIT in QAG.
00407 !      However, if this yields no improvement it is advised to analyze the
00408 !      integrand to determine the integration difficulties.  If the position
00409 !      of a local difficulty can be determined, such as a singularity or
00410 !      discontinuity within the interval) one will probably gain from
00411 !      splitting up the interval at this point and calling the integrator
00412 !      on the subranges.  If possible, an appropriate special-purpose
00413 !      integrator should be used which is designed for handling the type
00414 !      of difficulty involved.
00415 !    2, the occurrence of roundoff error is detected, which prevents the
00416 !      requested tolerance from being achieved.
00417 !    3, extremely bad integrand behavior occurs at some points of the
00418 !      integration interval.
00419 !    6, the input is invalid, because EPSABS < 0 and EPSREL < 0.
00420 !
00421 !    Workspace, real ALIST(LIMIT), BLIST(LIMIT), contains in entries 1
```

```
00422 !    through LAST the left and right ends of the partition subintervals.
00423 !
00424 !    Workspace, real RLIST(LIMIT), contains in entries 1 through LAST
00425 !    the integral approximations on the subintervals.
00426 !
00427 !    Workspace, real ELIST(LIMIT), contains in entries 1 through LAST
00428 !    the absolute error estimates on the subintervals.
00429 !
00430 !    Output, integer IORD(LIMIT), the first K elements of which are pointers
00431 !    to the error estimates over the subintervals, such that
00432 !    elist(iord(1)), ..., elist(iord(k)) form a decreasing sequence, with
00433 !    k = last if last <= (limit/2+2), and k = limit+1-last otherwise.
00434 !
00435 !    Output, integer LAST, the number of subintervals actually produced
00436 !    in the subdivision process.
00437 !
00438 !  Local parameters:
00439 !
00440 !    alist     - list of left end points of all subintervals
00441 !                  considered up to now
00442 !    blist     - list of right end points of all subintervals
00443 !                  considered up to now
00444 !    elist(i)  - error estimate applying to rlist(i)
00445 !    maxerr    - pointer to the interval with largest error estimate
00446 !    errmax    - elist(maxerr)
00447 !    area      - sum of the integrals over the subintervals
00448 !    errsum    - sum of the errors over the subintervals
00449 !    errbnd    - requested accuracy max(epsabs,epsrel*abs(result))
00450 !    *****1    - variable for the left subinterval
00451 !    *****2    - variable for the right subinterval
00452 !    last      - index for subdivision
00453 !
00454   implicit none
00455
00456   integer limit
00457
00458   real(kind=params_wp) a
00459   real(kind=params_wp) abserr
00460   real(kind=params_wp) alist(limit)
00461   real(kind=params_wp) area
00462   real(kind=params_wp) area1
00463   real(kind=params_wp) area12
00464   real(kind=params_wp) area2
00465   real(kind=params_wp) a1
00466   real(kind=params_wp) a2
00467   real(kind=params_wp) b
00468   real(kind=params_wp) blist(limit)
00469   real(kind=params_wp) b1
00470   real(kind=params_wp) b2
00471   real(kind=params_wp) c
00472   real(kind=params_wp) defabs
00473   real(kind=params_wp) defab1
00474   real(kind=params_wp) defab2
00475   real(kind=params_wp) elist(limit)
00476   real(kind=params_wp) epsabs
00477   real(kind=params_wp) epsrel
00478   real(kind=params_wp) errbnd
00479   real(kind=params_wp) errmax
00480   real(kind=params_wp) error1
00481   real(kind=params_wp) error2
00482   real(kind=params_wp) erro12
00483   real(kind=params_wp) errsum
00484   real(kind=params_wp), external :: f
00485   integer ier
00486   integer iord(limit)
00487   integer iroff1
00488   integer iroff2
00489   integer key
00490   integer keyf
00491   integer last
00492   integer maxerr
00493   integer neval
00494   integer nrmax
00495   real(kind=params_wp) resabs
00496   real(kind=params_wp) result
00497   real(kind=params_wp) rlist(limit)
00498 !
00499 !  Test on validity of parameters.
00500 !
00501   ier = 0
00502   neval = 0
00503   last = 0
00504   result = 0.0e+00
00505   abserr = 0.0e+00
00506   alist(1) = a
00507   blist(1) = b
00508   rlist(1) = 0.0e+00
```

```
00509   elist(1) = 0.0e+00
00510   iord(1) = 0
00511
00512   if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
00513     ier = 6
00514     return
00515   end if
00516 !
00517 !  First approximation to the integral.
00518 !
00519   keyf = key
00520   keyf = max( keyf, 1 )
00521   keyf = min( keyf, 6 )
00522
00523   c = keyf
00524   neval = 0
00525
00526   if ( keyf == 1 ) then
00527     call qk15 ( f, a, b, result, abserr, defabs, resabs )
00528   else if ( keyf == 2 ) then
00529     call qk21 ( f, a, b, result, abserr, defabs, resabs )
00530   else if ( keyf == 3 ) then
00531     call qk31 ( f, a, b, result, abserr, defabs, resabs )
00532   else if ( keyf == 4 ) then
00533     call qk41 ( f, a, b, result, abserr, defabs, resabs )
00534   else if ( keyf == 5 ) then
00535     call qk51 ( f, a, b, result, abserr, defabs, resabs )
00536   else if ( keyf == 6 ) then
00537     call qk61 ( f, a, b, result, abserr, defabs, resabs )
00538   end if
00539
00540   last = 1
00541   rlist(1) = result
00542   elist(1) = abserr
00543   iord(1) = 1
00544 !
00545 !  Test on accuracy.
00546 !
00547   errbnd = max( epsabs, epsrel * abs( result ) )
00548
00549   if ( abserr <= 5.0e+01 * epsilon( defabs ) * defabs .and. &
00550     errbnd < abserr ) then
00551     ier = 2
00552   end if
00553
00554   if ( limit == 1 ) then
00555     ier = 1
00556   end if
00557
00558   if ( ier /= 0 .or. &
00559     ( abserr <= errbnd .and. abserr /= resabs ) .or. &
00560     abserr == 0.0e+00 ) then
00561
00562     if ( keyf /= 1 ) then
00563       neval = (10*keyf+1) * (2*neval+1)
00564     else
00565       neval = 30 * neval + 15
00566     end if
00567
00568     return
00569
00570   end if
00571 !
00572 !  Initialization.
00573 !
00574   errmax = abserr
00575   maxerr = 1
00576   area = result
00577   errsum = abserr
00578   nrmax = 1
00579   iroff1 = 0
00580   iroff2 = 0
00581
00582   do last = 2, limit
00583 !
00584 !  Bisect the subinterval with the largest error estimate.
00585 !
00586     a1 = alist(maxerr)
00587     b1 = 0.5e+00 * ( alist(maxerr) + blist(maxerr) )
00588     a2 = b1
00589     b2 = blist(maxerr)
00590
00591     if ( keyf == 1 ) then
00592       call qk15 ( f, a1, b1, area1, error1, resabs, defab1 )
00593     else if ( keyf == 2 ) then
00594       call qk21 ( f, a1, b1, area1, error1, resabs, defab1 )
00595     else if ( keyf == 3 ) then
```

```
00596       call qk31 ( f, a1, b1, area1, error1, resabs, defab1 )
00597     else if ( keyf == 4 ) then
00598       call qk41 ( f, a1, b1, area1, error1, resabs, defab1)
00599     else if ( keyf == 5 ) then
00600       call qk51 ( f, a1, b1, area1, error1, resabs, defab1 )
00601     else if ( keyf == 6 ) then
00602       call qk61 ( f, a1, b1, area1, error1, resabs, defab1 )
00603     end if
00604
00605     if ( keyf == 1 ) then
00606       call qk15 ( f, a2, b2, area2, error2, resabs, defab2 )
00607     else if ( keyf == 2 ) then
00608       call qk21 ( f, a2, b2, area2, error2, resabs, defab2 )
00609     else if ( keyf == 3 ) then
00610       call qk31 ( f, a2, b2, area2, error2, resabs, defab2 )
00611     else if ( keyf == 4 ) then
00612       call qk41 ( f, a2, b2, area2, error2, resabs, defab2 )
00613     else if ( keyf == 5 ) then
00614       call qk51 ( f, a2, b2, area2, error2, resabs, defab2 )
00615     else if ( keyf == 6 ) then
00616       call qk61 ( f, a2, b2, area2, error2, resabs, defab2 )
00617     end if
00618 !
00619 !  Improve previous approximations to integral and error and
00620 !  test for accuracy.
00621 !
00622     neval = neval + 1
00623     area12 = area1 + area2
00624     erro12 = error1 + error2
00625     errsum = errsum + erro12 - errmax
00626     area = area + area12 - rlist(maxerr)
00627
00628     if ( defab1 /= error1 .and. defab2 /= error2 ) then
00629
00630       if ( abs( rlist(maxerr) - area12 ) <= 1.0e-05 * abs( area12 ) &
00631         .and. 9.9e-01 * errmax <= erro12 ) then
00632         iroff1 = iroff1 + 1
00633       end if
00634
00635       if ( 10 < last .and. errmax < erro12 ) then
00636         iroff2 = iroff2 + 1
00637       end if
00638
00639     end if
00640
00641     rlist(maxerr) = area1
00642     rlist(last) = area2
00643     errbnd = max( epsabs, epsrel * abs( area ) )
00644 !
00645 !  Test for roundoff error and eventually set error flag.
00646 !
00647     if ( errbnd < errsum ) then
00648
00649       if ( 6 <= iroff1 .or. 20 <= iroff2 ) then
00650         ier = 2
00651       end if
00652 !
00653 !  Set error flag in the case that the number of subintervals
00654 !  equals limit.
00655 !
00656       if ( last == limit ) then
00657         ier = 1
00658       end if
00659 !
00660 !  Set error flag in the case of bad integrand behavior
00661 !  at a point of the integration range.
00662 !
00663       if ( max( abs( a1 ), abs( b2 ) ) <= ( 1.0e+00_params_wp + c * 1.0e+03 * &
00664         epsilon( a1 ) ) * ( abs( a2 ) + 1.0e+04 * tiny( a2 ) ) ) then
00665         ier = 3
00666       end if
00667
00668     end if
00669 !
00670 !  Append the newly-created intervals to the list.
00671 !
00672     if ( error2 <= error1 ) then
00673       alist(last) = a2
00674       blist(maxerr) = b1
00675       blist(last) = b2
00676       elist(maxerr) = error1
00677       elist(last) = error2
00678     else
00679       alist(maxerr) = a2
00680       alist(last) = a1
00681       blist(last) = b1
00682       rlist(maxerr) = area2
```

```
00683       rlist(last) = area1
00684       elist(maxerr) = error2
00685       elist(last) = error1
00686     end if
00687 !
00688 !  Call QSORT to maintain the descending ordering
00689 !  in the list of error estimates and select the subinterval
00690 !  with the largest error estimate (to be bisected next).
00691 !
00692     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
00693
00694     if ( ier /= 0 .or. errsum <= errbnd ) then
00695       exit
00696     end if
00697
00698   end do
00699 !
00700 !  Compute final result.
00701 !
00702   result = sum( rlist(1:last) )
00703
00704   abserr = errsum
00705
00706   if ( keyf /= 1 ) then
00707     neval = ( 10 * keyf + 1 ) * ( 2 * neval + 1 )
00708   else
00709     neval = 30 * neval + 15
00710   end if
00711
00712   return
00713 end subroutine qage
00714 subroutine qagi ( f, bound, inf, epsabs, epsrel, result, abserr, neval, ier )
00715
00716 !*******************************************************************************80
00717 !
00718 !! QAGI estimates an integral over a semi-infinite or infinite interval.
00719 !
00720 !  Discussion:
00721 !
00722 !    The routine calculates an approximation RESULT to a definite integral
00723 !      I = integral of F over (A, +Infinity),
00724 !    or
00725 !      I = integral of F over (-Infinity,A)
00726 !    or
00727 !      I = integral of F over (-Infinity,+Infinity),
00728 !    hopefully satisfying
00729 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
00730 !
00731 !  Author:
00732 !
00733 !    Robert Piessens, Elise de Doncker-Kapenger,
00734 !    Christian Ueberhuber, David Kahaner
00735 !
00736 !  Reference:
00737 !
00738 !    Robert Piessens, Elise de Doncker-Kapenger,
00739 !    Christian Ueberhuber, David Kahaner,
00740 !    QUADPACK, a Subroutine Package for Automatic Integration,
00741 !    Springer Verlag, 1983
00742 !
00743 !  Parameters:
00744 !
00745 !    Input, external real F, the name of the function routine, of the form
00746 !      function f ( x )
00747 !      real f
00748 !      real x
00749 !    which evaluates the integrand function.
00750 !
00751 !    Input, real BOUND, the value of the finite endpoint of the integration
00752 !    range, if any, that is, if INF is 1 or -1.
00753 !
00754 !    Input, integer INF, indicates the type of integration range.
00755 !    1:  (  BOUND,    +Infinity),
00756 !    -1: ( -Infinity,  BOUND),
00757 !    2:  ( -Infinity, +Infinity).
00758 !
00759 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
00760 !
00761 !    Output, real RESULT, the estimated value of the integral.
00762 !
00763 !    Output, real ABSERR, an estimate of || I - RESULT ||.
00764 !
00765 !    Output, integer NEVAL, the number of times the integral was evaluated.
00766 !
00767 !    Output, integer IER, error indicator.
00768 !    0, normal and reliable termination of the routine.  It is assumed that
00769 !      the requested accuracy has been achieved.
```

```
00770 !    > 0,  abnormal termination of the routine.  The estimates for result
00771 !      and error are less reliable.  It is assumed that the requested
00772 !      accuracy has not been achieved.
00773 !    1, maximum number of subdivisions allowed has been achieved.  One can
00774 !      allow more subdivisions by increasing the data value of LIMIT in QAGI
00775 !      (and taking the according dimension adjustments into account).
00776 !      However, if this yields no improvement it is advised to analyze the
00777 !      integrand in order to determine the integration difficulties.  If the
00778 !      position of a local difficulty can be determined (e.g. singularity,
00779 !      discontinuity within the interval) one will probably gain from
00780 !      splitting up the interval at this point and calling the integrator
00781 !      on the subranges.  If possible, an appropriate special-purpose
00782 !      integrator should be used, which is designed for handling the type
00783 !      of difficulty involved.
00784 !    2, the occurrence of roundoff error is detected, which prevents the
00785 !      requested tolerance from being achieved.  The error may be
00786 !      under-estimated.
00787 !    3, extremely bad integrand behavior occurs at some points of the
00788 !      integration interval.
00789 !    4, the algorithm does not converge.  Roundoff error is detected in the
00790 !      extrapolation table.  It is assumed that the requested tolerance
00791 !      cannot be achieved, and that the returned result is the best which
00792 !      can be obtained.
00793 !    5, the integral is probably divergent, or slowly convergent.  It must
00794 !      be noted that divergence can occur with any other value of IER.
00795 !    6, the input is invalid, because INF /= 1 and INF /= -1 and INF /= 2, or
00796 !      epsabs < 0 and epsrel < 0.  result, abserr, neval are set to zero.
00797 !
00798 !  Local parameters:
00799 !
00800 !           the dimension of rlist2 is determined by the value of
00801 !           limexp in QEXTR.
00802 !
00803 !           alist     - list of left end points of all subintervals
00804 !                       considered up to now
00805 !           blist     - list of right end points of all subintervals
00806 !                       considered up to now
00807 !           rlist(i)  - approximation to the integral over
00808 !                       (alist(i),blist(i))
00809 !           rlist2    - array of dimension at least (limexp+2),
00810 !                       containing the part of the epsilon table
00811 !                       which is still needed for further computations
00812 !           elist(i)  - error estimate applying to rlist(i)
00813 !           maxerr    - pointer to the interval with largest error
00814 !                       estimate
00815 !           errmax    - elist(maxerr)
00816 !           erlast    - error on the interval currently subdivided
00817 !                       (before that subdivision has taken place)
00818 !           area      - sum of the integrals over the subintervals
00819 !           errsum    - sum of the errors over the subintervals
00820 !           errbnd    - requested accuracy max(epsabs,epsrel*
00821 !                       abs(result))
00822 !           *****1    - variable for the left subinterval
00823 !           *****2    - variable for the right subinterval
00824 !           last      - index for subdivision
00825 !           nres      - number of calls to the extrapolation routine
00826 !           numrl2    - number of elements currently in rlist2. if an
00827 !                       appropriate approximation to the compounded
00828 !                       integral has been obtained, it is put in
00829 !                       rlist2(numrl2) after numrl2 has been increased
00830 !                       by one.
00831 !           small     - length of the smallest interval considered up
00832 !                       to now, multiplied by 1.5
00833 !           erlarg    - sum of the errors over the intervals larger
00834 !                       than the smallest interval considered up to now
00835 !           extrap    - logical variable denoting that the routine
00836 !                       is attempting to perform extrapolation. i.e.
00837 !                       before subdividing the smallest interval we
00838 !                       try to decrease the value of erlarg.
00839 !           noext     - logical variable denoting that extrapolation
00840 !                       is no longer allowed (true-value)
00841 !
00842   implicit none
00843
00844   integer, parameter :: limit = 500
00845
00846   real(kind=params_wp) abseps
00847   real(kind=params_wp) abserr
00848   real(kind=params_wp) alist(limit)
00849   real(kind=params_wp) area
00850   real(kind=params_wp) area1
00851   real(kind=params_wp) area12
00852   real(kind=params_wp) area2
00853   real(kind=params_wp) a1
00854   real(kind=params_wp) a2
00855   real(kind=params_wp) blist(limit)
00856   real(kind=params_wp) boun
```

```
00857   real(kind=params_wp) bound
00858   real(kind=params_wp) b1
00859   real(kind=params_wp) b2
00860   real(kind=params_wp) correc
00861   real(kind=params_wp) defabs
00862   real(kind=params_wp) defab1
00863   real(kind=params_wp) defab2
00864   real(kind=params_wp) dres
00865   real(kind=params_wp) elist(limit)
00866   real(kind=params_wp) epsabs
00867   real(kind=params_wp) epsrel
00868   real(kind=params_wp) erlarg
00869   real(kind=params_wp) erlast
00870   real(kind=params_wp) errbnd
00871   real(kind=params_wp) errmax
00872   real(kind=params_wp) error1
00873   real(kind=params_wp) error2
00874   real(kind=params_wp) erro12
00875   real(kind=params_wp) errsum
00876   real(kind=params_wp) ertest
00877   logical extrap
00878   real(kind=params_wp), external :: f
00879   integer id
00880   integer ier
00881   integer ierro
00882   integer inf
00883   integer iord(limit)
00884   integer iroff1
00885   integer iroff2
00886   integer iroff3
00887   integer jupbnd
00888   integer k
00889   integer ksgn
00890   integer ktmin
00891   integer last
00892   integer maxerr
00893   integer neval
00894   logical noext
00895   integer nres
00896   integer nrmax
00897   integer numrl2
00898   real(kind=params_wp) resabs
00899   real(kind=params_wp) reseps
00900   real(kind=params_wp) result
00901   real(kind=params_wp) res3la(3)
00902   real(kind=params_wp) rlist(limit)
00903   real(kind=params_wp) rlist2(52)
00904   real(kind=params_wp) small
00905 !
00906 !  Test on validity of parameters.
00907 !
00908   ier = 0
00909   neval = 0
00910   last = 0
00911   result = 0.0e+00
00912   abserr = 0.0e+00
00913   alist(1) = 0.0e+00
00914   blist(1) = 1.0e+00_params_wp
00915   rlist(1) = 0.0e+00
00916   elist(1) = 0.0e+00
00917   iord(1) = 0
00918
00919   if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
00920     ier = 6
00921     return
00922   end if
00923 !
00924 !  First approximation to the integral.
00925 !
00926 !  Determine the interval to be mapped onto (0,1).
00927 !  If INF = 2 the integral is computed as i = i1+i2, where
00928 !  i1 = integral of f over (-infinity,0),
00929 !  i2 = integral of f over (0,+infinity).
00930 !
00931   if ( inf == 2 ) then
00932     boun = 0.0e+00_params_wp
00933   else
00934     boun = bound
00935   end if
00936
00937   call qk15i ( f, boun, inf, 0.0e+00_params_wp, 1.0e+00_params_wp, result, abserr, defabs, resabs )
00938 !
00939 !  Test on accuracy.
00940 !
00941   last = 1
00942   rlist(1) = result
00943   elist(1) = abserr
```

```
00944   iord(1) = 1
00945   dres = abs( result )
00946   errbnd = max( epsabs, epsrel * dres )
00947
00948   if ( abserr <= 100.0e+00 * epsilon( defabs ) * defabs .and. &
00949     errbnd < abserr ) then
00950     ier = 2
00951   end if
00952
00953   if ( limit == 1 ) then
00954     ier = 1
00955   end if
00956
00957   if ( ier /= 0 .or. (abserr <= errbnd .and. abserr /= resabs ) .or. &
00958     abserr == 0.0e+00 ) go to 130
00959 !
00960 !  Initialization.
00961 !
00962   rlist2(1) = result
00963   errmax = abserr
00964   maxerr = 1
00965   area = result
00966   errsum = abserr
00967   abserr = huge( abserr )
00968   nrmax = 1
00969   nres = 0
00970   ktmin = 0
00971   numrl2 = 2
00972   extrap = .false.
00973   noext = .false.
00974   ierro = 0
00975   iroff1 = 0
00976   iroff2 = 0
00977   iroff3 = 0
00978
00979   if ( ( 1.0e+00_params_wp - 5.0e+01 * epsilon( defabs ) ) * defabs <= dres ) then
00980     ksgn = 1
00981   else
00982     ksgn = -1
00983   end if
00984
00985   do last = 2, limit
00986 !
00987 !  Bisect the subinterval with nrmax-th largest error estimate.
00988 !
00989     a1 = alist(maxerr)
00990     b1 = 5.0e-01 * ( alist(maxerr) + blist(maxerr) )
00991     a2 = b1
00992     b2 = blist(maxerr)
00993     erlast = errmax
00994     call qk15i ( f, boun, inf, a1, b1, area1, error1, resabs, defab1 )
00995     call qk15i ( f, boun, inf, a2, b2, area2, error2, resabs, defab2 )
00996 !
00997 !  Improve previous approximations to integral and error
00998 !  and test for accuracy.
00999 !
01000     area12 = area1 + area2
01001     erro12 = error1 + error2
01002     errsum = errsum + erro12 - errmax
01003     area = area + area12 - rlist(maxerr)
01004
01005     if ( defab1 /= error1 .and. defab2 /= error2 ) then
01006
01007       if ( abs( rlist(maxerr) - area12 ) <= 1.0e-05 * abs( area12 ) &
01008         .and. 9.9e-01 * errmax <= erro12 ) then
01009
01010         if ( extrap ) then
01011           iroff2 = iroff2 + 1
01012         end if
01013
01014         if ( .not. extrap ) then
01015           iroff1 = iroff1 + 1
01016         end if
01017
01018       end if
01019
01020       if ( 10 < last .and. errmax < erro12 ) then
01021         iroff3 = iroff3 + 1
01022       end if
01023
01024     end if
01025
01026     rlist(maxerr) = area1
01027     rlist(last) = area2
01028     errbnd = max( epsabs, epsrel * abs( area ) )
01029 !
01030 !  Test for roundoff error and eventually set error flag.
```

```
01031 !
01032     if ( 10 <= iroff1 + iroff2 .or. 20 <= iroff3 ) then
01033       ier = 2
01034     end if
01035
01036     if ( 5 <= iroff2 ) then
01037       ierro = 3
01038     end if
01039 !
01040 !  Set error flag in the case that the number of subintervals equals LIMIT.
01041 !
01042     if ( last == limit ) then
01043       ier = 1
01044     end if
01045 !
01046 !  Set error flag in the case of bad integrand behavior
01047 !  at some points of the integration range.
01048 !
01049     if ( max( abs(a1), abs(b2) ) <= (1.0e+00_params_wp + 1.0e+03 * epsilon( a1 ) ) * &
01050     ( abs(a2) + 1.0e+03 * tiny( a2 ) )) then
01051       ier = 4
01052     end if
01053 !
01054 !  Append the newly-created intervals to the list.
01055 !
01056     if ( error2 <= error1 ) then
01057       alist(last) = a2
01058       blist(maxerr) = b1
01059       blist(last) = b2
01060       elist(maxerr) = error1
01061       elist(last) = error2
01062     else
01063       alist(maxerr) = a2
01064       alist(last) = a1
01065       blist(last) = b1
01066       rlist(maxerr) = area2
01067       rlist(last) = area1
01068       elist(maxerr) = error2
01069       elist(last) = error1
01070     end if
01071 !
01072 !  Call QSORT to maintain the descending ordering
01073 !  in the list of error estimates and select the subinterval
01074 !  with NRMAX-th largest error estimate (to be bisected next).
01075 !
01076     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
01077
01078     if ( errsum <= errbnd ) go to 115
01079
01080     if ( ier /= 0 ) then
01081       exit
01082     end if
01083
01084     if ( last == 2 ) then
01085       small = 3.75e-01
01086       erlarg = errsum
01087       ertest = errbnd
01088       rlist2(2) = area
01089       cycle
01090     end if
01091
01092     if ( noext ) then
01093       cycle
01094     end if
01095
01096     erlarg = erlarg - erlast
01097
01098     if ( small < abs( b1 - a1 ) ) then
01099       erlarg = erlarg + erro12
01100     end if
01101 !
01102 !  Test whether the interval to be bisected next is the
01103 !  smallest interval.
01104 !
01105     if ( .not. extrap ) then
01106
01107       if ( small < abs( blist(maxerr) - alist(maxerr) ) ) then
01108         cycle
01109       end if
01110
01111       extrap = .true.
01112       nrmax = 2
01113
01114     end if
01115
01116     if ( ierro == 3 .or. erlarg <= ertest ) then
01117       go to 60
```

```
01118     end if
01119 !
01120 !  The smallest interval has the largest error.
01121 !  before bisecting decrease the sum of the errors over the
01122 !  larger intervals (erlarg) and perform extrapolation.
01123 !
01124     id = nrmax
01125     jupbnd = last
01126
01127     if ( (2+limit/2) < last ) then
01128       jupbnd = limit + 3 - last
01129     end if
01130
01131     do k = id, jupbnd
01132       maxerr = iord(nrmax)
01133       errmax = elist(maxerr)
01134       if ( small < abs( blist(maxerr) - alist(maxerr) ) ) then
01135         go to 90
01136       end if
01137       nrmax = nrmax + 1
01138     end do
01139 !
01140 !  Extrapolate.
01141 !
01142 60  continue
01143
01144     numrl2 = numrl2 + 1
01145     rlist2(numrl2) = area
01146     call qextr ( numrl2, rlist2, reseps, abseps, res3la, nres )
01147     ktmin = ktmin+1
01148
01149     if ( 5 < ktmin .and. abserr < 1.0e-03 * errsum ) then
01150       ier = 5
01151     end if
01152
01153     if ( abseps < abserr ) then
01154
01155       ktmin = 0
01156       abserr = abseps
01157       result = reseps
01158       correc = erlarg
01159       ertest = max( epsabs, epsrel * abs(reseps) )
01160
01161       if ( abserr <= ertest ) then
01162         exit
01163       end if
01164
01165     end if
01166 !
01167 !  Prepare bisection of the smallest interval.
01168 !
01169     if ( numrl2 == 1 ) then
01170       noext = .true.
01171     end if
01172
01173     if ( ier == 5 ) then
01174       exit
01175     end if
01176
01177     maxerr = iord(1)
01178     errmax = elist(maxerr)
01179     nrmax = 1
01180     extrap = .false.
01181     small = small * 5.0e-01
01182     erlarg = errsum
01183
01184 90  continue
01185
01186   end do
01187 !
01188 !  Set final result and error estimate.
01189 !
01190   if ( abserr == huge( abserr ) ) then
01191     go to 115
01192   end if
01193
01194   if ( ( ier + ierro ) == 0 ) then
01195     go to 110
01196   end if
01197
01198   if ( ierro == 3 ) then
01199     abserr = abserr + correc
01200   end if
01201
01202   if ( ier == 0 ) then
01203     ier = 3
01204   end if
```

```
01205
01206   if ( result /= 0.0e+00 .and. area /= 0.0e+00) then
01207     go to 105
01208   end if
01209
01210   if ( errsum < abserr ) then
01211     go to 115
01212   end if
01213
01214   if ( area == 0.0e+00 ) then
01215     go to 130
01216   end if
01217
01218   go to 110
01219
01220 105 continue
01221
01222   if ( errsum / abs( area ) < abserr / abs( result )  ) then
01223     go to 115
01224   end if
01225 !
01226 !  Test on divergence
01227 !
01228 110 continue
01229
01230   if ( ksgn == (-1) .and. &
01231   max( abs(result), abs(area) ) <=  defabs * 1.0e-02) go to 130
01232
01233   if ( 1.0e-02 > (result/area) .or. &
01234     (result/area) > 1.0e+02 .or. &
01235     errsum > abs(area)) then
01236     ier = 6
01237   end if
01238
01239   go to 130
01240 !
01241 !  Compute global integral sum.
01242 !
01243   115 continue
01244
01245   result = sum( rlist(1:last) )
01246
01247   abserr = errsum
01248   130 continue
01249
01250   neval = 30 * last - 15
01251   if ( inf == 2 ) then
01252     neval = 2 * neval
01253   end if
01254
01255   if ( 2 < ier ) then
01256     ier = ier - 1
01257   end if
01258
01259   return
01260 end subroutine qagi
01261 subroutine qagp ( f, a, b, npts2, points, epsabs, epsrel, result, abserr, &
01262   neval, ier )
01263
01264 !*******************************************************************************80
01265 !
01266 !! QAGP computes a definite integral.
01267 !
01268 !  Discussion:
01269 !
01270 !    The routine calculates an approximation RESULT to a definite integral
01271 !      I = integral of F over (A,B),
01272 !    hopefully satisfying
01273 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
01274 !
01275 !    Interior break points of the integration interval,
01276 !    where local difficulties of the integrand may occur, such as
01277 !    singularities or discontinuities, are provided by the user.
01278 !
01279 !  Author:
01280 !
01281 !    Robert Piessens, Elise de Doncker-Kapenger,
01282 !    Christian Ueberhuber, David Kahaner
01283 !
01284 !  Reference:
01285 !
01286 !    Robert Piessens, Elise de Doncker-Kapenger,
01287 !    Christian Ueberhuber, David Kahaner,
01288 !    QUADPACK, a Subroutine Package for Automatic Integration,
01289 !    Springer Verlag, 1983
01290 !
01291 !  Parameters:
```

```
01292 !
01293 !    Input, external real F, the name of the function routine, of the form
01294 !      function f ( x )
01295 !      real f
01296 !      real x
01297 !    which evaluates the integrand function.
01298 !
01299 !    Input, real A, B, the limits of integration.
01300 !
01301 !    Input, integer NPTS2, the number of user-supplied break points within
01302 !    the integration range, plus 2.  NPTS2 must be at least 2.
01303 !
01304 !    Input/output, real POINTS(NPTS2), contains the user provided interior
01305 !    breakpoints in entries 1 through NPTS2-2.  If these points are not
01306 !    in ascending order on input, they will be sorted.
01307 !
01308 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
01309 !
01310 !    Output, real RESULT, the estimated value of the integral.
01311 !
01312 !    Output, real ABSERR, an estimate of || I - RESULT ||.
01313 !
01314 !    Output, integer NEVAL, the number of times the integral was evaluated.
01315 !
01316 !    Output, integer IER, return flag.
01317 !                     ier = 0 normal and reliable termination of the
01318 !                             routine. it is assumed that the requested
01319 !                             accuracy has been achieved.
01320 !                     ier > 0 abnormal termination of the routine.
01321 !                             the estimates for integral and error are
01322 !                             less reliable. it is assumed that the
01323 !                             requested accuracy has not been achieved.
01324 !                     ier = 1 maximum number of subdivisions allowed
01325 !                             has been achieved. one can allow more
01326 !                             subdivisions by increasing the data value
01327 !                             of limit in qagp(and taking the according
01328 !                             dimension adjustments into account).
01329 !                             however, if this yields no improvement
01330 !                             it is advised to analyze the integrand
01331 !                             in order to determine the integration
01332 !                             difficulties. if the position of a local
01333 !                             difficulty can be determined (i.e.
01334 !                             singularity, discontinuity within the
01335 !                             interval), it should be supplied to the
01336 !                             routine as an element of the vector
01337 !                             points. if necessary, an appropriate
01338 !                             special-purpose integrator must be used,
01339 !                             which is designed for handling the type
01340 !                             of difficulty involved.
01341 !                         = 2 the occurrence of roundoff error is
01342 !                             detected, which prevents the requested
01343 !                             tolerance from being achieved.
01344 !                             the error may be under-estimated.
01345 !                         = 3 extremely bad integrand behavior occurs
01346 !                             at some points of the integration
01347 !                             interval.
01348 !                         = 4 the algorithm does not converge. roundoff
01349 !                             error is detected in the extrapolation
01350 !                             table. it is presumed that the requested
01351 !                             tolerance cannot be achieved, and that
01352 !                             the returned result is the best which
01353 !                             can be obtained.
01354 !                         = 5 the integral is probably divergent, or
01355 !                             slowly convergent. it must be noted that
01356 !                             divergence can occur with any other value
01357 !                             of ier > 0.
01358 !                         = 6 the input is invalid because
01359 !                             npts2 < 2 or
01360 !                             break points are specified outside
01361 !                             the integration range or
01362 !                             epsabs < 0 and epsrel < 0,
01363 !                             or limit < npts2.
01364 !                             result, abserr, neval are set to zero.
01365 !
01366 !  Local parameters:
01367 !
01368 !           the dimension of rlist2 is determined by the value of
01369 !           limexp in QEXTR (rlist2 should be of dimension
01370 !           (limexp+2) at least).
01371 !
01372 !          alist     - list of left end points of all subintervals
01373 !                      considered up to now
01374 !          blist     - list of right end points of all subintervals
01375 !                      considered up to now
01376 !          rlist(i)  - approximation to the integral over
01377 !                      (alist(i),blist(i))
01378 !          rlist2    - array of dimension at least limexp+2
```

```
01379 !                       containing the part of the epsilon table which
01380 !                       is still needed for further computations
01381 !           elist(i)  - error estimate applying to rlist(i)
01382 !           maxerr    - pointer to the interval with largest error
01383 !                       estimate
01384 !           errmax    - elist(maxerr)
01385 !           erlast    - error on the interval currently subdivided
01386 !                       (before that subdivision has taken place)
01387 !           area      - sum of the integrals over the subintervals
01388 !           errsum    - sum of the errors over the subintervals
01389 !           errbnd    - requested accuracy max(epsabs,epsrel*
01390 !                       abs(result))
01391 !           *****1    - variable for the left subinterval
01392 !           *****2    - variable for the right subinterval
01393 !           last      - index for subdivision
01394 !           nres      - number of calls to the extrapolation routine
01395 !           numrl2    - number of elements in rlist2. if an appropriate
01396 !                       approximation to the compounded integral has
01397 !                       obtained, it is put in rlist2(numrl2) after
01398 !                       numrl2 has been increased by one.
01399 !           erlarg    - sum of the errors over the intervals larger
01400 !                       than the smallest interval considered up to now
01401 !           extrap    - logical variable denoting that the routine
01402 !                       is attempting to perform extrapolation. i.e.
01403 !                       before subdividing the smallest interval we
01404 !                       try to decrease the value of erlarg.
01405 !           noext     - logical variable denoting that extrapolation is
01406 !                       no longer allowed (true-value)
01407 !
01408   implicit none
01409
01410   integer, parameter :: limit = 500
01411
01412   real(kind=params_wp) a
01413   real(kind=params_wp) abseps
01414   real(kind=params_wp) abserr
01415   real(kind=params_wp) alist(limit)
01416   real(kind=params_wp) area
01417   real(kind=params_wp) area1
01418   real(kind=params_wp) area12
01419   real(kind=params_wp) area2
01420   real(kind=params_wp) a1
01421   real(kind=params_wp) a2
01422   real(kind=params_wp) b
01423   real(kind=params_wp) blist(limit)
01424   real(kind=params_wp) b1
01425   real(kind=params_wp) b2
01426   real(kind=params_wp) correc
01427   real(kind=params_wp) defabs
01428   real(kind=params_wp) defab1
01429   real(kind=params_wp) defab2
01430   real(kind=params_wp) dres
01431   real(kind=params_wp) elist(limit)
01432   real(kind=params_wp) epsabs
01433   real(kind=params_wp) epsrel
01434   real(kind=params_wp) erlarg
01435   real(kind=params_wp) erlast
01436   real(kind=params_wp) errbnd
01437   real(kind=params_wp) errmax
01438   real(kind=params_wp) error1
01439   real(kind=params_wp) erro12
01440   real(kind=params_wp) error2
01441   real(kind=params_wp) errsum
01442   real(kind=params_wp) ertest
01443   logical extrap
01444   real(kind=params_wp), external :: f
01445   integer i
01446   integer id
01447   integer ier
01448   integer ierro
01449   integer ind1
01450   integer ind2
01451   integer iord(limit)
01452   integer iroff1
01453   integer iroff2
01454   integer iroff3
01455   integer j
01456   integer jlow
01457   integer jupbnd
01458   integer k
01459   integer ksgn
01460   integer ktmin
01461   integer last
01462   integer levcur
01463   integer level(limit)
01464   integer levmax
01465   integer maxerr
```

```
01466   integer ndin(40)
01467   integer neval
01468   integer nint
01469   logical noext
01470   integer npts
01471   integer npts2
01472   integer nres
01473   integer nrmax
01474   integer numrl2
01475   real(kind=params_wp) points(40)
01476   real(kind=params_wp) pts(40)
01477   real(kind=params_wp) resa
01478   real(kind=params_wp) resabs
01479   real(kind=params_wp) reseps
01480   real(kind=params_wp) result
01481   real(kind=params_wp) res3la(3)
01482   real(kind=params_wp) rlist(limit)
01483   real(kind=params_wp) rlist2(52)
01484   real(kind=params_wp) sign
01485   real(kind=params_wp) temp
01486 !
01487 !  Test on validity of parameters.
01488 !
01489   ier = 0
01490   neval = 0
01491   last = 0
01492   result = 0.0e+00
01493   abserr = 0.0e+00
01494   alist(1) = a
01495   blist(1) = b
01496   rlist(1) = 0.0e+00
01497   elist(1) = 0.0e+00
01498   iord(1) = 0
01499   level(1) = 0
01500   npts = npts2 - 2
01501
01502   if ( npts2 < 2 ) then
01503     ier = 6
01504     return
01505   else if ( limit <= npts .or. ( epsabs < 0.0e+00 .and. &
01506     epsrel < 0.0e+00) ) then
01507     ier = 6
01508     return
01509   end if
01510 !
01511 !  If any break points are provided, sort them into an
01512 !  ascending sequence.
01513 !
01514   if ( b < a ) then
01515     sign = -1.0e+00_params_wp
01516   else
01517     sign = +1.0e+00_params_wp
01518   end if
01519
01520   pts(1) = min( a, b )
01521
01522   do i = 1, npts
01523     pts(i+1) = points(i)
01524   end do
01525
01526   pts(npts+2) = max( a, b )
01527   nint = npts+1
01528   a1 = pts(1)
01529
01530   if ( npts /= 0 ) then
01531
01532     do i = 1, nint
01533       do j = i+1, nint+1
01534         if ( pts(j) < pts(i) ) then
01535           temp   = pts(i)
01536           pts(i) = pts(j)
01537           pts(j) = temp
01538         end if
01539       end do
01540     end do
01541
01542     if ( pts(1) /= min( a, b ) .or. pts(nint+1) /= max( a, b ) ) then
01543       ier = 6
01544       return
01545     end if
01546
01547   end if
01548 !
01549 !  Compute first integral and error approximations.
01550 !
01551   resabs = 0.0e+00
01552
```

```
01553   do i = 1, nint
01554
01555     b1 = pts(i+1)
01556     call qk21 ( f, a1, b1, area1, error1, defabs, resa )
01557     abserr = abserr + error1
01558     result = result + area1
01559     ndin(i) = 0
01560
01561     if ( error1 == resa .and. error1 /= 0.0e+00 ) then
01562       ndin(i) = 1
01563     end if
01564
01565     resabs = resabs + defabs
01566     level(i) = 0
01567     elist(i) = error1
01568     alist(i) = a1
01569     blist(i) = b1
01570     rlist(i) = area1
01571     iord(i) = i
01572     a1 = b1
01573
01574   end do
01575
01576   errsum = 0.0e+00
01577
01578   do i = 1, nint
01579     if ( ndin(i) == 1 ) then
01580       elist(i) = abserr
01581     end if
01582     errsum = errsum + elist(i)
01583   end do
01584 !
01585 !  Test on accuracy.
01586 !
01587   last = nint
01588   neval = 21 * nint
01589   dres = abs( result )
01590   errbnd = max( epsabs, epsrel * dres )
01591
01592   if ( abserr <= 1.0e+02 * epsilon( resabs ) * resabs .and. &
01593     abserr > errbnd ) then
01594     ier = 2
01595   end if
01596
01597   if ( nint /= 1 ) then
01598
01599     do i = 1, npts
01600
01601       jlow = i+1
01602       ind1 = iord(i)
01603
01604       do j = jlow, nint
01605         ind2 = iord(j)
01606         if ( elist(ind1) <= elist(ind2) ) then
01607           ind1 = ind2
01608           k = j
01609         end if
01610       end do
01611
01612       if ( ind1 /= iord(i) ) then
01613         iord(k) = iord(i)
01614         iord(i) = ind1
01615       end if
01616
01617     end do
01618
01619     if ( limit < npts2 ) then
01620       ier = 1
01621     end if
01622
01623   end if
01624
01625   if ( ier /= 0 .or. abserr <= errbnd ) then
01626     return
01627   end if
01628 !
01629 !  Initialization
01630 !
01631   rlist2(1) = result
01632   maxerr = iord(1)
01633   errmax = elist(maxerr)
01634   area = result
01635   nrmax = 1
01636   nres = 0
01637   numrl2 = 1
01638   ktmin = 0
01639   extrap = .false.
```

```
01640   noext = .false.
01641   erlarg = errsum
01642   ertest = errbnd
01643   levmax = 1
01644   iroff1 = 0
01645   iroff2 = 0
01646   iroff3 = 0
01647   ierro = 0
01648   abserr = huge( abserr )
01649
01650   if ( dres >= ( 1.0e+00_params_wp - 0.5e+00 * epsilon( resabs ) ) * resabs ) then
01651     ksgn = 1
01652   else
01653     ksgn = -1
01654   end if
01655
01656   do last = npts2, limit
01657 !
01658 !  Bisect the subinterval with the NRMAX-th largest error estimate.
01659 !
01660     levcur = level(maxerr) + 1
01661     a1 = alist(maxerr)
01662     b1 = 0.5e+00 * ( alist(maxerr) + blist(maxerr) )
01663     a2 = b1
01664     b2 = blist(maxerr)
01665     erlast = errmax
01666     call qk21 ( f, a1, b1, area1, error1, resa, defab1 )
01667     call qk21 ( f, a2, b2, area2, error2, resa, defab2 )
01668 !
01669 !  Improve previous approximations to integral and error
01670 !  and test for accuracy.
01671 !
01672     neval = neval + 42
01673     area12 = area1 + area2
01674     erro12 = error1 + error2
01675     errsum = errsum + erro12 -errmax
01676     area = area + area12 - rlist(maxerr)
01677
01678     if ( defab1 /= error1 .and. defab2 /= error2 ) then
01679
01680       if ( abs( rlist( maxerr ) - area12 ) <= 1.0e-05 * abs(area12) .and. &
01681         erro12 >= 9.9e-01 * errmax ) then
01682
01683         if ( extrap ) then
01684           iroff2 = iroff2+1
01685         else
01686           iroff1 = iroff1+1
01687         end if
01688
01689       end if
01690
01691       if ( last > 10 .and. erro12 > errmax ) then
01692         iroff3 = iroff3 + 1
01693       end if
01694
01695     end if
01696
01697     level(maxerr) = levcur
01698     level(last) = levcur
01699     rlist(maxerr) = area1
01700     rlist(last) = area2
01701     errbnd = max( epsabs, epsrel * abs( area ) )
01702 !
01703 !  Test for roundoff error and eventually set error flag.
01704 !
01705     if ( 10 <= iroff1 + iroff2 .or. 20 <= iroff3 ) then
01706       ier = 2
01707     end if
01708
01709     if ( 5 <= iroff2 ) then
01710       ierro = 3
01711     end if
01712 !
01713 !  Set error flag in the case that the number of subintervals
01714 !  equals limit.
01715 !
01716     if ( last == limit ) then
01717       ier = 1
01718     end if
01719 !
01720 !  Set error flag in the case of bad integrand behavior
01721 !  at a point of the integration range
01722 !
01723     if ( max( abs(a1), abs(b2)) <= ( 1.0e+00_params_wp + 1.0e+03 * epsilon( a1 ) )* &
01724     ( abs( a2 ) + 1.0e+03 * tiny( a2 ) ) ) then
01725       ier = 4
01726     end if
```

```
01727 !
01728 !  Append the newly-created intervals to the list.
01729 !
01730     if ( error2 <= error1 ) then
01731       alist(last) = a2
01732       blist(maxerr) = b1
01733       blist(last) = b2
01734       elist(maxerr) = error1
01735       elist(last) = error2
01736     else
01737       alist(maxerr) = a2
01738       alist(last) = a1
01739       blist(last) = b1
01740       rlist(maxerr) = area2
01741       rlist(last) = area1
01742       elist(maxerr) = error2
01743       elist(last) = error1
01744     end if
01745 !
01746 !  Call QSORT to maintain the descending ordering
01747 !  in the list of error estimates and select the subinterval
01748 !  with nrmax-th largest error estimate (to be bisected next).
01749 !
01750     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
01751
01752     if ( errsum <= errbnd ) then
01753       go to 190
01754     end if
01755
01756     if ( ier /= 0 ) then
01757       exit
01758     end if
01759
01760     if ( noext ) then
01761       cycle
01762     end if
01763
01764     erlarg = erlarg - erlast
01765
01766     if ( levcur+1 <= levmax ) then
01767       erlarg = erlarg + erro12
01768     end if
01769 !
01770 !  Test whether the interval to be bisected next is the
01771 !  smallest interval.
01772 !
01773     if ( .not. extrap ) then
01774
01775       if ( level(maxerr)+1 <= levmax ) then
01776         cycle
01777       end if
01778
01779       extrap = .true.
01780       nrmax = 2
01781
01782     end if
01783 !
01784 !  The smallest interval has the largest error.
01785 !  Before bisecting decrease the sum of the errors over the
01786 !  larger intervals (erlarg) and perform extrapolation.
01787 !
01788     if ( ierro /= 3 .and. erlarg > ertest ) then
01789
01790       id = nrmax
01791       jupbnd = last
01792       if ( last > (2+limit/2) ) then
01793         jupbnd = limit+3-last
01794       end if
01795
01796       do k = id, jupbnd
01797         maxerr = iord(nrmax)
01798         errmax = elist(maxerr)
01799         if ( level(maxerr)+1 <= levmax ) then
01800           go to 160
01801         end if
01802         nrmax = nrmax + 1
01803       end do
01804
01805     end if
01806 !
01807 !  Perform extrapolation.
01808 !
01809     numrl2 = numrl2 + 1
01810     rlist2(numrl2) = area
01811
01812     if ( numrl2 <= 2 ) then
01813       go to 155
```

```
01814     end if
01815
01816     call qextr ( numrl2, rlist2, reseps, abseps, res3la, nres )
01817     ktmin = ktmin+1
01818
01819     if ( 5 < ktmin .and. abserr < 1.0e-03 * errsum ) then
01820       ier = 5
01821     end if
01822
01823     if ( abseps < abserr ) then
01824
01825       ktmin = 0
01826       abserr = abseps
01827       result = reseps
01828       correc = erlarg
01829       ertest = max( epsabs, epsrel * abs(reseps) )
01830
01831       if ( abserr < ertest ) then
01832         exit
01833       end if
01834
01835     end if
01836 !
01837 !  Prepare bisection of the smallest interval.
01838 !
01839     if ( numrl2 == 1 ) then
01840       noext = .true.
01841     end if
01842
01843     if ( 5 <= ier ) then
01844       exit
01845     end if
01846
01847 155 continue
01848
01849     maxerr = iord(1)
01850     errmax = elist(maxerr)
01851     nrmax = 1
01852     extrap = .false.
01853     levmax = levmax + 1
01854     erlarg = errsum
01855
01856 160 continue
01857
01858   end do
01859 !
01860 !  Set the final result.
01861 !
01862   if ( abserr == huge( abserr ) ) then
01863     go to 190
01864   end if
01865
01866   if ( ( ier + ierro ) == 0 ) then
01867     go to 180
01868   end if
01869
01870   if ( ierro == 3 ) then
01871     abserr = abserr + correc
01872   end if
01873
01874   if ( ier == 0 ) then
01875     ier = 3
01876   end if
01877
01878   if ( result /= 0.0e+00 .and. area /= 0.0e+00 ) then
01879     go to 175
01880   end if
01881
01882   if ( errsum < abserr ) then
01883     go to 190
01884   end if
01885
01886   if ( area == 0.0e+00 ) then
01887     go to 210
01888   end if
01889
01890   go to 180
01891
01892 175 continue
01893
01894   if ( abserr / abs(result) > errsum / abs(area) ) then
01895     go to 190
01896   end if
01897 !
01898 !  Test on divergence.
01899 !
01900   180 continue
```

```
01901
01902   if ( ksgn == (-1) .and. max( abs(result),abs(area)) <=  &
01903     resabs*1.0e-02 ) go to 210
01904
01905   if ( 1.0e-02 > (result/area) .or. (result/area) > 1.0e+02 .or. &
01906     errsum > abs(area) ) then
01907     ier = 6
01908   end if
01909
01910   go to 210
01911 !
01912 !  Compute global integral sum.
01913 !
01914 190 continue
01915
01916   result = sum( rlist(1:last) )
01917
01918   abserr = errsum
01919
01920 210 continue
01921
01922   if ( 2 < ier ) then
01923     ier = ier - 1
01924   end if
01925
01926   result = result * sign
01927
01928   return
01929 end subroutine qagp
01930 subroutine qags ( f, a, b, epsabs, epsrel, result, abserr, neval, ier )
01931
01932 !*****************************************************************************80
01933 !
01934 !! QAGS estimates the integral of a function.
01935 !
01936 !  Discussion:
01937 !
01938 !    The routine calculates an approximation RESULT to a definite integral
01939 !      I = integral of F over (A,B),
01940 !    hopefully satisfying
01941 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
01942 !
01943 !  Author:
01944 !
01945 !    Robert Piessens, Elise de Doncker-Kapenger,
01946 !    Christian Ueberhuber, David Kahaner
01947 !
01948 !  Reference:
01949 !
01950 !    Robert Piessens, Elise de Doncker-Kapenger,
01951 !    Christian Ueberhuber, David Kahaner,
01952 !    QUADPACK, a Subroutine Package for Automatic Integration,
01953 !    Springer Verlag, 1983
01954 !
01955 !  Parameters:
01956 !
01957 !    Input, external real F, the name of the function routine, of the form
01958 !      function f ( x )
01959 !      real f
01960 !      real x
01961 !    which evaluates the integrand function.
01962 !
01963 !    Input, real A, B, the limits of integration.
01964 !
01965 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
01966 !
01967 !    Output, real RESULT, the estimated value of the integral.
01968 !
01969 !    Output, real ABSERR, an estimate of || I - RESULT ||.
01970 !
01971 !    Output, integer NEVAL, the number of times the integral was evaluated.
01972 !
01973 !    Output, integer IER, error flag.
01974 !                      ier = 0 normal and reliable termination of the
01975 !                              routine. it is assumed that the requested
01976 !                              accuracy has been achieved.
01977 !                      ier > 0 abnormal termination of the routine
01978 !                              the estimates for integral and error are
01979 !                              less reliable. it is assumed that the
01980 !                              requested accuracy has not been achieved.
01981 !                          = 1 maximum number of subdivisions allowed
01982 !                              has been achieved. one can allow more sub-
01983 !                              divisions by increasing the data value of
01984 !                              limit in qags (and taking the according
01985 !                              dimension adjustments into account).
01986 !                              however, if this yields no improvement
01987 !                              it is advised to analyze the integrand
```

```
01988 !                              in order to determine the integration
01989 !                              difficulties. if the position of a
01990 !                              local difficulty can be determined (e.g.
01991 !                              singularity, discontinuity within the
01992 !                              interval) one will probably gain from
01993 !                              splitting up the interval at this point
01994 !                              and calling the integrator on the sub-
01995 !                              ranges. if possible, an appropriate
01996 !                              special-purpose integrator should be used,
01997 !                              which is designed for handling the type
01998 !                              of difficulty involved.
01999 !                            = 2 the occurrence of roundoff error is detec-
02000 !                              ted, which prevents the requested
02001 !                              tolerance from being achieved.
02002 !                              the error may be under-estimated.
02003 !                            = 3 extremely bad integrand behavior occurs
02004 !                              at some  points of the integration
02005 !                              interval.
02006 !                            = 4 the algorithm does not converge. roundoff
02007 !                              error is detected in the extrapolation
02008 !                              table. it is presumed that the requested
02009 !                              tolerance cannot be achieved, and that the
02010 !                              returned result is the best which can be
02011 !                              obtained.
02012 !                            = 5 the integral is probably divergent, or
02013 !                              slowly convergent. it must be noted that
02014 !                              divergence can occur with any other value
02015 !                              of ier.
02016 !                            = 6 the input is invalid, because
02017 !                              epsabs < 0 and epsrel < 0,
02018 !                              result, abserr and neval are set to zero.
02019 !
02020 !  Local Parameters:
02021 !
02022 !           alist     - list of left end points of all subintervals
02023 !                       considered up to now
02024 !           blist     - list of right end points of all subintervals
02025 !                       considered up to now
02026 !           rlist(i)  - approximation to the integral over
02027 !                       (alist(i),blist(i))
02028 !           rlist2    - array of dimension at least limexp+2 containing
02029 !                       the part of the epsilon table which is still
02030 !                       needed for further computations
02031 !           elist(i)  - error estimate applying to rlist(i)
02032 !           maxerr    - pointer to the interval with largest error
02033 !                       estimate
02034 !           errmax    - elist(maxerr)
02035 !           erlast    - error on the interval currently subdivided
02036 !                       (before that subdivision has taken place)
02037 !           area      - sum of the integrals over the subintervals
02038 !           errsum    - sum of the errors over the subintervals
02039 !           errbnd    - requested accuracy max(epsabs,epsrel*
02040 !                       abs(result))
02041 !           *****1    - variable for the left interval
02042 !           *****2    - variable for the right interval
02043 !           last      - index for subdivision
02044 !           nres      - number of calls to the extrapolation routine
02045 !           numrl2    - number of elements currently in rlist2. if an
02046 !                       appropriate approximation to the compounded
02047 !                       integral has been obtained it is put in
02048 !                       rlist2(numrl2) after numrl2 has been increased
02049 !                       by one.
02050 !           small     - length of the smallest interval considered
02051 !                       up to now, multiplied by 1.5
02052 !           erlarg    - sum of the errors over the intervals larger
02053 !                       than the smallest interval considered up to now
02054 !           extrap    - logical variable denoting that the routine is
02055 !                       attempting to perform extrapolation i.e. before
02056 !                       subdividing the smallest interval we try to
02057 !                       decrease the value of erlarg.
02058 !           noext     - logical variable denoting that extrapolation
02059 !                       is no longer allowed (true value)
02060 !
02061   implicit none
02062
02063   integer, parameter :: limit = 500
02064
02065   real(kind=params_wp) a
02066   real(kind=params_wp) abseps
02067   real(kind=params_wp) abserr
02068   real(kind=params_wp) alist(limit)
02069   real(kind=params_wp) area
02070   real(kind=params_wp) area1
02071   real(kind=params_wp) area12
02072   real(kind=params_wp) area2
02073   real(kind=params_wp) a1
02074   real(kind=params_wp) a2
```

```
02075   real(kind=params_wp) b
02076   real(kind=params_wp) blist(limit)
02077   real(kind=params_wp) b1
02078   real(kind=params_wp) b2
02079   real(kind=params_wp) correc
02080   real(kind=params_wp) defabs
02081   real(kind=params_wp) defab1
02082   real(kind=params_wp) defab2
02083   real(kind=params_wp) dres
02084   real(kind=params_wp) elist(limit)
02085   real(kind=params_wp) epsabs
02086   real(kind=params_wp) epsrel
02087   real(kind=params_wp) erlarg
02088   real(kind=params_wp) erlast
02089   real(kind=params_wp) errbnd
02090   real(kind=params_wp) errmax
02091   real(kind=params_wp) error1
02092   real(kind=params_wp) error2
02093   real(kind=params_wp) erro12
02094   real(kind=params_wp) errsum
02095   real(kind=params_wp) ertest
02096   logical extrap
02097   real(kind=params_wp), external :: f
02098   integer id
02099   integer ier
02100   integer ierro
02101   integer iord(limit)
02102   integer iroff1
02103   integer iroff2
02104   integer iroff3
02105   integer jupbnd
02106   integer k
02107   integer ksgn
02108   integer ktmin
02109   integer last
02110   logical noext
02111   integer maxerr
02112   integer neval
02113   integer nres
02114   integer nrmax
02115   integer numrl2
02116   real(kind=params_wp) resabs
02117   real(kind=params_wp) reseps
02118   real(kind=params_wp) result
02119   real(kind=params_wp) res3la(3)
02120   real(kind=params_wp) rlist(limit)
02121   real(kind=params_wp) rlist2(52)
02122   real(kind=params_wp) small
02123 !
02124 !  The dimension of rlist2 is determined by the value of
02125 !  limexp in QEXTR (rlist2 should be of dimension
02126 !  (limexp+2) at least).
02127 !
02128 !  Test on validity of parameters.
02129 !
02130   ier = 0
02131   neval = 0
02132   last = 0
02133   result = 0.0e+00
02134   abserr = 0.0e+00
02135   alist(1) = a
02136   blist(1) = b
02137   rlist(1) = 0.0e+00
02138   elist(1) = 0.0e+00
02139
02140   if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
02141     ier = 6
02142     return
02143   end if
02144 !
02145 !  First approximation to the integral.
02146 !
02147   ierro = 0
02148   call qk21 ( f, a, b, result, abserr, defabs, resabs )
02149 !
02150 !  Test on accuracy.
02151 !
02152   dres = abs( result )
02153   errbnd = max( epsabs, epsrel * dres )
02154   last = 1
02155   rlist(1) = result
02156   elist(1) = abserr
02157   iord(1) = 1
02158
02159   if ( abserr <= 1.0e+02 * epsilon( defabs ) * defabs .and. &
02160     abserr > errbnd ) then
02161     ier = 2
```

```
02162   end if
02163
02164   if ( limit == 1 ) then
02165     ier = 1
02166   end if
02167
02168   if ( ier /= 0 .or. (abserr <= errbnd .and. abserr /= resabs ) .or. &
02169     abserr == 0.0e+00 ) go to 140
02170 !
02171 !  Initialization.
02172 !
02173   rlist2(1) = result
02174   errmax = abserr
02175   maxerr = 1
02176   area = result
02177   errsum = abserr
02178   abserr = huge( abserr )
02179   nrmax = 1
02180   nres = 0
02181   numrl2 = 2
02182   ktmin = 0
02183   extrap = .false.
02184   noext = .false.
02185   iroff1 = 0
02186   iroff2 = 0
02187   iroff3 = 0
02188
02189   if ( dres >= (1.0e+00_params_wp-5.0e+01* epsilon( defabs ) ) * defabs ) then
02190     ksgn = 1
02191   else
02192     ksgn = -1
02193   end if
02194
02195   do last = 2, limit
02196 !
02197 !  Bisect the subinterval with the nrmax-th largest error estimate.
02198 !
02199     a1 = alist(maxerr)
02200     b1 = 5.0e-01 * ( alist(maxerr) + blist(maxerr) )
02201     a2 = b1
02202     b2 = blist(maxerr)
02203     erlast = errmax
02204     call qk21 ( f, a1, b1, area1, error1, resabs, defab1 )
02205     call qk21 ( f, a2, b2, area2, error2, resabs, defab2 )
02206 !
02207 !  Improve previous approximations to integral and error
02208 !  and test for accuracy.
02209 !
02210     area12 = area1+area2
02211     erro12 = error1+error2
02212     errsum = errsum+erro12-errmax
02213     area = area+area12-rlist(maxerr)
02214
02215     if ( defab1 == error1 .or. defab2 == error2 ) go to 15
02216
02217     if ( abs( rlist(maxerr) - area12) > 1.0e-05 * abs(area12) &
02218       .or. erro12 < 9.9e-01 * errmax ) go to 10
02219
02220     if ( extrap ) then
02221       iroff2 = iroff2+1
02222     else
02223       iroff1 = iroff1+1
02224     end if
02225
02226 10  continue
02227
02228     if ( last > 10 .and. erro12 > errmax ) then
02229       iroff3 = iroff3+1
02230     end if
02231
02232 15  continue
02233
02234     rlist(maxerr) = area1
02235     rlist(last) = area2
02236     errbnd = max( epsabs, epsrel*abs(area) )
02237 !
02238 !  Test for roundoff error and eventually set error flag.
02239 !
02240     if ( iroff1+iroff2 >= 10 .or. iroff3 >= 20 ) then
02241       ier = 2
02242     end if
02243
02244     if ( iroff2 >= 5 ) then
02245       ierro = 3
02246     end if
02247 !
02248 !  Set error flag in the case that the number of subintervals
```

```
02249 !  equals limit.
02250 !
02251     if ( last == limit ) then
02252       ier = 1
02253     end if
02254 !
02255 !  Set error flag in the case of bad integrand behavior
02256 !  at a point of the integration range.
02257 !
02258     if ( max( abs(a1),abs(b2)) <= (1.0e+00_params_wp+1.0e+03* epsilon( a1 ) )* &
02259       (abs(a2)+1.0e+03* tiny( a2 ) ) ) then
02260       ier = 4
02261     end if
02262 !
02263 !  Append the newly-created intervals to the list.
02264 !
02265     if ( error2 <= error1 ) then
02266       alist(last) = a2
02267       blist(maxerr) = b1
02268       blist(last) = b2
02269       elist(maxerr) = error1
02270       elist(last) = error2
02271     else
02272       alist(maxerr) = a2
02273       alist(last) = a1
02274       blist(last) = b1
02275       rlist(maxerr) = area2
02276       rlist(last) = area1
02277       elist(maxerr) = error2
02278       elist(last) = error1
02279     end if
02280 !
02281 !  Call QSORT to maintain the descending ordering
02282 !  in the list of error estimates and select the subinterval
02283 !  with nrmax-th largest error estimate (to be bisected next).
02284 !
02285     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
02286
02287     if ( errsum <= errbnd ) go to 115
02288
02289     if ( ier /= 0 ) then
02290       exit
02291     end if
02292
02293     if ( last == 2 ) go to 80
02294     if ( noext ) go to 90
02295
02296     erlarg = erlarg-erlast
02297
02298     if ( abs(b1-a1) > small ) then
02299       erlarg = erlarg+erro12
02300     end if
02301 !
02302 !  Test whether the interval to be bisected next is the
02303 !  smallest interval.
02304 !
02305     if ( .not. extrap ) then
02306       if ( abs(blist(maxerr)-alist(maxerr)) > small ) go to 90
02307       extrap = .true.
02308       nrmax = 2
02309     end if
02310
02311 !40  continue
02312 !
02313 !  The smallest interval has the largest error.
02314 !  Before bisecting decrease the sum of the errors over the
02315 !  larger intervals (erlarg) and perform extrapolation.
02316 !
02317     if ( ierro /= 3 .and. erlarg > ertest ) then
02318
02319       id = nrmax
02320       jupbnd = last
02321
02322       if ( last > (2+limit/2) ) then
02323         jupbnd = limit+3-last
02324       end if
02325
02326       do k = id, jupbnd
02327         maxerr = iord(nrmax)
02328         errmax = elist(maxerr)
02329         if ( abs(blist(maxerr)-alist(maxerr)) > small ) then
02330           go to 90
02331         end if
02332         nrmax = nrmax+1
02333       end do
02334
02335     end if
```

```
02336 !
02337 !  Perform extrapolation.
02338 !
02339 !60  continue
02340
02341     numrl2 = numrl2+1
02342     rlist2(numrl2) = area
02343     call qextr ( numrl2, rlist2, reseps, abseps, res3la, nres )
02344     ktmin = ktmin+1
02345
02346     if ( ktmin > 5 .and. abserr < 1.0e-03 * errsum ) then
02347       ier = 5
02348     end if
02349
02350     if ( abseps < abserr ) then
02351
02352       ktmin = 0
02353       abserr = abseps
02354       result = reseps
02355       correc = erlarg
02356       ertest = max( epsabs,epsrel*abs(reseps))
02357
02358       if ( abserr <= ertest ) then
02359         exit
02360       end if
02361
02362     end if
02363 !
02364 !  Prepare bisection of the smallest interval.
02365 !
02366     if ( numrl2 == 1 ) then
02367       noext = .true.
02368     end if
02369
02370     if ( ier == 5 ) then
02371       exit
02372     end if
02373
02374     maxerr = iord(1)
02375     errmax = elist(maxerr)
02376     nrmax = 1
02377     extrap = .false.
02378     small = small * 5.0e-01
02379     erlarg = errsum
02380     go to 90
02381
02382 80  continue
02383
02384     small = abs( b - a ) * 3.75e-01
02385     erlarg = errsum
02386     ertest = errbnd
02387     rlist2(2) = area
02388
02389 90  continue
02390
02391   end do
02392 !
02393 !  Set final result and error estimate.
02394 !
02395   if ( abserr == huge( abserr ) ) then
02396     go to 115
02397   end if
02398
02399   if ( ier + ierro == 0 ) then
02400     go to 110
02401   end if
02402
02403   if ( ierro == 3 ) then
02404     abserr = abserr + correc
02405   end if
02406
02407   if ( ier == 0 ) then
02408     ier = 3
02409   end if
02410
02411   if ( result /= 0.0e+00.and.area /= 0.0e+00 ) then
02412     go to 105
02413   end if
02414
02415   if ( abserr > errsum ) go to 115
02416   if ( area == 0.0e+00 ) go to 130
02417   go to 110
02418
02419 105 continue
02420
02421   if ( abserr/abs(result) > errsum/abs(area) ) go to 115
02422 !
```

```
02423 !  Test on divergence.
02424 !
02425 110 continue
02426
02427   if ( ksgn == (-1).and.max( abs(result),abs(area)) <=  &
02428    defabs*1.0e-02 ) go to 130
02429
02430   if ( 1.0e-02 > (result/area) .or. (result/area) > 1.0e+02 &
02431    .or. errsum > abs(area) ) then
02432     ier = 6
02433   end if
02434
02435   go to 130
02436 !
02437 !  Compute global integral sum.
02438 !
02439 115 continue
02440
02441   result = sum( rlist(1:last) )
02442
02443   abserr = errsum
02444
02445 130 continue
02446
02447   if ( 2 < ier ) then
02448     ier = ier - 1
02449   end if
02450
02451 140 continue
02452
02453   neval = 42*last-21
02454
02455   return
02456 end subroutine qags
02457 subroutine qawc ( f, a, b, c, epsabs, epsrel, result, abserr, neval, ier )
02458
02459 !*****************************************************************************80
02460 !
02461 !! QAWC computes a Cauchy principal value.
02462 !
02463 !  Discussion:
02464 !
02465 !    The routine calculates an approximation RESULT to a Cauchy principal
02466 !    value
02467 !      I = integral of F*W over (A,B),
02468 !    with
02469 !      W(X) = 1 / (X-C),
02470 !    with C distinct from A and B, hopefully satisfying
02471 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
02472 !
02473 !  Author:
02474 !
02475 !    Robert Piessens, Elise de Doncker-Kapenger,
02476 !    Christian Ueberhuber, David Kahaner
02477 !
02478 !  Reference:
02479 !
02480 !    Robert Piessens, Elise de Doncker-Kapenger,
02481 !    Christian Ueberhuber, David Kahaner,
02482 !    QUADPACK, a Subroutine Package for Automatic Integration,
02483 !    Springer Verlag, 1983
02484 !
02485 !  Parameters:
02486 !
02487 !    Input, external real F, the name of the function routine, of the form
02488 !      function f ( x )
02489 !      real f
02490 !      real x
02491 !    which evaluates the integrand function.
02492 !
02493 !    Input, real A, B, the limits of integration.
02494 !
02495 !    Input, real C, a parameter in the weight function, which must
02496 !    not be equal to A or B.
02497 !
02498 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
02499 !
02500 !    Output, real RESULT, the estimated value of the integral.
02501 !
02502 !    Output, real ABSERR, an estimate of || I - RESULT ||.
02503 !
02504 !    Output, integer NEVAL, the number of times the integral was evaluated.
02505 !
02506 !            ier    - integer
02507 !                     ier = 0 normal and reliable termination of the
02508 !                             routine. it is assumed that the requested
02509 !                             accuracy has been achieved.
```

```
02510 !                     ier > 0 abnormal termination of the routine
02511 !                             the estimates for integral and error are
02512 !                             less reliable. it is assumed that the
02513 !                             requested accuracy has not been achieved.
02514 !                     ier = 1 maximum number of subdivisions allowed
02515 !                             has been achieved. one can allow more sub-
02516 !                             divisions by increasing the data value of
02517 !                             limit in qawc (and taking the according
02518 !                             dimension adjustments into account).
02519 !                             however, if this yields no improvement it
02520 !                             is advised to analyze the integrand in
02521 !                             order to determine the integration
02522 !                             difficulties. if the position of a local
02523 !                             difficulty can be determined (e.g.
02524 !                             singularity, discontinuity within the
02525 !                             interval one will probably gain from
02526 !                             splitting up the interval at this point
02527 !                             and calling appropriate integrators on the
02528 !                             subranges.
02529 !                         = 2 the occurrence of roundoff error is detec-
02530 !                             ted, which prevents the requested
02531 !                             tolerance from being achieved.
02532 !                         = 3 extremely bad integrand behavior occurs
02533 !                             at some points of the integration
02534 !                             interval.
02535 !                         = 6 the input is invalid, because
02536 !                             c = a or c = b or
02537 !                             epsabs < 0 and epsrel < 0,
02538 !                             result, abserr, neval are set to zero.
02539 !
02540 !  Local parameters:
02541 !
02542 !    LIMIT is the maximum number of subintervals allowed in the
02543 !    subdivision process of qawce. take care that limit >= 1.
02544 !
02545   implicit none
02546
02547   integer, parameter :: limit = 500
02548
02549   real(kind=params_wp) a
02550   real(kind=params_wp) abserr
02551   real(kind=params_wp) alist(limit)
02552   real(kind=params_wp) b
02553   real(kind=params_wp) blist(limit)
02554   real(kind=params_wp) elist(limit)
02555   real(kind=params_wp) c
02556   real(kind=params_wp) epsabs
02557   real(kind=params_wp) epsrel
02558   real(kind=params_wp), external :: f
02559   integer ier
02560   integer iord(limit)
02561   integer last
02562   integer neval
02563   real(kind=params_wp) result
02564   real(kind=params_wp) rlist(limit)
02565
02566   call qawce ( f, a, b, c, epsabs, epsrel, limit, result, abserr, neval, ier, &
02567     alist, blist, rlist, elist, iord, last )
02568
02569   return
02570 end subroutine qawc
02571 subroutine qawce ( f, a, b, c, epsabs, epsrel, limit, result, abserr, neval, &
02572   ier, alist, blist, rlist, elist, iord, last )
02573
02574 !*******************************************************************************80
02575 !
02576 !! QAWCE computes a Cauchy principal value.
02577 !
02578 !  Discussion:
02579 !
02580 !    The routine calculates an approximation RESULT to a Cauchy principal
02581 !    value
02582 !      I = integral of F*W over (A,B),
02583 !    with
02584 !      W(X) = 1 / ( X - C ),
02585 !    with C distinct from A and B, hopefully satisfying
02586 !      | I - RESULT | <= max ( EPSABS, EPSREL * |I| ).
02587 !
02588 !  Author:
02589 !
02590 !    Robert Piessens, Elise de Doncker-Kapenger,
02591 !    Christian Ueberhuber, David Kahaner
02592 !
02593 !  Reference:
02594 !
02595 !    Robert Piessens, Elise de Doncker-Kapenger,
02596 !    Christian Ueberhuber, David Kahaner,
```

```
02597 !    QUADPACK, a Subroutine Package for Automatic Integration,
02598 !    Springer Verlag, 1983
02599 !
02600 !  Parameters:
02601 !
02602 !    Input, external real F, the name of the function routine, of the form
02603 !      function f ( x )
02604 !      real(kind=params_wp) f
02605 !      real x
02606 !    which evaluates the integrand function.
02607 !
02608 !    Input, real A, B, the limits of integration.
02609 !
02610 !    Input, real C, a parameter in the weight function, which cannot be
02611 !    equal to A or B.
02612 !
02613 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
02614 !
02615 !    Input, integer LIMIT, the upper bound on the number of subintervals that
02616 !    will be used in the partition of [A,B].  LIMIT is typically 500.
02617 !
02618 !    Output, real RESULT, the estimated value of the integral.
02619 !
02620 !    Output, real ABSERR, an estimate of || I - RESULT ||.
02621 !
02622 !    Output, integer NEVAL, the number of times the integral was evaluated.
02623 !
02624 !            ier    - integer
02625 !                     ier = 0 normal and reliable termination of the
02626 !                             routine. it is assumed that the requested
02627 !                             accuracy has been achieved.
02628 !                     ier > 0 abnormal termination of the routine
02629 !                             the estimates for integral and error are
02630 !                             less reliable. it is assumed that the
02631 !                             requested accuracy has not been achieved.
02632 !                     ier = 1 maximum number of subdivisions allowed
02633 !                             has been achieved. one can allow more sub-
02634 !                             divisions by increasing the value of
02635 !                             limit. however, if this yields no
02636 !                             improvement it is advised to analyze the
02637 !                             integrand, in order to determine the
02638 !                             integration difficulties.  if the position
02639 !                             of a local difficulty can be determined
02640 !                             (e.g. singularity, discontinuity within
02641 !                             the interval) one will probably gain
02642 !                             from splitting up the interval at this
02643 !                             point and calling appropriate integrators
02644 !                             on the subranges.
02645 !                         = 2 the occurrence of roundoff error is detec-
02646 !                             ted, which prevents the requested
02647 !                             tolerance from being achieved.
02648 !                         = 3 extremely bad integrand behavior occurs
02649 !                             at some interior points of the integration
02650 !                             interval.
02651 !                         = 6 the input is invalid, because
02652 !                             c = a or c = b or
02653 !                             epsabs < 0 and epsrel < 0,
02654 !                             or limit < 1.
02655 !                             result, abserr, neval, rlist(1), elist(1),
02656 !                             iord(1) and last are set to zero.
02657 !                             alist(1) and blist(1) are set to a and b
02658 !                             respectively.
02659 !
02660 !    Workspace, real ALIST(LIMIT), BLIST(LIMIT), contains in entries 1
02661 !    through LAST the left and right ends of the partition subintervals.
02662 !
02663 !    Workspace, real RLIST(LIMIT), contains in entries 1 through LAST
02664 !    the integral approximations on the subintervals.
02665 !
02666 !    Workspace, real ELIST(LIMIT), contains in entries 1 through LAST
02667 !    the absolute error estimates on the subintervals.
02668 !
02669 !            iord    - integer
02670 !                      vector of dimension at least limit, the first k
02671 !                      elements of which are pointers to the error
02672 !                      estimates over the subintervals, so that
02673 !                      elist(iord(1)), ...,  elist(iord(k)) with
02674 !                      k = last if last <= (limit/2+2), and
02675 !                      k = limit+1-last otherwise, form a decreasing
02676 !                      sequence.
02677 !
02678 !            last    - integer
02679 !                      number of subintervals actually produced in
02680 !                      the subdivision process
02681 !
02682 !  Local parameters:
02683 !
```

```
02684 !           alist     - list of left end points of all subintervals
02685 !                       considered up to now
02686 !           blist     - list of right end points of all subintervals
02687 !                       considered up to now
02688 !           rlist(i)  - approximation to the integral over
02689 !                       (alist(i),blist(i))
02690 !           elist(i)  - error estimate applying to rlist(i)
02691 !           maxerr    - pointer to the interval with largest error
02692 !                       estimate
02693 !           errmax    - elist(maxerr)
02694 !           area      - sum of the integrals over the subintervals
02695 !           errsum    - sum of the errors over the subintervals
02696 !           errbnd    - requested accuracy max(epsabs,epsrel*
02697 !                       abs(result))
02698 !           *****1    - variable for the left subinterval
02699 !           *****2    - variable for the right subinterval
02700 !           last      - index for subdivision
02701 !
02702   implicit none
02703
02704   integer limit
02705
02706   real(kind=params_wp) a
02707   real(kind=params_wp) aa
02708   real(kind=params_wp) abserr
02709   real(kind=params_wp) alist(limit)
02710   real(kind=params_wp) area
02711   real(kind=params_wp) area1
02712   real(kind=params_wp) area12
02713   real(kind=params_wp) area2
02714   real(kind=params_wp) a1
02715   real(kind=params_wp) a2
02716   real(kind=params_wp) b
02717   real(kind=params_wp) bb
02718   real(kind=params_wp) blist(limit)
02719   real(kind=params_wp) b1
02720   real(kind=params_wp) b2
02721   real(kind=params_wp) c
02722   real(kind=params_wp) elist(limit)
02723   real(kind=params_wp) epsabs
02724   real(kind=params_wp) epsrel
02725   real(kind=params_wp) errbnd
02726   real(kind=params_wp) errmax
02727   real(kind=params_wp) error1
02728   real(kind=params_wp) error2
02729   real(kind=params_wp) erro12
02730   real(kind=params_wp) errsum
02731   real(kind=params_wp), external :: f
02732   integer ier
02733   integer iord(limit)
02734   integer iroff1
02735   integer iroff2
02736   integer krule
02737   integer last
02738   integer maxerr
02739   integer nev
02740   integer neval
02741   integer nrmax
02742   real(kind=params_wp) result
02743   real(kind=params_wp) rlist(limit)
02744 !
02745 !  Test on validity of parameters.
02746 !
02747   ier = 0
02748   neval = 0
02749   last = 0
02750   alist(1) = a
02751   blist(1) = b
02752   rlist(1) = 0.0e+00
02753   elist(1) = 0.0e+00
02754   iord(1) = 0
02755   result = 0.0e+00
02756   abserr = 0.0e+00
02757
02758   if ( c == a  ) then
02759     ier = 6
02760     return
02761   else if ( c == b ) then
02762     ier = 6
02763     return
02764   else if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
02765     ier = 6
02766     return
02767   end if
02768 !
02769 !  First approximation to the integral.
02770 !
```

```
02771   if ( a <= b ) then
02772     aa = a
02773     bb = b
02774   else
02775     aa = b
02776     bb = a
02777   end if
02778
02779   krule = 1
02780   call qc25c ( f, aa, bb, c, result, abserr, krule, neval )
02781   last = 1
02782   rlist(1) = result
02783   elist(1) = abserr
02784   iord(1) = 1
02785   alist(1) = a
02786   blist(1) = b
02787 !
02788 !  Test on accuracy.
02789 !
02790   errbnd = max( epsabs, epsrel * abs(result) )
02791
02792   if ( limit == 1 ) then
02793     ier = 1
02794     go to 70
02795   end if
02796
02797   if ( abserr < min( 1.0e-02 * abs(result), errbnd)  ) then
02798     go to 70
02799   end if
02800 !
02801 !  Initialization
02802 !
02803   alist(1) = aa
02804   blist(1) = bb
02805   rlist(1) = result
02806   errmax = abserr
02807   maxerr = 1
02808   area = result
02809   errsum = abserr
02810   nrmax = 1
02811   iroff1 = 0
02812   iroff2 = 0
02813
02814   do last = 2, limit
02815 !
02816 !  Bisect the subinterval with nrmax-th largest error estimate.
02817 !
02818     a1 = alist(maxerr)
02819     b1 = 5.0e-01*(alist(maxerr)+blist(maxerr))
02820     b2 = blist(maxerr)
02821
02822     if ( c <= b1 .and. a1 < c ) then
02823       b1 = 5.0e-01*(c+b2)
02824     end if
02825
02826     if ( b1 < c .and. c < b2 ) then
02827       b1 = 5.0e-01 * ( a1 + c )
02828     end if
02829
02830     a2 = b1
02831     krule = 2
02832
02833     call qc25c ( f, a1, b1, c, area1, error1, krule, nev )
02834     neval = neval+nev
02835
02836     call qc25c ( f, a2, b2, c, area2, error2, krule, nev )
02837     neval = neval+nev
02838 !
02839 !  Improve previous approximations to integral and error
02840 !  and test for accuracy.
02841 !
02842     area12 = area1 + area2
02843     erro12 = error1 + error2
02844     errsum = errsum + erro12 - errmax
02845     area = area + area12 - rlist(maxerr)
02846
02847     if ( abs( rlist(maxerr)-area12) < 1.0e-05 * abs(area12) &
02848       .and. erro12 >= 9.9e-01 * errmax .and. krule == 0 ) &
02849       iroff1 = iroff1+1
02850
02851     if ( last > 10.and.erro12 > errmax .and. krule == 0 ) then
02852       iroff2 = iroff2+1
02853     end if
02854
02855     rlist(maxerr) = area1
02856     rlist(last) = area2
02857     errbnd = max( epsabs, epsrel * abs(area) )
```

```
02858
02859     if ( errsum > errbnd ) then
02860 !
02861 !  Test for roundoff error and eventually set error flag.
02862 !
02863       if ( iroff1 >= 6 .and. iroff2 > 20 ) then
02864         ier = 2
02865       end if
02866 !
02867 !  Set error flag in the case that number of interval
02868 !  bisections exceeds limit.
02869 !
02870       if ( last == limit ) then
02871         ier = 1
02872       end if
02873 !
02874 !  Set error flag in the case of bad integrand behavior at
02875 !  a point of the integration range.
02876 !
02877       if ( max( abs(a1), abs(b2) ) <= ( 1.0e+00_params_wp + 1.0e+03 * epsilon( a1 ) ) &
02878         *( abs(a2)+1.0e+03* tiny( a2 ) )) then
02879         ier = 3
02880       end if
02881
02882     end if
02883 !
02884 !  Append the newly-created intervals to the list.
02885 !
02886     if ( error2 <= error1 ) then
02887       alist(last) = a2
02888       blist(maxerr) = b1
02889       blist(last) = b2
02890       elist(maxerr) = error1
02891       elist(last) = error2
02892     else
02893       alist(maxerr) = a2
02894       alist(last) = a1
02895       blist(last) = b1
02896       rlist(maxerr) = area2
02897       rlist(last) = area1
02898       elist(maxerr) = error2
02899       elist(last) = error1
02900     end if
02901 !
02902 !  Call QSORT to maintain the descending ordering
02903 !  in the list of error estimates and select the subinterval
02904 !  with NRMAX-th largest error estimate (to be bisected next).
02905 !
02906     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
02907
02908     if ( ier /= 0 .or. errsum <= errbnd ) then
02909       exit
02910     end if
02911
02912   end do
02913 !
02914 !  Compute final result.
02915 !
02916   result = sum( rlist(1:last) )
02917
02918   abserr = errsum
02919
02920 70 continue
02921
02922   if ( aa == b ) then
02923     result = - result
02924   end if
02925
02926   return
02927 end subroutine qawce
02928 subroutine qawf ( f, a, omega, integr, epsabs, result, abserr, neval, ier )
02929
02930 !*****************************************************************************80
02931 !
02932 !! QAWF computes Fourier integrals over the interval [ A, +Infinity ).
02933 !
02934 !  Discussion:
02935 !
02936 !    The routine calculates an approximation RESULT to a definite integral
02937 !
02938 !      I = integral of F*COS(OMEGA*X)
02939 !    or
02940 !      I = integral of F*SIN(OMEGA*X)
02941 !
02942 !    over the interval [A,+Infinity), hopefully satisfying
02943 !
02944 !      || I - RESULT || <= EPSABS.
```

```
02945 !
02946 !    If OMEGA = 0 and INTEGR = 1, the integral is calculated by means
02947 !    of QAGI, and IER has the meaning as described in the comments of QAGI.
02948 !
02949 !  Author:
02950 !
02951 !    Robert Piessens, Elise de Doncker-Kapenger,
02952 !    Christian Ueberhuber, David Kahaner
02953 !
02954 !  Reference:
02955 !
02956 !    Robert Piessens, Elise de Doncker-Kapenger,
02957 !    Christian Ueberhuber, David Kahaner,
02958 !    QUADPACK, a Subroutine Package for Automatic Integration,
02959 !    Springer Verlag, 1983
02960 !
02961 !  Parameters:
02962 !
02963 !    Input, external real F, the name of the function routine, of the form
02964 !      function f ( x )
02965 !      real(kind=params_wp) f
02966 !      real x
02967 !    which evaluates the integrand function.
02968 !
02969 !    Input, real A, the lower limit of integration.
02970 !
02971 !    Input, real OMEGA, the parameter in the weight function.
02972 !
02973 !    Input, integer INTEGR, indicates which weight functions is used
02974 !    = 1, w(x) = cos(omega*x)
02975 !    = 2, w(x) = sin(omega*x)
02976 !
02977 !    Input, real EPSABS, the absolute accuracy requested.
02978 !
02979 !    Output, real RESULT, the estimated value of the integral.
02980 !
02981 !    Output, real ABSERR, an estimate of || I - RESULT ||.
02982 !
02983 !    Output, integer NEVAL, the number of times the integral was evaluated.
02984 !
02985 !            ier    - integer
02986 !                     ier = 0 normal and reliable termination of the
02987 !                             routine. it is assumed that the
02988 !                             requested accuracy has been achieved.
02989 !                     ier > 0 abnormal termination of the routine.
02990 !                             the estimates for integral and error are
02991 !                             less reliable. it is assumed that the
02992 !                             requested accuracy has not been achieved.
02993 !                   if omega /= 0
02994 !                    ier = 6 the input is invalid because
02995 !                            (integr /= 1 and integr /= 2) or
02996 !                             epsabs <= 0
02997 !                             result, abserr, neval, lst are set to
02998 !                             zero.
02999 !                        = 7 abnormal termination of the computation
03000 !                            of one or more subintegrals
03001 !                        = 8 maximum number of cycles allowed
03002 !                            has been achieved, i.e. of subintervals
03003 !                            (a+(k-1)c,a+kc) where
03004 !                            c = (2*int(abs(omega))+1)*pi/abs(omega),
03005 !                            for k = 1, 2, ...
03006 !                        = 9 the extrapolation table constructed for
03007 !                            convergence acceleration of the series
03008 !                            formed by the integral contributions
03009 !                            over the cycles, does not converge to
03010 !                            within the requested accuracy.
03011 !
03012 !  Local parameters:
03013 !
03014 !    Integer LIMLST, gives an upper bound on the number of cycles, LIMLST >= 3.
03015 !    if limlst < 3, the routine will end with ier = 6.
03016 !
03017 !    Integer MAXP1, an upper bound on the number of Chebyshev moments which
03018 !    can be stored, i.e. for the intervals of lengths abs(b-a)*2**(-l),
03019 !    l = 0,1, ..., maxp1-2, maxp1 >= 1.  if maxp1 < 1, the routine will end
03020 !    with ier = 6.
03021 !
03022   implicit none
03023
03024   integer, parameter :: limit = 500
03025   integer, parameter :: limlst = 50
03026   integer, parameter :: maxp1 = 21
03027
03028   real(kind=params_wp) a
03029   real(kind=params_wp) abserr
03030   real(kind=params_wp) alist(limit)
03031   real(kind=params_wp) blist(limit)
```

```
03032   real(kind=params_wp) chebmo(maxp1,25)
03033   real(kind=params_wp) elist(limit)
03034   real(kind=params_wp) epsabs
03035   real(kind=params_wp) erlst(limlst)
03036   real(kind=params_wp), external :: f
03037   integer ier
03038   integer integr
03039   integer iord(limit)
03040   integer ierlst(limlst)
03041   integer lst
03042   integer neval
03043   integer nnlog(limit)
03044   real(kind=params_wp) omega
03045   real(kind=params_wp) result
03046   real(kind=params_wp) rlist(limit)
03047   real(kind=params_wp) rslst(limlst)
03048
03049   ier = 6
03050   neval = 0
03051   result = 0.0e+00
03052   abserr = 0.0e+00
03053
03054   if ( limlst < 3 .or. maxp1 < 1 ) then
03055     return
03056   end if
03057
03058   call qawfe ( f, a, omega, integr, epsabs, limlst, limit, maxp1, &
03059     result, abserr, neval, ier, rslst, erlst, ierlst, lst, alist, blist, &
03060     rlist, elist, iord, nnlog, chebmo )
03061
03062   return
03063 end subroutine qawf
03064 subroutine qawfe ( f, a, omega, integr, epsabs, limlst, limit, maxp1, &
03065   result, abserr, neval, ier, rslst, erlst, ierlst, lst, alist, blist, &
03066   rlist, elist, iord, nnlog, chebmo )
03067
03068 !*****************************************************************************80
03069 !
03070 !! QAWFE computes Fourier integrals.
03071 !
03072 !  Discussion:
03073 !
03074 !    The routine calculates an approximation RESULT to a definite integral
03075 !      I = integral of F*COS(OMEGA*X) or F*SIN(OMEGA*X) over (A,+Infinity),
03076 !    hopefully satisfying
03077 !      || I - RESULT || <= EPSABS.
03078 !
03079 !  Author:
03080 !
03081 !    Robert Piessens, Elise de Doncker-Kapenger,
03082 !    Christian Ueberhuber, David Kahaner
03083 !
03084 !  Reference:
03085 !
03086 !    Robert Piessens, Elise de Doncker-Kapenger,
03087 !    Christian Ueberhuber, David Kahaner,
03088 !    QUADPACK, a Subroutine Package for Automatic Integration,
03089 !    Springer Verlag, 1983
03090 !
03091 !  Parameters:
03092 !
03093 !    Input, external real F, the name of the function routine, of the form
03094 !      function f ( x )
03095 !      real f
03096 !      real x
03097 !    which evaluates the integrand function.
03098 !
03099 !    Input, real A, the lower limit of integration.
03100 !
03101 !    Input, real OMEGA, the parameter in the weight function.
03102 !
03103 !    Input, integer INTEGR, indicates which weight function is used
03104 !    = 1      w(x) = cos(omega*x)
03105 !    = 2      w(x) = sin(omega*x)
03106 !
03107 !    Input, real EPSABS, the absolute accuracy requested.
03108 !
03109 !    Input, integer LIMLST, an upper bound on the number of cycles.
03110 !    LIMLST must be at least 1.  In fact, if LIMLST < 3, the routine
03111 !    will end with IER= 6.
03112 !
03113 !    Input, integer LIMIT, an upper bound on the number of subintervals
03114 !    allowed in the partition of each cycle, limit >= 1.
03115 !
03116 !            maxp1  - integer
03117 !                     gives an upper bound on the number of
03118 !                     Chebyshev moments which can be stored, i.e.
```

```
03119 !                    for the intervals of lengths abs(b-a)*2**(-l),
03120 !                    l=0,1, ..., maxp1-2, maxp1 >= 1
03121 !
03122 !    Output, real RESULT, the estimated value of the integral.
03123 !
03124 !    Output, real ABSERR, an estimate of || I - RESULT ||.
03125 !
03126 !    Output, integer NEVAL, the number of times the integral was evaluated.
03127 !
03128 !            ier    - ier = 0 normal and reliable termination of
03129 !                             the routine. it is assumed that the
03130 !                             requested accuracy has been achieved.
03131 !                     ier > 0 abnormal termination of the routine
03132 !                             the estimates for integral and error
03133 !                             are less reliable. it is assumed that
03134 !                             the requested accuracy has not been
03135 !                             achieved.
03136 !                   if omega /= 0
03137 !                    ier = 6 the input is invalid because
03138 !                             (integr /= 1 and integr /= 2) or
03139 !                              epsabs <= 0 or limlst < 3.
03140 !                              result, abserr, neval, lst are set
03141 !                              to zero.
03142 !                         = 7 bad integrand behavior occurs within
03143 !                             one or more of the cycles. location
03144 !                             and type of the difficulty involved
03145 !                             can be determined from the vector ierlst.
03146 !                             here lst is the number of cycles actually
03147 !                             needed (see below).
03148 !                             ierlst(k) = 1 the maximum number of
03149 !                                           subdivisions (= limit)
03150 !                                           has been achieved on the
03151 !                                           k th cycle.
03152 !                                       = 2 occurence of roundoff
03153 !                                           error is detected and
03154 !                                           prevents the tolerance
03155 !                                           imposed on the k th cycle
03156 !                                           from being acheived.
03157 !                                       = 3 extremely bad integrand
03158 !                                           behavior occurs at some
03159 !                                           points of the k th cycle.
03160 !                                       = 4 the integration procedure
03161 !                                           over the k th cycle does
03162 !                                           not converge (to within the
03163 !                                           required accuracy) due to
03164 !                                           roundoff in the
03165 !                                           extrapolation procedure
03166 !                                           invoked on this cycle. it
03167 !                                           is assumed that the result
03168 !                                           on this interval is the
03169 !                                           best which can be obtained.
03170 !                                       = 5 the integral over the k th
03171 !                                           cycle is probably divergent
03172 !                                           or slowly convergent. it
03173 !                                           must be noted that
03174 !                                           divergence can occur with
03175 !                                           any other value of
03176 !                                           ierlst(k).
03177 !                         = 8 maximum number of  cycles  allowed
03178 !                             has been achieved, i.e. of subintervals
03179 !                             (a+(k-1)c,a+kc) where
03180 !                             c = (2*int(abs(omega))+1)*pi/abs(omega),
03181 !                             for k = 1, 2, ..., lst.
03182 !                             one can allow more cycles by increasing
03183 !                             the value of limlst (and taking the
03184 !                             according dimension adjustments into
03185 !                             account).
03186 !                             examine the array iwork which contains
03187 !                             the error flags over the cycles, in order
03188 !                             to eventual look for local integration
03189 !                             difficulties.
03190 !                             if the position of a local difficulty can
03191 !                             be determined (e.g. singularity,
03192 !                             discontinuity within the interval)
03193 !                             one will probably gain from splitting
03194 !                             up the interval at this point and
03195 !                             calling appopriate integrators on the
03196 !                             subranges.
03197 !                         = 9 the extrapolation table constructed for
03198 !                             convergence acceleration of the series
03199 !                             formed by the integral contributions
03200 !                             over the cycles, does not converge to
03201 !                             within the required accuracy.
03202 !                             as in the case of ier = 8, it is advised
03203 !                             to examine the array iwork which contains
03204 !                             the error flags on the cycles.
03205 !                   if omega = 0 and integr = 1,
```

```
03206 !                    the integral is calculated by means of qagi
03207 !                    and ier = ierlst(1) (with meaning as described
03208 !                    for ierlst(k), k = 1).
03209 !
03210 !            rslst  - real
03211 !                     vector of dimension at least limlst
03212 !                     rslst(k) contains the integral contribution
03213 !                     over the interval (a+(k-1)c,a+kc) where
03214 !                     c = (2*int(abs(omega))+1)*pi/abs(omega),
03215 !                     k = 1, 2, ..., lst.
03216 !                     note that, if omega = 0, rslst(1) contains
03217 !                     the value of the integral over (a,infinity).
03218 !
03219 !            erlst  - real
03220 !                     vector of dimension at least limlst
03221 !                     erlst(k) contains the error estimate
03222 !                     corresponding with rslst(k).
03223 !
03224 !            ierlst - integer
03225 !                     vector of dimension at least limlst
03226 !                     ierlst(k) contains the error flag corresponding
03227 !                     with rslst(k). for the meaning of the local error
03228 !                     flags see description of output parameter ier.
03229 !
03230 !            lst    - integer
03231 !                     number of subintervals needed for the integration
03232 !                     if omega = 0 then lst is set to 1.
03233 !
03234 !            alist, blist, rlist, elist - real
03235 !                     vector of dimension at least limit,
03236 !
03237 !            iord, nnlog - integer
03238 !                     vector of dimension at least limit, providing
03239 !                     space for the quantities needed in the
03240 !                     subdivision process of each cycle
03241 !
03242 !            chebmo - real
03243 !                     array of dimension at least (maxp1,25),
03244 !                     providing space for the Chebyshev moments
03245 !                     needed within the cycles
03246 !
03247 !  Local parameters:
03248 !
03249 !           c1, c2    - end points of subinterval (of length
03250 !                       cycle)
03251 !           cycle     - (2*int(abs(omega))+1)*pi/abs(omega)
03252 !           psum      - vector of dimension at least (limexp+2)
03253 !                       (see routine qextr)
03254 !                       psum contains the part of the epsilon table
03255 !                       which is still needed for further computations.
03256 !                       each element of psum is a partial sum of
03257 !                       the series which should sum to the value of
03258 !                       the integral.
03259 !           errsum    - sum of error estimates over the
03260 !                       subintervals, calculated cumulatively
03261 !           epsa      - absolute tolerance requested over current
03262 !                       subinterval
03263 !           chebmo    - array containing the modified Chebyshev
03264 !                       moments (see also routine qc25o)
03265 !
03266   implicit none
03267
03268   integer limit
03269   integer limlst
03270   integer maxp1
03271
03272   real(kind=params_wp) a
03273   real(kind=params_wp) abseps
03274   real(kind=params_wp) abserr
03275   real(kind=params_wp) alist(limit)
03276   real(kind=params_wp) blist(limit)
03277   real(kind=params_wp) chebmo(maxp1,25)
03278   real(kind=params_wp) correc
03279   real(kind=params_wp) cycle
03280   real(kind=params_wp) c1
03281   real(kind=params_wp) c2
03282   real(kind=params_wp) dl
03283 ! real dla
03284   real(kind=params_wp) drl
03285   real(kind=params_wp) elist(limit)
03286   real(kind=params_wp) ep
03287   real(kind=params_wp) eps
03288   real(kind=params_wp) epsa
03289   real(kind=params_wp) epsabs
03290   real(kind=params_wp) erlst(limlst)
03291   real(kind=params_wp) errsum
03292   real(kind=params_wp), external :: f
```

```
03293   real(kind=params_wp) fact
03294   integer ier
03295   integer ierlst(limlst)
03296   integer integr
03297   integer iord(limit)
03298   integer ktmin
03299   integer l
03300   integer ll
03301   integer lst
03302   integer momcom
03303   integer nev
03304   integer neval
03305   integer nnlog(limit)
03306   integer nres
03307   integer numrl2
03308   real(kind=params_wp) omega
03309   real(kind=params_wp), parameter :: p = 0.9e+00
03310   real(kind=params_wp), parameter :: pi = 3.1415926535897932e+00
03311   real(kind=params_wp) p1
03312   real(kind=params_wp) psum(52)
03313   real(kind=params_wp) reseps
03314   real(kind=params_wp) result
03315   real(kind=params_wp) res3la(3)
03316   real(kind=params_wp) rlist(limit)
03317   real(kind=params_wp) rslst(limlst)
03318 !
03319 !  The dimension of  psum  is determined by the value of
03320 !  limexp in QEXTR (psum must be
03321 !  of dimension (limexp+2) at least).
03322 !
03323 !  Test on validity of parameters.
03324 !
03325   result = 0.0e+00
03326   abserr = 0.0e+00
03327   neval = 0
03328   lst = 0
03329   ier = 0
03330
03331   if ( (integr /= 1 .and. integr /= 2 ) .or. &
03332     epsabs <= 0.0e+00 .or. &
03333     limlst < 3 ) then
03334     ier = 6
03335     return
03336   end if
03337
03338   if ( omega == 0.0e+00 ) then
03339
03340     if ( integr == 1 ) then
03341       call qagi ( f, 0.0e+00_params_wp, 1, epsabs, 0.0e+00_params_wp, result, abserr, neval, ier )
03342     else
03343       result = 0.0e+00
03344       abserr = 0.0e+00
03345       neval = 0
03346       ier = 0
03347     end if
03348
03349     rslst(1) = result
03350     erlst(1) = abserr
03351     ierlst(1) = ier
03352     lst = 1
03353
03354     return
03355   end if
03356 !
03357 !  Initializations.
03358 !
03359   l = int( abs( omega ) )
03360   dl = 2 * l + 1
03361   cycle = dl * pi / abs( omega )
03362   ier = 0
03363   ktmin = 0
03364   neval = 0
03365   numrl2 = 0
03366   nres = 0
03367   c1 = a
03368   c2 = cycle+a
03369   p1 = 1.0e+00_params_wp-p
03370   eps = epsabs
03371
03372   if ( epsabs > tiny( epsabs ) / p1 ) then
03373     eps = epsabs * p1
03374   end if
03375
03376   ep = eps
03377   fact = 1.0e+00_params_wp
03378   correc = 0.0e+00
03379   abserr = 0.0e+00
```

```
03380    errsum = 0.0e+00
03381
03382    do lst = 1, limlst
03383  !
03384  !   Integrate over current subinterval.
03385  !
03386  !    dla = lst
03387       epsa = eps * fact
03388
03389       call qfour ( f, c1, c2, omega, integr, epsa, 0.0e+00_params_wp, limit, lst, maxp1, &
03390         rslst(lst), erlst(lst), nev, ierlst(lst), alist, blist, rlist, elist, &
03391         iord, nnlog, momcom, chebmo )
03392
03393       neval = neval + nev
03394       fact = fact * p
03395       errsum = errsum + erlst(lst)
03396       drl = 5.0e+01 * abs(rslst(lst))
03397  !
03398  !   Test on accuracy with partial sum.
03399  !
03400       if ((errsum+drl) <= epsabs.and.lst >= 6) then
03401         go to 80
03402       end if
03403
03404       correc = max( correc,erlst(lst))
03405
03406       if ( ierlst(lst) /= 0 ) then
03407         eps = max( ep,correc*p1)
03408         ier = 7
03409       end if
03410
03411       if ( ier == 7 .and. (errsum+drl) <= correc*1.0e+01.and. lst > 5) go to 80
03412
03413       numrl2 = numrl2+1
03414
03415       if ( lst <= 1 ) then
03416         psum(1) = rslst(1)
03417         go to 40
03418       end if
03419
03420       psum(numrl2) = psum(ll) + rslst(lst)
03421
03422       if ( lst == 2 ) then
03423         go to 40
03424       end if
03425  !
03426  !   Test on maximum number of subintervals
03427  !
03428       if ( lst == limlst ) then
03429         ier = 8
03430       end if
03431  !
03432  !   Perform new extrapolation
03433  !
03434       call qextr ( numrl2, psum, reseps, abseps, res3la, nres )
03435  !
03436  !   Test whether extrapolated result is influenced by roundoff
03437  !
03438       ktmin = ktmin + 1
03439
03440       if ( ktmin >= 15 .and. abserr <= 1.0e-03 * (errsum+drl) ) then
03441         ier = 9
03442       end  if
03443
03444       if ( abseps <= abserr .or. lst == 3 ) then
03445
03446         abserr = abseps
03447         result = reseps
03448         ktmin = 0
03449  !
03450  !   If IER is not 0, check whether direct result (partial
03451  !   sum) or extrapolated result yields the best integral
03452  !   approximation
03453  !
03454         if ( ( abserr + 1.0e+01 * correc ) <= epsabs ) then
03455           exit
03456         end if
03457
03458         if ( abserr <= epsabs .and. 1.0e+01 * correc >= epsabs ) then
03459           exit
03460         end if
03461
03462       end if
03463
03464       if ( ier /= 0 .and. ier /= 7 ) then
03465         exit
03466       end if
```

```
03467
03468 40  continue
03469
03470     ll = numrl2
03471     c1 = c2
03472     c2 = c2+cycle
03473
03474   end do
03475 !
03476 !  Set final result and error estimate.
03477 !
03478 !60 continue
03479
03480   abserr = abserr + 1.0e+01 * correc
03481
03482   if ( ier == 0 ) then
03483     return
03484   end if
03485
03486   if ( result /= 0.0e+00 .and. psum(numrl2) /= 0.0e+00) go to 70
03487
03488   if ( abserr > errsum ) then
03489     go to 80
03490   end if
03491
03492   if ( psum(numrl2) == 0.0e+00 ) then
03493     return
03494   end if
03495
03496 70 continue
03497
03498   if ( abserr / abs(result) <= (errsum+drl)/abs(psum(numrl2)) ) then
03499
03500     if ( ier >= 1 .and. ier /= 7 ) then
03501       abserr = abserr + drl
03502     end if
03503
03504     return
03505
03506   end if
03507
03508 80 continue
03509
03510   result = psum(numrl2)
03511   abserr = errsum + drl
03512
03513   return
03514 end subroutine qawfe
03515 subroutine qawo ( f, a, b, omega, integr, epsabs, epsrel, result, abserr, &
03516   neval, ier )
03517
03518 !*******************************************************************************80
03519 !
03520 !! QAWO computes the integrals of oscillatory integrands.
03521 !
03522 !  Discussion:
03523 !
03524 !    The routine calculates an approximation RESULT to a given
03525 !    definite integral
03526 !      I = Integral ( A <= X <= B ) F(X) * cos ( OMEGA * X ) dx
03527 !    or
03528 !      I = Integral ( A <= X <= B ) F(X) * sin ( OMEGA * X ) dx
03529 !    hopefully satisfying following claim for accuracy
03530 !      | I - RESULT | <= max ( epsabs, epsrel * |I| ).
03531 !
03532 !  Author:
03533 !
03534 !    Robert Piessens, Elise de Doncker-Kapenger,
03535 !    Christian Ueberhuber, David Kahaner
03536 !
03537 !  Reference:
03538 !
03539 !    Robert Piessens, Elise de Doncker-Kapenger,
03540 !    Christian Ueberhuber, David Kahaner,
03541 !    QUADPACK, a Subroutine Package for Automatic Integration,
03542 !    Springer Verlag, 1983
03543 !
03544 !  Parameters:
03545 !
03546 !    Input, external real F, the name of the function routine, of the form
03547 !      function f ( x )
03548 !      real f
03549 !      real x
03550 !    which evaluates the integrand function.
03551 !
03552 !    Input, real A, B, the limits of integration.
03553 !
```

```
03554 !    Input, real OMEGA, the parameter in the weight function.
03555 !
03556 !    Input, integer INTEGR, specifies the weight function:
03557 !    1, W(X) = cos ( OMEGA * X )
03558 !    2, W(X) = sin ( OMEGA * X )
03559 !
03560 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
03561 !
03562 !    Output, real RESULT, the estimated value of the integral.
03563 !
03564 !    Output, real ABSERR, an estimate of || I - RESULT ||.
03565 !
03566 !    Output, integer NEVAL, the number of times the integral was evaluated.
03567 !
03568 !            ier    - integer
03569 !                     ier = 0 normal and reliable termination of the
03570 !                             routine. it is assumed that the
03571 !                             requested accuracy has been achieved.
03572 !                     - ier > 0 abnormal termination of the routine.
03573 !                             the estimates for integral and error are
03574 !                             less reliable. it is assumed that the
03575 !                             requested accuracy has not been achieved.
03576 !                       ier = 1 maximum number of subdivisions allowed
03577 !                             (= leniw/2) has been achieved. one can
03578 !                             allow more subdivisions by increasing the
03579 !                             value of leniw (and taking the according
03580 !                             dimension adjustments into account).
03581 !                             however, if this yields no improvement it
03582 !                             is advised to analyze the integrand in
03583 !                             order to determine the integration
03584 !                             difficulties. if the position of a local
03585 !                             difficulty can be determined (e.g.
03586 !                             singularity, discontinuity within the
03587 !                             interval) one will probably gain from
03588 !                             splitting up the interval at this point
03589 !                             and calling the integrator on the
03590 !                             subranges. if possible, an appropriate
03591 !                             special-purpose integrator should
03592 !                             be used which is designed for handling
03593 !                             the type of difficulty involved.
03594 !                         = 2 the occurrence of roundoff error is
03595 !                             detected, which prevents the requested
03596 !                             tolerance from being achieved.
03597 !                             the error may be under-estimated.
03598 !                         = 3 extremely bad integrand behavior occurs
03599 !                             at some interior points of the integration
03600 !                             interval.
03601 !                         = 4 the algorithm does not converge. roundoff
03602 !                             error is detected in the extrapolation
03603 !                             table. it is presumed that the requested
03604 !                             tolerance cannot be achieved due to
03605 !                             roundoff in the extrapolation table,
03606 !                             and that the returned result is the best
03607 !                             which can be obtained.
03608 !                         = 5 the integral is probably divergent, or
03609 !                             slowly convergent. it must be noted that
03610 !                             divergence can occur with any other value
03611 !                             of ier.
03612 !                         = 6 the input is invalid, because
03613 !                             epsabs < 0 and epsrel < 0,
03614 !                             result, abserr, neval are set to zero.
03615 !
03616 !  Local parameters:
03617 !
03618 !    limit is the maximum number of subintervals allowed in the
03619 !    subdivision process of QFOUR. take care that limit >= 1.
03620 !
03621 !    maxp1 gives an upper bound on the number of Chebyshev moments
03622 !    which can be stored, i.e. for the intervals of lengths
03623 !    abs(b-a)*2**(-l), l = 0, 1, ... , maxp1-2. take care that
03624 !    maxp1 >= 1.
03625
03626   implicit none
03627
03628   integer, parameter :: limit = 500
03629   integer, parameter :: maxp1 = 21
03630
03631   real(kind=params_wp) a
03632   real(kind=params_wp) abserr
03633   real(kind=params_wp) alist(limit)
03634   real(kind=params_wp) b
03635   real(kind=params_wp) blist(limit)
03636   real(kind=params_wp) chebmo(maxp1,25)
03637   real(kind=params_wp) elist(limit)
03638   real(kind=params_wp) epsabs
03639   real(kind=params_wp) epsrel
03640   real(kind=params_wp), external :: f
```

```
03641   integer ier
03642   integer integr
03643   integer iord(limit)
03644   integer momcom
03645   integer neval
03646   integer nnlog(limit)
03647   real(kind=params_wp) omega
03648   real(kind=params_wp) result
03649   real(kind=params_wp) rlist(limit)
03650
03651   call qfour ( f, a, b, omega, integr, epsabs, epsrel, limit, 1, maxp1, &
03652     result, abserr, neval, ier, alist, blist, rlist, elist, iord, nnlog, &
03653     momcom, chebmo )
03654
03655   return
03656 end subroutine qawo
03657 subroutine qaws ( f, a, b, alfa, beta, integr, epsabs, epsrel, result, &
03658   abserr, neval, ier )
03659
03660 !*****************************************************************************80
03661 !
03662 !! QAWS estimates integrals with algebraico-logarithmic endpoint singularities.
03663 !
03664 !  Discussion:
03665 !
03666 !    This routine calculates an approximation RESULT to a given
03667 !    definite integral
03668 !      I = integral of f*w over (a,b)
03669 !    where w shows a singular behavior at the end points, see parameter
03670 !    integr, hopefully satisfying following claim for accuracy
03671 !      abs(i-result) <= max(epsabs,epsrel*abs(i)).
03672 !
03673 !  Author:
03674 !
03675 !    Robert Piessens, Elise de Doncker-Kapenger,
03676 !    Christian Ueberhuber, David Kahaner
03677 !
03678 !  Reference:
03679 !
03680 !    Robert Piessens, Elise de Doncker-Kapenger,
03681 !    Christian Ueberhuber, David Kahaner,
03682 !    QUADPACK, a Subroutine Package for Automatic Integration,
03683 !    Springer Verlag, 1983
03684 !
03685 !  Parameters:
03686 !
03687 !    Input, external real F, the name of the function routine, of the form
03688 !      function f ( x )
03689 !      real f
03690 !      real x
03691 !    which evaluates the integrand function.
03692 !
03693 !    Input, real A, B, the limits of integration.
03694 !
03695 !    Input, real ALFA, BETA, parameters used in the weight function.
03696 !    ALFA and BETA should be greater than -1.
03697 !
03698 !    Input, integer INTEGR, indicates which weight function is to be used
03699 !    = 1  (x-a)**alfa*(b-x)**beta
03700 !    = 2  (x-a)**alfa*(b-x)**beta*log(x-a)
03701 !    = 3  (x-a)**alfa*(b-x)**beta*log(b-x)
03702 !    = 4  (x-a)**alfa*(b-x)**beta*log(x-a)*log(b-x)
03703 !
03704 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
03705 !
03706 !    Output, real RESULT, the estimated value of the integral.
03707 !
03708 !    Output, real ABSERR, an estimate of || I - RESULT ||.
03709 !
03710 !    Output, integer NEVAL, the number of times the integral was evaluated.
03711 !
03712 !            ier    - integer
03713 !                     ier = 0 normal and reliable termination of the
03714 !                             routine. it is assumed that the requested
03715 !                             accuracy has been achieved.
03716 !                     ier > 0 abnormal termination of the routine
03717 !                             the estimates for the integral and error
03718 !                             are less reliable. it is assumed that the
03719 !                             requested accuracy has not been achieved.
03720 !                     ier = 1 maximum number of subdivisions allowed
03721 !                             has been achieved. one can allow more
03722 !                             subdivisions by increasing the data value
03723 !                             of limit in qaws (and taking the according
03724 !                             dimension adjustments into account).
03725 !                             however, if this yields no improvement it
03726 !                             is advised to analyze the integrand, in
03727 !                             order to determine the integration
```

```
03728 !                            difficulties which prevent the requested
03729 !                            tolerance from being achieved. in case of
03730 !                            a jump discontinuity or a local
03731 !                            singularity of algebraico-logarithmic type
03732 !                            at one or more interior points of the
03733 !                            integration range, one should proceed by
03734 !                            splitting up the interval at these points
03735 !                            and calling the integrator on the
03736 !                            subranges.
03737 !                        = 2 the occurrence of roundoff error is
03738 !                            detected, which prevents the requested
03739 !                            tolerance from being achieved.
03740 !                        = 3 extremely bad integrand behavior occurs
03741 !                            at some points of the integration
03742 !                            interval.
03743 !                        = 6 the input is invalid, because
03744 !                            b <= a or alfa <= (-1) or beta <= (-1) or
03745 !                            integr < 1 or integr > 4 or
03746 !                            epsabs < 0 and epsrel < 0,
03747 !                            result, abserr, neval are set to zero.
03748 !
03749 !  Local parameters:
03750 !
03751 !    LIMIT is the maximum number of subintervals allowed in the
03752 !    subdivision process of qawse. take care that limit >= 2.
03753 !
03754   implicit none
03755
03756   integer, parameter :: limit = 500
03757
03758   real(kind=params_wp) a
03759   real(kind=params_wp) abserr
03760   real(kind=params_wp) alfa
03761   real(kind=params_wp) alist(limit)
03762   real(kind=params_wp) b
03763   real(kind=params_wp) blist(limit)
03764   real(kind=params_wp) beta
03765   real(kind=params_wp) elist(limit)
03766   real(kind=params_wp) epsabs
03767   real(kind=params_wp) epsrel
03768   real(kind=params_wp), external :: f
03769   integer ier
03770   integer integr
03771   integer iord(limit)
03772   integer last
03773   integer neval
03774   real(kind=params_wp) result
03775   real(kind=params_wp) rlist(limit)
03776
03777   call qawse ( f, a, b, alfa, beta, integr, epsabs, epsrel, limit, result, &
03778     abserr, neval, ier, alist, blist, rlist, elist, iord, last )
03779
03780   return
03781 end subroutine qaws
03782 subroutine qawse ( f, a, b, alfa, beta, integr, epsabs, epsrel, limit, &
03783   result, abserr, neval, ier, alist, blist, rlist, elist, iord, last )
03784
03785 !*****************************************************************************80
03786 !
03787 !! QAWSE estimates integrals with algebraico-logarithmic endpoint singularities.
03788 !
03789 !  Discussion:
03790 !
03791 !    This routine calculates an approximation RESULT to an integral
03792 !      I = integral of F(X) * W(X) over (a,b),
03793 !    where W(X) shows a singular behavior at the endpoints, hopefully
03794 !    satisfying:
03795 !      | I - RESULT | <= max ( epsabs, epsrel * |I| ).
03796 !
03797 !  Author:
03798 !
03799 !    Robert Piessens, Elise de Doncker-Kapenger,
03800 !    Christian Ueberhuber, David Kahaner
03801 !
03802 !  Reference:
03803 !
03804 !    Robert Piessens, Elise de Doncker-Kapenger,
03805 !    Christian Ueberhuber, David Kahaner,
03806 !    QUADPACK, a Subroutine Package for Automatic Integration,
03807 !    Springer Verlag, 1983
03808 !
03809 !  Parameters:
03810 !
03811 !    Input, external real F, the name of the function routine, of the form
03812 !      function f ( x )
03813 !      real f
03814 !      real x
```

```
03815 !    which evaluates the integrand function.
03816 !
03817 !    Input, real A, B, the limits of integration.
03818 !
03819 !    Input, real ALFA, BETA, parameters used in the weight function.
03820 !    ALFA and BETA should be greater than -1.
03821 !
03822 !    Input, integer INTEGR, indicates which weight function is used:
03823 !    = 1  (x-a)**alfa*(b-x)**beta
03824 !    = 2  (x-a)**alfa*(b-x)**beta*log(x-a)
03825 !    = 3  (x-a)**alfa*(b-x)**beta*log(b-x)
03826 !    = 4  (x-a)**alfa*(b-x)**beta*log(x-a)*log(b-x)
03827 !
03828 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
03829 !
03830 !    Input, integer LIMIT, an upper bound on the number of subintervals
03831 !    in the partition of (A,B), LIMIT >= 2.  If LIMIT < 2, the routine
03832 !     will end with IER = 6.
03833 !
03834 !    Output, real RESULT, the estimated value of the integral.
03835 !
03836 !    Output, real ABSERR, an estimate of || I - RESULT ||.
03837 !
03838 !    Output, integer NEVAL, the number of times the integral was evaluated.
03839 !
03840 !            ier    - integer
03841 !                     ier = 0 normal and reliable termination of the
03842 !                             routine. it is assumed that the requested
03843 !                             accuracy has been achieved.
03844 !                     ier > 0 abnormal termination of the routine
03845 !                             the estimates for the integral and error
03846 !                             are less reliable. it is assumed that the
03847 !                             requested accuracy has not been achieved.
03848 !                         = 1 maximum number of subdivisions allowed
03849 !                             has been achieved. one can allow more
03850 !                             subdivisions by increasing the value of
03851 !                             limit. however, if this yields no
03852 !                             improvement it is advised to analyze the
03853 !                             integrand, in order to determine the
03854 !                             integration difficulties which prevent
03855 !                             the requested tolerance from being
03856 !                             achieved. in case of a jump discontinuity
03857 !                             or a local singularity of algebraico-
03858 !                             logarithmic type at one or more interior
03859 !                             points of the integration range, one
03860 !                             should proceed by splitting up the
03861 !                             interval at these points and calling the
03862 !                             integrator on the subranges.
03863 !                         = 2 the occurrence of roundoff error is
03864 !                             detected, which prevents the requested
03865 !                             tolerance from being achieved.
03866 !                         = 3 extremely bad integrand behavior occurs
03867 !                             at some points of the integration
03868 !                             interval.
03869 !                         = 6 the input is invalid, because
03870 !                             b <= a or alfa <= (-1) or beta <= (-1) or
03871 !                             integr < 1 or integr > 4, or
03872 !                             epsabs < 0 and epsrel < 0,
03873 !                             or limit < 2.
03874 !                             result, abserr, neval, rlist(1), elist(1),
03875 !                             iord(1) and last are set to zero.
03876 !                             alist(1) and blist(1) are set to a and b
03877 !                             respectively.
03878 !
03879 !    Workspace, real ALIST(LIMIT), BLIST(LIMIT), contains in entries 1
03880 !    through LAST the left and right ends of the partition subintervals.
03881 !
03882 !    Workspace, real RLIST(LIMIT), contains in entries 1 through LAST
03883 !    the integral approximations on the subintervals.
03884 !
03885 !    Workspace, real ELIST(LIMIT), contains in entries 1 through LAST
03886 !    the absolute error estimates on the subintervals.
03887 !
03888 !            iord   - integer
03889 !                     vector of dimension at least limit, the first k
03890 !                     elements of which are pointers to the error
03891 !                     estimates over the subintervals, so that
03892 !                     elist(iord(1)), ..., elist(iord(k)) with k = last
03893 !                     if last <= (limit/2+2), and k = limit+1-last
03894 !                     otherwise, form a decreasing sequence.
03895 !
03896 !    Output, integer LAST, the number of subintervals actually produced in
03897 !    the subdivision process.
03898 !
03899 !  Local parameters:
03900 !
03901 !           alist     - list of left end points of all subintervals
```

```
03902 !                        considered up to now
03903 !           blist     - list of right end points of all subintervals
03904 !                        considered up to now
03905 !           rlist(i)  - approximation to the integral over
03906 !                        (alist(i),blist(i))
03907 !           elist(i)  - error estimate applying to rlist(i)
03908 !           maxerr    - pointer to the interval with largest error
03909 !                        estimate
03910 !           errmax    - elist(maxerr)
03911 !           area      - sum of the integrals over the subintervals
03912 !           errsum    - sum of the errors over the subintervals
03913 !           errbnd    - requested accuracy max(epsabs,epsrel*
03914 !                        abs(result))
03915 !           *****1    - variable for the left subinterval
03916 !           *****2    - variable for the right subinterval
03917 !           last      - index for subdivision
03918 !
03919   implicit none
03920
03921   integer limit
03922
03923   real(kind=params_wp) a
03924   real(kind=params_wp) abserr
03925   real(kind=params_wp) alfa
03926   real(kind=params_wp) alist(limit)
03927   real(kind=params_wp) area
03928   real(kind=params_wp) area1
03929   real(kind=params_wp) area12
03930   real(kind=params_wp) area2
03931   real(kind=params_wp) a1
03932   real(kind=params_wp) a2
03933   real(kind=params_wp) b
03934   real(kind=params_wp) beta
03935   real(kind=params_wp) blist(limit)
03936   real(kind=params_wp) b1
03937   real(kind=params_wp) b2
03938   real(kind=params_wp) centre
03939   real(kind=params_wp) elist(limit)
03940   real(kind=params_wp) epsabs
03941   real(kind=params_wp) epsrel
03942   real(kind=params_wp) errbnd
03943   real(kind=params_wp) errmax
03944   real(kind=params_wp) error1
03945   real(kind=params_wp) erro12
03946   real(kind=params_wp) error2
03947   real(kind=params_wp) errsum
03948   real(kind=params_wp), external :: f
03949   integer ier
03950   integer integr
03951   integer iord(limit)
03952   integer iroff1
03953   integer iroff2
03954   integer last
03955   integer maxerr
03956   integer nev
03957   integer neval
03958   integer nrmax
03959   real(kind=params_wp) resas1
03960   real(kind=params_wp) resas2
03961   real(kind=params_wp) result
03962   real(kind=params_wp) rg(25)
03963   real(kind=params_wp) rh(25)
03964   real(kind=params_wp) ri(25)
03965   real(kind=params_wp) rj(25)
03966   real(kind=params_wp) rlist(limit)
03967 !
03968 !  Test on validity of parameters.
03969 !
03970   ier = 0
03971   neval = 0
03972   last = 0
03973   rlist(1) = 0.0e+00
03974   elist(1) = 0.0e+00
03975   iord(1) = 0
03976   result = 0.0e+00
03977   abserr = 0.0e+00
03978
03979   if ( b <= a .or. &
03980     (epsabs < 0.0e+00 .and. epsrel < 0.0e+00) .or. &
03981     alfa <= (-1.0e+00_params_wp) .or. &
03982     beta <= (-1.0e+00_params_wp) .or. &
03983     integr < 1 .or. &
03984     integr > 4 .or. &
03985     limit < 2 ) then
03986     ier = 6
03987     return
03988   end if
```

```
03989 !
03990 !  Compute the modified Chebyshev moments.
03991 !
03992   call qmomo ( alfa, beta, ri, rj, rg, rh, integr )
03993 !
03994 !  Integrate over the intervals (a,(a+b)/2) and ((a+b)/2,b).
03995 !
03996   centre = 5.0e-01 * ( b + a )
03997
03998   call qc25s ( f, a, b, a, centre, alfa, beta, ri, rj, rg, rh, area1, &
03999     error1, resas1, integr, nev )
04000
04001   neval = nev
04002
04003   call qc25s ( f, a, b, centre, b, alfa, beta, ri, rj, rg, rh, area2, &
04004     error2, resas2, integr, nev )
04005
04006   last = 2
04007   neval = neval+nev
04008   result = area1+area2
04009   abserr = error1+error2
04010 !
04011 !  Test on accuracy.
04012 !
04013   errbnd = max( epsabs, epsrel * abs( result ) )
04014 !
04015 !  Initialization.
04016 !
04017   if ( error2 <= error1 ) then
04018     alist(1) = a
04019     alist(2) = centre
04020     blist(1) = centre
04021     blist(2) = b
04022     rlist(1) = area1
04023     rlist(2) = area2
04024     elist(1) = error1
04025     elist(2) = error2
04026   else
04027     alist(1) = centre
04028     alist(2) = a
04029     blist(1) = b
04030     blist(2) = centre
04031     rlist(1) = area2
04032     rlist(2) = area1
04033     elist(1) = error2
04034     elist(2) = error1
04035   end if
04036
04037   iord(1) = 1
04038   iord(2) = 2
04039
04040   if ( limit == 2 ) then
04041     ier = 1
04042     return
04043   end if
04044
04045   if ( abserr <= errbnd ) then
04046     return
04047   end if
04048
04049   errmax = elist(1)
04050   maxerr = 1
04051   nrmax = 1
04052   area = result
04053   errsum = abserr
04054   iroff1 = 0
04055   iroff2 = 0
04056
04057   do last = 3, limit
04058 !
04059 !  Bisect the subinterval with largest error estimate.
04060 !
04061     a1 = alist(maxerr)
04062     b1 = 5.0e-01 * ( alist(maxerr) + blist(maxerr) )
04063     a2 = b1
04064     b2 = blist(maxerr)
04065
04066     call qc25s ( f, a, b, a1, b1, alfa, beta, ri, rj, rg, rh, area1, &
04067       error1, resas1, integr, nev )
04068
04069     neval = neval + nev
04070
04071     call qc25s ( f, a, b, a2, b2, alfa, beta, ri, rj, rg, rh, area2, &
04072       error2, resas2, integr, nev )
04073
04074     neval = neval + nev
04075 !
```

```
04076 !  Improve previous approximations integral and error and
04077 !  test for accuracy.
04078 !
04079     area12 = area1+area2
04080     erro12 = error1+error2
04081     errsum = errsum+erro12-errmax
04082     area = area+area12-rlist(maxerr)
04083 !
04084 !  Test for roundoff error.
04085 !
04086     if ( a /= a1 .and. b /= b2 ) then
04087
04088       if ( resas1 /= error1 .and. resas2 /= error2 ) then
04089
04090         if ( abs( rlist(maxerr) - area12 ) < 1.0e-05 * abs( area12 ) &
04091           .and.erro12 >= 9.9e-01*errmax ) then
04092           iroff1 = iroff1 + 1
04093         end if
04094
04095         if ( last > 10 .and. erro12 > errmax ) then
04096           iroff2 = iroff2 + 1
04097         end if
04098
04099       end if
04100
04101     end if
04102
04103     rlist(maxerr) = area1
04104     rlist(last) = area2
04105 !
04106 !  Test on accuracy.
04107 !
04108     errbnd = max( epsabs, epsrel * abs( area ) )
04109
04110     if ( errsum > errbnd ) then
04111 !
04112 !  Set error flag in the case that the number of interval
04113 !  bisections exceeds limit.
04114 !
04115       if ( last == limit ) then
04116         ier = 1
04117       end if
04118 !
04119 !  Set error flag in the case of roundoff error.
04120 !
04121       if ( iroff1 >= 6 .or. iroff2 >= 20 ) then
04122         ier = 2
04123      end if
04124 !
04125 !  Set error flag in the case of bad integrand behavior
04126 !  at interior points of integration range.
04127 !
04128       if ( max( abs(a1),abs(b2)) <= (1.0e+00_params_wp+1.0e+03* epsilon( a1 ) )* &
04129         ( abs(a2) + 1.0e+03* tiny( a2) )) then
04130         ier = 3
04131       end if
04132
04133     end if
04134 !
04135 !  Append the newly-created intervals to the list.
04136 !
04137     if ( error2 <= error1 ) then
04138       alist(last) = a2
04139       blist(maxerr) = b1
04140       blist(last) = b2
04141       elist(maxerr) = error1
04142       elist(last) = error2
04143     else
04144       alist(maxerr) = a2
04145       alist(last) = a1
04146       blist(last) = b1
04147       rlist(maxerr) = area2
04148       rlist(last) = area1
04149       elist(maxerr) = error2
04150       elist(last) = error1
04151     end if
04152 !
04153 !  Call QSORT to maintain the descending ordering
04154 !  in the list of error estimates and select the subinterval
04155 !  with largest error estimate (to be bisected next).
04156 !
04157     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
04158
04159     if ( ier /= 0 .or. errsum <= errbnd ) then
04160       exit
04161     end if
04162
```

```
04163   end do
04164 !
04165 !  Compute final result.
04166 !
04167   result = sum( rlist(1:last) )
04168
04169   abserr = errsum
04170
04171   return
04172 end subroutine qawse
04173 subroutine qc25c ( f, a, b, c, result, abserr, krul, neval )
04174
04175 !*****************************************************************************80
04176 !
04177 !! QC25C returns integration rules for Cauchy Principal Value integrals.
04178 !
04179 !  Discussion:
04180 !
04181 !    This routine estimates
04182 !      I = integral of F(X) * W(X) over (a,b)
04183 !    with error estimate, where
04184 !      w(x) = 1/(x-c)
04185 !
04186 !  Author:
04187 !
04188 !    Robert Piessens, Elise de Doncker-Kapenger,
04189 !    Christian Ueberhuber, David Kahaner
04190 !
04191 !  Reference:
04192 !
04193 !    Robert Piessens, Elise de Doncker-Kapenger,
04194 !    Christian Ueberhuber, David Kahaner,
04195 !    QUADPACK, a Subroutine Package for Automatic Integration,
04196 !    Springer Verlag, 1983
04197 !
04198 !  Parameters:
04199 !
04200 !    Input, external real F, the name of the function routine, of the form
04201 !      function f ( x )
04202 !      real f
04203 !      real x
04204 !    which evaluates the integrand function.
04205 !
04206 !    Input, real A, B, the limits of integration.
04207 !
04208 !    Input, real C, the parameter in the weight function.
04209 !
04210 !    Output, real RESULT, the estimated value of the integral.
04211 !    RESULT is computed by using a generalized Clenshaw-Curtis method if
04212 !    C lies within ten percent of the integration interval.  In the
04213 !    other case the 15-point Kronrod rule obtained by optimal addition
04214 !    of abscissae to the 7-point Gauss rule, is applied.
04215 !
04216 !    Output, real ABSERR, an estimate of || I - RESULT ||.
04217 !
04218 !          krul   - integer
04219 !                   key which is decreased by 1 if the 15-point
04220 !                   Gauss-Kronrod scheme has been used
04221 !
04222 !    Output, integer NEVAL, the number of times the integral was evaluated.
04223 !
04224 !  Local parameters:
04225 !
04226 !          fval   - value of the function f at the points
04227 !                   cos(k*pi/24),  k = 0, ..., 24
04228 !          cheb12 - Chebyshev series expansion coefficients, for the
04229 !                   function f, of degree 12
04230 !          cheb24 - Chebyshev series expansion coefficients, for the
04231 !                   function f, of degree 24
04232 !          res12  - approximation to the integral corresponding to the
04233 !                   use of cheb12
04234 !          res24  - approximation to the integral corresponding to the
04235 !                   use of cheb24
04236 !          qwgtc  - external function subprogram defining the weight
04237 !                   function
04238 !          hlgth  - half-length of the interval
04239 !          centr  - mid point of the interval
04240 !
04241   implicit none
04242
04243   real(kind=params_wp) a
04244   real(kind=params_wp) abserr
04245   real(kind=params_wp) ak22
04246   real(kind=params_wp) amom0
04247   real(kind=params_wp) amom1
04248   real(kind=params_wp) amom2
04249   real(kind=params_wp) b
```

```
04250   real(kind=params_wp) c
04251   real(kind=params_wp) cc
04252   real(kind=params_wp) centr
04253   real(kind=params_wp) cheb12(13)
04254   real(kind=params_wp) cheb24(25)
04255   real(kind=params_wp), external :: f
04256   real(kind=params_wp) fval(25)
04257   real(kind=params_wp) hlgth
04258   integer i
04259   integer isym
04260   integer k
04261   integer kp
04262   integer krul
04263   integer neval
04264   real(kind=params_wp) p2
04265   real(kind=params_wp) p3
04266   real(kind=params_wp) p4
04267 !  real(kind=params_wp), external :: qwgtc
04268   real(kind=params_wp) resabs
04269   real(kind=params_wp) resasc
04270   real(kind=params_wp) result
04271   real(kind=params_wp) res12
04272   real(kind=params_wp) res24
04273   real(kind=params_wp) u
04274   real(kind=params_wp), parameter, dimension ( 11 ) :: x = (/ &
04275     9.914448613738104e-01, 9.659258262890683e-01, &
04276     9.238795325112868e-01, 8.660254037844386e-01, &
04277     7.933533402912352e-01, 7.071067811865475e-01, &
04278     6.087614290087206e-01, 5.000000000000000e-01, &
04279     3.826834323650898e-01, 2.588190451025208e-01, &
04280     1.305261922200516e-01 /)
04281 !
04282 !  Check the position of C.
04283 !
04284   cc = ( 2.0e+00 * c - b - a ) / ( b - a )
04285 !
04286 !  Apply the 15-point Gauss-Kronrod scheme.
04287 !
04288   if ( abs( cc ) >= 1.1e+00 ) then
04289     krul = krul - 1
04290     call qk15w ( f, qwgtc, c, p2, p3, p4, kp, a, b, result, abserr, &
04291       resabs, resasc )
04292     neval = 15
04293     if ( resasc == abserr ) then
04294       krul = krul+1
04295     end if
04296     return
04297   end if
04298 !
04299 !  Use the generalized Clenshaw-Curtis method.
04300 !
04301   hlgth = 5.0e-01 * ( b - a )
04302   centr = 5.0e-01 * ( b + a )
04303   neval = 25
04304   fval(1) = 5.0e-01 * f(hlgth+centr)
04305   fval(13) = f(centr)
04306   fval(25) = 5.0e-01 * f(centr-hlgth)
04307
04308   do i = 2, 12
04309     u = hlgth * x(i-1)
04310     isym = 26 - i
04311     fval(i) = f(u+centr)
04312     fval(isym) = f(centr-u)
04313   end do
04314 !
04315 !  Compute the Chebyshev series expansion.
04316 !
04317   call qcheb ( x, fval, cheb12, cheb24 )
04318 !
04319 !  The modified Chebyshev moments are computed by forward
04320 !  recursion, using AMOM0 and AMOM1 as starting values.
04321 !
04322   amom0 = log( abs( ( 1.0e+00_params_wp - cc ) / ( 1.0e+00_params_wp + cc ) ) )
04323   amom1 = 2.0e+00 + cc * amom0
04324   res12 = cheb12(1) * amom0 + cheb12(2) * amom1
04325   res24 = cheb24(1) * amom0 + cheb24(2) * amom1
04326
04327   do k = 3, 13
04328     amom2 = 2.0e+00 * cc * amom1 - amom0
04329     ak22 = ( k - 2 ) * ( k - 2 )
04330     if ( ( k / 2 ) * 2 == k ) then
04331       amom2 = amom2 - 4.0e+00 / ( ak22 - 1.0e+00_params_wp )
04332     end if
04333     res12 = res12 + cheb12(k) * amom2
04334     res24 = res24 + cheb24(k) * amom2
04335     amom0 = amom1
04336     amom1 = amom2
```

```
04337   end do
04338
04339   do k = 14, 25
04340     amom2 = 2.0e+00 * cc * amom1 - amom0
04341     ak22 = ( k - 2 ) * ( k - 2 )
04342     if ( ( k / 2 ) * 2 == k ) then
04343       amom2 = amom2 - 4.0e+00 / ( ak22 - 1.0e+00_params_wp )
04344     end if
04345     res24 = res24 + cheb24(k) * amom2
04346     amom0 = amom1
04347     amom1 = amom2
04348   end do
04349
04350   result = res24
04351   abserr = abs( res24 - res12 )
04352
04353   return
04354 end subroutine qc25c
04355 subroutine qc25o ( f, a, b, omega, integr, nrmom, maxp1, ksave, result, &
04356   abserr, neval, resabs, resasc, momcom, chebmo )
04357
04358 !*****************************************************************************80
04359 !
04360 !! QC25O returns integration rules for integrands with a COS or SIN factor.
04361 !
04362 !  Discussion:
04363 !
04364 !    This routine estimates the integral
04365 !      I = integral of f(x) * w(x) over (a,b)
04366 !    where
04367 !      w(x) = cos(omega*x)
04368 !    or
04369 !      w(x) = sin(omega*x),
04370 !    and estimates
04371 !      J = integral ( A <= X <= B ) |F(X)| dx.
04372 !
04373 !    For small values of OMEGA or small intervals (a,b) the 15-point
04374 !    Gauss-Kronrod rule is used.  In all other cases a generalized
04375 !    Clenshaw-Curtis method is used, that is, a truncated Chebyshev
04376 !    expansion of the function F is computed on (a,b), so that the
04377 !    integrand can be written as a sum of terms of the form W(X)*T(K,X),
04378 !    where T(K,X) is the Chebyshev polynomial of degree K.  The Chebyshev
04379 !    moments are computed with use of a linear recurrence relation.
04380 !
04381 !  Author:
04382 !
04383 !    Robert Piessens, Elise de Doncker-Kapenger,
04384 !    Christian Ueberhuber, David Kahaner
04385 !
04386 !  Reference:
04387 !
04388 !    Robert Piessens, Elise de Doncker-Kapenger,
04389 !    Christian Ueberhuber, David Kahaner,
04390 !    QUADPACK, a Subroutine Package for Automatic Integration,
04391 !    Springer Verlag, 1983
04392 !
04393 !  Parameters:
04394 !
04395 !    Input, external real F, the name of the function routine, of the form
04396 !      function f ( x )
04397 !      real f
04398 !      real x
04399 !    which evaluates the integrand function.
04400 !
04401 !    Input, real A, B, the limits of integration.
04402 !
04403 !    Input, real OMEGA, the parameter in the weight function.
04404 !
04405 !    Input, integer INTEGR, indicates which weight function is to be used
04406 !    = 1, w(x) = cos(omega*x)
04407 !    = 2, w(x) = sin(omega*x)
04408 !
04409 !    ?, integer NRMOM, the length of interval (a,b) is equal to the length
04410 !    of the original integration interval divided by
04411 !    2**nrmom (we suppose that the routine is used in an
04412 !    adaptive integration process, otherwise set
04413 !    nrmom = 0).  nrmom must be zero at the first call.
04414 !
04415 !           maxp1  - integer
04416 !                    gives an upper bound on the number of Chebyshev
04417 !                    moments which can be stored, i.e. for the intervals
04418 !                    of lengths abs(bb-aa)*2**(-l), l = 0,1,2, ...,
04419 !                    maxp1-2.
04420 !
04421 !           ksave  - integer
04422 !                    key which is one when the moments for the
04423 !                    current interval have been computed
```

```
04424 !
04425 !    Output, real RESULT, the estimated value of the integral.
04426 !
04427 !           abserr - real
04428 !                    estimate of the modulus of the absolute
04429 !                    error, which should equal or exceed abs(i-result)
04430 !
04431 !    Output, integer NEVAL, the number of times the integral was evaluated.
04432 !
04433 !    Output, real RESABS, approximation to the integral J.
04434 !
04435 !    Output, real RESASC, approximation to the integral of abs(F-I/(B-A)).
04436 !
04437 !         on entry and return
04438 !           momcom - integer
04439 !                    for each interval length we need to compute
04440 !                    the Chebyshev moments. momcom counts the number
04441 !                    of intervals for which these moments have already
04442 !                    been computed. if nrmom < momcom or ksave = 1,
04443 !                    the Chebyshev moments for the interval (a,b)
04444 !                    have already been computed and stored, otherwise
04445 !                    we compute them and we increase momcom.
04446 !
04447 !           chebmo - real
04448 !                    array of dimension at least (maxp1,25) containing
04449 !                    the modified Chebyshev moments for the first momcom
04450 !                    interval lengths
04451 !
04452 !  Local parameters:
04453 !
04454 !    maxp1 gives an upper bound
04455 !           on the number of Chebyshev moments which can be
04456 !           computed, i.e. for the interval (bb-aa), ...,
04457 !           (bb-aa)/2**(maxp1-2).
04458 !           should this number be altered, the first dimension of
04459 !           chebmo needs to be adapted.
04460 !
04461 !    x contains the values cos(k*pi/24)
04462 !           k = 1, ...,11, to be used for the Chebyshev expansion of f
04463 !
04464 !           centr  - mid point of the integration interval
04465 !           hlgth  - half length of the integration interval
04466 !           fval   - value of the function f at the points
04467 !                    (b-a)*0.5*cos(k*pi/12) + (b+a)*0.5
04468 !                    k = 0, ...,24
04469 !           cheb12 - coefficients of the Chebyshev series expansion
04470 !                    of degree 12, for the function f, in the
04471 !                    interval (a,b)
04472 !           cheb24 - coefficients of the Chebyshev series expansion
04473 !                    of degree 24, for the function f, in the
04474 !                    interval (a,b)
04475 !           resc12 - approximation to the integral of
04476 !                    cos(0.5*(b-a)*omega*x)*f(0.5*(b-a)*x+0.5*(b+a))
04477 !                    over (-1,+1), using the Chebyshev series
04478 !                    expansion of degree 12
04479 !           resc24 - approximation to the same integral, using the
04480 !                    Chebyshev series expansion of degree 24
04481 !           ress12 - the analogue of resc12 for the sine
04482 !           ress24 - the analogue of resc24 for the sine
04483 !
04484   implicit none
04485
04486   integer maxp1
04487
04488   real(kind=params_wp) a
04489   real(kind=params_wp) abserr
04490   real(kind=params_wp) ac
04491   real(kind=params_wp) an
04492   real(kind=params_wp) an2
04493   real(kind=params_wp) as
04494   real(kind=params_wp) asap
04495   real(kind=params_wp) ass
04496   real(kind=params_wp) b
04497   real(kind=params_wp) centr
04498   real(kind=params_wp) chebmo(maxp1,25)
04499   real(kind=params_wp) cheb12(13)
04500   real(kind=params_wp) cheb24(25)
04501   real(kind=params_wp) conc
04502   real(kind=params_wp) cons
04503   real(kind=params_wp) cospar
04504   real(kind=params_wp) d(28)
04505   real(kind=params_wp) d1(28)
04506   real(kind=params_wp) d2(28)
04507   real(kind=params_wp) d3(28)
04508   real(kind=params_wp) estc
04509   real(kind=params_wp) ests
04510   real(kind=params_wp), external :: f
```

```
04511   real(kind=params_wp) fval(25)
04512   real(kind=params_wp) hlgth
04513   integer i
04514   integer integr
04515   integer isym
04516   integer j
04517   integer k
04518   integer ksave
04519   integer m
04520   integer momcom
04521   integer neval
04522   integer, parameter :: nmac = 28
04523   integer noeql
04524   integer noequ
04525   integer nrmom
04526   real(kind=params_wp) omega
04527   real(kind=params_wp) parint
04528   real(kind=params_wp) par2
04529   real(kind=params_wp) par22
04530   real(kind=params_wp) p2
04531   real(kind=params_wp) p3
04532   real(kind=params_wp) p4
04533 !  real(kind=params_wp), external :: qwgto
04534   real(kind=params_wp) resabs
04535   real(kind=params_wp) resasc
04536   real(kind=params_wp) resc12
04537   real(kind=params_wp) resc24
04538   real(kind=params_wp) ress12
04539   real(kind=params_wp) ress24
04540   real(kind=params_wp) result
04541   real(kind=params_wp) sinpar
04542   real(kind=params_wp) v(28)
04543   real(kind=params_wp), dimension ( 11 ) :: x = (/ &
04544     9.914448613738104e-01,     9.659258262890683e-01, &
04545     9.238795325112868e-01,     8.660254037844386e-01, &
04546     7.933533402912352e-01,     7.071067811865475e-01, &
04547     6.087614290087206e-01,     5.000000000000000e-01, &
04548     3.826834323650898e-01,     2.588190451025208e-01, &
04549     1.305261922200516e-01 /)
04550
04551   centr = 5.0e-01 * ( b + a )
04552   hlgth = 5.0e-01 * ( b - a )
04553   parint = omega * hlgth
04554 !
04555 !  Compute the integral using the 15-point Gauss-Kronrod
04556 !  formula if the value of the parameter in the integrand
04557 !  is small or if the length of the integration interval
04558 !  is less than (bb-aa)/2**(maxpl-2), where (aa,bb) is the
04559 !  original integration interval.
04560 !
04561   if ( abs( parint ) <= 2.0e+00 ) then
04562
04563     call qk15w ( f, qwgto, omega, p2, p3, p4, integr, a, b, result, &
04564       abserr, resabs, resasc )
04565
04566     neval = 15
04567     return
04568
04569   end if
04570 !
04571 !  Compute the integral using the generalized clenshaw-curtis method.
04572 !
04573   conc = hlgth * cos(centr*omega)
04574   cons = hlgth * sin(centr*omega)
04575   resasc = huge( resasc )
04576   neval = 25
04577 !
04578 !  Check whether the Chebyshev moments for this interval
04579 !  have already been computed.
04580 !
04581   if ( nrmom < momcom .or. ksave == 1 ) then
04582     go to 140
04583   end if
04584 !
04585 !  Compute a new set of Chebyshev moments.
04586 !
04587   m = momcom + 1
04588   par2 = parint * parint
04589   par22 = par2 + 2.0e+00
04590   sinpar = sin(parint)
04591   cospar = cos(parint)
04592 !
04593 !  Compute the Chebyshev moments with respect to cosine.
04594 !
04595   v(1) = 2.0e+00 * sinpar / parint
04596   v(2) = (8.0e+00*cospar+(par2+par2-8.0e+00)*sinpar/ parint)/par2
04597   v(3) = (3.2e+01*(par2-1.2e+01)*cospar+(2.0e+00* &
```

```
04598       ((par2-8.0e+01)*par2+1.92e+02)*sinpar)/ &
04599       parint)/(par2*par2)
04600     ac = 8.0e+00*cospar
04601     as = 2.4e+01*parint*sinpar
04602
04603     if ( abs( parint ) > 2.4e+01 ) then
04604       go to 70
04605     end if
04606 !
04607 !  Compute the Chebyshev moments as the solutions of a boundary value
04608 !  problem with one initial value (v(3)) and one end value computed
04609 !  using an asymptotic formula.
04610 !
04611     noequ = nmac-3
04612     noeq1 = noequ-1
04613     an = 6.0e+00
04614
04615     do k = 1, noeq1
04616       an2 = an*an
04617       d(k) = -2.0e+00*(an2-4.0e+00) * (par22-an2-an2)
04618       d2(k) = (an-1.0e+00_params_wp)*(an-2.0e+00) * par2
04619       d1(k) = (an+3.0e+00)*(an+4.0e+00) * par2
04620       v(k+3) = as-(an2-4.0e+00) * ac
04621       an = an+2.0e+00
04622     end do
04623
04624     an2 = an*an
04625     d(noequ) = -2.0e+00*(an2-4.0e+00) * (par22-an2-an2)
04626     v(noequ+3) = as - ( an2 - 4.0e+00 ) * ac
04627     v(4) = v(4) - 5.6e+01 * par2 * v(3)
04628     ass = parint * sinpar
04629     asap = (((((2.10e+02*par2-1.0e+00_params_wp)*cospar-(1.05e+02*par2 &
04630        -6.3e+01)*ass)/an2-(1.0e+00_params_wp-1.5e+01*par2)*cospar &
04631        +1.5e+01*ass)/an2-cospar+3.0e+00*ass)/an2-cospar)/an2
04632     v(noequ+3) = v(noequ+3)-2.0e+00*asap*par2*(an-1.0e+00_params_wp)* &
04633         (an-2.0e+00)
04634 !
04635 !  Solve the tridiagonal system by means of Gaussian
04636 !  elimination with partial pivoting.
04637 !
04638     d3(1:noequ) = 0.0e+00
04639
04640     d2(noequ) = 0.0e+00
04641
04642     do i = 1, noeq1
04643
04644       if ( abs(d1(i)) > abs(d(i)) ) then
04645         an = d1(i)
04646         d1(i) = d(i)
04647         d(i) = an
04648         an = d2(i)
04649         d2(i) = d(i+1)
04650         d(i+1) = an
04651         d3(i) = d2(i+1)
04652         d2(i+1) = 0.0e+00
04653         an = v(i+4)
04654         v(i+4) = v(i+3)
04655         v(i+3) = an
04656       end if
04657
04658       d(i+1) = d(i+1)-d2(i)*d1(i)/d(i)
04659       d2(i+1) = d2(i+1)-d3(i)*d1(i)/d(i)
04660       v(i+4) = v(i+4)-v(i+3)*d1(i)/d(i)
04661
04662     end do
04663
04664     v(noequ+3) = v(noequ+3) / d(noequ)
04665     v(noequ+2) = (v(noequ+2)-d2(noeq1)*v(noequ+3))/d(noeq1)
04666
04667     do i = 2, noeq1
04668       k = noequ-i
04669       v(k+3) = (v(k+3)-d3(k)*v(k+5)-d2(k)*v(k+4))/d(k)
04670     end do
04671
04672     go to 90
04673 !
04674 !  Compute the Chebyshev moments by means of forward recursion
04675 !
04676 70 continue
04677
04678     an = 4.0e+00
04679
04680     do i = 4, 13
04681       an2 = an*an
04682       v(i) = ((an2-4.0e+00)*(2.0e+00*(par22-an2-an2)*v(i-1)-ac) &
04683        +as-par2*(an+1.0e+00_params_wp)*(an+2.0e+00)*v(i-2))/ &
04684        (par2*(an-1.0e+00_params_wp)*(an-2.0e+00))
```

```
04685       an = an+2.0e+00
04686     end do
04687
04688   90 continue
04689
04690     do j = 1, 13
04691       chebmo(m,2*j-1) = v(j)
04692     end do
04693   !
04694   !  Compute the Chebyshev moments with respect to sine.
04695   !
04696     v(1) = 2.0e+00*(sinpar-parint*cospar)/par2
04697     v(2) = (1.8e+01-4.8e+01/par2)*sinpar/par2 &
04698        +(-2.0e+00+4.8e+01/par2)*cospar/parint
04699     ac = -2.4e+01*parint*cospar
04700     as = -8.0e+00*sinpar
04701     chebmo(m,2) = v(1)
04702     chebmo(m,4) = v(2)
04703
04704     if ( abs(parint) <= 2.4e+01 ) then
04705
04706       do k = 3, 12
04707         an = k
04708         chebmo(m,2*k) = -sinpar/(an*(2.0e+00*an-2.0e+00)) &
04709                     -2.5e-01*parint*(v(k+1)/an-v(k)/(an-1.0e+00_params_wp))
04710       end do
04711   !
04712   !  Compute the Chebyshev moments by means of forward recursion.
04713   !
04714     else
04715
04716       an = 3.0e+00
04717
04718       do i = 3, 12
04719         an2 = an*an
04720         v(i) = ((an2-4.0e+00)*(2.0e+00*(par22-an2-an2)*v(i-1)+as) &
04721          +ac-par2*(an+1.0e+00_params_wp)*(an+2.0e+00)*v(i-2)) &
04722          /(par2*(an-1.0e+00_params_wp)*(an-2.0e+00))
04723         an = an+2.0e+00
04724         chebmo(m,2*i) = v(i)
04725       end do
04726
04727     end if
04728
04729   140 continue
04730
04731     if ( nrmom < momcom ) then
04732       m = nrmom + 1
04733     end if
04734
04735     if ( momcom < maxp1 - 1 .and. nrmom >= momcom ) then
04736       momcom = momcom + 1
04737     end if
04738   !
04739   !  Compute the coefficients of the Chebyshev expansions
04740   !  of degrees 12 and 24 of the function F.
04741   !
04742     fval(1) = 5.0e-01 * f(centr+hlgth)
04743     fval(13) = f(centr)
04744     fval(25) = 5.0e-01 * f(centr-hlgth)
04745
04746     do i = 2, 12
04747       isym = 26-i
04748       fval(i) = f(hlgth*x(i-1)+centr)
04749       fval(isym) = f(centr-hlgth*x(i-1))
04750     end do
04751
04752     call qcheb ( x, fval, cheb12, cheb24 )
04753   !
04754   !  Compute the integral and error estimates.
04755   !
04756     resc12 = cheb12(13) * chebmo(m,13)
04757     ress12 = 0.0e+00
04758     estc = abs( cheb24(25)*chebmo(m,25))+abs((cheb12(13)- &
04759       cheb24(13))*chebmo(m,13) )
04760     ests = 0.0e+00
04761     k = 11
04762
04763     do j = 1, 6
04764       resc12 = resc12+cheb12(k)*chebmo(m,k)
04765       ress12 = ress12+cheb12(k+1)*chebmo(m,k+1)
04766       estc = estc+abs((cheb12(k)-cheb24(k))*chebmo(m,k))
04767       ests = ests+abs((cheb12(k+1)-cheb24(k+1))*chebmo(m,k+1))
04768       k = k-2
04769     end do
04770
04771     resc24 = cheb24(25)*chebmo(m,25)
```

```
04772   ress24 = 0.0e+00
04773   resabs = abs(cheb24(25))
04774   k = 23
04775
04776   do j = 1, 12
04777
04778     resc24 = resc24+cheb24(k)*chebmo(m,k)
04779     ress24 = ress24+cheb24(k+1)*chebmo(m,k+1)
04780     resabs = resabs+abs(cheb24(k))+abs(cheb24(k+1))
04781
04782     if ( j <= 5 ) then
04783       estc = estc+abs(cheb24(k)*chebmo(m,k))
04784       ests = ests+abs(cheb24(k+1)*chebmo(m,k+1))
04785     end if
04786
04787     k = k-2
04788
04789   end do
04790
04791   resabs = resabs * abs( hlgth )
04792
04793   if ( integr == 1 ) then
04794     result = conc * resc24-cons*ress24
04795     abserr = abs( conc * estc ) + abs( cons * ests )
04796   else
04797     result = conc*ress24+cons*resc24
04798     abserr = abs(conc*ests)+abs(cons*estc)
04799   end if
04800
04801   return
04802 end subroutine qc25o
04803 subroutine qc25s ( f, a, b, bl, br, alfa, beta, ri, rj, rg, rh, result, &
04804   abserr, resasc, integr, neval )
04805
04806 !*****************************************************************************80
04807 !
04808 !! QC25S returns rules for algebraico-logarithmic end point singularities.
04809 !
04810 !  Discussion:
04811 !
04812 !    This routine computes
04813 !      i = integral of F(X) * W(X) over (bl,br),
04814 !    with error estimate, where the weight function W(X) has a singular
04815 !    behavior of algebraico-logarithmic type at the points
04816 !    a and/or b.
04817 !
04818 !    The interval (bl,br) is a subinterval of (a,b).
04819 !
04820 !  Author:
04821 !
04822 !    Robert Piessens, Elise de Doncker-Kapenger,
04823 !    Christian Ueberhuber, David Kahaner
04824 !
04825 !  Reference:
04826 !
04827 !    Robert Piessens, Elise de Doncker-Kapenger,
04828 !    Christian Ueberhuber, David Kahaner,
04829 !    QUADPACK, a Subroutine Package for Automatic Integration,
04830 !    Springer Verlag, 1983
04831 !
04832 !  Parameters:
04833 !
04834 !    Input, external real F, the name of the function routine, of the form
04835 !      function f ( x )
04836 !      real f
04837 !      real x
04838 !    which evaluates the integrand function.
04839 !
04840 !    Input, real A, B, the limits of integration.
04841 !
04842 !    Input, real BL, BR, the lower and upper limits of integration.
04843 !    A <= BL < BR <= B.
04844 !
04845 !    Input, real ALFA, BETA, parameters in the weight function.
04846 !
04847 !    Input, real RI(25), RJ(25), RG(25), RH(25), modified Chebyshev moments
04848 !    for the application of the generalized Clenshaw-Curtis method,
04849 !    computed in QMOMO.
04850 !
04851 !    Output, real RESULT, the estimated value of the integral, computed by
04852 !    using a generalized clenshaw-curtis method if bl = a or br = b.
04853 !    In all other cases the 15-point Kronrod rule is applied, obtained by
04854 !    optimal addition of abscissae to the 7-point Gauss rule.
04855 !
04856 !    Output, real ABSERR, an estimate of || I - RESULT ||.
04857 !
04858 !    Output, real RESASC, approximation to the integral of abs(F*W-I/(B-A)).
```

```
04859 !
04860 !    Input, integer INTEGR,  determines the weight function
04861 !    1, w(x) = (x-a)**alfa*(b-x)**beta
04862 !    2, w(x) = (x-a)**alfa*(b-x)**beta*log(x-a)
04863 !    3, w(x) = (x-a)**alfa*(b-x)**beta*log(b-x)
04864 !    4, w(x) = (x-a)**alfa*(b-x)**beta*log(x-a)*log(b-x)
04865 !
04866 !    Output, integer NEVAL, the number of times the integral was evaluated.
04867 !
04868 !  Local Parameters:
04869 !
04870 !           fval   - value of the function f at the points
04871 !                    (br-bl)*0.5*cos(k*pi/24)+(br+bl)*0.5
04872 !                    k = 0, ..., 24
04873 !           cheb12 - coefficients of the Chebyshev series expansion
04874 !                    of degree 12, for the function f, in the interval
04875 !                    (bl,br)
04876 !           cheb24 - coefficients of the Chebyshev series expansion
04877 !                    of degree 24, for the function f, in the interval
04878 !                    (bl,br)
04879 !           res12  - approximation to the integral obtained from cheb12
04880 !           res24  - approximation to the integral obtained from cheb24
04881 !           qwgts  - external function subprogram defining the four
04882 !                    possible weight functions
04883 !           hlgth  - half-length of the interval (bl,br)
04884 !           centr  - mid point of the interval (bl,br)
04885 !
04886 !           the vector x contains the values cos(k*pi/24)
04887 !           k = 1, ..., 11, to be used for the computation of the
04888 !           Chebyshev series expansion of f.
04889 !
04890   implicit none
04891
04892   real(kind=params_wp) a
04893   real(kind=params_wp) abserr
04894   real(kind=params_wp) alfa
04895   real(kind=params_wp) b
04896   real(kind=params_wp) beta
04897   real(kind=params_wp) bl
04898   real(kind=params_wp) br
04899   real(kind=params_wp) centr
04900   real(kind=params_wp) cheb12(13)
04901   real(kind=params_wp) cheb24(25)
04902   real(kind=params_wp) dc
04903   real(kind=params_wp), external :: f
04904   real(kind=params_wp) factor
04905   real(kind=params_wp) fix
04906   real(kind=params_wp) fval(25)
04907   real(kind=params_wp) hlgth
04908   integer i
04909   integer integr
04910   integer isym
04911   integer neval
04912 !  real(kind=params_wp), external :: qwgts
04913   real(kind=params_wp) resabs
04914   real(kind=params_wp) resasc
04915   real(kind=params_wp) result
04916   real(kind=params_wp) res12
04917   real(kind=params_wp) res24
04918   real(kind=params_wp) rg(25)
04919   real(kind=params_wp) rh(25)
04920   real(kind=params_wp) ri(25)
04921   real(kind=params_wp) rj(25)
04922   real(kind=params_wp) u
04923   real(kind=params_wp), dimension ( 11 ) :: x = (/ &
04924     9.914448613738104e-01,     9.659258262890683e-01, &
04925     9.238795325112868e-01,     8.660254037844386e-01, &
04926     7.933533402912352e-01,     7.071067811865475e-01, &
04927     6.087614290087206e-01,     5.000000000000000e-01, &
04928     3.826834323650898e-01,     2.588190451025208e-01, &
04929     1.305261922200516e-01 /)
04930
04931   neval = 25
04932
04933   if ( bl == a .and. (alfa /= 0.0e+00 .or. integr == 2 .or. integr == 4)) then
04934     go to 10
04935   end if
04936
04937   if ( br == b .and. (beta /= 0.0e+00 .or. integr == 3 .or. integr == 4)) &
04938     go to 140
04939 !
04940 !  If a > bl and b < br, apply the 15-point Gauss-Kronrod scheme.
04941 !
04942   call qk15w ( f, qwgts, a, b, alfa, beta, integr, bl, br, result, abserr, &
04943     resabs, resasc )
04944
04945   neval = 15
```

```
04946   return
04947 !
04948 !  This part of the program is executed only if a = bl.
04949 !
04950 !  Compute the Chebyshev series expansion of the function
04951 !  f1 = (0.5*(b+b-br-a)-0.5*(br-a)*x)**beta*f(0.5*(br-a)*x+0.5*(br+a))
04952 !
04953 10 continue
04954
04955   hlgth = 5.0e-01*(br-bl)
04956   centr = 5.0e-01*(br+bl)
04957   fix = b-centr
04958   fval(1) = 5.0e-01*f(hlgth+centr)*(fix-hlgth)**beta
04959   fval(13) = f(centr)*(fix**beta)
04960   fval(25) = 5.0e-01*f(centr-hlgth)*(fix+hlgth)**beta
04961
04962   do i = 2, 12
04963     u = hlgth*x(i-1)
04964     isym = 26-i
04965     fval(i) = f(u+centr)*(fix-u)**beta
04966     fval(isym) = f(centr-u)*(fix+u)**beta
04967   end do
04968
04969   factor = hlgth**(alfa+1.0e+00_params_wp)
04970   result = 0.0e+00
04971   abserr = 0.0e+00
04972   res12 = 0.0e+00
04973   res24 = 0.0e+00
04974
04975   if ( integr > 2 ) go to 70
04976
04977   call qcheb ( x, fval, cheb12, cheb24 )
04978 !
04979 !  integr = 1  (or 2)
04980 !
04981   do i = 1, 13
04982     res12 = res12+cheb12(i)*ri(i)
04983     res24 = res24+cheb24(i)*ri(i)
04984   end do
04985
04986   do i = 14, 25
04987     res24 = res24 + cheb24(i) * ri(i)
04988   end do
04989
04990   if ( integr == 1 ) go to 130
04991 !
04992 !  integr = 2
04993 !
04994   dc = log( br - bl )
04995   result = res24 * dc
04996   abserr = abs((res24-res12)*dc)
04997   res12 = 0.0e+00
04998   res24 = 0.0e+00
04999
05000   do i = 1, 13
05001     res12 = res12+cheb12(i)*rg(i)
05002     res24 = res24+cheb24(i)*rg(i)
05003   end do
05004
05005   do i = 14, 25
05006     res24 = res24+cheb24(i)*rg(i)
05007   end do
05008
05009   go to 130
05010 !
05011 !  Compute the Chebyshev series expansion of the function
05012 !  F4 = f1*log(0.5*(b+b-br-a)-0.5*(br-a)*x)
05013 !
05014 70 continue
05015
05016   fval(1) = fval(1) * log( fix - hlgth )
05017   fval(13) = fval(13) * log( fix )
05018   fval(25) = fval(25) * log( fix + hlgth )
05019
05020   do i = 2, 12
05021     u = hlgth*x(i-1)
05022     isym = 26-i
05023     fval(i) = fval(i) * log( fix - u )
05024     fval(isym) = fval(isym) * log( fix + u )
05025   end do
05026
05027   call qcheb ( x, fval, cheb12, cheb24 )
05028 !
05029 !  integr = 3  (or 4)
05030 !
05031   do i = 1, 13
05032     res12 = res12+cheb12(i)*ri(i)
```

```
05033       res24 = res24+cheb24(i)*ri(i)
05034     end do
05035
05036     do i = 14, 25
05037       res24 = res24+cheb24(i)*ri(i)
05038     end do
05039
05040     if ( integr == 3 ) then
05041       go to 130
05042     end if
05043 !
05044 !  integr = 4
05045 !
05046     dc = log( br - bl )
05047     result = res24*dc
05048     abserr = abs((res24-res12)*dc)
05049     res12 = 0.0e+00
05050     res24 = 0.0e+00
05051
05052     do i = 1, 13
05053       res12 = res12+cheb12(i)*rg(i)
05054       res24 = res24+cheb24(i)*rg(i)
05055     end do
05056
05057     do i = 14, 25
05058       res24 = res24+cheb24(i)*rg(i)
05059     end do
05060
05061 130 continue
05062
05063     result = (result+res24)*factor
05064     abserr = (abserr+abs(res24-res12))*factor
05065     go to 270
05066 !
05067 !  This part of the program is executed only if b = br.
05068 !
05069 !  Compute the Chebyshev series expansion of the function
05070 !  f2 = (0.5*(b+bl-a-a)+0.5*(b-bl)*x)**alfa*f(0.5*(b-bl)*x+0.5*(b+bl))
05071 !
05072 140 continue
05073
05074     hlgth = 5.0e-01*(br-bl)
05075     centr = 5.0e-01*(br+bl)
05076     fix = centr-a
05077     fval(1) = 5.0e-01*f(hlgth+centr)*(fix+hlgth)**alfa
05078     fval(13) = f(centr)*(fix**alfa)
05079     fval(25) = 5.0e-01*f(centr-hlgth)*(fix-hlgth)**alfa
05080
05081     do i = 2, 12
05082       u = hlgth*x(i-1)
05083       isym = 26-i
05084       fval(i) = f(u+centr)*(fix+u)**alfa
05085       fval(isym) = f(centr-u)*(fix-u)**alfa
05086     end do
05087
05088     factor = hlgth**(beta+1.0e+00_params_wp)
05089     result = 0.0e+00
05090     abserr = 0.0e+00
05091     res12 = 0.0e+00
05092     res24 = 0.0e+00
05093
05094     if ( integr == 2 .or. integr == 4 ) then
05095       go to 200
05096     end if
05097 !
05098 !  integr = 1  (or 3)
05099 !
05100     call qcheb ( x, fval, cheb12, cheb24 )
05101
05102     do i = 1, 13
05103       res12 = res12+cheb12(i)*rj(i)
05104       res24 = res24+cheb24(i)*rj(i)
05105     end do
05106
05107     do i = 14, 25
05108       res24 = res24+cheb24(i)*rj(i)
05109     end do
05110
05111     if ( integr == 1 ) go to 260
05112 !
05113 !  integr = 3
05114 !
05115     dc = log( br - bl )
05116     result = res24*dc
05117     abserr = abs((res24-res12)*dc)
05118     res12 = 0.0e+00
05119     res24 = 0.0e+00
```

```
05120
05121   do i = 1, 13
05122     res12 = res12+cheb12(i)*rh(i)
05123     res24 = res24+cheb24(i)*rh(i)
05124   end do
05125
05126   do i = 14, 25
05127     res24 = res24+cheb24(i)*rh(i)
05128   end do
05129
05130   go to 260
05131 !
05132 !  Compute the Chebyshev series expansion of the function
05133 !  f3 = f2*log(0.5*(b-bl)*x+0.5*(b+bl-a-a))
05134 !
05135 200 continue
05136
05137   fval(1) = fval(1) * log( hlgth + fix )
05138   fval(13) = fval(13) * log( fix )
05139   fval(25) = fval(25) * log( fix - hlgth )
05140
05141   do i = 2, 12
05142     u = hlgth*x(i-1)
05143     isym = 26-i
05144     fval(i) = fval(i) * log(u+fix)
05145     fval(isym) = fval(isym) * log(fix-u)
05146   end do
05147
05148   call qcheb ( x, fval, cheb12, cheb24 )
05149 !
05150 !  integr = 2  (or 4)
05151 !
05152   do i = 1, 13
05153     res12 = res12+cheb12(i)*rj(i)
05154     res24 = res24+cheb24(i)*rj(i)
05155   end do
05156
05157   do i = 14, 25
05158     res24 = res24+cheb24(i)*rj(i)
05159   end do
05160
05161   if ( integr == 2 ) go to 260
05162
05163   dc = log(br-bl)
05164   result = res24*dc
05165   abserr = abs((res24-res12)*dc)
05166   res12 = 0.0e+00
05167   res24 = 0.0e+00
05168 !
05169 !  integr = 4
05170 !
05171   do i = 1, 13
05172     res12 = res12+cheb12(i)*rh(i)
05173     res24 = res24+cheb24(i)*rh(i)
05174   end do
05175
05176   do i = 14, 25
05177     res24 = res24+cheb24(i)*rh(i)
05178   end do
05179
05180 260 continue
05181
05182   result = (result+res24)*factor
05183   abserr = (abserr+abs(res24-res12))*factor
05184
05185 270 continue
05186
05187   return
05188 end subroutine qc25s
05189 subroutine qcheb ( x, fval, cheb12, cheb24 )
05190
05191 !*****************************************************************************80
05192 !
05193 !! QCHEB computes the Chebyshev series expansion.
05194 !
05195 !  Discussion:
05196 !
05197 !    This routine computes the Chebyshev series expansion
05198 !    of degrees 12 and 24 of a function using a fast Fourier transform method
05199 !
05200 !      f(x) = sum(k=1, ...,13) (cheb12(k)*t(k-1,x)),
05201 !      f(x) = sum(k=1, ...,25) (cheb24(k)*t(k-1,x)),
05202 !
05203 !    where T(K,X) is the Chebyshev polynomial of degree K.
05204 !
05205 !  Author:
05206 !
```

```
05207 !    Robert Piessens, Elise de Doncker-Kapenger,
05208 !    Christian Ueberhuber, David Kahaner
05209 !
05210 !  Reference:
05211 !
05212 !    Robert Piessens, Elise de Doncker-Kapenger,
05213 !    Christian Ueberhuber, David Kahaner,
05214 !    QUADPACK, a Subroutine Package for Automatic Integration,
05215 !    Springer Verlag, 1983
05216 !
05217 !  Parameters:
05218 !
05219 !    Input, real X(11), contains the values of COS(K*PI/24), for K = 1 to 11.
05220 !
05221 !    Input/output, real FVAL(25), the function values at the points
05222 !    (b+a+(b-a)*cos(k*pi/24))/2, k = 0, ...,24, where (a,b) is the
05223 !    approximation interval.  FVAL(1) and FVAL(25) are divided by two
05224 !    These values are destroyed at output.
05225 !
05226 !    Output, real CHEB12(13), the Chebyshev coefficients for degree 12.
05227 !
05228 !    Output, real CHEB24(25), the Chebyshev coefficients for degree 24.
05229 !
05230   implicit none
05231
05232   real(kind=params_wp) alam
05233   real(kind=params_wp) alam1
05234   real(kind=params_wp) alam2
05235   real(kind=params_wp) cheb12(13)
05236   real(kind=params_wp) cheb24(25)
05237   real(kind=params_wp) fval(25)
05238   integer i
05239   integer j
05240   real(kind=params_wp) part1
05241   real(kind=params_wp) part2
05242   real(kind=params_wp) part3
05243   real(kind=params_wp) v(12)
05244   real(kind=params_wp) x(11)
05245
05246   do i = 1, 12
05247     j = 26-i
05248     v(i) = fval(i)-fval(j)
05249     fval(i) = fval(i)+fval(j)
05250   end do
05251
05252   alam1 = v(1)-v(9)
05253   alam2 = x(6)*(v(3)-v(7)-v(11))
05254   cheb12(4) = alam1+alam2
05255   cheb12(10) = alam1-alam2
05256   alam1 = v(2)-v(8)-v(10)
05257   alam2 = v(4)-v(6)-v(12)
05258   alam = x(3)*alam1+x(9)*alam2
05259   cheb24(4) = cheb12(4)+alam
05260   cheb24(22) = cheb12(4)-alam
05261   alam = x(9)*alam1-x(3)*alam2
05262   cheb24(10) = cheb12(10)+alam
05263   cheb24(16) = cheb12(10)-alam
05264   part1 = x(4)*v(5)
05265   part2 = x(8)*v(9)
05266   part3 = x(6)*v(7)
05267   alam1 = v(1)+part1+part2
05268   alam2 = x(2)*v(3)+part3+x(10)*v(11)
05269   cheb12(2) = alam1+alam2
05270   cheb12(12) = alam1-alam2
05271   alam = x(1)*v(2)+x(3)*v(4)+x(5)*v(6)+x(7)*v(8) &
05272     +x(9)*v(10)+x(11)*v(12)
05273   cheb24(2) = cheb12(2)+alam
05274   cheb24(24) = cheb12(2)-alam
05275   alam = x(11)*v(2)-x(9)*v(4)+x(7)*v(6)-x(5)*v(8) &
05276     +x(3)*v(10)-x(1)*v(12)
05277   cheb24(12) = cheb12(12)+alam
05278   cheb24(14) = cheb12(12)-alam
05279   alam1 = v(1)-part1+part2
05280   alam2 = x(10)*v(3)-part3+x(2)*v(11)
05281   cheb12(6) = alam1+alam2
05282   cheb12(8) = alam1-alam2
05283   alam = x(5)*v(2)-x(9)*v(4)-x(1)*v(6) &
05284     -x(11)*v(8)+x(3)*v(10)+x(7)*v(12)
05285   cheb24(6) = cheb12(6)+alam
05286   cheb24(20) = cheb12(6)-alam
05287   alam = x(7)*v(2)-x(3)*v(4)-x(11)*v(6)+x(1)*v(8) &
05288     -x(9)*v(10)-x(5)*v(12)
05289   cheb24(8) = cheb12(8)+alam
05290   cheb24(18) = cheb12(8)-alam
05291
05292   do i = 1, 6
05293     j = 14-i
```

```
05294     v(i) = fval(i)-fval(j)
05295     fval(i) = fval(i)+fval(j)
05296   end do
05297
05298   alam1 = v(1)+x(8)*v(5)
05299   alam2 = x(4)*v(3)
05300   cheb12(3) = alam1+alam2
05301   cheb12(11) = alam1-alam2
05302   cheb12(7) = v(1)-v(5)
05303   alam = x(2)*v(2)+x(6)*v(4)+x(10)*v(6)
05304   cheb24(3) = cheb12(3)+alam
05305   cheb24(23) = cheb12(3)-alam
05306   alam = x(6)*(v(2)-v(4)-v(6))
05307   cheb24(7) = cheb12(7)+alam
05308   cheb24(19) = cheb12(7)-alam
05309   alam = x(10)*v(2)-x(6)*v(4)+x(2)*v(6)
05310   cheb24(11) = cheb12(11)+alam
05311   cheb24(15) = cheb12(11)-alam
05312
05313   do i = 1, 3
05314     j = 8-i
05315     v(i) = fval(i)-fval(j)
05316     fval(i) = fval(i)+fval(j)
05317   end do
05318
05319   cheb12(5) = v(1)+x(8)*v(3)
05320   cheb12(9) = fval(1)-x(8)*fval(3)
05321   alam = x(4)*v(2)
05322   cheb24(5) = cheb12(5)+alam
05323   cheb24(21) = cheb12(5)-alam
05324   alam = x(8)*fval(2)-fval(4)
05325   cheb24(9) = cheb12(9)+alam
05326   cheb24(17) = cheb12(9)-alam
05327   cheb12(1) = fval(1)+fval(3)
05328   alam = fval(2)+fval(4)
05329   cheb24(1) = cheb12(1)+alam
05330   cheb24(25) = cheb12(1)-alam
05331   cheb12(13) = v(1)-v(3)
05332   cheb24(13) = cheb12(13)
05333   alam = 1.0e+00_params_wp/6.0e+00
05334
05335   do i = 2, 12
05336     cheb12(i) = cheb12(i)*alam
05337   end do
05338
05339   alam = 5.0e-01*alam
05340   cheb12(1) = cheb12(1)*alam
05341   cheb12(13) = cheb12(13)*alam
05342
05343   do i = 2, 24
05344     cheb24(i) = cheb24(i)*alam
05345   end do
05346
05347   cheb24(1) = 0.5e+00 * alam*cheb24(1)
05348   cheb24(25) = 0.5e+00 * alam*cheb24(25)
05349
05350   return
05351 end subroutine qcheb
05352 subroutine qextr ( n, epstab, result, abserr, res3la, nres )
05353
05354 !***********************************************************************80
05355 !
05356 !! QEXTR carries out the Epsilon extrapolation algorithm.
05357 !
05358 !  Discussion:
05359 !
05360 !    The routine determines the limit of a given sequence of approximations,
05361 !    by means of the epsilon algorithm of P. Wynn.  An estimate of the
05362 !    absolute error is also given.  The condensed epsilon table is computed.
05363 !    Only those elements needed for the computation of the next diagonal
05364 !    are preserved.
05365 !
05366 !  Author:
05367 !
05368 !    Robert Piessens, Elise de Doncker-Kapenger,
05369 !    Christian Ueberhuber, David Kahaner
05370 !
05371 !  Reference:
05372 !
05373 !    Robert Piessens, Elise de Doncker-Kapenger,
05374 !    Christian Ueberhuber, David Kahaner,
05375 !    QUADPACK, a Subroutine Package for Automatic Integration,
05376 !    Springer Verlag, 1983
05377 !
05378 !  Parameters:
05379 !
05380 !    Input, integer N, indicates the entry of EPSTAB which contains
```

```
05381 !    the new element in the first column of the epsilon table.
05382 !
05383 !    Input/output, real EPSTAB(52), the two lower diagonals of the triangular
05384 !    epsilon table.  The elements are numbered starting at the right-hand
05385 !    corner of the triangle.
05386 !
05387 !    Output, real RESULT, the estimated value of the integral.
05388 !
05389 !    Output, real ABSERR, estimate of the absolute error computed from
05390 !    RESULT and the 3 previous results.
05391 !
05392 !    ?, real RES3LA(3), the last 3 results.
05393 !
05394 !    Input/output, integer NRES, the number of calls to the routine.  This
05395 !    should be zero on the first call, and is automatically updated
05396 !    before return.
05397 !
05398 !  Local Parameters:
05399 !
05400 !           e0     - the 4 elements on which the
05401 !           e1       computation of a new element in
05402 !           e2       the epsilon table is based
05403 !           e3                 e0
05404 !                          e3   e1    new
05405 !                               e2
05406 !           newelm - number of elements to be computed in the new
05407 !                    diagonal
05408 !           error  - error = abs(e1-e0)+abs(e2-e1)+abs(new-e2)
05409 !           result - the element in the new diagonal with least value
05410 !                    of error
05411 !           limexp is the maximum number of elements the epsilon table
05412 !           can contain. if this number is reached, the upper diagonal
05413 !           of the epsilon table is deleted.
05414 !
05415   implicit none
05416
05417   real(kind=params_wp) abserr
05418   real(kind=params_wp) delta1
05419   real(kind=params_wp) delta2
05420   real(kind=params_wp) delta3
05421   real(kind=params_wp) epsinf
05422   real(kind=params_wp) epstab(52)
05423   real(kind=params_wp) error
05424   real(kind=params_wp) err1
05425   real(kind=params_wp) err2
05426   real(kind=params_wp) err3
05427   real(kind=params_wp) e0
05428   real(kind=params_wp) e1
05429   real(kind=params_wp) e1abs
05430   real(kind=params_wp) e2
05431   real(kind=params_wp) e3
05432   integer i
05433   integer ib
05434   integer ib2
05435   integer ie
05436   integer indx
05437   integer k1
05438   integer k2
05439   integer k3
05440   integer limexp
05441   integer n
05442   integer newelm
05443   integer nres
05444   integer num
05445   real(kind=params_wp) res
05446   real(kind=params_wp) result
05447   real(kind=params_wp) res3la(3)
05448   real(kind=params_wp) ss
05449   real(kind=params_wp) tol1
05450   real(kind=params_wp) tol2
05451   real(kind=params_wp) tol3
05452
05453   nres = nres+1
05454   abserr = huge( abserr )
05455   result = epstab(n)
05456
05457   if ( n < 3 ) then
05458     abserr = max( abserr,0.5e+00* epsilon( result ) *abs(result))
05459     return
05460   end if
05461
05462   limexp = 50
05463   epstab(n+2) = epstab(n)
05464   newelm = (n-1)/2
05465   epstab(n) = huge( epstab(n) )
05466   num = n
05467   k1 = n
```

```
05468
05469   do i = 1, newelm
05470
05471     k2 = k1-1
05472     k3 = k1-2
05473     res = epstab(k1+2)
05474     e0 = epstab(k3)
05475     e1 = epstab(k2)
05476     e2 = res
05477     e1abs = abs(e1)
05478     delta2 = e2-e1
05479     err2 = abs(delta2)
05480     tol2 = max( abs(e2),e1abs)* epsilon( e2 )
05481     delta3 = e1-e0
05482     err3 = abs(delta3)
05483     tol3 = max( e1abs,abs(e0))* epsilon( e0 )
05484 !
05485 !  If e0, e1 and e2 are equal to within machine accuracy, convergence
05486 !  is assumed.
05487 !
05488     if ( err2 <= tol2 .and. err3 <= tol3 ) then
05489       result = res
05490       abserr = err2+err3
05491       abserr = max( abserr,0.5e+00* epsilon( result ) *abs(result))
05492       return
05493     end if
05494
05495     e3 = epstab(k1)
05496     epstab(k1) = e1
05497     delta1 = e1-e3
05498     err1 = abs(delta1)
05499     tol1 = max( e1abs,abs(e3))* epsilon( e3 )
05500 !
05501 !  If two elements are very close to each other, omit a part
05502 !  of the table by adjusting the value of N.
05503 !
05504     if ( err1 <= tol1 .or. err2 <= tol2 .or. err3 <= tol3 ) go to 20
05505
05506     ss = 1.0e+00_params_wp/delta1+1.0e+00_params_wp/delta2-1.0e+00_params_wp/delta3
05507     epsinf = abs( ss*e1 )
05508 !
05509 !  Test to detect irregular behavior in the table, and
05510 !  eventually omit a part of the table adjusting the value of N.
05511 !
05512     if ( epsinf > 1.0e-04 ) go to 30
05513
05514 20  continue
05515
05516     n = i+i-1
05517     exit
05518 !
05519 !  Compute a new element and eventually adjust the value of RESULT.
05520 !
05521 30  continue
05522
05523     res = e1+1.0e+00_params_wp/ss
05524     epstab(k1) = res
05525     k1 = k1-2
05526     error = err2+abs(res-e2)+err3
05527
05528     if ( error <= abserr ) then
05529       abserr = error
05530       result = res
05531     end if
05532
05533   end do
05534 !
05535 !  Shift the table.
05536 !
05537   if ( n == limexp ) then
05538     n = 2*(limexp/2)-1
05539   end if
05540
05541   if ( (num/2)*2 == num ) then
05542     ib = 2
05543   else
05544     ib = 1
05545   end if
05546
05547   ie = newelm+1
05548
05549   do i = 1, ie
05550     ib2 = ib+2
05551     epstab(ib) = epstab(ib2)
05552     ib = ib2
05553   end do
05554
```

```
05555   if ( num /= n ) then
05556
05557     indx = num-n+1
05558
05559     do i = 1, n
05560       epstab(i)= epstab(indx)
05561       indx = indx+1
05562     end do
05563
05564   end if
05565
05566   if ( nres < 4 ) then
05567     res3la(nres) = result
05568     abserr = huge( abserr )
05569   else
05570     abserr = abs(result-res3la(3))+abs(result-res3la(2)) &
05571       +abs(result-res3la(1))
05572     res3la(1) = res3la(2)
05573     res3la(2) = res3la(3)
05574     res3la(3) = result
05575   end if
05576
05577   abserr = max( abserr,0.5e+00* epsilon( result ) *abs(result))
05578
05579   return
05580 end subroutine qextr
05581 subroutine qfour ( f, a, b, omega, integr, epsabs, epsrel, limit, icall, &
05582   maxp1, result, abserr, neval, ier, alist, blist, rlist, elist, iord, &
05583   nnlog, momcom, chebmo )
05584
05585 !*****************************************************************************80
05586 !
05587 !! QFOUR estimates the integrals of oscillatory functions.
05588 !
05589 !  Discussion:
05590 !
05591 !    This routine calculates an approximation RESULT to a definite integral
05592 !      I = integral of F(X) * COS(OMEGA*X)
05593 !    or
05594 !      I = integral of F(X) * SIN(OMEGA*X)
05595 !    over (A,B), hopefully satisfying:
05596 !      | I - RESULT | <= max ( epsabs, epsrel * |I| ) ).
05597 !
05598 !    QFOUR is called by QAWO and QAWF.  It can also be called directly in
05599 !    a user-written program.  In the latter case it is possible for the
05600 !    user to determine the first dimension of array CHEBMO(MAXP1,25).
05601 !    See also parameter description of MAXP1.  Additionally see
05602 !    parameter description of ICALL for eventually re-using
05603 !    Chebyshev moments computed during former call on subinterval
05604 !    of equal length abs(B-A).
05605 !
05606 !  Author:
05607 !
05608 !    Robert Piessens, Elise de Doncker-Kapenger,
05609 !    Christian Ueberhuber, David Kahaner
05610 !
05611 !  Reference:
05612 !
05613 !    Robert Piessens, Elise de Doncker-Kapenger,
05614 !    Christian Ueberhuber, David Kahaner,
05615 !    QUADPACK, a Subroutine Package for Automatic Integration,
05616 !    Springer Verlag, 1983
05617 !
05618 !  Parameters:
05619 !
05620 !    Input, external real F, the name of the function routine, of the form
05621 !      function f ( x )
05622 !      real f
05623 !      real x
05624 !    which evaluates the integrand function.
05625 !
05626 !    Input, real A, B, the limits of integration.
05627 !
05628 !    Input, real OMEGA, the multiplier of X in the weight function.
05629 !
05630 !    Input, integer INTEGR, indicates the weight functions to be used.
05631 !    = 1, w(x) = cos(omega*x)
05632 !    = 2, w(x) = sin(omega*x)
05633 !
05634 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
05635 !
05636 !    Input, integer LIMIT, the maximum number of subintervals of [A,B]
05637 !    that can be generated.
05638 !
05639 !    icall  - integer
05640 !                     if qfour is to be used only once, ICALL must
05641 !                     be set to 1.  assume that during this call, the
```

```
05642 !                     Chebyshev moments (for clenshaw-curtis integration
05643 !                     of degree 24) have been computed for intervals of
05644 !                     lenghts (abs(b-a))*2**(-l), l=0,1,2,...momcom-1.
05645 !                     the Chebyshev moments already computed can be
05646 !                     re-used in subsequent calls, if qfour must be
05647 !                     called twice or more times on intervals of the
05648 !                     same length abs(b-a). from the second call on, one
05649 !                     has to put then ICALL > 1.
05650 !                     if ICALL < 1, the routine will end with ier = 6.
05651 !
05652 !            maxp1  - integer
05653 !                     gives an upper bound on the number of
05654 !                     Chebyshev moments which can be stored, i.e.
05655 !                     for the intervals of lenghts abs(b-a)*2**(-l),
05656 !                     l=0,1, ..., maxp1-2, maxp1 >= 1.
05657 !                     if maxp1 < 1, the routine will end with ier = 6.
05658 !                     increasing (decreasing) the value of maxp1
05659 !                     decreases (increases) the computational time but
05660 !                     increases (decreases) the required memory space.
05661 !
05662 !    Output, real RESULT, the estimated value of the integral.
05663 !
05664 !    Output, real ABSERR, an estimate of || I - RESULT ||.
05665 !
05666 !    Output, integer NEVAL, the number of times the integral was evaluated.
05667 !
05668 !            ier    - integer
05669 !                     ier = 0 normal and reliable termination of the
05670 !                             routine. it is assumed that the
05671 !                             requested accuracy has been achieved.
05672 !                   - ier > 0 abnormal termination of the routine.
05673 !                             the estimates for integral and error are
05674 !                             less reliable. it is assumed that the
05675 !                             requested accuracy has not been achieved.
05676 !                     ier = 1 maximum number of subdivisions allowed
05677 !                             has been achieved. one can allow more
05678 !                             subdivisions by increasing the value of
05679 !                             limit (and taking according dimension
05680 !                             adjustments into account). however, if
05681 !                             this yields no improvement it is advised
05682 !                             to analyze the integrand, in order to
05683 !                             determine the integration difficulties.
05684 !                             if the position of a local difficulty can
05685 !                             be determined (e.g. singularity,
05686 !                             discontinuity within the interval) one
05687 !                             will probably gain from splitting up the
05688 !                             interval at this point and calling the
05689 !                             integrator on the subranges. if possible,
05690 !                             an appropriate special-purpose integrator
05691 !                             should be used which is designed for
05692 !                             handling the type of difficulty involved.
05693 !                         = 2 the occurrence of roundoff error is
05694 !                             detected, which prevents the requested
05695 !                             tolerance from being achieved.
05696 !                             the error may be under-estimated.
05697 !                         = 3 extremely bad integrand behavior occurs
05698 !                             at some points of the integration
05699 !                             interval.
05700 !                         = 4 the algorithm does not converge. roundoff
05701 !                             error is detected in the extrapolation
05702 !                             table. it is presumed that the requested
05703 !                             tolerance cannot be achieved due to
05704 !                             roundoff in the extrapolation table, and
05705 !                             that the returned result is the best which
05706 !                             can be obtained.
05707 !                         = 5 the integral is probably divergent, or
05708 !                             slowly convergent. it must be noted that
05709 !                             divergence can occur with any other value
05710 !                             of ier > 0.
05711 !                         = 6 the input is invalid, because
05712 !                             epsabs < 0 and epsrel < 0,
05713 !                             or (integr /= 1 and integr /= 2) or
05714 !                             ICALL < 1 or maxp1 < 1.
05715 !                             result, abserr, neval, last, rlist(1),
05716 !                             elist(1), iord(1) and nnlog(1) are set to
05717 !                             zero. alist(1) and blist(1) are set to a
05718 !                             and b respectively.
05719 !
05720 !    Workspace, real ALIST(LIMIT), BLIST(LIMIT), contains in entries 1
05721 !    through LAST the left and right ends of the partition subintervals.
05722 !
05723 !    Workspace, real RLIST(LIMIT), contains in entries 1 through LAST
05724 !    the integral approximations on the subintervals.
05725 !
05726 !    Workspace, real ELIST(LIMIT), contains in entries 1 through LAST
05727 !    the absolute error estimates on the subintervals.
05728 !
```

```
05729 !            iord   - integer
05730 !                     vector of dimension at least limit, the first k
05731 !                     elements of which are pointers to the error
05732 !                     estimates over the subintervals, such that
05733 !                     elist(iord(1)), ..., elist(iord(k)), form
05734 !                     a decreasing sequence, with k = last
05735 !                     if last <= (limit/2+2), and
05736 !                     k = limit+1-last otherwise.
05737 !
05738 !            nnlog  - integer
05739 !                     vector of dimension at least limit, indicating the
05740 !                     subdivision levels of the subintervals, i.e.
05741 !                     iwork(i) = l means that the subinterval numbered
05742 !                     i is of length abs(b-a)*2**(1-l)
05743 !
05744 !         on entry and return
05745 !            momcom - integer
05746 !                     indicating that the Chebyshev moments have been
05747 !                     computed for intervals of lengths
05748 !                     (abs(b-a))*2**(-l), l=0,1,2, ..., momcom-1,
05749 !                     momcom < maxp1
05750 !
05751 !            chebmo - real
05752 !                     array of dimension (maxp1,25) containing the
05753 !                     Chebyshev moments
05754 !
05755 !  Local Parameters:
05756 !
05757 !           alist     - list of left end points of all subintervals
05758 !                       considered up to now
05759 !           blist     - list of right end points of all subintervals
05760 !                       considered up to now
05761 !           rlist(i)  - approximation to the integral over
05762 !                       (alist(i),blist(i))
05763 !           rlist2    - array of dimension at least limexp+2 containing
05764 !                       the part of the epsilon table which is still
05765 !                       needed for further computations
05766 !           elist(i)  - error estimate applying to rlist(i)
05767 !           maxerr    - pointer to the interval with largest error
05768 !                       estimate
05769 !           errmax    - elist(maxerr)
05770 !           erlast    - error on the interval currently subdivided
05771 !           area      - sum of the integrals over the subintervals
05772 !           errsum    - sum of the errors over the subintervals
05773 !           errbnd    - requested accuracy max(epsabs,epsrel*
05774 !                       abs(result))
05775 !           *****1    - variable for the left subinterval
05776 !           *****2    - variable for the right subinterval
05777 !           last      - index for subdivision
05778 !           nres      - number of calls to the extrapolation routine
05779 !           numrl2    - number of elements in rlist2. if an appropriate
05780 !                       approximation to the compounded integral has
05781 !                       been obtained it is put in rlist2(numrl2) after
05782 !                       numrl2 has been increased by one
05783 !           small     - length of the smallest interval considered
05784 !                       up to now, multiplied by 1.5
05785 !           erlarg    - sum of the errors over the intervals larger
05786 !                       than the smallest interval considered up to now
05787 !           extrap    - logical variable denoting that the routine is
05788 !                       attempting to perform extrapolation, i.e. before
05789 !                       subdividing the smallest interval we try to
05790 !                       decrease the value of erlarg
05791 !           noext     - logical variable denoting that extrapolation
05792 !                       is no longer allowed (true value)
05793 !
05794   implicit none
05795
05796   integer limit
05797   integer maxp1
05798
05799   real(kind=params_wp) a
05800   real(kind=params_wp) abseps
05801   real(kind=params_wp) abserr
05802   real(kind=params_wp) alist(limit)
05803   real(kind=params_wp) area
05804   real(kind=params_wp) area1
05805   real(kind=params_wp) area12
05806   real(kind=params_wp) area2
05807   real(kind=params_wp) a1
05808   real(kind=params_wp) a2
05809   real(kind=params_wp) b
05810   real(kind=params_wp) blist(limit)
05811   real(kind=params_wp) b1
05812   real(kind=params_wp) b2
05813   real(kind=params_wp) chebmo(maxp1,25)
05814   real(kind=params_wp) correc
05815   real(kind=params_wp) defab1
```

```
05816   real(kind=params_wp) defab2
05817   real(kind=params_wp) defabs
05818   real(kind=params_wp) domega
05819   real(kind=params_wp) dres
05820   real(kind=params_wp) elist(limit)
05821   real(kind=params_wp) epsabs
05822   real(kind=params_wp) epsrel
05823   real(kind=params_wp) erlarg
05824   real(kind=params_wp) erlast
05825   real(kind=params_wp) errbnd
05826   real(kind=params_wp) errmax
05827   real(kind=params_wp) error1
05828   real(kind=params_wp) erro12
05829   real(kind=params_wp) error2
05830   real(kind=params_wp) errsum
05831   real(kind=params_wp) ertest
05832   logical extall
05833   logical extrap
05834   real(kind=params_wp), external :: f
05835   integer icall
05836   integer id
05837   integer ier
05838   integer ierro
05839   integer integr
05840   integer iord(limit)
05841   integer iroff1
05842   integer iroff2
05843   integer iroff3
05844   integer jupbnd
05845   integer k
05846   integer ksgn
05847   integer ktmin
05848   integer last
05849   integer maxerr
05850   integer momcom
05851   integer nev
05852   integer neval
05853   integer nnlog(limit)
05854   logical noext
05855   integer nres
05856   integer nrmax
05857   integer nrmom
05858   integer numrl2
05859   real(kind=params_wp) omega
05860   real(kind=params_wp) resabs
05861   real(kind=params_wp) reseps
05862   real(kind=params_wp) result
05863   real(kind=params_wp) res3la(3)
05864   real(kind=params_wp) rlist(limit)
05865   real(kind=params_wp) rlist2(52)
05866   real(kind=params_wp) small
05867   real(kind=params_wp) width
05868 !
05869 !  the dimension of rlist2 is determined by  the value of
05870 !  limexp in QEXTR (rlist2 should be of dimension
05871 !  (limexp+2) at least).
05872 !
05873 !  Test on validity of parameters.
05874 !
05875   ier = 0
05876   neval = 0
05877   last = 0
05878   result = 0.0e+00
05879   abserr = 0.0e+00
05880   alist(1) = a
05881   blist(1) = b
05882   rlist(1) = 0.0e+00
05883   elist(1) = 0.0e+00
05884   iord(1) = 0
05885   nnlog(1) = 0
05886
05887   if ( (integr /= 1.and.integr /= 2) .or. (epsabs < 0.0e+00.and. &
05888     epsrel < 0.0e+00) .or. icall < 1 .or. maxp1 < 1 ) then
05889     ier = 6
05890     return
05891   end if
05892 !
05893 !  First approximation to the integral.
05894 !
05895   domega = abs( omega )
05896   nrmom = 0
05897
05898   if ( icall <= 1 ) then
05899     momcom = 0
05900   end if
05901
05902   call qc25o ( f, a, b, domega, integr, nrmom, maxp1, 0, result, abserr, &
```

```
05903       neval, defabs, resabs, momcom, chebmo )
05904 !
05905 !  Test on accuracy.
05906 !
05907     dres = abs(result)
05908     errbnd = max( epsabs,epsrel*dres)
05909     rlist(1) = result
05910     elist(1) = abserr
05911     iord(1) = 1
05912     if ( abserr <= 1.0e+02* epsilon( defabs ) *defabs .and. &
05913       abserr > errbnd ) ier = 2
05914
05915     if ( limit == 1 ) then
05916       ier = 1
05917     end if
05918
05919     if ( ier /= 0 .or. abserr <= errbnd ) then
05920       go to 200
05921     end if
05922 !
05923 !  Initializations
05924 !
05925     errmax = abserr
05926     maxerr = 1
05927     area = result
05928     errsum = abserr
05929     abserr = huge( abserr )
05930     nrmax = 1
05931     extrap = .false.
05932     noext = .false.
05933     ierro = 0
05934     iroff1 = 0
05935     iroff2 = 0
05936     iroff3 = 0
05937     ktmin = 0
05938     small = abs(b-a)*7.5e-01
05939     nres = 0
05940     numrl2 = 0
05941     extall = .false.
05942
05943     if ( 5.0e-01*abs(b-a)*domega <= 2.0e+00) then
05944       numrl2 = 1
05945       extall = .true.
05946       rlist2(1) = result
05947     end if
05948
05949     if ( 2.5e-01 * abs(b-a) * domega <= 2.0e+00 ) then
05950       extall = .true.
05951     end if
05952
05953     if ( dres >= (1.0e+00_params_wp-5.0e+01* epsilon( defabs ) )*defabs ) then
05954       ksgn = 1
05955     else
05956       ksgn = -1
05957     end if
05958 !
05959 !  main do-loop
05960 !
05961     do last = 2, limit
05962 !
05963 !  Bisect the subinterval with the nrmax-th largest error estimate.
05964 !
05965       nrmom = nnlog(maxerr)+1
05966       a1 = alist(maxerr)
05967       b1 = 5.0e-01*(alist(maxerr)+blist(maxerr))
05968       a2 = b1
05969       b2 = blist(maxerr)
05970       erlast = errmax
05971
05972       call qc25o ( f, a1, b1, domega, integr, nrmom, maxp1, 0, area1, &
05973         error1, nev, resabs, defab1, momcom, chebmo )
05974
05975       neval = neval+nev
05976
05977       call qc25o ( f, a2, b2, domega, integr, nrmom, maxp1, 1, area2, &
05978         error2, nev, resabs, defab2, momcom, chebmo )
05979
05980       neval = neval+nev
05981 !
05982 !  Improve previous approximations to integral and error and
05983 !  test for accuracy.
05984 !
05985       area12 = area1+area2
05986       erro12 = error1+error2
05987       errsum = errsum+erro12-errmax
05988       area = area+area12-rlist(maxerr)
05989       if ( defab1 == error1 .or. defab2 == error2 ) go to 25
```

```
05990     if ( abs(rlist(maxerr)-area12) > 1.0e-05*abs(area12) &
05991     .or. erro12 < 9.9e-01*errmax ) go to 20
05992     if ( extrap ) iroff2 = iroff2+1
05993
05994     if ( .not.extrap ) then
05995       iroff1 = iroff1+1
05996     end if
05997
05998 20  continue
05999
06000     if ( last > 10.and.erro12 > errmax ) iroff3 = iroff3+1
06001
06002 25  continue
06003
06004     rlist(maxerr) = area1
06005     rlist(last) = area2
06006     nnlog(maxerr) = nrmom
06007     nnlog(last) = nrmom
06008     errbnd = max( epsabs,epsrel*abs(area))
06009 !
06010 !  Test for roundoff error and eventually set error flag
06011 !
06012     if ( iroff1+iroff2 >= 10 .or. iroff3 >= 20 ) ier = 2
06013
06014     if ( iroff2 >= 5) ierro = 3
06015 !
06016 !  Set error flag in the case that the number of subintervals
06017 !  equals limit.
06018 !
06019     if ( last == limit ) then
06020       ier = 1
06021     end if
06022 !
06023 !  Set error flag in the case of bad integrand behavior at
06024 !  a point of the integration range.
06025 !
06026     if ( max( abs(a1),abs(b2)) <= (1.0e+00_params_wp+1.0e+03* epsilon( a1 ) ) &
06027     *(abs(a2)+1.0e+03* tiny( a2 ) )) then
06028       ier = 4
06029     end if
06030 !
06031 !  Append the newly-created intervals to the list.
06032 !
06033     if ( error2 <= error1 ) then
06034       alist(last) = a2
06035       blist(maxerr) = b1
06036       blist(last) = b2
06037       elist(maxerr) = error1
06038       elist(last) = error2
06039     else
06040       alist(maxerr) = a2
06041       alist(last) = a1
06042       blist(last) = b1
06043       rlist(maxerr) = area2
06044       rlist(last) = area1
06045       elist(maxerr) = error2
06046       elist(last) = error1
06047     end if
06048 !
06049 !  Call QSORT to maintain the descending ordering
06050 !  in the list of error estimates and select the subinterval
06051 !  with nrmax-th largest error estimate (to be bisected next).
06052 !
06053
06054     call qsort ( limit, last, maxerr, errmax, elist, iord, nrmax )
06055
06056     if ( errsum <= errbnd ) then
06057       go to 170
06058     end if
06059
06060     if ( ier /= 0 ) then
06061       exit
06062     end if
06063
06064     if ( last == 2 .and. extall ) go to 120
06065
06066     if ( noext ) then
06067       cycle
06068     end if
06069
06070     if ( .not. extall ) go to 50
06071     erlarg = erlarg-erlast
06072     if ( abs(b1-a1) > small ) erlarg = erlarg+erro12
06073     if ( extrap ) go to 70
06074 !
06075 !  Test whether the interval to be bisected next is the
06076 !  smallest interval.
```

```
06077 !
06078 50  continue
06079
06080     width = abs(blist(maxerr)-alist(maxerr))
06081
06082     if ( width > small ) then
06083       cycle
06084     end if
06085
06086     if ( extall ) go to 60
06087 !
06088 !  Test whether we can start with the extrapolation procedure
06089 !  (we do this if we integrate over the next interval with
06090 !  use of a Gauss-Kronrod rule - see QC25O).
06091 !
06092     small = small*5.0e-01
06093
06094     if ( 2.5e-01*width*domega > 2.0e+00 ) then
06095       cycle
06096     end if
06097
06098     extall = .true.
06099     go to 130
06100
06101 60  continue
06102
06103     extrap = .true.
06104     nrmax = 2
06105
06106 70  continue
06107
06108     if ( ierro == 3 .or. erlarg <= ertest ) go to 90
06109 !
06110 !  The smallest interval has the largest error.
06111 !  Before bisecting decrease the sum of the errors over the
06112 !  larger intervals (ERLARG) and perform extrapolation.
06113 !
06114     jupbnd = last
06115
06116     if ( last > (limit/2+2) ) then
06117       jupbnd = limit+3-last
06118     end if
06119
06120     id = nrmax
06121
06122     do k = id, jupbnd
06123       maxerr = iord(nrmax)
06124       errmax = elist(maxerr)
06125       if ( abs(blist(maxerr)-alist(maxerr)) > small ) go to 140
06126       nrmax = nrmax+1
06127     end do
06128 !
06129 !  Perform extrapolation.
06130 !
06131 90  continue
06132
06133     numrl2 = numrl2+1
06134     rlist2(numrl2) = area
06135
06136     if ( numrl2 < 3 ) go to 110
06137
06138     call qextr ( numrl2, rlist2, reseps, abseps, res3la, nres )
06139     ktmin = ktmin+1
06140
06141     if ( ktmin > 5.and.abserr < 1.0e-03*errsum ) then
06142       ier = 5
06143     end if
06144
06145     if ( abseps >= abserr ) go to 100
06146
06147     ktmin = 0
06148     abserr = abseps
06149     result = reseps
06150     correc = erlarg
06151     ertest = max( epsabs, epsrel*abs(reseps))
06152
06153     if ( abserr <= ertest ) then
06154       exit
06155     end if
06156 !
06157 !  Prepare bisection of the smallest interval.
06158 !
06159 100 continue
06160
06161     if ( numrl2 == 1 ) then
06162       noext = .true.
06163     end if
```

```
06164
06165     if ( ier == 5 ) then
06166       exit
06167     end if
06168
06169 110 continue
06170
06171     maxerr = iord(1)
06172     errmax = elist(maxerr)
06173     nrmax = 1
06174     extrap = .false.
06175     small = small*5.0e-01
06176     erlarg = errsum
06177     cycle
06178
06179 120 continue
06180
06181     small = small * 5.0e-01
06182     numrl2 = numrl2 + 1
06183     rlist2(numrl2) = area
06184
06185 130 continue
06186
06187     ertest = errbnd
06188     erlarg = errsum
06189
06190 140 continue
06191
06192   end do
06193 !
06194 !  set the final result.
06195 !
06196   if ( abserr == huge( abserr ) .or. nres == 0 ) then
06197     go to 170
06198   end if
06199
06200   if ( ier+ierro == 0 ) go to 165
06201   if ( ierro == 3 ) abserr = abserr+correc
06202   if ( ier == 0 ) ier = 3
06203   if ( result /= 0.0e+00.and.area /= 0.0e+00 ) go to 160
06204   if ( abserr > errsum ) go to 170
06205   if ( area == 0.0e+00 ) go to 190
06206   go to 165
06207
06208 160 continue
06209
06210   if ( abserr/abs(result) > errsum/abs(area) ) go to 170
06211 !
06212 !  Test on divergence.
06213 !
06214   165 continue
06215
06216   if ( ksgn == (-1) .and. max( abs(result),abs(area)) <=  &
06217    defabs*1.0e-02 ) go to 190
06218
06219   if ( 1.0e-02 > (result/area) .or. (result/area) > 1.0e+02 &
06220    .or. errsum >= abs(area) ) ier = 6
06221
06222   go to 190
06223 !
06224 !  Compute global integral sum.
06225 !
06226 170 continue
06227
06228   result = sum( rlist(1:last) )
06229
06230   abserr = errsum
06231
06232 190 continue
06233
06234   if (ier > 2) ier=ier-1
06235
06236 200 continue
06237
06238   if ( integr == 2 .and. omega < 0.0e+00 ) then
06239     result = -result
06240   end if
06241
06242   return
06243 end subroutine qfour
06244 subroutine qk15 ( f, a, b, result, abserr, resabs, resasc )
06245
06246 !*******************************************************************************80
06247 !
06248 !! QK15 carries out a 15 point Gauss-Kronrod quadrature rule.
06249 !
06250 !  Discussion:
```

```
06251 !
06252 !    This routine approximates
06253 !      I = integral ( A <= X <= B ) F(X) dx
06254 !    with an error estimate, and
06255 !      J = integral ( A <= X <= B ) | F(X) | dx
06256 !
06257 !  Author:
06258 !
06259 !    Robert Piessens, Elise de Doncker-Kapenger,
06260 !    Christian Ueberhuber, David Kahaner
06261 !
06262 !  Reference:
06263 !
06264 !    Robert Piessens, Elise de Doncker-Kapenger,
06265 !    Christian Ueberhuber, David Kahaner,
06266 !    QUADPACK, a Subroutine Package for Automatic Integration,
06267 !    Springer Verlag, 1983
06268 !
06269 !  Parameters:
06270 !
06271 !    Input, external real F, the name of the function routine, of the form
06272 !      function f ( x )
06273 !      real f
06274 !      real x
06275 !    which evaluates the integrand function.
06276 !
06277 !    Input, real A, B, the limits of integration.
06278 !
06279 !    Output, real RESULT, the estimated value of the integral.
06280 !    RESULT is computed by applying the 15-point Kronrod rule (RESK)
06281 !    obtained by optimal addition of abscissae to the 7-point Gauss rule
06282 !    (RESG).
06283 !
06284 !    Output, real ABSERR, an estimate of | I - RESULT |.
06285 !
06286 !    Output, real RESABS, approximation to the integral of the absolute
06287 !    value of F.
06288 !
06289 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
06290 !    over [A,B].
06291 !
06292 !  Local Parameters:
06293 !
06294 !           the abscissae and weights are given for the interval (-1,1).
06295 !           because of symmetry only the positive abscissae and their
06296 !           corresponding weights are given.
06297 !
06298 !           xgk    - abscissae of the 15-point Kronrod rule
06299 !                    xgk(2), xgk(4), ...  abscissae of the 7-point
06300 !                    Gauss rule
06301 !                    xgk(1), xgk(3), ...  abscissae which are optimally
06302 !                    added to the 7-point Gauss rule
06303 !
06304 !           wgk    - weights of the 15-point Kronrod rule
06305 !
06306 !           wg     - weights of the 7-point Gauss rule
06307 !
06308 !           centr  - mid point of the interval
06309 !           hlgth  - half-length of the interval
06310 !           absc   - abscissa
06311 !           fval*  - function value
06312 !           resg   - result of the 7-point Gauss formula
06313 !           resk   - result of the 15-point Kronrod formula
06314 !           reskh  - approximation to the mean value of f over (a,b),
06315 !                    i.e. to i/(b-a)
06316 !
06317   implicit none
06318
06319   real(kind=params_wp) a
06320   real(kind=params_wp) absc
06321   real(kind=params_wp) abserr
06322   real(kind=params_wp) b
06323   real(kind=params_wp) centr
06324   real(kind=params_wp) dhlgth
06325   real(kind=params_wp), external :: f
06326   real(kind=params_wp) fc
06327   real(kind=params_wp) fsum
06328   real(kind=params_wp) fval1
06329   real(kind=params_wp) fval2
06330   real(kind=params_wp) fv1(7)
06331   real(kind=params_wp) fv2(7)
06332   real(kind=params_wp) hlgth
06333   integer j
06334   integer jtw
06335   integer jtwm1
06336   real(kind=params_wp) resabs
06337   real(kind=params_wp) resasc
```

```
06338   real(kind=params_wp) resg
06339   real(kind=params_wp) resk
06340   real(kind=params_wp) reskh
06341   real(kind=params_wp) result
06342   real(kind=params_wp) wg(4)
06343   real(kind=params_wp) wgk(8)
06344   real(kind=params_wp) xgk(8)
06345
06346   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8)/ &
06347        9.914553711208126e-01,    9.491079123427585e-01, &
06348        8.648644233597691e-01,    7.415311855993944e-01, &
06349        5.860872354676911e-01,    4.058451513773972e-01, &
06350        2.077849550078985e-01,    0.0e+00               /
06351   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8)/ &
06352        2.293532201052922e-02,    6.309209262997855e-02, &
06353        1.047900103222502e-01,    1.406532597155259e-01, &
06354        1.690047266392679e-01,    1.903505780647854e-01, &
06355        2.044329400752989e-01,    2.094821410847278e-01/
06356   data wg(1),wg(2),wg(3),wg(4)/ &
06357        1.294849661688697e-01,    2.797053914892767e-01, &
06358        3.818300505051189e-01,    4.179591836734694e-01/
06359 !
06360   centr = 5.0e-01*(a+b)
06361   hlgth = 5.0e-01*(b-a)
06362   dhlgth = abs(hlgth)
06363 !
06364 !  Compute the 15-point Kronrod approximation to the integral,
06365 !  and estimate the absolute error.
06366 !
06367   fc = f(centr)
06368   resg = fc*wg(4)
06369   resk = fc*wgk(8)
06370   resabs = abs(resk)
06371
06372   do j = 1, 3
06373     jtw = j*2
06374     absc = hlgth*xgk(jtw)
06375     fval1 = f(centr-absc)
06376     fval2 = f(centr+absc)
06377     fv1(jtw) = fval1
06378     fv2(jtw) = fval2
06379     fsum = fval1+fval2
06380     resg = resg+wg(j)*fsum
06381     resk = resk+wgk(jtw)*fsum
06382     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
06383   end do
06384
06385   do j = 1, 4
06386     jtwm1 = j*2-1
06387     absc = hlgth*xgk(jtwm1)
06388     fval1 = f(centr-absc)
06389     fval2 = f(centr+absc)
06390     fv1(jtwm1) = fval1
06391     fv2(jtwm1) = fval2
06392     fsum = fval1+fval2
06393     resk = resk+wgk(jtwm1)*fsum
06394     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
06395   end do
06396
06397   reskh = resk * 5.0e-01
06398   resasc = wgk(8)*abs(fc-reskh)
06399
06400   do j = 1, 7
06401     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
06402   end do
06403
06404   result = resk*hlgth
06405   resabs = resabs*dhlgth
06406   resasc = resasc*dhlgth
06407   abserr = abs((resk-resg)*hlgth)
06408
06409   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00 ) then
06410     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02_params_wp*abserr/resasc)**1.5e+00_params_wp)
06411   end if
06412
06413   if ( resabs > tiny( resabs ) / (5.0e+01* epsilon( resabs ) ) ) then
06414     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
06415   end if
06416
06417   return
06418 end subroutine qk15
06419 subroutine qk15i ( f, boun, inf, a, b, result, abserr, resabs, resasc )
06420
06421 !***************************************************************************80
06422 !
06423 !! QK15I applies a 15 point Gauss-Kronrod quadrature on an infinite interval.
06424 !
```

```
06425 !  Discussion:
06426 !
06427 !    The original infinite integration range is mapped onto the interval
06428 !    (0,1) and (a,b) is a part of (0,1).  The routine then computes:
06429 !
06430 !    i = integral of transformed integrand over (a,b),
06431 !    j = integral of abs(transformed integrand) over (a,b).
06432 !
06433 !  Author:
06434 !
06435 !    Robert Piessens, Elise de Doncker-Kapenger,
06436 !    Christian Ueberhuber, David Kahaner
06437 !
06438 !  Reference:
06439 !
06440 !    Robert Piessens, Elise de Doncker-Kapenger,
06441 !    Christian Ueberhuber, David Kahaner,
06442 !    QUADPACK, a Subroutine Package for Automatic Integration,
06443 !    Springer Verlag, 1983
06444 !
06445 !  Parameters:
06446 !
06447 !    Input, external real F, the name of the function routine, of the form
06448 !      function f ( x )
06449 !      real f
06450 !      real x
06451 !    which evaluates the integrand function.
06452 !
06453 !    Input, real BOUN, the finite bound of the original integration range,
06454 !    or zero if INF is 2.
06455 !
06456 !    Input, integer INF, indicates the type of the interval.
06457 !    -1: the original interval is (-infinity,BOUN),
06458 !    +1, the original interval is (BOUN,+infinity),
06459 !    +2, the original interval is (-infinity,+infinity) and
06460 !    the integral is computed as the sum of two integrals, one
06461 !    over (-infinity,0) and one over (0,+infinity).
06462 !
06463 !    Input, real A, B, the limits of integration, over a subrange of [0,1].
06464 !
06465 !    Output, real RESULT, the estimated value of the integral.
06466 !    RESULT is computed by applying the 15-point Kronrod rule (RESK) obtained
06467 !    by optimal addition of abscissae to the 7-point Gauss rule (RESG).
06468 !
06469 !    Output, real ABSERR, an estimate of | I - RESULT |.
06470 !
06471 !    Output, real RESABS, approximation to the integral of the absolute
06472 !    value of F.
06473 !
06474 !    Output, real RESASC, approximation to the integral of the
06475 !    transformated integrand | F-I/(B-A) | over [A,B].
06476 !
06477 !  Local Parameters:
06478 !
06479 !           centr  - mid point of the interval
06480 !           hlgth  - half-length of the interval
06481 !           absc*  - abscissa
06482 !           tabsc* - transformed abscissa
06483 !           fval*  - function value
06484 !           resg   - result of the 7-point Gauss formula
06485 !           resk   - result of the 15-point Kronrod formula
06486 !           reskh  - approximation to the mean value of the transformed
06487 !                    integrand over (a,b), i.e. to i/(b-a)
06488 !
06489   implicit none
06490
06491   real(kind=params_wp) a
06492   real(kind=params_wp) absc
06493   real(kind=params_wp) absc1
06494   real(kind=params_wp) absc2
06495   real(kind=params_wp) abserr
06496   real(kind=params_wp) b
06497   real(kind=params_wp) boun
06498   real(kind=params_wp) centr
06499   real(kind=params_wp) dinf
06500   real(kind=params_wp), external :: f
06501   real(kind=params_wp) fc
06502   real(kind=params_wp) fsum
06503   real(kind=params_wp) fval1
06504   real(kind=params_wp) fval2
06505   real(kind=params_wp) fv1(7)
06506   real(kind=params_wp) fv2(7)
06507   real(kind=params_wp) hlgth
06508   integer inf
06509   integer j
06510   real(kind=params_wp) resabs
06511   real(kind=params_wp) resasc
```

```
06512   real(kind=params_wp) resg
06513   real(kind=params_wp) resk
06514   real(kind=params_wp) reskh
06515   real(kind=params_wp) result
06516   real(kind=params_wp) tabsc1
06517   real(kind=params_wp) tabsc2
06518   real(kind=params_wp) wg(8)
06519   real(kind=params_wp) wgk(8)
06520   real(kind=params_wp) xgk(8)
06521 !
06522 !  the abscissae and weights are supplied for the interval
06523 !  (-1,1).  because of symmetry only the positive abscissae and
06524 !  their corresponding weights are given.
06525 !
06526 !           xgk    - abscissae of the 15-point Kronrod rule
06527 !                    xgk(2), xgk(4), ... abscissae of the 7-point Gauss
06528 !                    rule
06529 !                    xgk(1), xgk(3), ...  abscissae which are optimally
06530 !                    added to the 7-point Gauss rule
06531 !
06532 !           wgk    - weights of the 15-point Kronrod rule
06533 !
06534 !           wg     - weights of the 7-point Gauss rule, corresponding
06535 !                    to the abscissae xgk(2), xgk(4), ...
06536 !                    wg(1), wg(3), ... are set to zero.
06537 !
06538   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8)/ &
06539        9.914553711208126e-01,     9.491079123427585e-01, &
06540        8.648644233597691e-01,     7.415311855993944e-01, &
06541        5.860872354676911e-01,     4.058451513773972e-01, &
06542        2.077849550078985e-01,     0.0000000000000000e+00/
06543
06544   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8)/ &
06545        2.293532201052922e-02,     6.309209262997855e-02, &
06546        1.047900103222502e-01,     1.406532597155259e-01, &
06547        1.690047266392679e-01,     1.903505780647854e-01, &
06548        2.044329400752989e-01,     2.094821410847278e-01/
06549
06550   data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8)/ &
06551        0.0000000000000000e+00,     1.294849661688697e-01, &
06552        0.0000000000000000e+00,     2.797053914892767e-01, &
06553        0.0000000000000000e+00,     3.818300505051189e-01, &
06554        0.0000000000000000e+00,     4.179591836734694e-01/
06555
06556   dinf = min( 1, inf )
06557
06558   centr = 5.0e-01*(a+b)
06559   hlgth = 5.0e-01*(b-a)
06560   tabsc1 = boun+dinf*(1.0e+00_params_wp-centr)/centr
06561   fval1 = f(tabsc1)
06562   if ( inf == 2 ) fval1 = fval1+f(-tabsc1)
06563   fc = (fval1/centr)/centr
06564 !
06565 !  Compute the 15-point Kronrod approximation to the integral,
06566 !  and estimate the error.
06567 !
06568   resg = wg(8)*fc
06569   resk = wgk(8)*fc
06570   resabs = abs(resk)
06571
06572   do j = 1, 7
06573
06574     absc = hlgth*xgk(j)
06575     absc1 = centr-absc
06576     absc2 = centr+absc
06577     tabsc1 = boun+dinf*(1.0e+00_params_wp-absc1)/absc1
06578     tabsc2 = boun+dinf*(1.0e+00_params_wp-absc2)/absc2
06579     fval1 = f(tabsc1)
06580     fval2 = f(tabsc2)
06581
06582     if ( inf == 2 ) then
06583       fval1 = fval1+f(-tabsc1)
06584       fval2 = fval2+f(-tabsc2)
06585     end if
06586
06587     fval1 = (fval1/absc1)/absc1
06588     fval2 = (fval2/absc2)/absc2
06589     fv1(j) = fval1
06590     fv2(j) = fval2
06591     fsum = fval1+fval2
06592     resg = resg+wg(j)*fsum
06593     resk = resk+wgk(j)*fsum
06594     resabs = resabs+wgk(j)*(abs(fval1)+abs(fval2))
06595   end do
06596
06597   reskh = resk * 5.0e-01
06598   resasc = wgk(8) * abs(fc-reskh)
```

```
06599
06600   do j = 1, 7
06601     resasc = resasc + wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
06602   end do
06603
06604   result = resk * hlgth
06605   resasc = resasc * hlgth
06606   resabs = resabs * hlgth
06607   abserr = abs( ( resk - resg ) * hlgth )
06608
06609   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) then
06610     abserr = resasc* min( 1.0e+00_params_wp,(2.0e+02_params_wp*abserr/resasc)**1.5e+00_params_wp)
06611   end if
06612
06613   if ( resabs > tiny( resabs ) / ( 5.0e+01 * epsilon( resabs ) ) ) then
06614     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
06615   end if
06616
06617   return
06618 end subroutine qk15i
06619 subroutine qk15w ( f, w, p1, p2, p3, p4, kp, a, b, result, abserr, resabs, &
06620   resasc )
06621
06622 !*******************************************************************************80
06623 !
06624 !! QK15W applies a 15 point Gauss-Kronrod rule for a weighted integrand.
06625 !
06626 !  Discussion:
06627 !
06628 !    This routine approximates
06629 !      i = integral of f*w over (a,b),
06630 !    with error estimate, and
06631 !      j = integral of abs(f*w) over (a,b)
06632 !
06633 !  Author:
06634 !
06635 !    Robert Piessens, Elise de Doncker-Kapenger,
06636 !    Christian Ueberhuber, David Kahaner
06637 !
06638 !  Reference:
06639 !
06640 !    Robert Piessens, Elise de Doncker-Kapenger,
06641 !    Christian Ueberhuber, David Kahaner,
06642 !    QUADPACK, a Subroutine Package for Automatic Integration,
06643 !    Springer Verlag, 1983
06644 !
06645 !  Parameters:
06646 !
06647 !    Input, external real F, the name of the function routine, of the form
06648 !      function f ( x )
06649 !      real f
06650 !      real x
06651 !    which evaluates the integrand function.
06652 !
06653 !              w      - real
06654 !                       function subprogram defining the integrand
06655 !                       weight function w(x). the actual name for w
06656 !                       needs to be declared e x t e r n a l in the
06657 !                       calling program.
06658 !
06659 !    ?, real P1, P2, P3, P4, parameters in the weight function
06660 !
06661 !    Input, integer KP, key for indicating the type of weight function
06662 !
06663 !    Input, real A, B, the limits of integration.
06664 !
06665 !    Output, real RESULT, the estimated value of the integral.
06666 !    RESULT is computed by applying the 15-point Kronrod rule (RESK) obtained by
06667 !    optimal addition of abscissae to the 7-point Gauss rule (RESG).
06668 !
06669 !    Output, real ABSERR, an estimate of | I - RESULT |.
06670 !
06671 !    Output, real RESABS, approximation to the integral of the absolute
06672 !    value of F.
06673 !
06674 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
06675 !    over [A,B].
06676 !
06677 !  Local Parameters:
06678 !
06679 !           centr  - mid point of the interval
06680 !           hlgth  - half-length of the interval
06681 !           absc*  - abscissa
06682 !           fval*  - function value
06683 !           resg   - result of the 7-point Gauss formula
06684 !           resk   - result of the 15-point Kronrod formula
06685 !           reskh  - approximation to the mean value of f*w over (a,b),
```

```
06686 !                   i.e. to i/(b-a)
06687 !
06688   implicit none
06689
06690   real(kind=params_wp) a
06691   real(kind=params_wp) absc
06692   real(kind=params_wp) absc1
06693   real(kind=params_wp) absc2
06694   real(kind=params_wp) abserr
06695   real(kind=params_wp) b
06696   real(kind=params_wp) centr
06697   real(kind=params_wp) dhlgth
06698   real(kind=params_wp), external :: f
06699   real(kind=params_wp) fc
06700   real(kind=params_wp) fsum
06701   real(kind=params_wp) fval1
06702   real(kind=params_wp) fval2
06703   real(kind=params_wp) fv1(7)
06704   real(kind=params_wp) fv2(7)
06705   real(kind=params_wp) hlgth
06706   integer j
06707   integer jtw
06708   integer jtwm1
06709   integer kp
06710   real(kind=params_wp) p1
06711   real(kind=params_wp) p2
06712   real(kind=params_wp) p3
06713   real(kind=params_wp) p4
06714   real(kind=params_wp) resabs
06715   real(kind=params_wp) resasc
06716   real(kind=params_wp) resg
06717   real(kind=params_wp) resk
06718   real(kind=params_wp) reskh
06719   real(kind=params_wp) result
06720   real(kind=params_wp), external :: w
06721   real(kind=params_wp), dimension ( 4 ) :: wg = (/ &
06722     1.294849661688697e-01,     2.797053914892767e-01, &
06723     3.818300505051889e-01,     4.179591836734694e-01 /)
06724   real(kind=params_wp) wgk(8)
06725   real(kind=params_wp) xgk(8)
06726 !
06727 !  the abscissae and weights are given for the interval (-1,1).
06728 !  because of symmetry only the positive abscissae and their
06729 !  corresponding weights are given.
06730 !
06731 !           xgk    - abscissae of the 15-point Gauss-Kronrod rule
06732 !                    xgk(2), xgk(4), ... abscissae of the 7-point Gauss
06733 !                    rule
06734 !                    xgk(1), xgk(3), ... abscissae which are optimally
06735 !                    added to the 7-point Gauss rule
06736 !
06737 !           wgk    - weights of the 15-point Gauss-Kronrod rule
06738 !
06739 !           wg     - weights of the 7-point Gauss rule
06740 !
06741   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8)/ &
06742        9.914553711208126e-01,     9.491079123427585e-01, &
06743        8.648644233597691e-01,     7.415311855993944e-01, &
06744        5.860872354676911e-01,     4.058451513773972e-01, &
06745        2.077849550789850e-01,     0.000000000000000e+00/
06746
06747   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8)/ &
06748        2.293532201052922e-02,     6.309209262997855e-02, &
06749        1.047900103222502e-01,     1.406532597155259e-01, &
06750        1.690047266392679e-01,     1.903505780647854e-01, &
06751        2.044329400752989e-01,     2.094821410847278e-01/
06752 !
06753   centr = 5.0e-01*(a+b)
06754   hlgth = 5.0e-01*(b-a)
06755   dhlgth = abs(hlgth)
06756 !
06757 !  Compute the 15-point Kronrod approximation to the integral,
06758 !  and estimate the error.
06759 !
06760   fc = f(centr)*w(centr,p1,p2,p3,p4,kp)
06761   resg = wg(4)*fc
06762   resk = wgk(8)*fc
06763   resabs = abs(resk)
06764
06765   do j = 1, 3
06766     jtw = j*2
06767     absc = hlgth*xgk(jtw)
06768     absc1 = centr-absc
06769     absc2 = centr+absc
06770     fval1 = f(absc1)*w(absc1,p1,p2,p3,p4,kp)
06771     fval2 = f(absc2)*w(absc2,p1,p2,p3,p4,kp)
06772     fv1(jtw) = fval1
```

```
06773     fv2(jtw) = fval2
06774     fsum = fval1+fval2
06775     resg = resg+wg(j)*fsum
06776     resk = resk+wgk(jtw)*fsum
06777     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
06778   end do
06779
06780   do j = 1, 4
06781     jtwm1 = j*2-1
06782     absc = hlgth*xgk(jtwm1)
06783     absc1 = centr-absc
06784     absc2 = centr+absc
06785     fval1 = f(absc1)*w(absc1,p1,p2,p3,p4,kp)
06786     fval2 = f(absc2)*w(absc2,p1,p2,p3,p4,kp)
06787     fv1(jtwm1) = fval1
06788     fv2(jtwm1) = fval2
06789     fsum = fval1+fval2
06790     resk = resk+wgk(jtwm1)*fsum
06791     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
06792   end do
06793
06794   reskh = resk*5.0e-01
06795   resasc = wgk(8)*abs(fc-reskh)
06796
06797   do j = 1, 7
06798     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
06799   end do
06800
06801   result = resk*hlgth
06802   resabs = resabs*dhlgth
06803   resasc = resasc*dhlgth
06804   abserr = abs((resk-resg)*hlgth)
06805
06806   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) then
06807     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00_params_wp)
06808   end if
06809
06810   if ( resabs > tiny( resabs ) /(5.0e+01* epsilon( resabs ) ) ) then
06811     abserr = max( ( epsilon( resabs ) * 5.0e+01)*resabs,abserr)
06812   end if
06813
06814   return
06815 end subroutine qk15w
06816 subroutine qk21 ( f, a, b, result, abserr, resabs, resasc )
06817
06818 !*****************************************************************************80
06819 !
06820 !! QK21 carries out a 21 point Gauss-Kronrod quadrature rule.
06821 !
06822 !  Discussion:
06823 !
06824 !    This routine approximates
06825 !      I = integral ( A <= X <= B ) F(X) dx
06826 !    with an error estimate, and
06827 !      J = integral ( A <= X <= B ) | F(X) | dx
06828 !
06829 !  Author:
06830 !
06831 !    Robert Piessens, Elise de Doncker-Kapenger,
06832 !    Christian Ueberhuber, David Kahaner
06833 !
06834 !  Reference:
06835 !
06836 !    Robert Piessens, Elise de Doncker-Kapenger,
06837 !    Christian Ueberhuber, David Kahaner,
06838 !    QUADPACK, a Subroutine Package for Automatic Integration,
06839 !    Springer Verlag, 1983
06840 !
06841 !  Parameters:
06842 !
06843 !    Input, external real F, the name of the function routine, of the form
06844 !      function f ( x )
06845 !      real f
06846 !      real x
06847 !    which evaluates the integrand function.
06848 !
06849 !    Input, real A, B, the limits of integration.
06850 !
06851 !    Output, real RESULT, the estimated value of the integral.
06852 !    RESULT is computed by applying the 21-point Kronrod rule (resk)
06853 !    obtained by optimal addition of abscissae to the 10-point Gauss
06854 !    rule (resg).
06855 !
06856 !    Output, real ABSERR, an estimate of | I - RESULT |.
06857 !
06858 !    Output, real RESABS, approximation to the integral of the absolute
06859 !    value of F.
```

```
06860 !
06861 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
06862 !    over [A,B].
06863 !
06864   implicit none
06865
06866   real(kind=params_wp) a
06867   real(kind=params_wp) absc
06868   real(kind=params_wp) abserr
06869   real(kind=params_wp) b
06870   real(kind=params_wp) centr
06871   real(kind=params_wp) dhlgth
06872   real(kind=params_wp), external :: f
06873   real(kind=params_wp) fc
06874   real(kind=params_wp) fsum
06875   real(kind=params_wp) fval1
06876   real(kind=params_wp) fval2
06877   real(kind=params_wp) fv1(10)
06878   real(kind=params_wp) fv2(10)
06879   real(kind=params_wp) hlgth
06880   integer j
06881   integer jtw
06882   integer jtwm1
06883   real(kind=params_wp) resabs
06884   real(kind=params_wp) resasc
06885   real(kind=params_wp) resg
06886   real(kind=params_wp) resk
06887   real(kind=params_wp) reskh
06888   real(kind=params_wp) result
06889   real(kind=params_wp) wg(5)
06890   real(kind=params_wp) wgk(11)
06891   real(kind=params_wp) xgk(11)
06892 !
06893 !           the abscissae and weights are given for the interval (-1,1).
06894 !           because of symmetry only the positive abscissae and their
06895 !           corresponding weights are given.
06896 !
06897 !           xgk    - abscissae of the 21-point Kronrod rule
06898 !                    xgk(2), xgk(4), ...  abscissae of the 10-point
06899 !                    Gauss rule
06900 !                    xgk(1), xgk(3), ...  abscissae which are optimally
06901 !                    added to the 10-point Gauss rule
06902 !
06903 !           wgk    - weights of the 21-point Kronrod rule
06904 !
06905 !           wg     - weights of the 10-point Gauss rule
06906 !
06907   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
06908     xgk(9),xgk(10),xgk(11)/ &
06909        9.956571630258081e-01,     9.739065285171717e-01, &
06910        9.301574913557082e-01,     8.650633666889845e-01, &
06911        7.808177265864169e-01,     6.794095682990244e-01, &
06912        5.627571346686047e-01,     4.333953941292472e-01, &
06913        2.943928627014602e-01,     1.488743389816312e-01, &
06914        0.000000000000000e+00/
06915 !
06916   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
06917     wgk(9),wgk(10),wgk(11)/ &
06918        1.169463886737187e-02,     3.255816230796473e-02, &
06919        5.475589657435200e-02,     7.503967481091995e-02, &
06920        9.312545458369761e-02,     1.093871588022976e-01, &
06921        1.234919762620659e-01,     1.347092173114733e-01, &
06922        1.427759385770601e-01,     1.477391049013385e-01, &
06923        1.494455540029169e-01/
06924 !
06925   data wg(1),wg(2),wg(3),wg(4),wg(5)/ &
06926        6.667134430868814e-02,     1.494513491505806e-01, &
06927        2.190863625159820e-01,     2.692667193099964e-01, &
06928        2.955242247147529e-01/
06929 !
06930 !
06931 !           list of major variables
06932 !
06933 !           centr  - mid point of the interval
06934 !           hlgth  - half-length of the interval
06935 !           absc   - abscissa
06936 !           fval*  - function value
06937 !           resg   - result of the 10-point Gauss formula
06938 !           resk   - result of the 21-point Kronrod formula
06939 !           reskh  - approximation to the mean value of f over (a,b),
06940 !                    i.e. to i/(b-a)
06941 !
06942   centr = 5.0e-01*(a+b)
06943   hlgth = 5.0e-01*(b-a)
06944   dhlgth = abs(hlgth)
06945 !
06946 !  Compute the 21-point Kronrod approximation to the
```

```
06947 !  integral, and estimate the absolute error.
06948 !
06949   resg = 0.0e+00
06950   fc = f(centr)
06951   resk = wgk(11)*fc
06952   resabs = abs(resk)
06953
06954   do j = 1, 5
06955     jtw = 2*j
06956     absc = hlgth*xgk(jtw)
06957     fval1 = f(centr-absc)
06958     fval2 = f(centr+absc)
06959     fv1(jtw) = fval1
06960     fv2(jtw) = fval2
06961     fsum = fval1+fval2
06962     resg = resg+wg(j)*fsum
06963     resk = resk+wgk(jtw)*fsum
06964     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
06965   end do
06966
06967   do j = 1, 5
06968     jtwm1 = 2*j-1
06969     absc = hlgth*xgk(jtwm1)
06970     fval1 = f(centr-absc)
06971     fval2 = f(centr+absc)
06972     fv1(jtwm1) = fval1
06973     fv2(jtwm1) = fval2
06974     fsum = fval1+fval2
06975     resk = resk+wgk(jtwm1)*fsum
06976     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
06977   end do
06978
06979   reskh = resk*5.0e-01
06980   resasc = wgk(11)*abs(fc-reskh)
06981
06982   do j = 1, 10
06983     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
06984   end do
06985
06986   result = resk*hlgth
06987   resabs = resabs*dhlgth
06988   resasc = resasc*dhlgth
06989   abserr = abs((resk-resg)*hlgth)
06990
06991   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) then
06992     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
06993   end if
06994
06995   if ( resabs > tiny( resabs ) /(5.0e+01* epsilon( resabs ) )) then
06996     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
06997   end if
06998
06999   return
07000 end subroutine qk21
07001 subroutine qk31 ( f, a, b, result, abserr, resabs, resasc )
07002
07003 !*******************************************************************************80
07004 !
07005 !! QK31 carries out a 31 point Gauss-Kronrod quadrature rule.
07006 !
07007 !  Discussion:
07008 !
07009 !    This routine approximates
07010 !      I = integral ( A <= X <= B ) F(X) dx
07011 !    with an error estimate, and
07012 !      J = integral ( A <= X <= B ) | F(X) | dx
07013 !
07014 !  Author:
07015 !
07016 !    Robert Piessens, Elise de Doncker-Kapenger,
07017 !    Christian Ueberhuber, David Kahaner
07018 !
07019 !  Reference:
07020 !
07021 !    Robert Piessens, Elise de Doncker-Kapenger,
07022 !    Christian Ueberhuber, David Kahaner,
07023 !    QUADPACK, a Subroutine Package for Automatic Integration,
07024 !    Springer Verlag, 1983
07025 !
07026 !  Parameters:
07027 !
07028 !    Input, external real F, the name of the function routine, of the form
07029 !      function f ( x )
07030 !      real f
07031 !      real x
07032 !    which evaluates the integrand function.
07033 !
```

```
07034 !    Input, real A, B, the limits of integration.
07035 !
07036 !    Output, real RESULT, the estimated value of the integral.
07037 !                        result is computed by applying the 31-point
07038 !                        Gauss-Kronrod rule (resk), obtained by optimal
07039 !                        addition of abscissae to the 15-point Gauss
07040 !                        rule (resg).
07041 !
07042 !    Output, real ABSERR, an estimate of | I - RESULT |.
07043 !
07044 !    Output, real RESABS, approximation to the integral of the absolute
07045 !    value of F.
07046 !
07047 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
07048 !    over [A,B].
07049 !
07050   implicit none
07051
07052   real(kind=params_wp) a
07053   real(kind=params_wp) absc
07054   real(kind=params_wp) abserr
07055   real(kind=params_wp) b
07056   real(kind=params_wp) centr
07057   real(kind=params_wp) dhlgth
07058   real(kind=params_wp), external :: f
07059   real(kind=params_wp) fc
07060   real(kind=params_wp) fsum
07061   real(kind=params_wp) fval1
07062   real(kind=params_wp) fval2
07063   real(kind=params_wp) fv1(15)
07064   real(kind=params_wp) fv2(15)
07065   real(kind=params_wp) hlgth
07066   integer j
07067   integer jtw
07068   integer jtwm1
07069   real(kind=params_wp) resabs
07070   real(kind=params_wp) resasc
07071   real(kind=params_wp) resg
07072   real(kind=params_wp) resk
07073   real(kind=params_wp) reskh
07074   real(kind=params_wp) result
07075   real(kind=params_wp) wg(8)
07076   real(kind=params_wp) wgk(16)
07077   real(kind=params_wp) xgk(16)
07078 !
07079 !           the abscissae and weights are given for the interval (-1,1).
07080 !           because of symmetry only the positive abscissae and their
07081 !           corresponding weights are given.
07082 !
07083 !           xgk    - abscissae of the 31-point Kronrod rule
07084 !                    xgk(2), xgk(4), ...  abscissae of the 15-point
07085 !                    Gauss rule
07086 !                    xgk(1), xgk(3), ...  abscissae which are optimally
07087 !                    added to the 15-point Gauss rule
07088 !
07089 !           wgk    - weights of the 31-point Kronrod rule
07090 !
07091 !           wg     - weights of the 15-point Gauss rule
07092 !
07093   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
07094     xgk(9),xgk(10),xgk(11),xgk(12),xgk(13),xgk(14),xgk(15),xgk(16)/ &
07095        9.980022986933971e-01,   9.879925180204854e-01, &
07096        9.677390756791391e-01,   9.372733924007059e-01, &
07097        8.972645323440819e-01,   8.482065834104272e-01, &
07098        7.904185014424659e-01,   7.244177313601700e-01, &
07099        6.509967412974170e-01,   5.709721726085388e-01, &
07100        4.850818636402397e-01,   3.941513470775634e-01, &
07101        2.991800071531688e-01,   2.011940939974345e-01, &
07102        1.011420669187175e-01,   0.0e+00               /
07103   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
07104     wgk(9),wgk(10),wgk(11),wgk(12),wgk(13),wgk(14),wgk(15),wgk(16)/ &
07105        5.377479872923349e-03,   1.500794732931612e-02, &
07106        2.546084732671532e-02,   3.534636079137585e-02, &
07107        4.458975132476488e-02,   5.348152469092809e-02, &
07108        6.200956780067064e-02,   6.985412131872826e-02, &
07109        7.684968075772038e-02,   8.308050282313302e-02, &
07110        8.856444305621177e-02,   9.312659817082532e-02, &
07111        9.664272698362368e-02,   9.917359872179196e-02, &
07112        1.007698455238756e-01,   1.013300070147915e-01/
07113   data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8)/ &
07114        3.075324199611727e-02,   7.036604748810812e-02, &
07115        1.071592204671719e-01,   1.395706779261543e-01, &
07116        1.662692058169939e-01,   1.861610000155622e-01, &
07117        1.984314853271116e-01,   2.025782419255613e-01/
07118 !
07119 !
07120 !           list of major variables
```

```
07121 !
07122 !           centr  - mid point of the interval
07123 !           hlgth  - half-length of the interval
07124 !           absc   - abscissa
07125 !           fval*  - function value
07126 !           resg   - result of the 15-point Gauss formula
07127 !           resk   - result of the 31-point Kronrod formula
07128 !           reskh  - approximation to the mean value of f over (a,b),
07129 !                    i.e. to i/(b-a)
07130 !
07131   centr = 5.0e-01*(a+b)
07132   hlgth = 5.0e-01*(b-a)
07133   dhlgth = abs(hlgth)
07134 !
07135 !  Compute the 31-point Kronrod approximation to the integral,
07136 !  and estimate the absolute error.
07137 !
07138   fc = f(centr)
07139   resg = wg(8)*fc
07140   resk = wgk(16)*fc
07141   resabs = abs(resk)
07142
07143   do j = 1, 7
07144     jtw = j*2
07145     absc = hlgth*xgk(jtw)
07146     fval1 = f(centr-absc)
07147     fval2 = f(centr+absc)
07148     fv1(jtw) = fval1
07149     fv2(jtw) = fval2
07150     fsum = fval1+fval2
07151     resg = resg+wg(j)*fsum
07152     resk = resk+wgk(jtw)*fsum
07153     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
07154   end do
07155
07156   do j = 1, 8
07157     jtwm1 = j*2-1
07158     absc = hlgth*xgk(jtwm1)
07159     fval1 = f(centr-absc)
07160     fval2 = f(centr+absc)
07161     fv1(jtwm1) = fval1
07162     fv2(jtwm1) = fval2
07163     fsum = fval1+fval2
07164     resk = resk+wgk(jtwm1)*fsum
07165     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
07166   end do
07167
07168   reskh = resk*5.0e-01
07169   resasc = wgk(16)*abs(fc-reskh)
07170
07171   do j = 1, 15
07172     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
07173  end do
07174
07175   result = resk*hlgth
07176   resabs = resabs*dhlgth
07177   resasc = resasc*dhlgth
07178   abserr = abs((resk-resg)*hlgth)
07179
07180   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) &
07181     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
07182
07183   if ( resabs > tiny( resabs ) /(5.0e+01* epsilon( resabs ) )) then
07184     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
07185   end if
07186
07187   return
07188 end subroutine qk31
07189 subroutine qk41 ( f, a, b, result, abserr, resabs, resasc )
07190
07191 !*******************************************************************************80
07192 !
07193 !! QK41 carries out a 41 point Gauss-Kronrod quadrature rule.
07194 !
07195 !  Discussion:
07196 !
07197 !    This routine approximates
07198 !      I = integral ( A <= X <= B ) F(X) dx
07199 !    with an error estimate, and
07200 !      J = integral ( A <= X <= B ) | F(X) | dx
07201 !
07202 !  Author:
07203 !
07204 !    Robert Piessens, Elise de Doncker-Kapenger,
07205 !    Christian Ueberhuber, David Kahaner
07206 !
07207 !  Reference:
```

```
07208 !
07209 !    Robert Piessens, Elise de Doncker-Kapenger,
07210 !    Christian Ueberhuber, David Kahaner,
07211 !    QUADPACK, a Subroutine Package for Automatic Integration,
07212 !    Springer Verlag, 1983
07213 !
07214 !  Parameters:
07215 !
07216 !    Input, external real F, the name of the function routine, of the form
07217 !      function f ( x )
07218 !      real f
07219 !      real x
07220 !    which evaluates the integrand function.
07221 !
07222 !    Input, real A, B, the limits of integration.
07223 !
07224 !    Output, real RESULT, the estimated value of the integral.
07225 !                        result is computed by applying the 41-point
07226 !                        Gauss-Kronrod rule (resk) obtained by optimal
07227 !                        addition of abscissae to the 20-point Gauss
07228 !                        rule (resg).
07229 !
07230 !    Output, real ABSERR, an estimate of | I - RESULT |.
07231 !
07232 !    Output, real RESABS, approximation to the integral of the absolute
07233 !    value of F.
07234 !
07235 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
07236 !    over [A,B].
07237 !
07238 !  Local Parameters:
07239 !
07240 !           centr  - mid point of the interval
07241 !           hlgth  - half-length of the interval
07242 !           absc   - abscissa
07243 !           fval*  - function value
07244 !           resg   - result of the 20-point Gauss formula
07245 !           resk   - result of the 41-point Kronrod formula
07246 !           reskh  - approximation to mean value of f over (a,b), i.e.
07247 !                    to i/(b-a)
07248 !
07249   implicit none
07250
07251   real(kind=params_wp) a
07252   real(kind=params_wp) absc
07253   real(kind=params_wp) abserr
07254   real(kind=params_wp) b
07255   real(kind=params_wp) centr
07256   real(kind=params_wp) dhlgth
07257   real(kind=params_wp), external :: f
07258   real(kind=params_wp) fc
07259   real(kind=params_wp) fsum
07260   real(kind=params_wp) fval1
07261   real(kind=params_wp) fval2
07262   real(kind=params_wp) fv1(20)
07263   real(kind=params_wp) fv2(20)
07264   real(kind=params_wp) hlgth
07265   integer j
07266   integer jtw
07267   integer jtwm1
07268   real(kind=params_wp) resabs
07269   real(kind=params_wp) resasc
07270   real(kind=params_wp) resg
07271   real(kind=params_wp) resk
07272   real(kind=params_wp) reskh
07273   real(kind=params_wp) result
07274   real(kind=params_wp) wg(10)
07275   real(kind=params_wp) wgk(21)
07276   real(kind=params_wp) xgk(21)
07277 !
07278 !           the abscissae and weights are given for the interval (-1,1).
07279 !           because of symmetry only the positive abscissae and their
07280 !           corresponding weights are given.
07281 !
07282 !           xgk    - abscissae of the 41-point Gauss-Kronrod rule
07283 !                    xgk(2), xgk(4), ...  abscissae of the 20-point
07284 !                    Gauss rule
07285 !                    xgk(1), xgk(3), ...  abscissae which are optimally
07286 !                    added to the 20-point Gauss rule
07287 !
07288 !           wgk    - weights of the 41-point Gauss-Kronrod rule
07289 !
07290 !           wg     - weights of the 20-point Gauss rule
07291 !
07292   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
07293     xgk(9),xgk(10),xgk(11),xgk(12),xgk(13),xgk(14),xgk(15),xgk(16), &
07294     xgk(17),xgk(18),xgk(19),xgk(20),xgk(21)/ &
```

```
07295         9.988590315882777e-01,   9.931285991850949e-01, &
07296         9.815078774502503e-01,   9.639719272779138e-01, &
07297         9.408226338317548e-01,   9.122344282513259e-01, &
07298         8.782768112522820e-01,   8.391169718222188e-01, &
07299         7.950414288375512e-01,   7.463319064601508e-01, &
07300         6.932376563347514e-01,   6.360536807265150e-01, &
07301         5.751404468197103e-01,   5.108670019508271e-01, &
07302         4.435931752387251e-01,   3.737060887154196e-01, &
07303         3.016278681149130e-01,   2.277858511416451e-01, &
07304         1.526054652409227e-01,   7.652652113349733e-02, &
07305         0.0e+00                /
07306    data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
07307      wgk(9),wgk(10),wgk(11),wgk(12),wgk(13),wgk(14),wgk(15),wgk(16), &
07308      wgk(17),wgk(18),wgk(19),wgk(20),wgk(21)/ &
07309         3.073583718520532e-03,   8.600269855642942e-03, &
07310         1.462616925697125e-02,   2.038837346126652e-02, &
07311         2.588213360495116e-02,   3.128730677703280e-02, &
07312         3.660016975820080e-02,   4.166887332797369e-02, &
07313         4.643482186749767e-02,   5.094457392372869e-02, &
07314         5.519510534828599e-02,   5.911140088063957e-02, &
07315         6.265323755478117e-02,   6.583459713361842e-02, &
07316         6.864867292852162e-02,   7.105442355344407e-02, &
07317         7.303069033278667e-02,   7.458287540049919e-02, &
07318         7.570449768455667e-02,   7.637786767208074e-02, &
07319         7.660071191799966e-02/
07320    data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8),wg(9),wg(10)/ &
07321         1.761400713915212e-02,   4.060142980038694e-02, &
07322         6.267204833410906e-02,   8.327674157670475e-02, &
07323         1.019301198172404e-01,   1.181945319615184e-01, &
07324         1.316886384491766e-01,   1.420961093183821e-01, &
07325         1.491729864726037e-01,   1.527533871307259e-01/
07326 !
07327    centr = 5.0e-01*(a+b)
07328    hlgth = 5.0e-01*(b-a)
07329    dhlgth = abs(hlgth)
07330 !
07331 !  Compute 41-point Gauss-Kronrod approximation to the
07332 !  the integral, and estimate the absolute error.
07333 !
07334    resg = 0.0e+00
07335    fc = f(centr)
07336    resk = wgk(21)*fc
07337    resabs = abs(resk)
07338
07339    do j = 1, 10
07340      jtw = j*2
07341      absc = hlgth*xgk(jtw)
07342      fval1 = f(centr-absc)
07343      fval2 = f(centr+absc)
07344      fv1(jtw) = fval1
07345      fv2(jtw) = fval2
07346      fsum = fval1+fval2
07347      resg = resg+wg(j)*fsum
07348      resk = resk+wgk(jtw)*fsum
07349      resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
07350    end do
07351
07352    do j = 1, 10
07353      jtwm1 = j*2-1
07354      absc = hlgth*xgk(jtwm1)
07355      fval1 = f(centr-absc)
07356      fval2 = f(centr+absc)
07357      fv1(jtwm1) = fval1
07358      fv2(jtwm1) = fval2
07359      fsum = fval1+fval2
07360      resk = resk+wgk(jtwm1)*fsum
07361      resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
07362    end do
07363
07364    reskh = resk*5.0e-01
07365    resasc = wgk(21)*abs(fc-reskh)
07366
07367    do j = 1, 20
07368      resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
07369    end do
07370
07371    result = resk*hlgth
07372    resabs = resabs*dhlgth
07373    resasc = resasc*dhlgth
07374    abserr = abs((resk-resg)*hlgth)
07375
07376    if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) &
07377      abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
07378
07379    if ( resabs > tiny( resabs ) /(5.0e+01* epsilon( resabs ) )) then
07380      abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
07381    end if
```

```
07382
07383   return
07384 end subroutine qk41
07385 subroutine qk51 ( f, a, b, result, abserr, resabs, resasc )
07386
07387 !*******************************************************************************80
07388 !
07389 !! QK51 carries out a 51 point Gauss-Kronrod quadrature rule.
07390 !
07391 !  Discussion:
07392 !
07393 !    This routine approximates
07394 !      I = integral ( A <= X <= B ) F(X) dx
07395 !    with an error estimate, and
07396 !      J = integral ( A <= X <= B ) | F(X) | dx
07397 !
07398 !  Author:
07399 !
07400 !    Robert Piessens, Elise de Doncker-Kapenger,
07401 !    Christian Ueberhuber, David Kahaner
07402 !
07403 !  Reference:
07404 !
07405 !    Robert Piessens, Elise de Doncker-Kapenger,
07406 !    Christian Ueberhuber, David Kahaner,
07407 !    QUADPACK, a Subroutine Package for Automatic Integration,
07408 !    Springer Verlag, 1983
07409 !
07410 !  Parameters:
07411 !
07412 !    Input, external real F, the name of the function routine, of the form
07413 !      function f ( x )
07414 !      real f
07415 !      real x
07416 !    which evaluates the integrand function.
07417 !
07418 !    Input, real A, B, the limits of integration.
07419 !
07420 !    Output, real RESULT, the estimated value of the integral.
07421 !                        result is computed by applying the 51-point
07422 !                        Kronrod rule (resk) obtained by optimal addition
07423 !                        of abscissae to the 25-point Gauss rule (resg).
07424 !
07425 !    Output, real ABSERR, an estimate of | I - RESULT |.
07426 !
07427 !    Output, real RESABS, approximation to the integral of the absolute
07428 !    value of F.
07429 !
07430 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
07431 !    over [A,B].
07432 !
07433 !  Local Parameters:
07434 !
07435 !           centr  - mid point of the interval
07436 !           hlgth  - half-length of the interval
07437 !           absc   - abscissa
07438 !           fval*  - function value
07439 !           resg   - result of the 25-point Gauss formula
07440 !           resk   - result of the 51-point Kronrod formula
07441 !           reskh  - approximation to the mean value of f over (a,b),
07442 !                    i.e. to i/(b-a)
07443 !
07444   implicit none
07445
07446   real(kind=params_wp) a
07447   real(kind=params_wp) absc
07448   real(kind=params_wp) abserr
07449   real(kind=params_wp) b
07450   real(kind=params_wp) centr
07451   real(kind=params_wp) dhlgth
07452   real(kind=params_wp), external :: f
07453   real(kind=params_wp) fc
07454   real(kind=params_wp) fsum
07455   real(kind=params_wp) fval1
07456   real(kind=params_wp) fval2
07457   real(kind=params_wp) fv1(25)
07458   real(kind=params_wp) fv2(25)
07459   real(kind=params_wp) hlgth
07460   integer j
07461   integer jtw
07462   integer jtwm1
07463   real(kind=params_wp) resabs
07464   real(kind=params_wp) resasc
07465   real(kind=params_wp) resg
07466   real(kind=params_wp) resk
07467   real(kind=params_wp) reskh
07468   real(kind=params_wp) result
```

```
07469   real(kind=params_wp) wg(13)
07470   real(kind=params_wp) wgk(26)
07471   real(kind=params_wp) xgk(26)
07472 !
07473 !           the abscissae and weights are given for the interval (-1,1).
07474 !           because of symmetry only the positive abscissae and their
07475 !           corresponding weights are given.
07476 !
07477 !           xgk    - abscissae of the 51-point Kronrod rule
07478 !                    xgk(2), xgk(4), ...  abscissae of the 25-point
07479 !                    Gauss rule
07480 !                    xgk(1), xgk(3), ...  abscissae which are optimally
07481 !                    added to the 25-point Gauss rule
07482 !
07483 !           wgk    - weights of the 51-point Kronrod rule
07484 !
07485 !           wg     - weights of the 25-point Gauss rule
07486 !
07487   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
07488     xgk(9),xgk(10),xgk(11),xgk(12),xgk(13),xgk(14)/ &
07489        9.992621049926098e-01,   9.955569697904981e-01, &
07490        9.880357945340772e-01,   9.766639214595175e-01, &
07491        9.616149864258425e-01,   9.429745712289743e-01, &
07492        9.207471152817016e-01,   8.949919978782754e-01, &
07493        8.658470652932756e-01,   8.334426287608340e-01, &
07494        7.978737979985001e-01,   7.592592630373576e-01, &
07495        7.177664068130844e-01,   6.735663684734684e-01/
07496    data xgk(15),xgk(16),xgk(17),xgk(18),xgk(19),xgk(20),xgk(21), &
07497     xgk(22),xgk(23),xgk(24),xgk(25),xgk(26)/ &
07498        6.268100990103174e-01,   5.776629302412230e-01, &
07499        5.263252843347192e-01,   4.730027314457150e-01, &
07500        4.178853821930377e-01,   3.611723058093878e-01, &
07501        3.030895389311078e-01,   2.438668837209884e-01, &
07502        1.837189394210489e-01,   1.228646926107104e-01, &
07503        6.154448300568508e-02,   0.0e+00               /
07504   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
07505     wgk(9),wgk(10),wgk(11),wgk(12),wgk(13),wgk(14)/ &
07506        1.987383892330316e-03,   5.561932135356714e-03, &
07507        9.473973386174152e-03,   1.323622919557167e-02, &
07508        1.684781770912830e-02,   2.043537114588284e-02, &
07509        2.400994560695322e-02,   2.747531758785174e-02, &
07510        3.079230016738749e-02,   3.400213027432934e-02, &
07511        3.711627148341554e-02,   4.008382550403238e-02, &
07512        4.287284502017005e-02,   4.550291304992179e-02/
07513    data wgk(15),wgk(16),wgk(17),wgk(18),wgk(19),wgk(20),wgk(21), &
07514     wgk(22),wgk(23),wgk(24),wgk(25),wgk(26)/ &
07515        4.798253713883671e-02,   5.027767908071567e-02, &
07516        5.236288580640748e-02,   5.425112988854549e-02, &
07517        5.595081122041232e-02,   5.743711636156783e-02, &
07518        5.868968002239421e-02,   5.972034032417406e-02, &
07519        6.053945537604586e-02,   6.112850971705305e-02, &
07520        6.147118987142532e-02,   6.158081806783294e-02/
07521   data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8),wg(9),wg(10), &
07522     wg(11),wg(12),wg(13)/ &
07523        1.139379850102629e-02,   2.635498661503214e-02, &
07524        4.093915670130631e-02,   5.490469597583519e-02, &
07525        6.803833381235692e-02,   8.014070033500102e-02, &
07526        9.102826198296365e-02,   1.005359490670506e-01, &
07527        1.085196244742637e-01,   1.148582591457116e-01, &
07528        1.194557635357848e-01,   1.222424429903100e-01, &
07529        1.231760537267155e-01/
07530 !
07531   centr = 5.0e-01*(a+b)
07532   hlgth = 5.0e-01*(b-a)
07533   dhlgth = abs(hlgth)
07534 !
07535 !  Compute the 51-point Kronrod approximation to the integral,
07536 !  and estimate the absolute error.
07537 !
07538   fc = f(centr)
07539   resg = wg(13)*fc
07540   resk = wgk(26)*fc
07541   resabs = abs(resk)
07542
07543   do j = 1, 12
07544     jtw = j*2
07545     absc = hlgth*xgk(jtw)
07546     fval1 = f(centr-absc)
07547     fval2 = f(centr+absc)
07548     fv1(jtw) = fval1
07549     fv2(jtw) = fval2
07550     fsum = fval1+fval2
07551     resg = resg+wg(j)*fsum
07552     resk = resk+wgk(jtw)*fsum
07553     resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
07554   end do
07555
```

```
07556   do j = 1, 13
07557     jtwm1 = j*2-1
07558     absc = hlgth*xgk(jtwm1)
07559     fval1 = f(centr-absc)
07560     fval2 = f(centr+absc)
07561     fv1(jtwm1) = fval1
07562     fv2(jtwm1) = fval2
07563     fsum = fval1+fval2
07564     resk = resk+wgk(jtwm1)*fsum
07565     resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
07566   end do
07567
07568   reskh = resk*5.0e-01
07569   resasc = wgk(26)*abs(fc-reskh)
07570
07571   do j = 1, 25
07572     resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
07573   end do
07574
07575   result = resk*hlgth
07576   resabs = resabs*dhlgth
07577   resasc = resasc*dhlgth
07578   abserr = abs((resk-resg)*hlgth)
07579
07580   if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00) then
07581     abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
07582   end if
07583
07584   if ( resabs > tiny( resabs ) / (5.0e+01* epsilon( resabs ) ) ) then
07585     abserr = max(( epsilon( resabs ) *5.0e+01)*resabs,abserr)
07586   end if
07587
07588   return
07589 end subroutine qk51
07590 subroutine qk61 ( f, a, b, result, abserr, resabs, resasc )
07591
07592 !*******************************************************************************80
07593 !
07594 !! QK61 carries out a 61 point Gauss-Kronrod quadrature rule.
07595 !
07596 !  Discussion:
07597 !
07598 !    This routine approximates
07599 !      I = integral ( A <= X <= B ) F(X) dx
07600 !    with an error estimate, and
07601 !      J = integral ( A <= X <= B ) | F(X) | dx
07602 !
07603 !  Author:
07604 !
07605 !    Robert Piessens, Elise de Doncker-Kapenger,
07606 !    Christian Ueberhuber, David Kahaner
07607 !
07608 !  Reference:
07609 !
07610 !    Robert Piessens, Elise de Doncker-Kapenger,
07611 !    Christian Ueberhuber, David Kahaner,
07612 !    QUADPACK, a Subroutine Package for Automatic Integration,
07613 !    Springer Verlag, 1983
07614 !
07615 !  Parameters:
07616 !
07617 !    Input, external real F, the name of the function routine, of the form
07618 !      function f ( x )
07619 !      real f
07620 !      real x
07621 !    which evaluates the integrand function.
07622 !
07623 !    Input, real A, B, the limits of integration.
07624 !
07625 !    Output, real RESULT, the estimated value of the integral.
07626 !                    result is computed by applying the 61-point
07627 !                    Kronrod rule (resk) obtained by optimal addition of
07628 !                    abscissae to the 30-point Gauss rule (resg).
07629 !
07630 !    Output, real ABSERR, an estimate of | I - RESULT |.
07631 !
07632 !    Output, real RESABS, approximation to the integral of the absolute
07633 !    value of F.
07634 !
07635 !    Output, real RESASC, approximation to the integral | F-I/(B-A) |
07636 !    over [A,B].
07637 !
07638 !  Local Parameters:
07639 !
07640 !           centr  - mid point of the interval
07641 !           hlgth  - half-length of the interval
07642 !           absc   - abscissa
```

```
07643 !           fval*  - function value
07644 !           resg   - result of the 30-point Gauss rule
07645 !           resk   - result of the 61-point Kronrod rule
07646 !           reskh  - approximation to the mean value of f
07647 !                    over (a,b), i.e. to i/(b-a)
07648 !
07649   implicit none
07650
07651   real(kind=params_wp) a
07652   real(kind=params_wp) absc
07653   real(kind=params_wp) abserr
07654   real(kind=params_wp) b
07655   real(kind=params_wp) centr
07656   real(kind=params_wp) dhlgth
07657   real(kind=params_wp), external :: f
07658   real(kind=params_wp) fc
07659   real(kind=params_wp) fsum
07660   real(kind=params_wp) fval1
07661   real(kind=params_wp) fval2
07662   real(kind=params_wp) fv1(30)
07663   real(kind=params_wp) fv2(30)
07664   real(kind=params_wp) hlgth
07665   integer j
07666   integer jtw
07667   integer jtwm1
07668   real(kind=params_wp) resabs
07669   real(kind=params_wp) resasc
07670   real(kind=params_wp) resg
07671   real(kind=params_wp) resk
07672   real(kind=params_wp) reskh
07673   real(kind=params_wp) result
07674   real(kind=params_wp) wg(15)
07675   real(kind=params_wp) wgk(31)
07676   real(kind=params_wp) xgk(31)
07677 !
07678 !           the abscissae and weights are given for the
07679 !           interval (-1,1). because of symmetry only the positive
07680 !           abscissae and their corresponding weights are given.
07681 !
07682 !           xgk   - abscissae of the 61-point Kronrod rule
07683 !                   xgk(2), xgk(4)  ... abscissae of the 30-point
07684 !                   Gauss rule
07685 !                   xgk(1), xgk(3)  ... optimally added abscissae
07686 !                   to the 30-point Gauss rule
07687 !
07688 !           wgk   - weights of the 61-point Kronrod rule
07689 !
07690 !           wg    - weigths of the 30-point Gauss rule
07691 !
07692   data xgk(1),xgk(2),xgk(3),xgk(4),xgk(5),xgk(6),xgk(7),xgk(8), &
07693      xgk(9),xgk(10)/ &
07694        9.994844100504906e-01,     9.968934840746495e-01, &
07695        9.916309968704046e-01,     9.836681232797472e-01, &
07696        9.731163225011263e-01,     9.600218649683075e-01, &
07697        9.443744447485600e-01,     9.262000474292743e-01, &
07698        9.055733076999078e-01,     8.825605357920527e-01/
07699   data xgk(11),xgk(12),xgk(13),xgk(14),xgk(15),xgk(16),xgk(17), &
07700     xgk(18),xgk(19),xgk(20)/ &
07701        8.572052335460611e-01,     8.295657623827684e-01, &
07702        7.997278358218391e-01,     7.677774321048262e-01, &
07703        7.337900624532268e-01,     6.978504947933158e-01, &
07704        6.600610641266270e-01,     6.205261829892429e-01, &
07705        5.793452358263617e-01,     5.366241481420199e-01/
07706   data xgk(21),xgk(22),xgk(23),xgk(24),xgk(25),xgk(26),xgk(27), &
07707     xgk(28),xgk(29),xgk(30),xgk(31)/ &
07708        4.924804678617786e-01,     4.470337695380892e-01, &
07709        4.004012548303944e-01,     3.527047255308781e-01, &
07710        3.040732022736251e-01,     2.546369261678898e-01, &
07711        2.045251166823099e-01,     1.538699136085835e-01, &
07712        1.028069379667370e-01,     5.147184255531770e-02, &
07713        0.0e+00                   /
07714   data wgk(1),wgk(2),wgk(3),wgk(4),wgk(5),wgk(6),wgk(7),wgk(8), &
07715     wgk(9),wgk(10)/ &
07716        1.389013698677008e-03,     3.890461127099884e-03, &
07717        6.630703915931292e-03,     9.273279659517763e-03, &
07718        1.182301525349634e-02,     1.436972950704580e-02, &
07719        1.692088918905327e-02,     1.941414119394238e-02, &
07720        2.182803582160919e-02,     2.419116207808060e-02/
07721   data wgk(11),wgk(12),wgk(13),wgk(14),wgk(15),wgk(16),wgk(17), &
07722     wgk(18),wgk(19),wgk(20)/ &
07723        2.650995488233310e-02,     2.875404876504129e-02, &
07724        3.090725756238776e-02,     3.298144705748373e-02, &
07725        3.497933802806002e-02,     3.688236465182123e-02, &
07726        3.867894562472759e-02,     4.037453895153596e-02, &
07727        4.196981021516425e-02,     4.345253970135607e-02/
07728   data wgk(21),wgk(22),wgk(23),wgk(24),wgk(25),wgk(26),wgk(27), &
07729     wgk(28),wgk(29),wgk(30),wgk(31)/ &
```

```
07730         4.481480013316266e-02,     4.605923827100699e-02, &
07731         4.718554656929915e-02,     4.818586175708713e-02, &
07732         4.905543455502978e-02,     4.979568342707421e-02, &
07733         5.040592140278235e-02,     5.088179589874961e-02, &
07734         5.122154784925877e-02,     5.142612853745903e-02, &
07735         5.149472942945157e-02/
07736    data wg(1),wg(2),wg(3),wg(4),wg(5),wg(6),wg(7),wg(8)/ &
07737         7.968192496166606e-03,     1.846646831109096e-02, &
07738         2.878470788332337e-02,     3.879919256962705e-02, &
07739         4.840267283059405e-02,     5.749315621761907e-02, &
07740         6.597422988218050e-02,     7.375597473770521e-02/
07741    data wg(9),wg(10),wg(11),wg(12),wg(13),wg(14),wg(15)/ &
07742         8.075589522942022e-02,     8.689978720108298e-02, &
07743         9.212252223778613e-02,     9.636873717464426e-02, &
07744         9.959342058679527e-02,     1.017623897484055e-01, &
07745         1.028526528935588e-01/
07746
07747    centr = 5.0e-01*(b+a)
07748    hlgth = 5.0e-01*(b-a)
07749    dhlgth = abs(hlgth)
07750 !
07751 !  Compute the 61-point Kronrod approximation to the integral,
07752 !  and estimate the absolute error.
07753 !
07754    resg = 0.0e+00
07755    fc = f(centr)
07756    resk = wgk(31)*fc
07757    resabs = abs(resk)
07758
07759    do j = 1, 15
07760      jtw = j*2
07761      absc = hlgth*xgk(jtw)
07762      fval1 = f(centr-absc)
07763      fval2 = f(centr+absc)
07764      fv1(jtw) = fval1
07765      fv2(jtw) = fval2
07766      fsum = fval1+fval2
07767      resg = resg+wg(j)*fsum
07768      resk = resk+wgk(jtw)*fsum
07769      resabs = resabs+wgk(jtw)*(abs(fval1)+abs(fval2))
07770    end do
07771
07772    do j = 1, 15
07773      jtwm1 = j*2-1
07774      absc = hlgth*xgk(jtwm1)
07775      fval1 = f(centr-absc)
07776      fval2 = f(centr+absc)
07777      fv1(jtwm1) = fval1
07778      fv2(jtwm1) = fval2
07779      fsum = fval1+fval2
07780      resk = resk+wgk(jtwm1)*fsum
07781      resabs = resabs+wgk(jtwm1)*(abs(fval1)+abs(fval2))
07782    end do
07783
07784    reskh = resk * 5.0e-01
07785    resasc = wgk(31)*abs(fc-reskh)
07786
07787    do j = 1, 30
07788      resasc = resasc+wgk(j)*(abs(fv1(j)-reskh)+abs(fv2(j)-reskh))
07789    end do
07790
07791    result = resk*hlgth
07792    resabs = resabs*dhlgth
07793    resasc = resasc*dhlgth
07794    abserr = abs((resk-resg)*hlgth)
07795
07796    if ( resasc /= 0.0e+00 .and. abserr /= 0.0e+00) then
07797      abserr = resasc*min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
07798    end if
07799
07800    if ( resabs > tiny( resabs ) / (5.0e+01* epsilon( resabs ) )) then
07801      abserr = max( ( epsilon( resabs ) *5.0e+01)*resabs, abserr )
07802    end if
07803
07804    return
07805 end subroutine qk61
07806 subroutine qmomo ( alfa, beta, ri, rj, rg, rh, integr )
07807
07808 !***********************************************************************80
07809 !
07810 !! QMOMO computes modified Chebyshev moments.
07811 !
07812 !  Discussion:
07813 !
07814 !    This routine computes modified Chebyshev moments.
07815 !    The K-th modified Chebyshev moment is defined as the
07816 !    integral over (-1,1) of W(X)*T(K,X), where T(K,X) is the
```

```
07817 !    Chebyshev polynomial of degree K.
07818 !
07819 !  Author:
07820 !
07821 !    Robert Piessens, Elise de Doncker-Kapenger,
07822 !    Christian Ueberhuber, David Kahaner
07823 !
07824 !  Reference:
07825 !
07826 !    Robert Piessens, Elise de Doncker-Kapenger,
07827 !    Christian Ueberhuber, David Kahaner,
07828 !    QUADPACK, a Subroutine Package for Automatic Integration,
07829 !    Springer Verlag, 1983
07830 !
07831 !  Parameters:
07832 !
07833 !    Input, real ALFA, a parameter in the weight function w(x), ALFA > -1.
07834 !
07835 !    Input, real BETA, a parameter in the weight function w(x), BETA > -1.
07836 !
07837 !           ri     - real
07838 !                    vector of dimension 25
07839 !                    ri(k) is the integral over (-1,1) of
07840 !                    (1+x)**alfa*t(k-1,x), k = 1, ..., 25.
07841 !
07842 !           rj     - real
07843 !                    vector of dimension 25
07844 !                    rj(k) is the integral over (-1,1) of
07845 !                    (1-x)**beta*t(k-1,x), k = 1, ..., 25.
07846 !
07847 !           rg     - real
07848 !                    vector of dimension 25
07849 !                    rg(k) is the integral over (-1,1) of
07850 !                    (1+x)**alfa*log((1+x)/2)*t(k-1,x), k = 1, ...,25.
07851 !
07852 !           rh     - real
07853 !                    vector of dimension 25
07854 !                    rh(k) is the integral over (-1,1) of
07855 !                    (1-x)**beta*log((1-x)/2)*t(k-1,x), k = 1, ..., 25.
07856 !
07857 !           integr - integer
07858 !                    input parameter indicating the modified moments
07859 !                    to be computed
07860 !                    integr = 1 compute ri, rj
07861 !                           = 2 compute ri, rj, rg
07862 !                           = 3 compute ri, rj, rh
07863 !                           = 4 compute ri, rj, rg, rh
07864 !
07865   implicit none
07866
07867   real(kind=params_wp) alfa
07868   real(kind=params_wp) alfp1
07869   real(kind=params_wp) alfp2
07870   real(kind=params_wp) an
07871   real(kind=params_wp) anm1
07872   real(kind=params_wp) beta
07873   real(kind=params_wp) betp1
07874   real(kind=params_wp) betp2
07875   integer i
07876   integer im1
07877   integer integr
07878   real(kind=params_wp) ralf
07879   real(kind=params_wp) rbet
07880   real(kind=params_wp) rg(25)
07881   real(kind=params_wp) rh(25)
07882   real(kind=params_wp) ri(25)
07883   real(kind=params_wp) rj(25)
07884 !
07885   alfp1 = alfa+1.0e+00_params_wp
07886   betp1 = beta+1.0e+00_params_wp
07887   alfp2 = alfa+2.0e+00
07888   betp2 = beta+2.0e+00
07889   ralf = 2.0e+00**alfp1
07890   rbet = 2.0e+00**betp1
07891 !
07892 !  Compute RI, RJ using a forward recurrence relation.
07893 !
07894   ri(1) = ralf/alfp1
07895   rj(1) = rbet/betp1
07896   ri(2) = ri(1)*alfa/alfp2
07897   rj(2) = rj(1)*beta/betp2
07898   an = 2.0e+00
07899   anm1 = 1.0e+00_params_wp
07900
07901   do i = 3, 25
07902     ri(i) = -(ralf+an*(an-alfp2)*ri(i-1))/(anm1*(an+alfp1))
07903     rj(i) = -(rbet+an*(an-betp2)*rj(i-1))/(anm1*(an+betp1))
```

```
07904       anm1 = an
07905       an = an+1.0e+00_params_wp
07906     end do
07907
07908     if ( integr == 1 ) go to 70
07909     if ( integr == 3 ) go to 40
07910 !
07911 !  Compute RG using a forward recurrence relation.
07912 !
07913     rg(1) = -ri(1)/alfp1
07914     rg(2) = -(ralf+ralf)/(alfp2*alfp2)-rg(1)
07915     an = 2.0e+00
07916     anm1 = 1.0e+00_params_wp
07917     im1 = 2
07918
07919     do i = 3, 25
07920       rg(i) = -(an*(an-alfp2)*rg(im1)-an*ri(im1)+anm1*ri(i))/ &
07921       (anm1*(an+alfp1))
07922       anm1 = an
07923       an = an+1.0e+00_params_wp
07924       im1 = i
07925     end do
07926
07927     if ( integr == 2 ) go to 70
07928 !
07929 !  Compute RH using a forward recurrence relation.
07930 !
07931 40 continue
07932
07933     rh(1) = -rj(1) / betp1
07934     rh(2) = -(rbet+rbet)/(betp2*betp2)-rh(1)
07935     an = 2.0e+00
07936     anm1 = 1.0e+00_params_wp
07937     im1 = 2
07938
07939     do i = 3, 25
07940       rh(i) = -(an*(an-betp2)*rh(im1)-an*rj(im1)+ &
07941       anm1*rj(i))/(anm1*(an+betp1))
07942       anm1 = an
07943       an = an+1.0e+00_params_wp
07944       im1 = i
07945     end do
07946
07947     do i = 2, 25, 2
07948       rh(i) = -rh(i)
07949     end do
07950
07951     70 continue
07952
07953     do i = 2, 25, 2
07954       rj(i) = -rj(i)
07955     end do
07956
07957 !  90 continue
07958
07959     return
07960 end subroutine qmomo
07961 subroutine qng ( f, a, b, epsabs, epsrel, result, abserr, neval, ier )
07962
07963 !*******************************************************************************80
07964 !
07965 !! QNG estimates an integral, using non-adaptive integration.
07966 !
07967 !  Discussion:
07968 !
07969 !    The routine calculates an approximation RESULT to a definite integral
07970 !      I = integral of F over (A,B),
07971 !    hopefully satisfying
07972 !      || I - RESULT || <= max ( EPSABS, EPSREL * ||I|| ).
07973 !
07974 !    The routine is a simple non-adaptive automatic integrator, based on
07975 !    a sequence of rules with increasing degree of algebraic
07976 !    precision (Patterson, 1968).
07977 !
07978 !  Author:
07979 !
07980 !    Robert Piessens, Elise de Doncker-Kapenger,
07981 !    Christian Ueberhuber, David Kahaner
07982 !
07983 !  Reference:
07984 !
07985 !    Robert Piessens, Elise de Doncker-Kapenger,
07986 !    Christian Ueberhuber, David Kahaner,
07987 !    QUADPACK, a Subroutine Package for Automatic Integration,
07988 !    Springer Verlag, 1983
07989 !
07990 !  Parameters:
```

```
07991 !
07992 !    Input, external real F, the name of the function routine, of the form
07993 !      function f ( x )
07994 !      real(kind=params_wp) f
07995 !      real(kind=params_wp) x
07996 !    which evaluates the integrand function.
07997 !
07998 !    Input, real A, B, the limits of integration.
07999 !
08000 !    Input, real EPSABS, EPSREL, the absolute and relative accuracy requested.
08001 !
08002 !    Output, real RESULT, the estimated value of the integral.
08003 !    RESULT is obtained by applying the 21-point Gauss-Kronrod rule (RES21)
08004 !    obtained  by optimal addition of abscissae to the 10-point Gauss rule
08005 !    (RES10), or by applying the 43-point rule (RES43) obtained by optimal
08006 !    addition of abscissae to the 21-point Gauss-Kronrod rule, or by
08007 !    applying the 87-point rule (RES87) obtained by optimal addition of
08008 !    abscissae to the 43-point rule.
08009 !
08010 !    Output, real ABSERR, an estimate of || I - RESULT ||.
08011 !
08012 !    Output, integer NEVAL, the number of times the integral was evaluated.
08013 !
08014 !           ier    - ier = 0 normal and reliable termination of the
08015 !                            routine. it is assumed that the requested
08016 !                            accuracy has been achieved.
08017 !                    ier > 0 abnormal termination of the routine. it is
08018 !                            assumed that the requested accuracy has
08019 !                            not been achieved.
08020 !                    ier = 1 the maximum number of steps has been
08021 !                            executed. the integral is probably too
08022 !                            difficult to be calculated by qng.
08023 !                        = 6 the input is invalid, because
08024 !                            epsabs < 0 and epsrel < 0,
08025 !                            result, abserr and neval are set to zero.
08026 !
08027 !  Local Parameters:
08028 !
08029 !           centr  - mid point of the integration interval
08030 !           hlgth  - half-length of the integration interval
08031 !           fcentr - function value at mid point
08032 !           absc   - abscissa
08033 !           fval   - function value
08034 !           savfun - array of function values which have already
08035 !                    been computed
08036 !           res10  - 10-point Gauss result
08037 !           res21  - 21-point Kronrod result
08038 !           res43  - 43-point result
08039 !           res87  - 87-point result
08040 !           resabs - approximation to the integral of abs(f)
08041 !           resasc - approximation to the integral of abs(f-i/(b-a))
08042 !
08043   implicit none
08044
08045   real(kind=params_wp) a
08046   real(kind=params_wp) absc
08047   real(kind=params_wp) abserr
08048   real(kind=params_wp) b
08049   real(kind=params_wp) centr
08050   real(kind=params_wp) dhlgth
08051   real(kind=params_wp) epsabs
08052   real(kind=params_wp) epsrel
08053   real(kind=params_wp), external :: f
08054   real(kind=params_wp) fcentr
08055   real(kind=params_wp) fval
08056   real(kind=params_wp) fval1
08057   real(kind=params_wp) fval2
08058   real(kind=params_wp) fv1(5)
08059   real(kind=params_wp) fv2(5)
08060   real(kind=params_wp) fv3(5)
08061   real(kind=params_wp) fv4(5)
08062   real(kind=params_wp) hlgth
08063   integer ier
08064   integer ipx
08065   integer k
08066   integer l
08067   integer neval
08068   real(kind=params_wp) result
08069   real(kind=params_wp) res10
08070   real(kind=params_wp) res21
08071   real(kind=params_wp) res43
08072   real(kind=params_wp) res87
08073   real(kind=params_wp) resabs
08074   real(kind=params_wp) resasc
08075   real(kind=params_wp) reskh
08076   real(kind=params_wp) savfun(21)
08077   real(kind=params_wp) w10(5)
```

```
08078   real(kind=params_wp) w21a(5)
08079   real(kind=params_wp) w21b(6)
08080   real(kind=params_wp) w43a(10)
08081   real(kind=params_wp) w43b(12)
08082   real(kind=params_wp) w87a(21)
08083   real(kind=params_wp) w87b(23)
08084   real(kind=params_wp) x1(5)
08085   real(kind=params_wp) x2(5)
08086   real(kind=params_wp) x3(11)
08087   real(kind=params_wp) x4(22)
08088 !
08089 !           the following data statements contain the abscissae
08090 !           and weights of the integration rules used.
08091 !
08092 !           x1      abscissae common to the 10-, 21-, 43- and 87-point
08093 !                   rule
08094 !           x2      abscissae common to the 21-, 43- and 87-point rule
08095 !           x3      abscissae common to the 43- and 87-point rule
08096 !           x4      abscissae of the 87-point rule
08097 !           w10     weights of the 10-point formula
08098 !           w21a    weights of the 21-point formula for abscissae x1
08099 !           w21b    weights of the 21-point formula for abscissae x2
08100 !           w43a    weights of the 43-point formula for absissae x1, x3
08101 !           w43b    weights of the 43-point formula for abscissae x3
08102 !           w87a    weights of the 87-point formula for abscissae x1,
08103 !                   x2 and x3
08104 !           w87b    weights of the 87-point formula for abscissae x4
08105 !
08106   data x1(1),x1(2),x1(3),x1(4),x1(5)/ &
08107        9.739065285171717e-01,     8.650633666889845e-01, &
08108        6.794095682990244e-01,     4.333953941292472e-01, &
08109        1.488743389816312e-01/
08110   data x2(1),x2(2),x2(3),x2(4),x2(5)/ &
08111        9.956571630258081e-01,     9.301574913557082e-01, &
08112        7.808177265864169e-01,     5.627571346686047e-01, &
08113        2.943928627014602e-01/
08114   data x3(1),x3(2),x3(3),x3(4),x3(5),x3(6),x3(7),x3(8),x3(9),x3(10), &
08115     x3(11)/ &
08116        9.993333609019321e-01,     9.874334029080889e-01, &
08117        9.548079348142663e-01,     9.001486957483283e-01, &
08118        8.251983149831142e-01,     7.321483889893050e-01, &
08119        6.228479705377252e-01,     4.994795740710565e-01, &
08120        3.649016613465808e-01,     2.222549197766013e-01, &
08121        7.465061746138332e-02/
08122   data x4(1),x4(2),x4(3),x4(4),x4(5),x4(6),x4(7),x4(8),x4(9),x4(10), &
08123     x4(11),x4(12),x4(13),x4(14),x4(15),x4(16),x4(17),x4(18),x4(19), &
08124     x4(20),x4(21),x4(22)/          9.999029772627292e-01, &
08125        9.979898959866787e-01,     9.921754978606872e-01, &
08126        9.813581635727128e-01,     9.650576238583846e-01, &
08127        9.431676131336706e-01,     9.158064146855072e-01, &
08128        8.832216577713165e-01,     8.457107484624157e-01, &
08129        8.035576580352310e-01,     7.570057306854956e-01, &
08130        7.062732097873218e-01,     6.515894665011779e-01, &
08131        5.932233740579611e-01,     5.314936059708319e-01, &
08132        4.667636230420228e-01,     3.994248478592188e-01, &
08133        3.298748771061883e-01,     2.585035592021616e-01, &
08134        1.856953965683467e-01,     1.118422131799075e-01, &
08135        3.735212339461987e-02/
08136   data w10(1),w10(2),w10(3),w10(4),w10(5)/ &
08137        6.667134430868814e-02,     1.494513491505806e-01, &
08138        2.190863625159820e-01,     2.692667193099964e-01, &
08139        2.955242247147529e-01/
08140   data w21a(1),w21a(2),w21a(3),w21a(4),w21a(5)/ &
08141        3.255816230796473e-02,     7.503967481091995e-02, &
08142        1.093871588022976e-01,     1.347092173114733e-01, &
08143        1.477391049013385e-01/
08144   data w21b(1),w21b(2),w21b(3),w21b(4),w21b(5),w21b(6)/ &
08145        1.169463886737187e-02,     5.475589657435200e-02, &
08146        9.312545458369761e-02,     1.234919762620659e-01, &
08147        1.427759385770601e-01,     1.494455540029169e-01/
08148   data w43a(1),w43a(2),w43a(3),w43a(4),w43a(5),w43a(6),w43a(7), &
08149     w43a(8),w43a(9),w43a(10)/      1.629673428966656e-02, &
08150        3.752287612086950e-02,     5.469490205825544e-02, &
08151        6.735541460947809e-02,     7.387019963239395e-02, &
08152        5.768556059769796e-03,     2.737189059324884e-02, &
08153        4.656082691042883e-02,     6.174499520144256e-02, &
08154        7.138726726869340e-02/
08155   data w43b(1),w43b(2),w43b(3),w43b(4),w43b(5),w43b(6),w43b(7), &
08156     w43b(8),w43b(9),w43b(10),w43b(11),w43b(12)/ &
08157        1.844477640212414e-03,     1.079868958589165e-02, &
08158        2.189536386779543e-02,     3.259746397534569e-02, &
08159        4.216313793519181e-02,     5.074193960018458e-02, &
08160        5.837939554261925e-02,     6.474640495144589e-02, &
08161        6.956619791235648e-02,     7.282444147183321e-02, &
08162        7.450775101417512e-02,     7.472214751740301e-02/
08163   data w87a(1),w87a(2),w87a(3),w87a(4),w87a(5),w87a(6),w87a(7), &
08164     w87a(8),w87a(9),w87a(10),w87a(11),w87a(12),w87a(13),w87a(14), &
```

```
08165     w87a(15),w87a(16),w87a(17),w87a(18),w87a(19),w87a(20),w87a(21)/ &
08166        8.148377384149173e-03,     1.876143820156282e-02, &
08167        2.734745105005229e-02,     3.367770731163793e-02, &
08168        3.693509982042791e-02,     2.884872430211531e-03, &
08169        1.368594602271270e-02,     2.328041350288831e-02, &
08170        3.087249761171336e-02,     3.569363363941877e-02, &
08171        9.152833452022414e-04,     5.399280219300471e-03, &
08172        1.094767960111893e-02,     1.629873169678734e-02, &
08173        2.108156888920384e-02,     2.537096976925383e-02, &
08174        2.918969775647575e-02,     3.237320246720279e-02, &
08175        3.478309895036514e-02,     3.641222073135179e-02, &
08176        3.725387550304771e-02/
08177   data w87b(1),w87b(2),w87b(3),w87b(4),w87b(5),w87b(6),w87b(7), &
08178     w87b(8),w87b(9),w87b(10),w87b(11),w87b(12),w87b(13),w87b(14), &
08179     w87b(15),w87b(16),w87b(17),w87b(18),w87b(19),w87b(20),w87b(21), &
08180     w87b(22),w87b(23)/            2.741455637620724e-04, &
08181        1.807124155057943e-03,     4.096869282759165e-03, &
08182        6.758290051847379e-03,     9.549957672201647e-03, &
08183        1.232944765224485e-02,     1.501044734638895e-02, &
08184        1.754896798624319e-02,     1.993803778644089e-02, &
08185        2.219493596101229e-02,     2.433914712600081e-02, &
08186        2.637450541483921e-02,     2.828691078877120e-02, &
08187        3.005258112809270e-02,     3.164675137143993e-02, &
08188        3.305041341997850e-02,     3.425509970422606e-02, &
08189        3.526241266015668e-02,     3.607698962288870e-02, &
08190        3.669860449845609e-02,     3.712054926983258e-02, &
08191        3.733422875193504e-02,     3.736107376267902e-02/
08192 !
08193 !  Test on validity of parameters.
08194 !
08195   result = 0.0e+00
08196   abserr = 0.0e+00
08197   neval = 0
08198
08199   if ( epsabs < 0.0e+00 .and. epsrel < 0.0e+00 ) then
08200     ier = 6
08201     return
08202   end if
08203
08204   hlgth = 5.0e-01 * ( b - a )
08205   dhlgth = abs( hlgth )
08206   centr = 5.0e-01 * ( b + a )
08207   fcentr = f(centr)
08208   neval = 21
08209   ier = 1
08210 !
08211 !  Compute the integral using the 10- and 21-point formula.
08212 !
08213   do l = 1, 3
08214
08215     if ( l == 1 ) then
08216
08217       res10 = 0.0e+00
08218       res21 = w21b(6) * fcentr
08219       resabs = w21b(6) * abs(fcentr)
08220
08221       do k = 1, 5
08222         absc = hlgth * x1(k)
08223         fval1 = f(centr+absc)
08224         fval2 = f(centr-absc)
08225         fval = fval1 + fval2
08226         res10 = res10 + w10(k)*fval
08227         res21 = res21 + w21a(k)*fval
08228         resabs = resabs + w21a(k)*(abs(fval1)+abs(fval2))
08229         savfun(k) = fval
08230         fv1(k) = fval1
08231         fv2(k) = fval2
08232       end do
08233
08234       ipx = 5
08235
08236       do k = 1, 5
08237         ipx = ipx + 1
08238         absc = hlgth * x2(k)
08239         fval1 = f(centr+absc)
08240         fval2 = f(centr-absc)
08241         fval = fval1 + fval2
08242         res21 = res21 + w21b(k) * fval
08243         resabs = resabs + w21b(k) * ( abs( fval1 ) + abs( fval2 ) )
08244         savfun(ipx) = fval
08245         fv3(k) = fval1
08246         fv4(k) = fval2
08247       end do
08248 !
08249 !  Test for convergence.
08250 !
08251       result = res21 * hlgth
```

```
08252       resabs = resabs * dhlgth
08253       reskh = 5.0e-01 * res21
08254       resasc = w21b(6) * abs( fcentr - reskh )
08255
08256       do k = 1, 5
08257         resasc = resasc+w21a(k)*(abs(fv1(k)-reskh)+abs(fv2(k)-reskh)) &
08258                       +w21b(k)*(abs(fv3(k)-reskh)+abs(fv4(k)-reskh))
08259       end do
08260
08261       abserr = abs( ( res21 - res10 ) * hlgth )
08262       resasc = resasc * dhlgth
08263 !
08264 !  Compute the integral using the 43-point formula.
08265 !
08266     else if ( l == 2 ) then
08267
08268       res43 = w43b(12)*fcentr
08269       neval = 43
08270
08271       do k = 1, 10
08272         res43 = res43 + savfun(k) * w43a(k)
08273       end do
08274
08275       do k = 1, 11
08276         ipx = ipx + 1
08277         absc = hlgth * x3(k)
08278         fval = f(absc+centr) + f(centr-absc)
08279         res43 = res43 + fval * w43b(k)
08280         savfun(ipx) = fval
08281       end do
08282 !
08283 !  Test for convergence.
08284 !
08285       result = res43 * hlgth
08286       abserr = abs((res43-res21)*hlgth)
08287 !
08288 !  Compute the integral using the 87-point formula.
08289 !
08290     else if ( l == 3 ) then
08291
08292       res87 = w87b(23) * fcentr
08293       neval = 87
08294
08295       do k = 1, 21
08296         res87 = res87 + savfun(k) * w87a(k)
08297       end do
08298
08299       do k = 1, 22
08300         absc = hlgth * x4(k)
08301         res87 = res87 + w87b(k) * ( f(absc+centr) + f(centr-absc) )
08302       end do
08303
08304       result = res87 * hlgth
08305       abserr = abs( ( res87 - res43) * hlgth )
08306
08307     end if
08308
08309     if ( resasc /= 0.0e+00.and.abserr /= 0.0e+00 ) then
08310       abserr = resasc * min( 1.0e+00_params_wp,(2.0e+02*abserr/resasc)**1.5e+00)
08311     end if
08312
08313     if ( resabs > tiny( resabs ) / ( 5.0e+01 * epsilon( resabs ) ) ) then
08314       abserr = max(( epsilon( resabs ) *5.0e+01) * resabs, abserr )
08315     end if
08316
08317     if ( abserr <= max( epsabs, epsrel*abs(result))) then
08318       ier = 0
08319     end if
08320
08321     if ( ier == 0 ) then
08322       exit
08323     end if
08324
08325   end do
08326
08327   return
08328 end subroutine qng
08329 subroutine qsort ( limit, last, maxerr, ermax, elist, iord, nrmax )
08330
08331 !*****************************************************************************80
08332 !
08333 !! QSORT maintains the order of a list of local error estimates.
08334 !
08335 !  Discussion:
08336 !
08337 !    This routine maintains the descending ordering in the list of the
08338 !    local error estimates resulting from the interval subdivision process.
```

```
08339 !    At each call two error estimates are inserted using the sequential
08340 !    search top-down for the largest error estimate and bottom-up for the
08341 !    smallest error estimate.
08342 !
08343 !  Author:
08344 !
08345 !    Robert Piessens, Elise de Doncker-Kapenger,
08346 !    Christian Ueberhuber, David Kahaner
08347 !
08348 !  Reference:
08349 !
08350 !    Robert Piessens, Elise de Doncker-Kapenger,
08351 !    Christian Ueberhuber, David Kahaner,
08352 !    QUADPACK, a Subroutine Package for Automatic Integration,
08353 !    Springer Verlag, 1983
08354 !
08355 !  Parameters:
08356 !
08357 !    Input, integer LIMIT, the maximum number of error estimates the list can
08358 !    contain.
08359 !
08360 !    Input, integer LAST, the current number of error estimates.
08361 !
08362 !    Input/output, integer MAXERR, the index in the list of the NRMAX-th
08363 !    largest error.
08364 !
08365 !    Output, real ERMAX, the NRMAX-th largest error = ELIST(MAXERR).
08366 !
08367 !    Input, real ELIST(LIMIT), contains the error estimates.
08368 !
08369 !    Input/output, integer IORD(LAST).  The first K elements contain
08370 !    pointers to the error estimates such that ELIST(IORD(1)) through
08371 !    ELIST(IORD(K)) form a decreasing sequence, with
08372 !      K = LAST
08373 !    if
08374 !      LAST <= (LIMIT/2+2),
08375 !    and otherwise
08376 !      K = LIMIT+1-LAST.
08377 !
08378 !    Input/output, integer NRMAX.
08379 !
08380   implicit none
08381
08382   integer last
08383
08384   real(kind=params_wp) elist(last)
08385   real(kind=params_wp) ermax
08386   real(kind=params_wp) errmax
08387   real(kind=params_wp) errmin
08388   integer i
08389   integer ibeg
08390   integer iord(last)
08391   integer isucc
08392   integer j
08393   integer jbnd
08394   integer jupbn
08395   integer k
08396   integer limit
08397   integer maxerr
08398   integer nrmax
08399 !
08400 !  Check whether the list contains more than two error estimates.
08401 !
08402   if ( last <= 2 ) then
08403     iord(1) = 1
08404     iord(2) = 2
08405     go to 90
08406   end if
08407 !
08408 !  This part of the routine is only executed if, due to a
08409 !  difficult integrand, subdivision increased the error
08410 !  estimate. in the normal case the insert procedure should
08411 !  start after the nrmax-th largest error estimate.
08412 !
08413   errmax = elist(maxerr)
08414
08415   do i = 1, nrmax-1
08416
08417     isucc = iord(nrmax-1)
08418
08419     if ( errmax <= elist(isucc) ) then
08420       exit
08421     end if
08422
08423     iord(nrmax) = isucc
08424     nrmax = nrmax-1
08425
```

```
08426   end do
08427 !
08428 !  Compute the number of elements in the list to be maintained
08429 !  in descending order.  This number depends on the number of
08430 !  subdivisions still allowed.
08431 !
08432   jupbn = last
08433
08434   if ( (limit/2+2) < last ) then
08435     jupbn = limit+3-last
08436   end if
08437
08438   errmin = elist(last)
08439 !
08440 !  Insert errmax by traversing the list top-down, starting
08441 !  comparison from the element elist(iord(nrmax+1)).
08442 !
08443   jbnd = jupbn-1
08444   ibeg = nrmax+1
08445
08446   do i = ibeg, jbnd
08447     isucc = iord(i)
08448     if ( elist(isucc) <= errmax ) then
08449       go to 60
08450     end if
08451     iord(i-1) = isucc
08452   end do
08453
08454   iord(jbnd) = maxerr
08455   iord(jupbn) = last
08456   go to 90
08457 !
08458 !  Insert errmin by traversing the list bottom-up.
08459 !
08460 60 continue
08461
08462   iord(i-1) = maxerr
08463   k = jbnd
08464
08465   do j = i, jbnd
08466     isucc = iord(k)
08467     if ( errmin < elist(isucc) ) then
08468       go to 80
08469     end if
08470     iord(k+1) = isucc
08471     k = k-1
08472   end do
08473
08474   iord(i) = last
08475   go to 90
08476
08477 80 continue
08478
08479   iord(k+1) = last
08480 !
08481 !  Set maxerr and ermax.
08482 !
08483 90 continue
08484
08485   maxerr = iord(nrmax)
08486   ermax = elist(maxerr)
08487
08488   return
08489 end subroutine qsort
08490 function qwgtc ( x, c, p2, p3, p4, kp )
08491
08492 !*******************************************************************************80
08493 !
08494 !! QWGTC defines the weight function used by QC25C.
08495 !
08496 !  Discussion:
08497 !
08498 !    The weight function has the form 1 / ( X - C ).
08499 !
08500 !  Author:
08501 !
08502 !    Robert Piessens, Elise de Doncker-Kapenger,
08503 !    Christian Ueberhuber, David Kahaner
08504 !
08505 !  Reference:
08506 !
08507 !    Robert Piessens, Elise de Doncker-Kapenger,
08508 !    Christian Ueberhuber, David Kahaner,
08509 !    QUADPACK, a Subroutine Package for Automatic Integration,
08510 !    Springer Verlag, 1983
08511 !
08512 !  Parameters:
```

```
08513 !
08514 !    Input, real X, the point at which the weight function is evaluated.
08515 !
08516 !    Input, real C, the location of the singularity.
08517 !
08518 !    Input, real P2, P3, P4, parameters that are not used.
08519 !
08520 !    Input, integer KP, a parameter that is not used.
08521 !
08522 !    Output, real QWGTC, the value of the weight function at X.
08523 !
08524   implicit none
08525
08526   real(kind=params_wp) c
08527   integer kp
08528   real(kind=params_wp) p2
08529   real(kind=params_wp) p3
08530   real(kind=params_wp) p4
08531   real(kind=params_wp) qwgtc
08532   real(kind=params_wp) x
08533
08534   qwgtc = 1.0e+00_params_wp / ( x - c )
08535
08536   return
08537 end function qwgtc
08538 function qwgto ( x, omega, p2, p3, p4, integr )
08539
08540 !*******************************************************************************80
08541 !
08542 !! QWGTO defines the weight functions used by QC25O.
08543 !
08544 !  Author:
08545 !
08546 !    Robert Piessens, Elise de Doncker-Kapenger,
08547 !    Christian Ueberhuber, David Kahaner
08548 !
08549 !  Reference:
08550 !
08551 !    Robert Piessens, Elise de Doncker-Kapenger,
08552 !    Christian Ueberhuber, David Kahaner,
08553 !    QUADPACK, a Subroutine Package for Automatic Integration,
08554 !    Springer Verlag, 1983
08555 !
08556 !  Parameters:
08557 !
08558 !    Input, real X, the point at which the weight function is evaluated.
08559 !
08560 !    Input, real OMEGA, the factor multiplying X.
08561 !
08562 !    Input, real P2, P3, P4, parameters that are not used.
08563 !
08564 !    Input, integer INTEGR, specifies which weight function is used:
08565 !    1. W(X) = cos ( OMEGA * X )
08566 !    2, W(X) = sin ( OMEGA * X )
08567 !
08568 !    Output, real QWGTO, the value of the weight function at X.
08569 !
08570   implicit none
08571
08572   integer integr
08573   real(kind=params_wp) omega
08574   real(kind=params_wp) p2
08575   real(kind=params_wp) p3
08576   real(kind=params_wp) p4
08577   real(kind=params_wp) qwgto
08578   real(kind=params_wp) x
08579
08580   if ( integr == 1 ) then
08581     qwgto = cos( omega * x )
08582   else if ( integr == 2 ) then
08583     qwgto = sin( omega * x )
08584   end if
08585
08586   return
08587 end function qwgto
08588 function qwgts ( x, a, b, alfa, beta, integr )
08589
08590 !*******************************************************************************80
08591 !
08592 !! QWGTS defines the weight functions used by QC25S.
08593 !
08594 !  Author:
08595 !
08596 !    Robert Piessens, Elise de Doncker-Kapenger,
08597 !    Christian Ueberhuber, David Kahaner
08598 !
08599 !  Reference:
```

```
08600 !
08601 !    Robert Piessens, Elise de Doncker-Kapenger,
08602 !    Christian Ueberhuber, David Kahaner,
08603 !    QUADPACK, a Subroutine Package for Automatic Integration,
08604 !    Springer Verlag, 1983
08605 !
08606 !  Parameters:
08607 !
08608 !    Input, real X, the point at which the weight function is evaluated.
08609 !
08610 !    Input, real A, B, the endpoints of the integration interval.
08611 !
08612 !    Input, real ALFA, BETA, exponents that occur in the weight function.
08613 !
08614 !    Input, integer INTEGR, specifies which weight function is used:
08615 !    1. W(X) = (X-A)**ALFA * (B-X)**BETA
08616 !    2, W(X) = (X-A)**ALFA * (B-X)**BETA * log (X-A)
08617 !    3, W(X) = (X-A)**ALFA * (B-X)**BETA * log (B-X)
08618 !    4, W(X) = (X-A)**ALFA * (B-X)**BETA * log (X-A) * log(B-X)
08619 !
08620 !    Output, real QWGTS, the value of the weight function at X.
08621 !
08622   implicit none
08623
08624   real(kind=params_wp) a
08625   real(kind=params_wp) alfa
08626   real(kind=params_wp) b
08627   real(kind=params_wp) beta
08628   integer integr
08629   real(kind=params_wp) qwgts
08630   real(kind=params_wp) x
08631
08632   if ( integr == 1 ) then
08633     qwgts = ( x - a )**alfa * ( b - x )**beta
08634   else if ( integr == 2 ) then
08635     qwgts = ( x - a )**alfa * ( b - x )**beta * log( x - a )
08636   else if ( integr == 3 ) then
08637     qwgts = ( x - a )**alfa * ( b - x )**beta * log( b - x )
08638   else if ( integr == 4 ) then
08639     qwgts = ( x - a )**alfa * ( b - x )**beta * log( x - a ) * log( b - x )
08640   end if
08641
08642   return
08643 end function qwgts
08644 subroutine timestamp ( )
08645
08646 !*******************************************************************************80
08647 !
08648 !! TIMESTAMP prints the current YMDHMS date as a time stamp.
08649 !
08650 !  Example:
08651 !
08652 !    May 31 2001   9:45:54.872 AM
08653 !
08654 !  Modified:
08655 !
08656 !    31 May 2001
08657 !
08658 !  Author:
08659 !
08660 !    John Burkardt
08661 !
08662 !  Parameters:
08663 !
08664 !    None
08665 !
08666   implicit none
08667
08668   character ( len = 8 ) ampm
08669   integer d
08670   character ( len = 8 ) date
08671   integer h
08672   integer m
08673   integer mm
08674   character ( len = 9 ), parameter, dimension(12) :: month = (/ &
08675     'January  ', 'February ', 'March    ', 'April    ', &
08676     'May      ', 'June     ', 'July     ', 'August   ', &
08677     'September', 'October  ', 'November ', 'December ' /)
08678   integer n
08679   integer s
08680   character ( len = 10 )  time
08681   integer values(8)
08682   integer y
08683   character ( len = 5 ) zone
08684
08685   call date_and_time ( date, time, zone, values )
08686
```

```
08687   y = values(1)
08688   m = values(2)
08689   d = values(3)
08690   h = values(5)
08691   n = values(6)
08692   s = values(7)
08693   mm = values(8)
08694
08695   if ( h < 12 ) then
08696     ampm = 'AM'
08697   else if ( h == 12 ) then
08698     if ( n == 0 .and. s == 0 ) then
08699       ampm = 'Noon'
08700     else
08701       ampm = 'PM'
08702     end if
08703   else
08704     h = h - 12
08705     if ( h < 12 ) then
08706       ampm = 'PM'
08707     else if ( h == 12 ) then
08708       if ( n == 0 .and. s == 0 ) then
08709         ampm = 'Midnight'
08710       else
08711         ampm = 'AM'
08712       end if
08713     end if
08714   end if
08715
08716   write ( *, '(a,1x,i2,1x,i4,2x,i2,a1,i2.2,a1,i2.2,a1,i3.3,1x,a)' ) &
08717     trim( month(m) ), d, y, h, ':', n, ':', s, '.', mm, trim( ampm )
08718
08719   return
08720 end subroutine timestamp
08721
08722 end module quadpack
```

16.64 src/amns_driver/xfelem.f File Reference

Functions/Subroutines

- character ∗12 function xfelem (IZ0)

16.64.1 Function/Subroutine Documentation

16.64.1.1 xfelem()

```
character*12 function xfelem (
             integer IZ0 )
```

Definition at line 3 of file xfelem.f.

```
00004       IMPLICIT NONE
00005 C-----------------------------------------------------------------------
00006 C
00007 C ************* FORTRAN77 CHARACTER*12 FUNCTION: XFELEM ****************
00008 C
00009 C PURPOSE: TO RETURN THE NAME OF THE ELEMENT WITH NUCLEAR CHARGE IZ0
00010 C          (CHARACTER*12 FUNCTION VERSION OF 'XXELEM')
00011 C
00012 C CALLING PROGRAM: GENERAL USE
00013 C
00014 C FUNCTION:
00015 C
00016 C          (C*12) XFELEM  = FUNCTION NAME -
00017 C                           NAME OF ELEMENT WITH NUCLEAR CHARGE 'IZ0'
00018 C          (I*4)  IZ0     = ELEMENT NUCLEAR CHARGE
00019 C
00020 C          (C*12) NAMES() = NAMES OF FIRST 50 ELEMENTS.
00021 C                           ARRAY DIMENSION => NUCLEAR CHARGE
00022 C
00023 C NOTES:    IF NUCLEAR CHARGE IS OUT OF RANGE, I.E.NOT BETWEEN 1 & 50,
00024 C           THEN THE CHARACTER STRING 'XFELEM' IS RETURNED BLANK.
00025 C
00026 C ROUTINES: NONE
00027 C
00028 C
00029 C AUTHOR:   PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00030 C           K1/0/81
```

```
00031 C           JET EXT. 4569
00032 C
00033 C DATE:     13/02/91
00034 C
00035 C VERSION: 1.2
00036 C UPDATE:   17/09/99  HUGH SUMMERS - INCREASED ELEMENT NUMBER TO 92
00037 C
00038 C-----------------------------------------------------------------------
00039 C-----------------------------------------------------------------------
00040       INTEGER      IZ0
00041 C-----------------------------------------------------------------------
00042       CHARACTER*12 XFELEM , NAMES(92)
00043 C-----------------------------------------------------------------------
00044       DATA names/'HYDROGEN    ','HELIUM      ','LITHIUM     ',
00045      &           'BERYLLIUM   ','BORON       ','CARBON      ',
00046      &           'NITROGEN    ','OXYGEN      ','FLUORINE    ',
00047      &           'NEON        ','SODIUM      ','MAGNESIUM   ',
00048      &           'ALUMINIUM   ','SILICON     ','PHOSPHORUS  ',
00049      &           'SULPHUR     ','CHLORINE    ','ARGON       ',
00050      &           'POTASSIUM   ','CALCIUM     ','SCANDIUM    ',
00051      &           'TITANIUM    ','VANADIUM    ','CHROMIUM    ',
00052      &           'MANGANESE   ','IRON        ','COBALT      ',
00053      &           'NICKEL      ','COPPER      ','ZINC        ',
00054      &           'GALLIUM     ','GERMANIUM   ','ARSENIC     ',
00055      &           'SELENIUM    ','BROMINE     ','KRYPTON     ',
00056      &           'RUBIDIUM    ','STRONTIUM   ','YTTRIUM     ',
00057      &           'ZIRCONIUM   ','NIOBIUM     ','MOLYBDENUM  ',
00058      &           'TECHNETIUM  ','RUTHENIUM   ','RHODIUM     ',
00059      &           'PALLADIUM   ','SILVER      ','CADMIUM     ',
00060      &           'INDIUM      ','TIN         ','ANTIMONY    ',
00061      &           'TELLURIUM   ','IODINE      ','XENON       ',
00062      &           'CESIUM      ','BARIUM      ','LANTHANUM   ',
00063      &           'CERIUM      ','PRAESODYMIUM','NEODYMIUM   ',
00064      &           'PROMETHIUM  ','SAMARIUM    ','EUROPIUM    ',
00065      &           'GADOLINIUM  ','TERBIUM     ','DYSPROSIUM  ',
00066      &           'HOLMIUM     ','ERBIUM      ','THULIUM     ',
00067      &           'YTTERBIUM   ','LUTETIUM    ','HAFNIUM     ',
00068      &           'TANTALUM    ','TUNGSTEN    ','RHENIUM     ',
00069      &           'OSMIUM      ','IRIDIUM     ','PLATINUM    ',
00070      &           'GOLD        ','MERCURY     ','THALLIUM    ',
00071      &           'LEAD        ','BISMUTH     ','POLONIUM    ',
00072      &           'ASTATINE    ','RADON       ','FRANCIUM    ',
00073      &           'RADIUM      ','ACTINIUM    ','THORIUM     ',
00074      &           'PROTACTINIUM','URANIUM     '/
00075 C-----------------------------------------------------------------------
00076         IF ( (iz0.GT.92).OR.(iz0.LT.0) ) THEN
00077           xfelem = ' '
00078         ELSE
00079           xfelem = names(iz0)
00080         ENDIF
00081 C-----------------------------------------------------------------------
00082       RETURN
```

16.65 xfelem.f

```
00001 CX UNIX PORT - SCCS Info : Module @(#)$Header: /home/adascvs/fortran/adaslib/atomic/xfelem.for,v 1.2
       2004/07/06 15:30:02 whitefor Exp $ Date $Date: 2004/07/06 15:30:02 $
00002 CX
00003       FUNCTION xfelem ( IZ0 )
00004       IMPLICIT NONE
00005 C-----------------------------------------------------------------------
00006 C
00007 C ************* FORTRAN77 CHARACTER*12 FUNCTION: XFELEM ****************
00008 C
00009 C PURPOSE: TO RETURN THE NAME OF THE ELEMENT WITH NUCLEAR CHARGE IZ0
00010 C          (CHARACTER*12 FUNCTION VERSION OF 'XXELEM')
00011 C
00012 C CALLING PROGRAM: GENERAL USE
00013 C
00014 C FUNCTION:
00015 C
00016 C          (C*12) XFELEM  = FUNCTION NAME -
00017 C                           NAME OF ELEMENT WITH NUCLEAR CHARGE 'IZ0'
00018 C          (I*4)  IZ0     = ELEMENT NUCLEAR CHARGE
00019 C
00020 C          (C*12) NAMES() = NAMES OF FIRST 50 ELEMENTS.
00021 C                           ARRAY DIMENSION => NUCLEAR CHARGE
00022 C
00023 C NOTES:    IF NUCLEAR CHARGE IS OUT OF RANGE, I.E.NOT BETWEEN 1 & 50,
00024 C           THEN THE CHARACTER STRING 'XFELEM' IS RETURNED BLANK.
00025 C
00026 C ROUTINES: NONE
00027 C
00028 C
00029 C AUTHOR:   PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
```

```
00030 C           K1/0/81
00031 C           JET EXT. 4569
00032 C
00033 C DATE:     13/02/91
00034 C
00035 C VERSION: 1.2
00036 C UPDATE:   17/09/99  HUGH SUMMERS - INCREASED ELEMENT NUMBER TO 92
00037 C
00038 C-----------------------------------------------------------------------
00039 C-----------------------------------------------------------------------
00040       INTEGER      iz0
00041 C-----------------------------------------------------------------------
00042       CHARACTER*12 xfelem , names(92)
00043 C-----------------------------------------------------------------------
00044       DATA names/'HYDROGEN    ','HELIUM      ','LITHIUM     ',
00045      &           'BERYLLIUM   ','BORON       ','CARBON      ',
00046      &           'NITROGEN    ','OXYGEN      ','FLUORINE    ',
00047      &           'NEON        ','SODIUM      ','MAGNESIUM   ',
00048      &           'ALUMINIUM   ','SILICON     ','PHOSPHORUS  ',
00049      &           'SULPHUR     ','CHLORINE    ','ARGON       ',
00050      &           'POTASSIUM   ','CALCIUM     ','SCANDIUM    ',
00051      &           'TITANIUM    ','VANADIUM    ','CHROMIUM    ',
00052      &           'MANGANESE   ','IRON        ','COBALT      ',
00053      &           'NICKEL      ','COPPER      ','ZINC        ',
00054      &           'GALLIUM     ','GERMANIUM   ','ARSENIC     ',
00055      &           'SELENIUM    ','BROMINE     ','KRYPTON     ',
00056      &           'RUBIDIUM    ','STRONTIUM   ','YTTRIUM     ',
00057      &           'ZIRCONIUM   ','NIOBIUM     ','MOLYBDENUM  ',
00058      &           'TECHNETIUM  ','RUTHENIUM   ','RHODIUM     ',
00059      &           'PALLADIUM   ','SILVER      ','CADMIUM     ',
00060      &           'INDIUM      ','TIN         ','ANTIMONY    ',
00061      &           'TELLURIUM   ','IODINE      ','XENON       ',
00062      &           'CESIUM      ','BARIUM      ','LANTHANUM   ',
00063      &           'CERIUM      ','PRAESODYMIUM','NEODYMIUM   ',
00064      &           'PROMETHIUM  ','SAMARIUM    ','EUROPIUM    ',
00065      &           'GADOLINIUM  ','TERBIUM     ','DYSPROSIUM  ',
00066      &           'HOLMIUM     ','ERBIUM      ','THULIUM     ',
00067      &           'YTTERBIUM   ','LUTETIUM    ','HAFNIUM     ',
00068      &           'TANTALUM    ','TUNGSTEN    ','RHENIUM     ',
00069      &           'OSMIUM      ','IRIDIUM     ','PLATINUM    ',
00070      &           'GOLD        ','MERCURY     ','THALLIUM    ',
00071      &           'LEAD        ','BISMUTH     ','POLONIUM    ',
00072      &           'ASTATINE    ','RADON       ','FRANCIUM    ',
00073      &           'RADIUM      ','ACTINIUM    ','THORIUM     ',
00074      &           'PROTACTINIUM','URANIUM     '/
00075 C-----------------------------------------------------------------------
00076         IF ( (iz0.GT.92).OR.(iz0.LT.0) ) THEN
00077            xfelem = ' '
00078         ELSE
00079            xfelem = names(iz0)
00080         ENDIF
00081 C-----------------------------------------------------------------------
00082       RETURN
00083       END
```

16.66 src/amns_driver/xxcase.f File Reference

Functions/Subroutines

- subroutine xxcase (input, output, type)

16.66.1 Function/Subroutine Documentation

16.66.1.1 xxcase()

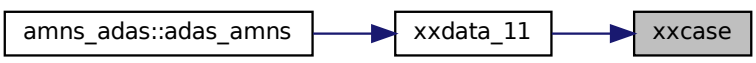
```
subroutine xxcase (
             character*(*) input,
             character*(*) output,
             character*2 type )
```

Definition at line 1 of file xxcase.f.

```
00002
00003       IMPLICIT NONE
00004
00005 C-----------------------------------------------------------------------
00006 C
00007 C  ***************** FORTRAN77 SUBROUTINE: XXCASE *********************
```

```
00008 C
00009 C  PURPOSE: Change a string of arbitrary size into all upper case
00010 C           or all lower case
00011 C
00012 C  CALLING PROGRAM: GENERAL USE.
00013 C
00014 C  INPUT    : (C*(*)) INPUT = Input String
00015 C  INPUT    : (C*2)   TYPE = Type of case to convert to:
00016 C                        'UC' -> Convert to Upper Case
00017 C                        'LC' -> Convert to Lower Case
00018 C                        Anything else -> No conversion
00019 C
00020 C  OUTPUT   : (C*(*)) OUTPUT = Output string in selected case
00021 C
00022 C  ROUTINES : NONE
00023 C
00024 C  AUTHOR   : Allan Whiteford,
00025 C             University of Strathclyde
00026 C
00027 C  VERSION  : 1.1
00028 C  DATE     : 05/09/2001
00029 C  MODIFIED : Allan Whiteford
00030 C             First version.
00031 C
00032 C  VERSION  : 1.2
00033 C  DATE     : 05/05/2005
00034 C  MODIFIED : Martin O'Mullane
00035 C             The routine converted length-1 rather than the whole
00036 C             input string.
00037 C
00038 C-----------------------------------------------------------------------
00039       integer i
00040       integer size
00041 C-----------------------------------------------------------------------
00042       character*(*) input
00043       character*(*) output
00044       character*2 type
00045 C-----------------------------------------------------------------------
00046
00047       size=len(input)
00048
00049       write(output(1:size),'(A)') input(1:size)
00050       i=1
00051
00052       if (type.eq.'UC' .or. type.eq.'uc') then
00053 10       if    ( ichar(input(i:i)) .ge. ichar('a')
00054      &    .and.   ichar(input(i:i)) .le. ichar('z')
00055      &          ) output(i:i)=char(ichar(input(i:i))+ichar('A')
00056      &                                             -ichar('a'))
00057
00058           i=i+1
00059           if (i .le. size) goto 10
00060         endif
00061
00062       if (type.eq.'LC' .or. type.eq.'lc') then
00063 20       if    ( ichar(input(i:i)) .ge. ichar('A')
00064      &    .and.   ichar(input(i:i)) .le. ichar('Z')
00065      &          ) output(i:i)=char(ichar(input(i:i))+ichar('a')
00066      &                                             -ichar('A'))
00067
00068           i=i+1
00069           if (i .le. size) goto 20
00070         endif
00071
```

Here is the caller graph for this function:



16.67 xxcase.f

```
00001       subroutine xxcase(input,output,type)
00002
```

```
00003       IMPLICIT NONE
00004
00005 C-----------------------------------------------------------------------
00006 C
00007 C  ****************** FORTRAN77 SUBROUTINE: XXCASE *********************
00008 C
00009 C  PURPOSE: Change a string of arbitrary size into all upper case
00010 C           or all lower case
00011 C
00012 C  CALLING PROGRAM: GENERAL USE.
00013 C
00014 C  INPUT    : (C*(*)) INPUT = Input String
00015 C  INPUT    : (C*2)   TYPE = Type of case to convert to:
00016 C                       'UC' -> Convert to Upper Case
00017 C                       'LC' -> Convert to Lower Case
00018 C                       Anything else -> No conversion
00019 C
00020 C  OUTPUT   : (C*(*)) OUTPUT = Output string in selected case
00021 C
00022 C  ROUTINES : NONE
00023 C
00024 C  AUTHOR   : Allan Whiteford,
00025 C             University of Strathclyde
00026 C
00027 C  VERSION  : 1.1
00028 C  DATE     : 05/09/2001
00029 C  MODIFIED : Allan Whiteford
00030 C             First version.
00031 C
00032 C  VERSION  : 1.2
00033 C  DATE     : 05/05/2005
00034 C  MODIFIED : Martin O'Mullane
00035 C             The routine converted length-1 rather than the whole
00036 C             input string.
00037 C
00038 C-----------------------------------------------------------------------
00039       integer i
00040       integer size
00041 C-----------------------------------------------------------------------
00042       character*(*) input
00043       character*(*) output
00044       character*2 type
00045 C-----------------------------------------------------------------------
00046
00047       size=len(input)
00048
00049       write(output(1:size),'(A)') input(1:size)
00050       i=1
00051
00052       if (type.eq.'UC' .or. type.eq.'uc') then
00053 10       if    ( ichar(input(i:i)) .ge. ichar('a')
00054      &    .and.   ichar(input(i:i)) .le. ichar('z')
00055      &          ) output(i:i)=char(ichar(input(i:i))+ichar('A')
00056      &                                             -ichar('a'))
00057
00058          i=i+1
00059          if (i .le. size) goto 10
00060       endif
00061
00062       if (type.eq.'LC' .or. type.eq.'lc') then
00063 20       if    ( ichar(input(i:i)) .ge. ichar('A')
00064      &    .and.   ichar(input(i:i)) .le. ichar('Z')
00065      &          ) output(i:i)=char(ichar(input(i:i))+ichar('a')
00066      &                                             -ichar('A'))
00067
00068          i=i+1
00069          if (i .le. size) goto 20
00070       endif
00071
00072       end
```

16.68 src/amns_driver/xxdata_11.f File Reference

Functions/Subroutines

- subroutine xxdata_11 (iunit, iclass, isdimd, iddimd, itdimd, ndptnl, ndptn, ndptnc, ndcnct, iz0, is1min, is1max, nptnl, nptn, nptnc, iptnla, iptna, iptnca, ncnct, icnctv, iblmx, ismax, dnr_ele, dnr_ams, isppr, ispbr, isstgr, idmax, itmax, ddens, dtev, drcof, lres, lstan, lptn)

16.68.1 Function/Subroutine Documentation

16.68.1.1 xxdata_11()

```
subroutine xxdata_11 (
             integer iunit,
             integer iclass,
             integer isdimd,
             integer iddimd,
             integer itdimd,
             integer ndptnl,
             integer ndptn,
             integer ndptnc,
             integer ndcnct,
             integer iz0,
             integer is1min,
             integer is1max,
             integer nptnl,
             integer, dimension(ndptnl) nptn,
             integer, dimension(ndptnl,ndptn) nptnc,
             integer, dimension(ndptnl) iptnla,
             integer, dimension(ndptnl,ndptn) iptna,
             integer, dimension(ndptnl,ndptn,ndptnc) iptnca,
             integer ncnct,
             integer, dimension(ndcnct) icnctv,
             integer iblmx,
             integer ismax,
             character dnr_ele,
             real*8 dnr_ams,
             integer, dimension(isdimd) isppr,
             integer, dimension(isdimd) ispbr,
             integer, dimension(isdimd) isstgr,
             integer idmax,
             integer itmax,
             real*8, dimension(iddimd) ddens,
             real*8, dimension(itdimd) dtev,
             real*8, dimension(isdimd,itdimd,iddimd) drcof,
             logical lres,
             logical lstan,
             logical lptn )
```

Definition at line 1 of file xxdata_11.f.

```
00014       implicit none
00015 c-----------------------------------------------------------------------
00016 c
00017 c  ****************** fortran77 subroutine: xxdata_11 ******************
00018 c
00019 c  purpose: to read a complete  adf11 file, check its class and
00020 c           determine its standard, resolved and partition organisation.
00021 c
00022 c  calling program: various
00023 c
00024 c  notes:    (1) A 'standard' adf11 file contains gcr data between one
00025 c                whole ionisation stage and another whole ionisation
00026 c                stage.
00027 c                A 'resolved' (or partial) adf11 file contains gcr data
00028 c                between a set of metastables of one ionisation stage
00029 c                and a set of metastables of another ionisation stage.
00030 c                A resolved file is distinguished from a standard file
00031 c                by the presence of a 'connection vector' in the adf11
00032 c                data file header lines.
00033 c                The connection vector specifies the number of meta-
00034 c                stables in each ionisation stage which are coupled
00035 c                together by gcr data.
00036 c            (2) A 'partitioned' adf11 file contains gcr data between
00037 c                clumps of ionisation stages or metastables or comb-
```

```
00038 c                inations of the two called 'partitions'.
00039 c                A 'partition level' is a specification of the
00040 c                partitions which span all the ionisation stages (and
00041 c                metastables) of an element.  Successive partition
00042 c                levels give a heirarchy corresponding to larger
00043 c                partitions and greater clumping.
00044 c                A 'superstage' is a set of partitions which are close-
00045 c                coupled.
00046 c                There are thus equivalences :
00047 c                       ionisation stage   -   superstage
00048 c                       metastable         -   partition
00049 c                       ion charge         -   superstage index
00050 c                A partitioned adf11 file may be standard (with each
00051 c                superstage comprising only one partition) or resolved.
00052 c                A partitioned file is distinguished by the presence of
00053 c                'partition specification block' in the adf11 data
00054 c                file header lines.
00055 c             (3) When  a partition specification block is present, it
00056 c                should be ordered from the highest partition level
00057 c                index to lowest partition level index.  Thus the first
00058 c                partition in the partition block has the least number
00059 c                of partitions and the last has the greatest number.
00060 c             (4) Twelev classes of adf11 data file may be read by the
00061 c                subroutine as follow:
00062 c
00063 c                class index    type      GCR data content
00064 c                -----------    ----      ----------------
00065 c                    1          acd       recombination coeffts
00066 c                    2          scd       ionisation coeffts
00067 c                    3          ccd       CX recombination coeffts
00068 c                    4          prb       recomb/brems power coeffts
00069 c                    5          prc       CX power coeffts
00070 c                    6          qcd       base meta. coupl. coeffts
00071 c                    7          xcd       parent meta. coupl. coeffts
00072 c                    8          plt       low level line power coeffts
00073 c                    9          pls       represent. line power coefft
00074 c                   10          zcd       effective charge
00075 c                   11          ycd       effective squared charge
00076 c                   12          ecd       effective ionisation potential
00077 c
00078 c             (5) A resolved adf11 file, with a connection vector, has a set
00079 c                of names and pointers at precise positions in the data file
00080 c                which are recognised.
00081 c                The names are different for partitioned and unpartitioned
00082 c                data files as follow:
00083 c
00084 c                       file      unpartitioned         partitioned
00085 c                       class        names                 names
00086 c
00087 c                       (all)         z1                   s1
00088 c
00089 c                                  (indices 1 and 2)     (indices 1 and 2)
00090 c                       ----       ----     ----         ----     ----
00091 c                       acd        iprt     igrd         ispp     ispb
00092 c                       scd        iprt     igrd         ispp     ispb
00093 c                       ccd        iprt     igrd         ispp     ispb
00094 c                       prb        iprt                  ispp
00095 c                       prc        iprt                  ispp
00096 c                       qcd        igrd     jgrd         ispb     jspb
00097 c                       xcd        iprt     jprt         ispp     jspp
00098 c                       plt        igrd                  ispb
00099 c                       pls        igrd                  ispb
00100 c                       zcd        igrd                  ispb
00101 c                       ycd        igrd                  ispb
00102 c                       ecd        igrd                  ispb
00103 c
00104 c              (6) In partitioned nomenclature: s=superstage; p=partition;
00105 c                 b=base (current superstage), p=parent (next up super-
00106 c                 stage), c=child (next down superstage). Thus arrays
00107 c                 'iprtr' and 'igrd' in old notation are now substituted
00108 c                 by 'isppr' and 'ispbr' respectively internally and in
00109 c                 external naming.
00110 c
00111 c
00112 c  subroutine:
00113 c
00114 c  input : (i*4)  iunit     = unit to which input file is allocated
00115 c  input : (i*4)  iclass    = class of data (1 - 12 ):
00116 c                               1-acd, 2-scd, 3-ccd, 4-prb, 5-prc
00117 c                               6-qcd, 7-xcd, 8-plt, 9-pls,10-zcd
00118 c                              11-ycd,12-ecd
00119 c
00120 c  input : (i*4)  isdimd    = maximum number of (sstage, parent, base)
00121 c                             blocks in isonuclear master files
00122 c  input : (i*4)  iddimd    = maximum number of dens values in
00123 c                             isonuclear master files
00124 c  input : (i*4)  itdimd    = maximum number of temp values in
```

```
00125 c                               isonuclear master files
00126 c  input : (i*4)  ndptnl    = maximum level of partitions
00127 c  input : (i*4)  ndptn     = maximum no. of partitions in one level
00128 c  input : (i*4)  ndptnc    = maximum no. of components in a partition
00129 c  input : (i*4)  ndcnct    = maximum number of elements in connection
00130 c                               vector
00131 c
00132 c  output: (i*4)  iz0       = nuclear charge
00133 c  output: (i*4)  is1min    = minimum ion charge + 1
00134 c                               (generalised to connection vector index)
00135 c  output: (i*4)  is1max    = maximum ion charge + 1
00136 c                               (note excludes the bare nucleus)
00137 c                               (generalised to connection vector index
00138 c                                and excludes last one which always remains
00139 c                                the bare nucleus)
00140 c  output: (i*4)  nptnl     = number of partition levels in block
00141 c  output: (i*4)  nptn()    = number of partitions in partition level
00142 c                               1st dim: partition level
00143 c  output: (i*4)  nptnc(,)  = number of components in partition
00144 c                               1st dim: partition level
00145 c                               2nd dim: member partition in partition level
00146 c  output: (i*4)  iptnla()  = partition level label (0=resolved root,1=
00147 c                                                    unresolved root)
00148 c                               1st dim: partition level index
00149 c  output: (i*4)  iptna(,)  = partition member label (labelling starts at 0)
00150 c                               1st dim: partition level index
00151 c                               2nd dim: member partition index in partition
00152 c                               level
00153 c  output: (i*4)  iptnca(„)= component label (labelling starts at 0)
00154 c                               1st dim: partition level index
00155 c                               2nd dim: member partition index in partition
00156 c                               level
00157 c                               3rd dim: component index of member partition
00158 c  output: (i*4)  ncnct     = number of elements in connection vector
00159 c  output: (i*4)  icnctv()  = connection vector of number of partitions
00160 c                               of each superstage in resolved case
00161 c                               including the bare nucleus
00162 c                               1st dim: connection vector index
00163 c
00164 c  output: (i*4)  iblmx     = number of (sstage, parent, base)
00165 c                               blocks in isonuclear master file
00166 c  output: (i*4)  ismax     = number of charge states
00167 c                               in isonuclear master file
00168 c                               (generalises to number of elements in
00169 c                                connection vector)
00170 c  output: (c*12) dnr_ele   = CX donor element name for iclass = 3 or 5
00171 c                               (blank if unset)
00172 c  output: (r*8)  dnr_ams   = CX donor element mass for iclass = 3 or 5
00173 c                               (0.0d0 if unset)
00174 c  output: (i*4)  isppr()   = 1st (parent) index for each partition block
00175 c                               1st dim: index of (sstage, parent, base)
00176 c                                        block in isonuclear master file
00177 c  output: (i*4)  ispbr()   = 2nd (base) index for each partition block
00178 c                               1st dim: index of (sstage, parent, base)
00179 c                                        block in isonuclear master file
00180 c  output: (i*4)  isstgr()  = s1 for each resolved data block
00181 c                               (generalises to connection vector index)
00182 c                               1st dim: index of (sstage, parent, base)
00183 c                                        block in isonuclear master file
00184 c
00185 c  output: (i*4)  idmax     = number of dens values in
00186 c                               isonuclear master files
00187 c  output: (i*4)  itmax     = number of temp values in
00188 c                               isonuclear master files
00189 c  output: (r*8)  ddens()   = log10(electron density(cm-3)) from adf11
00190 c  output: (r*8)  dtev()    = log10(electron temperature (eV) from adf11
00191 c  output: (r*8)  drcof(„) = if(iclass <=9):
00192 c                                  log10(coll.-rad. coefft.) from
00193 c                                  isonuclear master file
00194 c                             if(iclass >=10):
00195 c                                coll.-rad. coefft. from
00196 c                                isonuclear master file
00197 c                             1st dim: index of (sstage, parent, base)
00198 c                                      block in isonuclear master file
00199 c                             2nd dim: electron temperature index
00200 c                             3rd dim: electron density index
00201 c
00202 c  output: (l*4)  lres      = .true.  => partial file
00203 c                           = .false. => not partial file
00204 c  output: (l*4)  lstan     = .true.  => standard file
00205 c                           = .false. => not standard file
00206 c  output: (l*4)  lptn      = .true.  => partition block present
00207 c                           = .false. => partition block not present
00208 c
00209 c  routines:
00210 c          routine    source    brief description
00211 c          ------------------------------------------------------------
```

```
00212 c          i4unit     adas      fetch unit number for output of messages
00213 c          i4fctn     adas      convert string to integer form
00214 c          xfelem     adas      return element name given nuclear charge
00215 c          xxword     adas      extract position of number in buffer
00216 c          xxslen     adas      find string less front and tail blanks
00217 c          xxcase     adas      convert a string to upper or lower case
00218 c          xxrptn     adas      analyse an adf11 file partition block
00219 c
00220 c author:  h. p. summers, university of strathclyde
00221 c          ja7.08
00222 c          tel. 0141-548-4196
00223 c
00224 c date:    04/10/06
00225 c
00226 c version: 1.1                          date: 04/10/2006
00227 c modified: hugh summers
00228 c               - first edition.
00229 c
00230 c version: 1.2                          date: 21/01/2007
00231 c modified: Allan Whiteford
00232 c               - Commented out warning about lack of iclass,
00233 c                 all of the present ADAS files do not contain
00234 c                 this information
00235 c                 (first commit to CVS)
00236 c
00237 c version: 1.3                          date: 08/03/2007
00238 c modified: Hugh Summers
00239 c               - adjustments for revised ecd formats.
00240 c                 charge exchange donor/donor mass checks and
00241 c                 dnr_ele, dnr_ams added to parameter return.
00242 c
00243 c VERSION : 1.4
00244 c DATE    : 28-09-2009
00245 c MODIFIED: Martin O'Mullane
00246 c               - Do not write out warning about missing class
00247 c                 name in file (too many complaints about it).
00248 c
00249 c VERSION : 1.5
00250 c DATE    : 01-02-2010
00251 c MODIFIED: Martin O'Mullane
00252 c               - Incorrect warning message printed for a missing
00253 c                 or inconsistent element name in the adf11 file.
00254 c
00255 c-----------------------------------------------------------------------
00256       integer   nddash    , idword      , ndstack  , ndonors
00257 c-----------------------------------------------------------------------
00258       character csrch1*4   , csrch2*4     , csrch3*4
00259       character csrch4*4   , csrch5*4
00260       character cdash*8    , cpart*3
00261 c-----------------------------------------------------------------------
00262       parameter( nddash = 6  , idword = 256 , ndstack = 40 )
00263       parameter( csrch1 = 'iprt' , csrch2 = 'igrd'  ,csrch3 = '---/')
00264       parameter( csrch4 = 'ispp' , csrch5 = 'ispb')
00265       parameter( cdash = '--------' , cpart = '//#' )
00266       parameter( ndonors = 4 )
00267 c-----------------------------------------------------------------------
00268       integer   iunit     , i4unit   , i4fctn
00269       integer   iz0       , is1min   , is1max
00270       integer   iddimd    , itdimd   , isdimd
00271       integer   ndptnl    , ndptn    , ndptnc   , ndcnct
00272       integer   iblmx     , ismax    , itmax    , idmax
00273       integer   ischk
00274       integer   iclass    , iclass_file
00275       integer   i         , ic       , it         , id
00276       integer   icptn     , iabt     , iabt1      , iabt2    ,
00277      &          j         , ndash_line
00278       integer   nptnl     , ncnct    , ncptn_stack
00279       integer   nfirst    , iwords   , nwords   , ifirst     , ilast
00280 c-----------------------------------------------------------------------
00281       real*8    dnr_ams   , dmass
00282 c-----------------------------------------------------------------------
00283       character cstrg*80  , cstrg1*80, cterm*80   ,  chindi*4
00284       character xfelem*12 , str12*12 , dnr_ele*12
00285 c-----------------------------------------------------------------------
00286       logical   lres      , lstan    , lptn      , lresol
00287       logical   lptn_old  , lwarn    , ldonor    , ldmass
00288 c-----------------------------------------------------------------------
00289       integer   idash_linea(nddash)
00290       integer   nptn(ndptnl)          , nptnc(ndptnl,ndptn)
00291       integer   iptnla(ndptnl)        , iptna(ndptnl,ndptn)
00292       integer   iptnca(ndptnl,ndptn,ndptnc)
00293       integer   icnctv(ndcnct)
00294       integer   isstgr(isdimd)
00295       integer   ifirsta(idword)       , ilasta(idword)
00296       integer   ispbr(isdimd)         , isppr(isdimd)
00297 c-----------------------------------------------------------------------
00298       real*8    ddens(iddimd)         , dtev(itdimd)
```

```
00299       real*8    drcof(isdimd,itdimd,iddimd)
00300 c-----------------------------------------------------------------------
00301       character cclass(12)*4          , cpatrn(12)*4
00302       character cptrn1(12)*4          , cptrn2(12)*4
00303       character cptn_stack(ndstack)*80
00304       character cdonors(ndonors)*12
00305 c-----------------------------------------------------------------------
00306       data      cterm /'-----------------------------------------------
00307      &--------------------------------'/
00308       data      cclass/'/ACD','/SCD','/CCD','/PRB','/PRC',
00309      &                 '/QCD','/XCD','/PLT','/PLS','/ZCD',
00310      &                 '/YCD','/ECD'/
00311       data      cptrn1/'iprt','iprt','iprt','iprt','iprt',
00312      &                 'igrd','iprt','igrd','igrd','igrd',
00313      &                 'igrd','igrd'/
00314       data      cptrn2/'ispp','ispp','ispp','ispp','ispp',
00315      &                 'ispb','ispp','ispb','ispb','ispb',
00316      &                 'ispb','ispb'/
00317       data      cdonors/'HYDROGEN    ','DEUTERIUM   ','TRITIUM     ',
00318      &                  'HELIUM      '/
00319 c-----------------------------------------------------------------------
00320 c
00321 c-----------------------------------------------------------------------
00322 c  search for 'prnt' and count number of dash delimiters
00323 c-----------------------------------------------------------------------
00324
00325       ic = 0
00326       icptn = 0
00327       ndash_line = 0
00328
00329       lstan  = .false.
00330       lres = .false.
00331       lptn   = .false.
00332       nptnl  = 0
00333
00334    10 read(iunit,'(1a80)') cstrg
00335       ic = ic + 1
00336
00337
00338       call xxcase(cstrg,cstrgl,'lc')
00339
00340       if((index(cstrgl,csrch1).le.0) .and.
00341      &    (index(cstrgl,csrch2).le.0) .and.
00342      &    (index(cstrgl,csrch4).le.0) .and.
00343      &    (index(cstrgl,csrch5).le.0) .and.
00344      &    (index(cstrgl,csrch3).le.0)) then
00345
00346           if(index(cstrgl,cdash).gt.0) then
00347               ndash_line = ndash_line+1
00348               idash_linea(ndash_line)=ic
00349           endif
00350           if((cstrgl(1:3).eq.cpart).and.(.not.lptn)) then
00351               lptn = .true.
00352               icptn = ic
00353           endif
00354           go to 10
00355       else
00356           if((iclass.eq.1).or.(iclass.eq.3).or.
00357      &       (iclass.eq.4).or.(iclass.eq.5).or.
00358      &       (iclass.eq.6).or.(iclass.eq.7)) then
00359               if(ndash_line.eq.1) then
00360                   lstan = .true.
00361               elseif ((ndash_line.eq.2).and.lptn)then
00362                   lstan = .true.
00363               elseif ((ndash_line.eq.2).and.(.not.lptn))then
00364                   lres = .true.
00365               elseif ((ndash_line.eq.3).and.lptn)then
00366                   lres = .true.
00367               elseif ((ndash_line.eq.3).and.(.not.lptn))then
00368                   write(i4unit(-1),2002)'header lines faulty'
00369                   write(i4unit(-1),2003)
00370                   stop
00371               elseif (ndash_line.gt.3) then
00372                   write(i4unit(-1),2002)'header lines faulty'
00373                   write(i4unit(-1),2003)
00374                   stop
00375               endif
00376           elseif((iclass.eq.2).or.(iclass.eq.8).or.
00377      &           (iclass.eq.9).or.(iclass.eq.10).or.
00378      &           (iclass.eq.11).or.(iclass.eq.12)) then
00379               if(ndash_line.eq.1) then
00380                   lstan = .true.
00381               elseif ((ndash_line.eq.2).and.lptn)then
00382                   lstan = .true.
00383               elseif ((ndash_line.eq.2).and.(.not.lptn))then
00384                   lres = .true.
00385               elseif ((ndash_line.eq.3).and.lptn)then
```

```
00386                     lres = .true.
00387                   elseif ((ndash_line.eq.3).and.(.not.lptn))then
00388                     write(i4unit(-1),2002)'header lines faulty'
00389                     write(i4unit(-1),2003)
00390                     stop
00391                   elseif (ndash_line.gt.3) then
00392                     write(i4unit(-1),2002)'header lines faulty'
00393                     write(i4unit(-1),2003)
00394                     stop
00395                   endif
00396               endif
00397         endif
00398 c-----------------------------------------------------------------------
00399 c  assign look-up names according to partitioning & set warning logical
00400 c-----------------------------------------------------------------------
00401
00402         do i=1,12
00403           if(lptn) then
00404               cpatrn(i)=cptrn2(i)
00405           else
00406               cpatrn(i)=cptrn1(i)
00407           endif
00408         enddo
00409
00410         lwarn = .false.
00411
00412 c-----------------------------------------------------------------------
00413 c  rewind, separate and analyse header sections according to type
00414 c-----------------------------------------------------------------------
00415
00416         rewind(iunit)
00417
00418         ic =0
00419
00420 c-----------------------------------------------------------------------
00421 c  first line
00422 c-----------------------------------------------------------------------
00423         read(iunit,'(a80)')cstrg
00424         call xxcase(cstrg,cstrgl,'uc')
00425         ic=ic+1
00426
00427         iabt = 0
00428         iz0 = i4fctn(cstrg(1:5),iabt)
00429         if(iabt.gt.0) then
00430             write(i4unit(-1),2002)'faulty nuclear charge'
00431             write(i4unit(-1),2003)
00432             stop
00433         endif
00434         str12=xfelem(iz0)
00435         call xxslen(str12,ifirst,ilast)
00436         if(index(cstrgl,str12(ifirst:ilast)).le.0) then
00437               write(i4unit(-1),2000)'inconsistent or missing element',
00438      &                              ' name for iz0 = ',iz0
00439               write(i4unit(-1),2001)
00440         endif
00441
00442         iabt = 0
00443         idmax = i4fctn(cstrg(6:10),iabt)
00444         if(iabt.gt.0.or.idmax.le.0.or.idmax.gt.iddimd) then
00445               write(i4unit(-1),2002)'invalid number of densities',
00446      &                              idmax,' = 0 or > ',iddimd
00447               write(i4unit(-1),2003)
00448             stop
00449         endif
00450         iabt = 0
00451         itmax = i4fctn(cstrg(11:15),iabt)
00452         if(iabt.gt.0.or.itmax.le.0.or.itmax.gt.itdimd) then
00453             write(i4unit(-1),2002)'invalid number of ','temperatures',
00454      &                              itmax,' = 0 or > ',itdimd
00455             write(i4unit(-1),2003)
00456             stop
00457         endif
00458         iabt1 = 0
00459         is1min = i4fctn(cstrg(16:20),iabt1)
00460         iabt2 = 0
00461         is1max = i4fctn(cstrg(21:25),iabt2)
00462         iabt = iabt1 + iabt2
00463
00464         iclass_file = 0
00465         do i=1,12
00466           if(index(cstrgl,cclass(i)).gt.0) then
00467             iclass_file=i
00468           endif
00469         enddo
00470         if(iclass_file.le.0) then
00471 C             write(i4unit(-1),2000)'no class name given in file'
00472 C             write(i4unit(-1),2001)
```

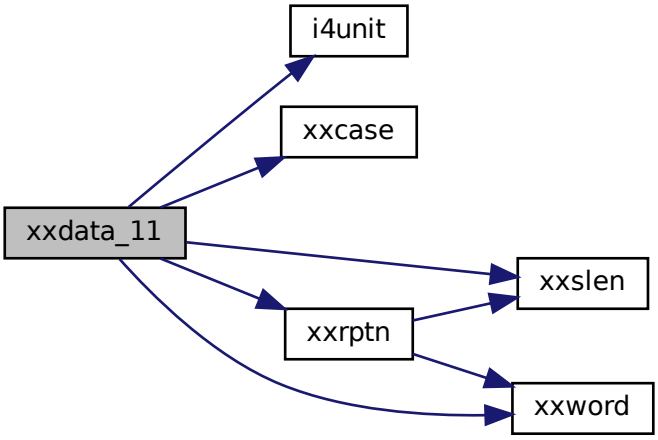
```
00473       elseif(iclass_file.ne.iclass) then
00474            write(i4unit(-1),2004)'file class name ',
00475      &                            cclass(iclass_file),
00476      &                           ' does not match iclass = ',iclass
00477            write(i4unit(-1),2003)
00478            stop
00479       endif
00480
00481 c-----------------------------------------------------------------------
00482 c  check if CX donor is explicitly named in dataset top line
00483 c-----------------------------------------------------------------------
00484       ldonor  = .false.
00485       ldmass  = .false.
00486       dnr_ele = ' '
00487       dnr_ams = 0.0d0
00488       if((iclass_file.eq.3).or.(iclass_file.eq.5)) then
00489           do i=1,ndonors
00490             call xxslen(cdonors(i),ifirst,ilast)
00491             if(index(cstrgl,':'//cdonors(i)(ifirst:ilast)).gt.0) then
00492                 ldonor=.true.
00493                 dnr_ele=cdonors(i)
00494             endif
00495           enddo
00496       endif
00497
00498 c-----------------------------------------------------------------------
00499 c  read connection vector if partial file
00500 c-----------------------------------------------------------------------
00501
00502       if(lres) then
00503           do i=2,idash_linea(1)
00504             read(iunit,'(1a80)')cstrg
00505           enddo
00506
00507           ncnct = 0
00508
00509           do i=idash_linea(1)+1,idash_linea(2)-1
00510             cstrg=' '
00511             read(iunit,'(1a80)')cstrg
00512             nfirst = 1
00513             iwords = idword
00514             call xxword( cstrg , ' '    , nfirst ,
00515      &                   iwords  ,
00516      &                   ifirsta , ilasta , nwords
00517      &                 )
00518             do j=1,nwords
00519               ncnct = ncnct + 1
00520               read(cstrg(ifirsta(j):ilasta(j)),*)icnctv(ncnct)
00521             enddo
00522           enddo
00523
00524       else
00525
00526           ncnct = 0
00527
00528       endif
00529
00530       rewind(iunit)
00531
00532 c-----------------------------------------------------------------------
00533 c  read partition data if present
00534 c-----------------------------------------------------------------------
00535 c
00536       lptn_old = lptn
00537       if(lptn_old) then
00538          do i=1,icptn-1
00539            read(iunit,'(1a80)')cstrg
00540          enddo
00541
00542           lresol = .false.
00543           call  xxrptn( iunit  , ndstack,
00544      &                  ndptnl , ndptn  , ndptnc ,
00545      &                  nptnl  , nptn   , nptnc  ,
00546      &                  iptnla , iptna  , iptnca ,
00547      &                  lresol , lptn   ,
00548      &                  cstrg  ,
00549      &                  ncptn_stack     ,cptn_stack
00550      &                )
00551
00552       endif
00553
00554 c------- now check is1min and is1max
00555
00556       if(.not.lptn) then
00557
00558           if((iabt.gt.0).or.(is1min.gt.is1max).or.(is1min.lt.1)
00559      &         .or.(is1max.gt.iz0)) then
```

```
00560                write(i4unit(-1),2002)'incorrect ion or partition limits'
00561                write(i4unit(-1),2003)
00562                stop
00563             endif
00564
00565          elseif(lstan.and.lptn) then
00566
00567             if((iabt.gt.0).or.(is1min.gt.is1max).or.(is1min.ne.1)
00568       &         .or.(is1max.ne.nptn(1)-1)) then
00569                write(i4unit(-1),2002)'incorrect ion or partition limits'
00570                write(i4unit(-1),2003)
00571                stop
00572             endif
00573
00574          elseif(lres.and.lptn) then
00575
00576             if((iabt.gt.0).or.(is1min.gt.is1max).or.(is1min.ne.1)
00577       &         .or.(is1max.ne.ncnct-1)) then
00578                write(i4unit(-1),2002)'incorrect ion or partition limits'
00579                write(i4unit(-1),2003)
00580                stop
00581             endif
00582
00583          endif
00584
00585 c-------------------------------------------------------------------------
00586 c  read temperatures and densities
00587 c-------------------------------------------------------------------------
00588
00589          rewind(iunit)
00590
00591          do i=1,idash_linea(ndash_line)
00592             read(iunit,'(1a80)')cstrg
00593          enddo
00594
00595          read(iunit,1000) ( ddens(i) , i = 1 , idmax )
00596          read(iunit,1000) ( dtev(i)  , i = 1 , itmax )
00597
00598 c----------------------------------------
00599 c  read parent and base metastable indices
00600 c----------------------------------------
00601
00602          chindi = cpatrn(iclass)
00603
00604          iblmx  = 0
00605    20    read(iunit,'(a80)',end=30)cstrg
00606          call xxcase(cstrg,cstrgl,'lc')
00607          if(cstrgl(2:80).ne.cterm(2:80).and.cstrgl(2:2).ne.' ') then
00608 c           if(lptn) then
00609             if(lres) then
00610                if( cstrgl(24:27) .eq. chindi) then
00611                   iblmx = iblmx  + 1
00612                   if((iclass.eq.4).or.(iclass.eq.5).or.
00613       &              (iclass.eq.8).or.(iclass.eq.9).or.
00614       &              (iclass.eq.10).or.(iclass.eq.11))then
00615                      read(cstrgl,1003)isppr(iblmx),
00616       &                               isstgr(iblmx)
00617                      ispbr(iblmx)=0
00618                   else
00619                      read(cstrgl,1002) isppr(iblmx), ispbr(iblmx),
00620       &                               isstgr(iblmx)
00621                   endif
00622                elseif(( cstrgl(24:27). eq. cptrn1(iclass)).or.
00623       &               ( cstrgl(24:27). eq. cptrn2(iclass))) then
00624                   iblmx = iblmx  + 1
00625                   if((iclass.eq.4).or.(iclass.eq.5).or.
00626       &              (iclass.eq.8).or.(iclass.eq.9).or.
00627       &              (iclass.eq.10).or.(iclass.eq.11))then
00628                      read(cstrgl,1003)isppr(iblmx),
00629       &                               isstgr(iblmx)
00630                      ispbr(iblmx)=0
00631                   else
00632                      read(cstrgl,1002) isppr(iblmx), ispbr(iblmx),
00633       &                               isstgr(iblmx)
00634                   endif
00635                   if(.not.lwarn) then
00636                      write(i4unit(-1),2006)'incorrect pointer code',
00637       &                                    ' for class: actual = ',
00638       &                                    cstrgl(24:27),
00639       &                                    ', expected = ',chindi
00640                      write(i4unit(-1),2001)
00641                      lwarn = .true.
00642                   endif
00643                else
00644                   write(i4unit(-1),2005)'incorrect pointer code for',
00645       &                                 ' class: actual = ',
00646       &                                 cstrgl(24:27),
```

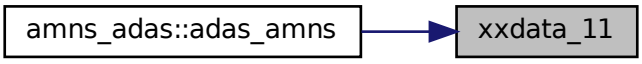
```
00647      &                                ', expected = ',chindi
00648                  write(i4unit(-1),2003)
00649                  stop
00650               endif
00651            else
00652               iblmx = iblmx + 1
00653               read(cstrgl,1001) isstgr(iblmx)
00654               isppr(iblmx) = 1
00655               if((iclass.eq.4).or.(iclass.eq.5).or.
00656      &           (iclass.eq.8).or.(iclass.eq.9).or.
00657      &           (iclass.eq.10).or.(iclass.eq.11))then
00658                   ispbr(iblmx) = 0
00659               else
00660                   ispbr(iblmx) = 1
00661               endif
00662            endif
00663 c-----------------------------------------
00664 c  get the donor mass for classes ccd and prc
00665 c-----------------------------------------
00666
00667            dmass   = 0.0d0
00668            if((iclass.eq.3).or.(iclass.eq.5))then
00669               if((cstrgl(46:47). eq. 'mh').or.
00670      &           (cstrgl(46:47). eq. 'md')) then
00671                       read(cstrgl(49:52),'(f4.2)')dmass
00672               endif
00673               if((dmass.gt.0.0d0).and.(dnr_ams.eq.0.0d0))then
00674                   dnr_ams=dmass
00675                   ldmass = .true.
00676               elseif((dmass.gt.0.d0).and.(dmass.eq.dnr_ams))then
00677                   continue
00678               else
00679                   ldmass  = .false.
00680                   dnr_ams = 0.0d0
00681               endif
00682            endif
00683
00684 c-------------------------
00685 c  read  final gcr values
00686 c-------------------------
00687
00688            do 25 it = 1 , itmax
00689              if(iclass.le.9) then
00690                 read(iunit,1000) ( drcof(iblmx,it,id) , id = 1 , idmax)
00691              else
00692                 read(iunit,1004) ( drcof(iblmx,it,id) , id = 1 , idmax)
00693              endif
00694    25      continue
00695            go to 20
00696         endif
00697 c
00698    30   close(iunit)
00699
00700 c-------------------------------------------------------------------------
00701 c  issue warnings if donor element and/or donor mass is ambiguous
00702 c-------------------------------------------------------------------------
00703         if(((iclass.eq.3).or.(iclass.eq.5)).and.(.not.ldonor)) then
00704             write(i4unit(-1),2000)'unspecified CX donor element'
00705             write(i4unit(-1),2001)
00706         endif
00707         if(((iclass.eq.3).or.(iclass.eq.5)).and.(.not.ldmass)) then
00708             write(i4unit(-1),2000)'unspecified CX donor mass'
00709             write(i4unit(-1),2001)
00710         endif
00711 c
00712 c-------------------------------------------------------------------------
00713 c      verify s1 set in master file consistent with is1min and is1max
00714 c      except for xcd and qcd cases
00715 c-------------------------------------------------------------------------
00716 c
00717         if(iclass.eq.6.or.iclass.eq.7) then
00718             ismax = is1max-is1min+1
00719         else
00720             if(iclass.eq.12) then
00721                 ischk=is1min-1
00722             else
00723                 ischk = is1min
00724             endif
00725
00726             do i=1,iblmx
00727               if(isstgr(i).ne.ischk) then
00728                  ischk = ischk+1
00729               endif
00730            enddo
00731
00732            if(((iclass.ne.10).and.(iclass.ne.11).and.(iclass.ne.12))
00733      &         .and.(ischk.eq.is1max)) then
```

```
00734                ismax = is1max-is1min+1
00735           elseif(((iclass.eq.10).or.(iclass.eq.11)).and.
00736      &         (ischk.eq.is1max+1)) then
00737                ismax = is1max-is1min+2
00738           elseif((iclass.eq.12).and.
00739      &         (ischk.eq.is1max)) then
00740                ismax = is1max-is1min+1
00741                ismax = is1max-is1min+2
00742           else
00743                 write(i4unit(-1),2002)'inconsistent s1 set in file'
00744                 write(i4unit(-1),2003)
00745                 stop
00746            endif
00747         endif
00748 c
00749 c-----------------------------------------------------------------------
00750 c     make up connection vectors and root partitions if appropriate
00751 c     in the resolved case with 0 root partition level, the sum over
00752 c     the connection vector gives the number of partitions
00753 c-----------------------------------------------------------------------
00754 c
00755       if ((.not.lres).and.(.not.lptn)) then
00756           ncnct = iz0+1
00757           do i=1,ncnct
00758              icnctv(i)=1
00759           enddo
00760       elseif ((.not.lres).and.lptn) then
00761           ncnct = is1max+1
00762           do i=1,ncnct
00763              icnctv(i)=1
00764           enddo
00765       endif
00766
00767       if ((.not.lres).and.(.not.lptn)) then
00768           nptnl   = 1
00769           iptnla(nptnl) = 1
00770           nptn(1) =  iz0+1
00771           do i=1,nptn(1)
00772             iptna(nptnl,i) = i-1
00773             nptnc(nptnl,i) = 1
00774             iptnca(1,iptna(nptnl,i)+1,nptnc(nptnl,i))=i-1
00775           enddo
00776       elseif (lres.and.(.not.lptn)) then
00777
00778           nptnl   = 1
00779           iptnla(nptnl) = 0
00780           nptn(1)=1
00781           do i=1,ncnct
00782             nptn(1)=nptn(1)+icnctv(i)
00783           enddo
00784           do i=1,nptn(1)
00785             iptna(nptnl,i) = i-1
00786             nptnc(nptnl,i) = 1
00787             iptnca(1,iptna(nptnl,i)+1,nptnc(nptnl,i))=i-1
00788           enddo
00789       endif
00790 c
00791 c-----------------------------------------------------------------------
00792 c
00793  1000 format(8f10.5)
00794  1001 format(57x,i2)
00795  1002 format(28x,i2,9x,i2,16x,i2)
00796  1003 format(28x,i2,9x,2x,16x,i2)
00797  1004 format(1p8d10.3)
00798 c
00799  2000 format(1x,30('*'),' xxdata_11 warning ',30('*')//
00800     &       2x,a,a,i3,a,i3 )
00801  2001 format(/1x,30('*'),' program continues ',30('*'))
00802  2002 format(1x,30('*'),'  xxdata_11 error  ',30('*')//
00803     &       2x,a,a,i3,a,i3 )
00804  2003 format(/1x,30('*'),' program terminated ',29('*'))
00805  2004 format(1x,30('*'),'  xxdata_11 error  ',30('*')//
00806     &       2x,a,a,a,i3)
00807  2005 format(1x,30('*'),'  xxdata_11 error  ',30('*')//
00808     &       2x,a,a,a4,a,a4 )
00809  2006 format(1x,30('*'),' xxdata_11 warning ',30('*')//
00810     &       2x,a,a,a4,a,a4 )
00811 c
00812 c-----------------------------------------------------------------------
00813 c
00814       return
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.69 xxdata_11.f

```
00001       subroutine  xxdata_11( iunit  , iclass ,
00002      &                       isdimd , iddimd , itdimd ,
00003      &                       ndptnl , ndptn  , ndptnc , ndcnct ,
00004      &                       iz0    , is1min , is1max ,
00005      &                       nptnl  , nptn   , nptnc  ,
00006      &                       iptnla , iptna  , iptnca ,
00007      &                       ncnct  , icnctv ,
00008      &                       iblmx  , ismax  , dnr_ele, dnr_ams,
00009      &                       isppr  , ispbr  , isstgr ,
00010      &                       idmax  , itmax  ,
00011      &                       ddens  , dtev   , drcof  ,
00012      &                       lres   , lstan  , lptn
00013      &                     )
00014        implicit none
00015 c-----------------------------------------------------------------------
00016 c
00017 c  ****************** fortran77 subroutine: xxdata_11 ******************
00018 c
00019 c  purpose: to read a complete  adf11 file, check its class and
00020 c           determine its standard, resolved and partition organisation.
00021 c
00022 c  calling program: various
00023 c
00024 c  notes:    (1) A 'standard' adf11 file contains gcr data between one
00025 c                whole ionisation stage and another whole ionisation
00026 c                stage.
00027 c                A 'resolved' (or partial) adf11 file contains gcr data
00028 c                between a set of metastables of one ionisation stage
00029 c                and a set of metastables of another ionisation stage.
```

```
00030 c                A resolved file is distinguished from a standard file
00031 c                by the presence of a 'connection vector' in the adf11
00032 c                data file header lines.
00033 c                The connection vector specifies the number of meta-
00034 c                stables in each ionisation stage which are coupled
00035 c                together by gcr data.
00036 c            (2) A 'partitioned' adf11 file contains gcr data between
00037 c                clumps of ionisation stages or metastables or comb-
00038 c                inations of the two called 'partitions'.
00039 c                A 'partition level' is a specification of the
00040 c                partitions which span all the ionisation stages (and
00041 c                metastables) of an element.  Successive partition
00042 c                levels give a heirarchy corresponding to larger
00043 c                partitions and greater clumping.
00044 c                A 'superstage' is a set of partitions which are close-
00045 c                coupled.
00046 c                There are thus equivalences :
00047 c                        ionisation stage   -   superstage
00048 c                        metastable         -   partition
00049 c                        ion charge         -   superstage index
00050 c                A partitioned adf11 file may be standard (with each
00051 c                superstage comprising only one partition) or resolved.
00052 c                A partitioned file is distinguished by the presence of
00053 c                'partition specification block' in the adf11 data
00054 c                file header lines.
00055 c            (3) When  a partition specification block is present, it
00056 c                should be ordered from the highest partition level
00057 c                index to lowest partition level index.  Thus the first
00058 c                partition in the partition block has the least number
00059 c                of partitions and the last has the greatest number.
00060 c            (4) Twelev classes of adf11 data file may be read by the
00061 c                subroutine as follow:
00062 c
00063 c                class index    type      GCR data content
00064 c                -----------    ----      ----------------
00065 c                    1          acd       recombination coeffts
00066 c                    2          scd       ionisation coeffts
00067 c                    3          ccd       CX recombination coeffts
00068 c                    4          prb       recomb/brems power coeffts
00069 c                    5          prc       CX power coeffts
00070 c                    6          qcd       base meta. coupl. coeffts
00071 c                    7          xcd       parent meta. coupl. coeffts
00072 c                    8          plt       low level line power coeffts
00073 c                    9          pls       represent. line power coefft
00074 c                   10          zcd       effective charge
00075 c                   11          ycd       effective squared charge
00076 c                   12          ecd       effective ionisation potential
00077 c
00078 c            (5) A resolved adf11 file, with a connection vector, has a set
00079 c                of names and pointers at precise positions in the data file
00080 c                which are recognised.
00081 c                The names are different for partitioned and unpartitioned
00082 c                data files as follow:
00083 c
00084 c                       file      unpartitioned         partitioned
00085 c                       class        names                 names
00086 c
00087 c                       (all)         z1                   s1
00088 c
00089 c                                (indices 1 and 2)     (indices 1 and 2)
00090 c                       ----       ----     ----       ----     ----
00091 c                       acd        iprt     igrd       ispp     ispb
00092 c                       scd        iprt     igrd       ispp     ispb
00093 c                       ccd        iprt     igrd       ispp     ispb
00094 c                       prb        iprt                ispp
00095 c                       prc        iprt                ispp
00096 c                       qcd        igrd     jgrd       ispb     jspb
00097 c                       xcd        iprt     jprt       ispp     jspp
00098 c                       plt        igrd                ispb
00099 c                       pls        igrd                ispb
00100 c                       zcd        igrd                ispb
00101 c                       ycd        igrd                ispb
00102 c                       ecd        igrd                ispb
00103 c
00104 c             (6) In partitioned nomenclature: s=superstage; p=partition;
00105 c                 b=base (current superstage), p=parent (next up super-
00106 c                 stage), c=child (next down superstage). Thus arrays
00107 c                 'iprtr' and 'igrd' in old notation are now substituted
00108 c                 by 'isppr' and 'ispbr' respectively internally and in
00109 c                 external naming.
00110 c
00111 c
00112 c  subroutine:
00113 c
00114 c  input : (i*4)  iunit     = unit to which input file is allocated
00115 c  input : (i*4)  iclass    = class of data (1 - 12 ):
00116 c                             1-acd, 2-scd, 3-ccd, 4-prb, 5-prc
```

```
00117 c                               6-qcd, 7-xcd, 8-plt, 9-pls,10-zcd
00118 c                              11-ycd,12-ecd
00119 c
00120 c  input : (i*4)  isdimd    = maximum number of (sstage, parent, base)
00121 c                            blocks in isonuclear master files
00122 c  input : (i*4)  iddimd    = maximum number of dens values in
00123 c                            isonuclear master files
00124 c  input : (i*4)  itdimd    = maximum number of temp values in
00125 c                            isonuclear master files
00126 c  input : (i*4)  ndptnl    = maximum level of partitions
00127 c  input : (i*4)  ndptn     = maximum no. of partitions in one level
00128 c  input : (i*4)  ndptnc    = maximum no. of components in a partition
00129 c  input : (i*4)  ndcnct    = maximum number of elements in connection
00130 c                            vector
00131 c
00132 c  output: (i*4)  iz0       = nuclear charge
00133 c  output: (i*4)  is1min    = minimum ion charge + 1
00134 c                            (generalised to connection vector index)
00135 c  output: (i*4)  is1max    = maximum ion charge + 1
00136 c                            (note excludes the bare nucleus)
00137 c                            (generalised to connection vector index
00138 c                             and excludes last one which always remains
00139 c                             the bare nucleus)
00140 c  output: (i*4)  nptnl     = number of partition levels in block
00141 c  output: (i*4)  nptn()    = number of partitions in partition level
00142 c                            1st dim: partition level
00143 c  output: (i*4)  nptnc(,)  = number of components in partition
00144 c                            1st dim: partition level
00145 c                            2nd dim: member partition in partition level
00146 c  output: (i*4)  iptnla()  = partition level label (0=resolved root,1=
00147 c                                                     unresolved root)
00148 c                            1st dim: partition level index
00149 c  output: (i*4)  iptna(,)  = partition member label (labelling starts at 0)
00150 c                            1st dim: partition level index
00151 c                            2nd dim: member partition index in partition
00152 c                            level
00153 c  output: (i*4)  iptnca(„)= component label (labelling starts at 0)
00154 c                            1st dim: partition level index
00155 c                            2nd dim: member partition index in partition
00156 c                            level
00157 c                            3rd dim: component index of member partition
00158 c  output: (i*4)  ncnct     = number of elements in connection vector
00159 c  output: (i*4)  icnctv()  = connection vector of number of partitions
00160 c                            of each superstage in resolved case
00161 c                            including the bare nucleus
00162 c                            1st dim: connection vector index
00163 c
00164 c  output: (i*4)  iblmx     = number of (sstage, parent, base)
00165 c                            blocks in isonuclear master file
00166 c  output: (i*4)  ismax     = number of charge states
00167 c                            in isonuclear master file
00168 c                            (generalises to number of elements in
00169 c                             connection vector)
00170 c  output: (c*12) dnr_ele   = CX donor element name for iclass = 3 or 5
00171 c                            (blank if unset)
00172 c  output: (r*8)  dnr_ams   = CX donor element mass for iclass = 3 or 5
00173 c                            (0.0d0 if unset)
00174 c  output: (i*4)  isppr()   = 1st (parent) index for each partition block
00175 c                            1st dim: index of (sstage, parent, base)
00176 c                                     block in isonuclear master file
00177 c  output: (i*4)  ispbr()   = 2nd (base) index for each partition block
00178 c                            1st dim: index of (sstage, parent, base)
00179 c                                     block in isonuclear master file
00180 c  output: (i*4)  isstgr()  = s1 for each resolved data block
00181 c                            (generalises to connection vector index)
00182 c                            1st dim: index of (sstage, parent, base)
00183 c                                     block in isonuclear master file
00184 c
00185 c  output: (i*4)  idmax     = number of dens values in
00186 c                            isonuclear master files
00187 c  output: (i*4)  itmax     = number of temp values in
00188 c                            isonuclear master files
00189 c  output: (r*8)  ddens()   = log10(electron density(cm-3)) from adf11
00190 c  output: (r*8)  dtev()    = log10(electron temperature (eV) from adf11
00191 c  output: (r*8)  drcof(„) = if(iclass <=9):
00192 c                               log10(coll.-rad. coefft.) from
00193 c                               isonuclear master file
00194 c                            if(iclass >=10):
00195 c                               coll.-rad. coefft. from
00196 c                               isonuclear master file
00197 c                            1st dim: index of (sstage, parent, base)
00198 c                                     block in isonuclear master file
00199 c                            2nd dim: electron temperature index
00200 c                            3rd dim: electron density index
00201 c
00202 c  output: (l*4)  lres      = .true.  => partial file
00203 c                           = .false. => not partial file
```

```
00204 c  output: (l*4)  lstan     = .true.  => standard file
00205 c                           = .false. => not standard file
00206 c  output: (l*4)  lptn      = .true.  => partition block present
00207 c                           = .false. => partition block not present
00208 c
00209 c  routines:
00210 c          routine    source    brief description
00211 c          ------------------------------------------------------------
00212 c          i4unit     adas      fetch unit number for output of messages
00213 c          i4fctn     adas      convert string to integer form
00214 c          xfelem     adas      return element name given nuclear charge
00215 c          xxword     adas      extract position of number in buffer
00216 c          xxslen     adas      find string less front and tail blanks
00217 c          xxcase     adas      convert a string to upper or lower case
00218 c          xxrptn     adas      analyse an adf11 file partition block
00219 c
00220 c author:  h. p. summers, university of strathclyde
00221 c          ja7.08
00222 c          tel. 0141-548-4196
00223 c
00224 c date:    04/10/06
00225 c
00226 c version: 1.1                          date: 04/10/2006
00227 c modified: hugh summers
00228 c               - first edition.
00229 c
00230 c version: 1.2                          date: 21/01/2007
00231 c modified: Allan Whiteford
00232 c               - Commented out warning about lack of iclass,
00233 c                 all of the present ADAS files do not contain
00234 c                 this information
00235 c                 (first commit to CVS)
00236 c
00237 c version: 1.3                          date: 08/03/2007
00238 c modified: Hugh Summers
00239 c               - adjustments for revised ecd formats.
00240 c                 charge exchange donor/donor mass checks and
00241 c                 dnr_ele, dnr_ams added to parameter return.
00242 c
00243 c VERSION : 1.4
00244 c DATE    : 28-09-2009
00245 c MODIFIED: Martin O'Mullane
00246 c               - Do not write out warning about missing class
00247 c                 name in file (too many complaints about it).
00248 c
00249 c VERSION : 1.5
00250 c DATE    : 01-02-2010
00251 c MODIFIED: Martin O'Mullane
00252 c               - Incorrect warning message printed for a missing
00253 c                 or inconsistent element name in the adf11 file.
00254 c
00255 c-----------------------------------------------------------------------
00256       integer   nddash    , idword      , ndstack  , ndonors
00257 c-----------------------------------------------------------------------
00258       character csrch1*4   , csrch2*4     , csrch3*4
00259       character csrch4*4   , csrch5*4
00260       character cdash*8    , cpart*3
00261 c-----------------------------------------------------------------------
00262       parameter ( nddash = 6  , idword = 256 , ndstack = 40 )
00263       parameter( csrch1 = 'iprt' , csrch2 = 'igrd'  ,csrch3 = '---/')
00264       parameter( csrch4 = 'ispp' , csrch5 = 'ispb')
00265       parameter( cdash = '--------' , cpart = '//#' )
00266       parameter( ndonors = 4 )
00267 c-----------------------------------------------------------------------
00268       integer   iunit     , i4unit   , i4fctn
00269       integer   iz0       , is1min   , is1max
00270       integer   iddimd    , itdimd   , isdimd
00271       integer   ndptnl    , ndptn    , ndptnc   , ndcnct
00272       integer   iblmx     , ismax    , itmax    , idmax
00273       integer   ischk
00274       integer   iclass    , iclass_file
00275       integer   i         , ic       , it         , id
00276       integer   icptn     , iabt     , iabt1      , iabt2    ,
00277      &          j         , ndash_line
00278       integer   nptnl     , ncnct    , ncptn_stack
00279       integer   nfirst    , iwords   , nwords   , ifirst     , ilast
00280 c-----------------------------------------------------------------------
00281       real*8    dnr_ams   , dmass
00282 c-----------------------------------------------------------------------
00283       character cstrg*80  , cstrg1*80, cterm*80   ,  chindi*4
00284       character xfelem*12 , str12*12 , dnr_ele*12
00285 c-----------------------------------------------------------------------
00286       logical   lres      , lstan     , lptn      , lresol
00287       logical   lptn_old  , lwarn     , ldonor    , ldmass
00288 c-----------------------------------------------------------------------
00289       integer   idash_linea(nddash)
00290       integer   nptn(ndptnl)          , nptnc(ndptnl,ndptn)
```

```
00291       integer   iptnla(ndptnl)        , iptna(ndptnl,ndptn)
00292       integer   iptnca(ndptnl,ndptn,ndptnc)
00293       integer   icnctv(ndcnct)
00294       integer   isstgr(isdimd)
00295       integer   ifirsta(idword)       , ilasta(idword)
00296       integer   ispbr(isdimd)         , isppr(isdimd)
00297 c-----------------------------------------------------------------------
00298       real*8    ddens(iddimd)         , dtev(itdimd)
00299       real*8    drcof(isdimd,itdimd,iddimd)
00300 c-----------------------------------------------------------------------
00301       character cclass(12)*4          , cpatrn(12)*4
00302       character cptrn1(12)*4          , cptrn2(12)*4
00303       character cptn_stack(ndstack)*80
00304       character cdonors(ndonors)*12
00305 c-----------------------------------------------------------------------
00306       data      cterm /'----------------------------------------------
00307      &--------------------------------'/
00308       data      cclass/'/ACD','/SCD','/CCD','/PRB','/PRC',
00309      &                 '/QCD','/XCD','/PLT','/PLS','/ZCD',
00310      &                 '/YCD','/ECD'/
00311       data      cptrn1/'iprt','iprt','iprt','iprt','iprt',
00312      &                 'igrd','iprt','igrd','igrd','igrd',
00313      &                 'igrd','igrd'/
00314       data      cptrn2/'ispp','ispp','ispp','ispp','ispp',
00315      &                 'ispb','ispp','ispb','ispb','ispb',
00316      &                 'ispb','ispb'/
00317       data      cdonors/'HYDROGEN    ','DEUTERIUM   ','TRITIUM     ',
00318      &                  'HELIUM      '/
00319 c-----------------------------------------------------------------------
00320 c
00321 c-----------------------------------------------------------------------
00322 c  search for 'prnt' and count number of dash delimiters
00323 c-----------------------------------------------------------------------
00324
00325       ic = 0
00326       icptn = 0
00327       ndash_line = 0
00328
00329       lstan  = .false.
00330       lres = .false.
00331       lptn   = .false.
00332       nptnl  = 0
00333
00334    10 read(iunit,'(1a80)') cstrg
00335       ic = ic + 1
00336
00337
00338       call xxcase(cstrg,cstrgl,'lc')
00339
00340       if((index(cstrgl,csrch1).le.0) .and.
00341      &    (index(cstrgl,csrch2).le.0) .and.
00342      &    (index(cstrgl,csrch4).le.0) .and.
00343      &    (index(cstrgl,csrch5).le.0) .and.
00344      &    (index(cstrgl,csrch3).le.0)) then
00345
00346           if(index(cstrgl,cdash).gt.0) then
00347               ndash_line = ndash_line+1
00348               idash_linea(ndash_line)=ic
00349           endif
00350           if((cstrgl(1:3).eq.cpart).and.(.not.lptn)) then
00351               lptn = .true.
00352               icptn = ic
00353           endif
00354           go to 10
00355       else
00356           if((iclass.eq.1).or.(iclass.eq.3).or.
00357      &       (iclass.eq.4).or.(iclass.eq.5).or.
00358      &       (iclass.eq.6).or.(iclass.eq.7)) then
00359               if(ndash_line.eq.1) then
00360                   lstan = .true.
00361               elseif ((ndash_line.eq.2).and.lptn)then
00362                   lstan = .true.
00363               elseif ((ndash_line.eq.2).and.(.not.lptn))then
00364                   lres = .true.
00365               elseif ((ndash_line.eq.3).and.lptn)then
00366                   lres = .true.
00367               elseif ((ndash_line.eq.3).and.(.not.lptn))then
00368                   write(i4unit(-1),2002)'header lines faulty'
00369                   write(i4unit(-1),2003)
00370                   stop
00371               elseif (ndash_line.gt.3) then
00372                   write(i4unit(-1),2002)'header lines faulty'
00373                   write(i4unit(-1),2003)
00374                   stop
00375               endif
00376           elseif((iclass.eq.2).or.(iclass.eq.8).or.
00377      &           (iclass.eq.9).or.(iclass.eq.10).or.
```

```
00378      &             (iclass.eq.11).or.(iclass.eq.12)) then
00379                 if(ndash_line.eq.1) then
00380                     lstan = .true.
00381                 elseif ((ndash_line.eq.2).and.lptn)then
00382                     lstan = .true.
00383                 elseif ((ndash_line.eq.2).and.(.not.lptn))then
00384                     lres = .true.
00385                 elseif ((ndash_line.eq.3).and.lptn)then
00386                     lres = .true.
00387                 elseif ((ndash_line.eq.3).and.(.not.lptn))then
00388                     write(i4unit(-1),2002)'header lines faulty'
00389                     write(i4unit(-1),2003)
00390                     stop
00391                 elseif (ndash_line.gt.3) then
00392                     write(i4unit(-1),2002)'header lines faulty'
00393                     write(i4unit(-1),2003)
00394                     stop
00395                 endif
00396             endif
00397        endif
00398 c-----------------------------------------------------------------------
00399 c  assign look-up names according to partitioning & set warning logical
00400 c-----------------------------------------------------------------------
00401
00402        do i=1,12
00403          if(lptn) then
00404              cpatrn(i)=cptrn2(i)
00405          else
00406              cpatrn(i)=cptrn1(i)
00407          endif
00408        enddo
00409
00410        lwarn = .false.
00411
00412 c-----------------------------------------------------------------------
00413 c  rewind, separate and analyse header sections according to type
00414 c-----------------------------------------------------------------------
00415
00416        rewind(iunit)
00417
00418        ic =0
00419
00420 c-----------------------------------------------------------------------
00421 c  first line
00422 c-----------------------------------------------------------------------
00423        read(iunit,'(a80)')cstrg
00424        call xxcase(cstrg,cstrgl,'uc')
00425        ic=ic+1
00426
00427        iabt = 0
00428        iz0 = i4fctn(cstrg(1:5),iabt)
00429        if(iabt.gt.0) then
00430             write(i4unit(-1),2002)'faulty nuclear charge'
00431             write(i4unit(-1),2003)
00432             stop
00433        endif
00434        str12=xfelem(iz0)
00435        call xxslen(str12,ifirst,ilast)
00436        if(index(cstrgl,str12(ifirst:ilast)).le.0) then
00437               write(i4unit(-1),2000)'inconsistent or missing element',
00438      &                              ' name for iz0 = ',iz0
00439               write(i4unit(-1),2001)
00440        endif
00441
00442        iabt = 0
00443        idmax = i4fctn(cstrg(6:10),iabt)
00444        if(iabt.gt.0.or.idmax.le.0.or.idmax.gt.iddimd) then
00445               write(i4unit(-1),2002)'invalid number of densities',
00446      &                              idmax,' = 0 or > ',iddimd
00447               write(i4unit(-1),2003)
00448             stop
00449        endif
00450        iabt = 0
00451        itmax = i4fctn(cstrg(11:15),iabt)
00452        if(iabt.gt.0.or.itmax.le.0.or.itmax.gt.itdimd) then
00453             write(i4unit(-1),2002)'invalid number of ','temperatures',
00454      &                              itmax,' = 0 or > ',itdimd
00455             write(i4unit(-1),2003)
00456             stop
00457        endif
00458        iabt1 = 0
00459        is1min = i4fctn(cstrg(16:20),iabt1)
00460        iabt2 = 0
00461        is1max = i4fctn(cstrg(21:25),iabt2)
00462        iabt = iabt1 + iabt2
00463
00464        iclass_file = 0
```

```
00465         do i=1,12
00466           if(index(cstrgl,cclass(i)).gt.0) then
00467             iclass_file=i
00468           endif
00469         enddo
00470         if(iclass_file.le.0) then
00471 C             write(i4unit(-1),2000)'no class name given in file'
00472 C             write(i4unit(-1),2001)
00473         elseif(iclass_file.ne.iclass) then
00474             write(i4unit(-1),2004)'file class name ',
00475      &                            cclass(iclass_file),
00476      &                           ' does not match iclass = ',iclass
00477             write(i4unit(-1),2003)
00478             stop
00479         endif
00480
00481 c-----------------------------------------------------------------------
00482 c  check if CX donor is explicitly named in dataset top line
00483 c-----------------------------------------------------------------------
00484         ldonor  = .false.
00485         ldmass  = .false.
00486         dnr_ele = ' '
00487         dnr_ams = 0.0d0
00488         if((iclass_file.eq.3).or.(iclass_file.eq.5)) then
00489             do i=1,ndonors
00490               call xxslen(cdonors(i),ifirst,ilast)
00491               if(index(cstrgl,':'//cdonors(i)(ifirst:ilast)).gt.0) then
00492                   ldonor=.true.
00493                   dnr_ele=cdonors(i)
00494               endif
00495             enddo
00496         endif
00497
00498 c-----------------------------------------------------------------------
00499 c  read connection vector if partial file
00500 c-----------------------------------------------------------------------
00501
00502         if(lres) then
00503             do i=2,idash_linea(1)
00504               read(iunit,'(1a80)')cstrg
00505             enddo
00506
00507             ncnct = 0
00508
00509             do i=idash_linea(1)+1,idash_linea(2)-1
00510               cstrg=' '
00511               read(iunit,'(1a80)')cstrg
00512               nfirst = 1
00513               iwords = idword
00514               call xxword( cstrg , ' '    , nfirst ,
00515      &                     iwords  ,
00516      &                     ifirsta , ilasta , nwords
00517      &                   )
00518               do j=1,nwords
00519                 ncnct = ncnct + 1
00520                 read(cstrg(ifirsta(j):ilasta(j)),*)icnctv(ncnct)
00521               enddo
00522             enddo
00523
00524         else
00525
00526             ncnct = 0
00527
00528         endif
00529
00530         rewind(iunit)
00531
00532 c-----------------------------------------------------------------------
00533 c  read partition data if present
00534 c-----------------------------------------------------------------------
00535 c
00536         lptn_old = lptn
00537         if(lptn_old) then
00538            do i=1,icptn-1
00539              read(iunit,'(1a80)')cstrg
00540            enddo
00541
00542             lresol = .false.
00543             call  xxrptn( iunit  , ndstack,
00544      &                    ndptnl , ndptn  , ndptnc ,
00545      &                    nptnl  , nptn   , nptnc  ,
00546      &                    iptnla , iptna  , iptnca ,
00547      &                    lresol , lptn   ,
00548      &                    cstrg  ,
00549      &                    ncptn_stack     ,cptn_stack
00550      &                  )
00551
```

```
00552        endif
00553
00554 c------- now check is1min and is1max
00555
00556        if(.not.lptn) then
00557
00558            if((iabt.gt.0).or.(is1min.gt.is1max).or.(is1min.lt.1)
00559      &         .or.(is1max.gt.iz0)) then
00560                write(i4unit(-1),2002)'incorrect ion or partition limits'
00561                write(i4unit(-1),2003)
00562                stop
00563            endif
00564
00565        elseif(lstan.and.lptn) then
00566
00567            if((iabt.gt.0).or.(is1min.gt.is1max).or.(is1min.ne.1)
00568      &         .or.(is1max.ne.nptn(1)-1)) then
00569                write(i4unit(-1),2002)'incorrect ion or partition limits'
00570                write(i4unit(-1),2003)
00571                stop
00572            endif
00573
00574        elseif(lres.and.lptn) then
00575
00576            if((iabt.gt.0).or.(is1min.gt.is1max).or.(is1min.ne.1)
00577      &         .or.(is1max.ne.ncnct-1)) then
00578                write(i4unit(-1),2002)'incorrect ion or partition limits'
00579                write(i4unit(-1),2003)
00580                stop
00581            endif
00582
00583        endif
00584
00585 c-------------------------------------------------------------------------
00586 c  read temperatures and densities
00587 c-------------------------------------------------------------------------
00588
00589        rewind(iunit)
00590
00591        do i=1,idash_linea(ndash_line)
00592           read(iunit,'(1a80)')cstrg
00593        enddo
00594
00595        read(iunit,1000) ( ddens(i) , i = 1 , idmax )
00596        read(iunit,1000) ( dtev(i)  , i = 1 , itmax )
00597
00598 c-----------------------------------------
00599 c  read parent and base metastable indices
00600 c-----------------------------------------
00601
00602        chindi = cpatrn(iclass)
00603
00604        iblmx  = 0
00605    20  read(iunit,'(a80)',end=30)cstrg
00606        call xxcase(cstrg,cstrgl,'lc')
00607        if(cstrgl(2:80).ne.cterm(2:80).and.cstrgl(2:2).ne.' ') then
00608 c           if(lptn) then
00609            if(lres) then
00610                if( cstrgl(24:27) .eq. chindi) then
00611                   iblmx = iblmx  + 1
00612                   if((iclass.eq.4).or.(iclass.eq.5).or.
00613      &               (iclass.eq.8).or.(iclass.eq.9).or.
00614      &               (iclass.eq.10).or.(iclass.eq.11))then
00615                       read(cstrgl,1003)isppr(iblmx),
00616      &                                 isstgr(iblmx)
00617                       ispbr(iblmx)=0
00618                   else
00619                       read(cstrgl,1002) isppr(iblmx), ispbr(iblmx),
00620      &                                 isstgr(iblmx)
00621                   endif
00622                elseif(( cstrgl(24:27). eq. cptrn1(iclass)).or.
00623      &                 ( cstrgl(24:27). eq. cptrn2(iclass))) then
00624                   iblmx = iblmx  + 1
00625                   if((iclass.eq.4).or.(iclass.eq.5).or.
00626      &               (iclass.eq.8).or.(iclass.eq.9).or.
00627      &               (iclass.eq.10).or.(iclass.eq.11))then
00628                       read(cstrgl,1003)isppr(iblmx),
00629      &                                 isstgr(iblmx)
00630                       ispbr(iblmx)=0
00631                   else
00632                      read(cstrgl,1002) isppr(iblmx), ispbr(iblmx),
00633      &                                 isstgr(iblmx)
00634                   endif
00635                   if(.not.lwarn) then
00636                       write(i4unit(-1),2006)'incorrect pointer code',
00637      &                                      ' for class: actual = ',
00638      &                                        cstrgl(24:27),
```

```
00639      &                                  ', expected = ',chindi
00640                        write(i4unit(-1),2001)
00641                        lwarn = .true.
00642                     endif
00643                  else
00644                     write(i4unit(-1),2005)'incorrect pointer code for',
00645      &                                     ' class: actual = ',
00646      &                                     cstrgl(24:27),
00647      &                                  ', expected = ',chindi
00648                     write(i4unit(-1),2003)
00649                     stop
00650                  endif
00651               else
00652                  iblmx = iblmx + 1
00653                  read(cstrgl,1001) isstgr(iblmx)
00654                  isppr(iblmx) = 1
00655                  if((iclass.eq.4).or.(iclass.eq.5).or.
00656      &              (iclass.eq.8).or.(iclass.eq.9).or.
00657      &              (iclass.eq.10).or.(iclass.eq.11))then
00658                     ispbr(iblmx) = 0
00659                  else
00660                     ispbr(iblmx) = 1
00661                  endif
00662               endif
00663 c-----------------------------------------
00664 c  get the donor mass for classes ccd and prc
00665 c-----------------------------------------
00666
00667               dmass   = 0.0d0
00668               if((iclass.eq.3).or.(iclass.eq.5))then
00669                  if((cstrgl(46:47). eq. 'mh').or.
00670      &              (cstrgl(46:47). eq. 'md')) then
00671                        read(cstrgl(49:52),'(f4.2)')dmass
00672                  endif
00673                  if((dmass.gt.0.0d0).and.(dnr_ams.eq.0.0d0))then
00674                     dnr_ams=dmass
00675                     ldmass = .true.
00676                  elseif((dmass.gt.0.d0).and.(dmass.eq.dnr_ams))then
00677                     continue
00678                  else
00679                     ldmass  = .false.
00680                     dnr_ams = 0.0d0
00681                  endif
00682               endif
00683
00684 c-------------------------
00685 c  read  final gcr values
00686 c-------------------------
00687
00688               do 25 it = 1 , itmax
00689                 if(iclass.le.9) then
00690                     read(iunit,1000) ( drcof(iblmx,it,id) , id = 1 , idmax)
00691                 else
00692                     read(iunit,1004) ( drcof(iblmx,it,id) , id = 1 , idmax)
00693                 endif
00694    25         continue
00695               go to 20
00696            endif
00697 c
00698    30   close(iunit)
00699
00700 c-----------------------------------------------------------------------
00701 c  issue warnings if donor element and/or donor mass is ambiguous
00702 c-----------------------------------------------------------------------
00703           if(((iclass.eq.3).or.(iclass.eq.5)).and.(.not.ldonor)) then
00704               write(i4unit(-1),2000)'unspecified CX donor element'
00705               write(i4unit(-1),2001)
00706           endif
00707           if(((iclass.eq.3).or.(iclass.eq.5)).and.(.not.ldmass)) then
00708               write(i4unit(-1),2000)'unspecified CX donor mass'
00709               write(i4unit(-1),2001)
00710           endif
00711 c
00712 c-----------------------------------------------------------------------
00713 c      verify s1 set in master file consistent with is1min and is1max
00714 c      except for xcd and qcd cases
00715 c-----------------------------------------------------------------------
00716 c
00717           if(iclass.eq.6.or.iclass.eq.7) then
00718               ismax = is1max-is1min+1
00719           else
00720               if(iclass.eq.12) then
00721                   ischk=is1min-1
00722               else
00723                   ischk = is1min
00724               endif
00725
```

```
00726            do i=1,iblmx
00727              if(isstgr(i).ne.ischk) then
00728                 ischk = ischk+1
00729              endif
00730           enddo
00731
00732           if(((iclass.ne.10).and.(iclass.ne.11).and.(iclass.ne.12))
00733      &         .and.(ischk.eq.is1max)) then
00734               ismax = is1max-is1min+1
00735           elseif(((iclass.eq.10).or.(iclass.eq.11)).and.
00736      &         (ischk.eq.is1max+1)) then
00737               ismax = is1max-is1min+2
00738           elseif((iclass.eq.12).and.
00739      &         (ischk.eq.is1max)) then
00740               ismax = is1max-is1min+1
00741               ismax = is1max-is1min+2
00742           else
00743                write(i4unit(-1),2002)'inconsistent s1 set in file'
00744                write(i4unit(-1),2003)
00745                stop
00746            endif
00747         endif
00748 c
00749 c-----------------------------------------------------------------------
00750 c     make up connection vectors and root partitions if appropriate
00751 c     in the resolved case with 0 root partition level, the sum over
00752 c     the connection vector gives the number of partitions
00753 c-----------------------------------------------------------------------
00754 c
00755       if ((.not.lres).and.(.not.lptn)) then
00756           ncnct = iz0+1
00757           do i=1,ncnct
00758              icnctv(i)=1
00759           enddo
00760       elseif ((.not.lres).and.lptn) then
00761           ncnct = is1max+1
00762           do i=1,ncnct
00763              icnctv(i)=1
00764           enddo
00765       endif
00766
00767       if ((.not.lres).and.(.not.lptn)) then
00768           nptnl   = 1
00769           iptnla(nptnl) = 1
00770           nptn(1) =  iz0+1
00771           do i=1,nptn(1)
00772             iptna(nptnl,i) = i-1
00773             nptnc(nptnl,i) = 1
00774             iptnca(1,iptna(nptnl,i)+1,nptnc(nptnl,i))=i-1
00775           enddo
00776       elseif (lres.and.(.not.lptn)) then
00777
00778           nptnl   = 1
00779           iptnla(nptnl) = 0
00780           nptn(1)=1
00781           do i=1,ncnct
00782             nptn(1)=nptn(1)+icnctv(i)
00783           enddo
00784           do i=1,nptn(1)
00785             iptna(nptnl,i) = i-1
00786             nptnc(nptnl,i) = 1
00787             iptnca(1,iptna(nptnl,i)+1,nptnc(nptnl,i))=i-1
00788           enddo
00789       endif
00790 c
00791 c-----------------------------------------------------------------------
00792 c
00793  1000 format(8f10.5)
00794  1001 format(57x,i2)
00795  1002 format(28x,i2,9x,i2,16x,i2)
00796  1003 format(28x,i2,9x,2x,16x,i2)
00797  1004 format(1p8d10.3)
00798 c
00799  2000 format(1x,30('*'),' xxdata_11 warning ',30('*')//
00800     &       2x,a,a,i3,a,i3 )
00801  2001 format(/1x,30('*'),' program continues ',30('*'))
00802  2002 format(1x,30('*'),'  xxdata_11 error  ',30('*')//
00803     &       2x,a,a,i3,a,i3 )
00804  2003 format(/1x,30('*'),' program terminated ',29('*'))
00805  2004 format(1x,30('*'),'  xxdata_11 error  ',30('*')//
00806     &       2x,a,a,a,i3)
00807  2005 format(1x,30('*'),'  xxdata_11 error  ',30('*')//
00808     &       2x,a,a,a4,a,a4 )
00809  2006 format(1x,30('*'),' xxdata_11 warning ',30('*')//
00810     &       2x,a,a,a4,a,a4 )
00811 c
00812 c-----------------------------------------------------------------------
```

```
00813 c
00814       return
00815       end
```

16.70 src/amns_driver/xxrptn.f File Reference

Functions/Subroutines

- subroutine xxrptn (iunit, ndstack, ndptnl, ndptn, ndptnc, nptnl, nptn, nptnc, iptnla, iptna, iptnca, lresol, lptn, cstrg, ncptn_stack, cptn_stack)

16.70.1 Function/Subroutine Documentation

16.70.1.1 xxrptn()

```
subroutine xxrptn (
             integer iunit,
             integer ndstack,
             integer ndptnl,
             integer ndptn,
             integer ndptnc,
             integer nptnl,
             integer, dimension(ndptnl) nptn,
             integer, dimension(ndptnl,ndptn) nptnc,
             integer, dimension(ndptnl) iptnla,
             integer, dimension(ndptnl,ndptn) iptna,
             integer, dimension(ndptnl,ndptn,ndptnc) iptnca,
             logical lresol,
             logical lptn,
             character cstrg,
             integer ncptn_stack,
             character, dimension(ndstack) cptn_stack )
```

Definition at line 1 of file xxrptn.f.

```
00009       implicit none
00010 c-----------------------------------------------------------------------
00011 c
00012 c  ****************** fortran77 subroutine: xxprtn *********************
00013 c
00014 c  Purpose:  To read and analyse a partition block in a datafile header
00015 c
00016 c  Calling program: adas416
00017 c
00018 c  Notes:  (1) Partition levels, partitions and partition components are
00019 c              labelled starting at 0 (but see (2)).
00020 c          (2) Partition level 0 labels the resolved root partition level
00021 c              partition level 1 labels the unresolved root partition
00022 c              level.
00023 c          (3) For an unresolved (standard) file, the partitions are each
00024 c              ionisation stage from the neutral to the bare nucleus and
00025 c              they are labelled by the ion charge.  Each partition has
00026 c              just the one component.
00027 c          (4) Distinguish the indexing (starting at 1) from the label
00028 c              (starting at 0) .
00029 c
00030 c  Subroutine:
00031 c
00032 c  input : (i*4)  iunit     = unit to which input file is allocated
00033 c  input : (i*4)  ndstack   = maximum no. of text lines in partition block
00034 c
00035 c  input : (i*4)  ndptnl    = maximum level of partitions
00036 c  input : (i*4)  ndptn     = maximum no. of partitions in one level
00037 c  input : (i*4)  ndptnc    = maximum no. of components in a partition
00038 c  input : (l*4)  lresol    = .true.  => resolved root partition
00039 c                           = .false. => standard root partition
00040 c
00041 c  output: (i*4)  nptnl     = number of partition levels in block
00042 c  output: (i*4)  nptn()    = number of partitions in partition level
00043 c                             1st dim: partition level
00044 c  output: (i*4)  nptnc(,)  = number of components in partition
```

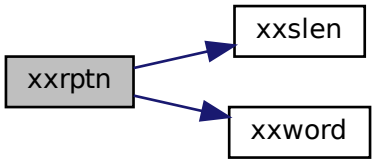
```
00045 c                              1st dim: partition level
00046 c                              2nd dim: member partition in partition level
00047 c  output: (i*4)  iptnla()  = partition level label (0=resolved root,1=
00048 c                                                    unresolved root)
00049 c                              1st dim: partition level index
00050 c  output: (i*4)  iptna(,)  = partition member label (labelling starts at 0)
00051 c                              1st dim: partition level index
00052 c                              2nd dim: member partition index in partition
00053 c                              level
00054 c  output: (i*4)  iptnca(„)= component label (labelling starts at 0)
00055 c                              1st dim: partition level index
00056 c                              2nd dim: member partition index in partition
00057 c                              level
00058 c                              3rd dim: component index of member partition
00059 c  output: (l*4)  lptn      = .true.  => partition block present
00060 c                           = .false. => partition block not present
00061 c  output: (c*80) cstrg     = string marking end of partition block
00062 c  output: (i*4)  ncptn_stack= number of text lines in partition block
00063 c  output: (c*80) cptn_stack()=text lines of partition block
00064 c                               1st dim: text line pointer
00065 c
00066 c
00067 c Routines:
00068 c          Routine    Source    Brief description
00069 c          ------------------------------------------------------------
00070 c          I4UNIT     ADAS      Fetch unit number for output of messages
00071 c          XXSLEN     ADAS      Find non-blank characters in string
00072 c          XXWORD     ADAS      Extract position of number in buffer
00073 c
00074 c Author:  H. P. Summers, university of strathclyde
00075 c          JA7.08
00076 c          tel. 0141-548-4196
00077 c
00078 c Date:    25/08/05
00079 c
00080 c Version: 1.1                          Date: 25/08/2005
00081 c Modified: Hugh Summers
00082 c               - First edition.
00083 c
00084 c Version: 1.2                          Date: 28/02/2008
00085 c Modified: Adam Foster
00086 c               - Increased length of strg to 1024
00087 c
00088 c Version: 1.3                          Date: 28/02/2008
00089 c Modified: Allan Whiteford
00090 c               - Added comments for Adam's change
00091 c               - Fixed capitalisation of comments section.
00092 c
00093 c-----------------------------------------------------------------------
00094 c-----------------------------------------------------------------------
00095       integer   idptnl   , idword
00096 c-----------------------------------------------------------------------
00097       parameter(idptnl = 4  , idword = 256 )
00098 c-----------------------------------------------------------------------
00099       integer   iunit     , i4unit
00100       integer   ndptnl    , ndptn      , ndptnc    , ndstack
00101       integer   nptnl
00102       integer   ifirst    , ilast
00103       integer   nchari    , ncharf
00104       integer   nfirst    , iwords     , nwords
00105       integer   nfirstc   , iwordsc    , nwordsc
00106       integer   i         , j          , k         , ic
00107       integer   ncptn_stack
00108 c-----------------------------------------------------------------------
00109       character cstrg*80
00110       Character strg*1024
00111 c-----------------------------------------------------------------------
00112       logical   lresol    , lptn
00113       logical   lptn_in
00114 c-----------------------------------------------------------------------
00115       integer   nptn(ndptnl)          , nptnc(ndptnl,ndptn)
00116       integer   iptnla(ndptnl)        , iptna(ndptnl,ndptn)
00117       integer   iptnca(ndptnl,ndptn,ndptnc)
00118       integer   ifirsta(idword)       , ilasta(idword)
00119       integer   ifirstac(idword)      , ilastac(idword)
00120       integer   ncpta(idptnl)
00121 c-----------------------------------------------------------------------
00122       character cptna(idptnl)*1024
00123       character cptn_stack(ndstack)*80
00124 c-----------------------------------------------------------------------
00125 c
00126 c-----------------------------------------------------------------------
00127 c  read partition data if present
00128 c-----------------------------------------------------------------------
00129
00130       if(idptnl.ne.ndptnl) then
00131         write(i4unit(-1),1001)'mismatch of internal dimensions'
```

```
00132           write(i4unit(-1),1002)
00133           stop
00134         endif
00135
00136         if(idptnl.ne.ndptnl) then
00137           write(i4unit(-1),1001)'mismatch of internal dimensions'
00138           write(i4unit(-1),1002)
00139           stop
00140         endif
00141
00142         do i=1,ndptnl
00143           cptna(i) = ' '
00144         enddo
00145 c
00146         lptn   = .false.
00147         cstrg  = ' '
00148         nptnl  = 0
00149         ncptn_stack = 0
00150
00151    10   read(iunit,'(1a80)') cstrg
00152
00153         if(cstrg(1:3).eq.'//#') then
00154             nchari = 0
00155             nptnl = nptnl + 1
00156             lptn = .true.
00157             lptn_in  = .true.
00158             call xxslen(cstrg,ifirst,ilast)
00159
00160             ncptn_stack=ncptn_stack+1
00161             cptn_stack(ncptn_stack)=' '
00162             cptn_stack(ncptn_stack)(1:ilast-ifirst+1)=cstrg(ifirst:ilast)
00163
00164             ncharf = nchari+ilast-ifirst+1
00165             cptna(nptnl)(nchari+1:ncharf) = cstrg(ifirst:ilast)
00166             nchari=ncharf
00167         elseif((cstrg(1:3).eq.'   ').and.lptn_in) then
00168             call xxslen(cstrg,ifirst,ilast)
00169
00170             ncptn_stack=ncptn_stack+1
00171             cptn_stack(ncptn_stack)=' '
00172             cptn_stack(ncptn_stack)(1:ilast-ifirst+7)=
00173      &                        '      '//cstrg(ifirst:ilast)
00174             ncharf=nchari+ilast-ifirst+2
00175             cptna(nptnl)(nchari+1: ncharf) = ' '//cstrg(ifirst:ilast)
00176
00177             nchari = ncharf
00178         elseif(cstrg(1:3).eq.'---') then
00179             lptn_in = .false.
00180             go to 15
00181         else
00182             write(i4unit(-1),1001)
00183             write(i4unit(-1),1002)
00184             stop
00185         endif
00186         go to 10
00187    15   continue
00188
00189 c
00190 c-----------------------------------------------------------------------
00191 c  preliminary checks
00192 c-----------------------------------------------------------------------
00193 c
00194         if(lptn) then
00195             do i = 1,nptnl
00196               read(cptna(i)(4:5),'(i2)')iptnla(i)
00197 c              write(i4unit(-1),*)'xxrptn: i,iptnla=',i,iptnla(i)
00198             enddo
00199             if((iptnla(1).gt.2).and.(nptnl.eq.1).and.(.not.lresol)) then
00200                 write(i4unit(-1),1001) 'partition level',i,' in error'
00201                 write(i4unit(-1),1002)
00202                 stop
00203             endif
00204             do i = 1,nptnl
00205               call xxslen(cptna(i),ifirst,ilast)
00206               if(cptna(i)(ilast:ilast).ne.'/') then
00207                   write(i4unit(-1),1001) 'partition level',i,
00208      &                                   ' not terminated'
00209                   write(i4unit(-1),1002)
00210               stop
00211               endif
00212             enddo
00213
00214          endif
00215 c
00216 c-----------------------------------------------------------------------
00217 c  analyse partitions
00218 c-----------------------------------------------------------------------
```

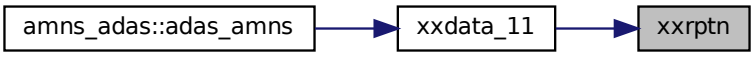
```
00219 c
00220       do i=1,nptnl
00221           nfirst = 1
00222           iwords = idword
00223           call xxslen(cptna(i),ifirst,ilast)
00224           call xxword( cptna(i)(ifirst+5:ilast) , '/'    , nfirst ,
00225      &                 iwords  ,
00226      &                 ifirsta , ilasta , nwords
00227      &               )
00228           if((nwords-2*int(nwords/2)).ne.0) then
00229                write(i4unit(-1),1001) 'partition level',i,
00230      &                                ' partition count wrong'
00231                write(i4unit(-1),1002)
00232                stop
00233           endif
00234
00235           nptn(i)=nwords/2
00236           ic = 0
00237 c          write(i4unit(-1),*)'xxrptn: i,nwords,nptn(i)=',
00238 c     &                               i,nwords,nptn(i)
00239           do j=1,nptn(i)
00240              strg=' '
00241              strg=cptna(i)(ifirsta(2*j-1)+5:ilasta(2*j-1)+5)
00242 c             write(i4unit(-1),*)'xxrptn-xxslen:strg=',strg
00243              call xxslen(strg,ifirst,ilast)
00244              read(strg(ifirst+1:ilast),'(i2)')iptna(i,j)
00245              strg = ' '
00246
00247
00248              strg=cptna(i)(ifirsta(2*j)+5:ilasta(2*j)+5)
00249
00250              iwordsc = idword
00251              nfirstc = 1
00252 c             write(i4unit(-1),*)'xxrptn-xxword:strg=',strg
00253 c             write(i4unit(-1),*)'xxrptn-xxword:nfirstc=',nfirstc
00254              call xxword( strg , ' '    , nfirstc ,
00255      &                    iwordsc  ,
00256      &                    ifirstac , ilastac , nwordsc
00257      &                  )
00258 c             write(i4unit(-1),*)'xxrptn:nwordsc=',nwordsc
00259              nptnc(i,j)=nwordsc
00260 c             write(i4unit(-1),*)'xxrptn: i,j,nptnc(i,j)=',
00261 c     &                                   i,j,nptnc(i,j)
00262 c             write(i4unit(-1),*)'xxrptn: ifirstac(1),ilastac(1)=',
00263 c     &                                   ifirstac(1),ilastac(1)
00264 c           write(i4unit(-1),*)'xxrptn:',strg(ifirstac(1):ilastac(1))
00265
00266
00267              do k=1,nptnc(i,j)
00268
00269                read(strg(ifirstac(k):ilastac(k)),*)iptnca(i,j,k)
00270 c-----------------------------------------------
00271 c  components should be sequential and contiguous.
00272 c-----------------------------------------------
00273                if(iptnca(i,j,k).ne.ic) then
00274                     write(i4unit(-1),1001) 'partition count',ic,
00275      &                                   ' out of sequence'
00276                     write(i4unit(-1),1002)
00277                     stop
00278                else
00279                     ic=ic+1
00280                endif
00281
00282              enddo
00283           enddo
00284
00285           ncpta(i)=ic
00286 c         write(i4unit(-1),*)'i,ncpta(i)=',i,ncpta(i)
00287
00288        enddo
00289 c
00290 c-------------------------------------------------------------------------
00291 c  final checks. level i+1 partition components should span
00292 c  partition i p-values
00293 c-------------------------------------------------------------------------
00294 c
00295       if (nptnl.ge.2)then
00296           do i=2,nptnl
00297             if(ncpta(i-1).ne.nptn(i))then
00298                write(i4unit(-1),1001) 'partition level',i,
00299      &                                ' count incorrect'
00300                write(i4unit(-1),1002)
00301                stop
00302             endif
00303           enddo
00304       endif
00305
```

```
00306       return
00307 c
00308 c-----------------------------------------------------------------------
00309 c
00310  1001 format(1x,32('*'),' xxrptn error ',32('*')//
00311     &       1x,'fault in input data file: ',a,i3,a)
00312  1002 format(/1x,29('*'),' program terminated ',29('*'))
00313
```

Here is the call graph for this function:



Here is the caller graph for this function:



16.71 xxrptn.f

```
00001       subroutine  xxrptn( iunit  , ndstack,
00002      &                     ndptnl , ndptn  , ndptnc ,
00003      &                     nptnl  , nptn   , nptnc  ,
00004      &                     iptnla , iptna  , iptnca ,
00005      &                     lresol , lptn   ,
00006      &                     cstrg  ,
00007      &                     ncptn_stack     , cptn_stack
00008      &                   )
00009       implicit none
00010 c-----------------------------------------------------------------------
00011 c
00012 c  ****************** fortran77 subroutine: xxprtn *********************
00013 c
00014 c  Purpose:  To read and analyse a partition block in a datafile header
00015 c
00016 c  Calling program: adas416
00017 c
00018 c  Notes:  (1) Partition levels, partitions and partition components are
00019 c              labelled starting at 0 (but see (2)).
00020 c          (2) Partition level 0 labels the resolved root partition level
00021 c              partition level 1 labels the unresolved root partition
00022 c              level.
00023 c          (3) For an unresolved (standard) file, the partitions are each
00024 c              ionisation stage from the neutral to the bare nucleus and
00025 c              they are labelled by the ion charge.  Each partition has
00026 c              just the one component.
00027 c          (4) Distinguish the indexing (starting at 1) from the label
00028 c              (starting at 0) .
00029 c
00030 c  Subroutine:
00031 c
00032 c  input : (i*4)  iunit     = unit to which input file is allocated
00033 c  input : (i*4)  ndstack   = maximum no. of text lines in partition block
00034 c
00035 c  input : (i*4)  ndptnl    = maximum level of partitions
```

```
00036 c  input : (i*4)  ndptn     = maximum no. of partitions in one level
00037 c  input : (i*4)  ndptnc    = maximum no. of components in a partition
00038 c  input : (l*4)  lresol    = .true.  => resolved root partition
00039 c                           = .false. => standard root partition
00040 c
00041 c  output: (i*4)  nptnl     = number of partition levels in block
00042 c  output: (i*4)  nptn()    = number of partitions in partition level
00043 c                             1st dim: partition level
00044 c  output: (i*4)  nptnc(,)  = number of components in partition
00045 c                             1st dim: partition level
00046 c                             2nd dim: member partition in partition level
00047 c  output: (i*4)  iptnla()  = partition level label (0=resolved root,1=
00048 c                                                     unresolved root)
00049 c                             1st dim: partition level index
00050 c  output: (i*4)  iptna(,)  = partition member label (labelling starts at 0)
00051 c                             1st dim: partition level index
00052 c                             2nd dim: member partition index in partition
00053 c                             level
00054 c  output: (i*4)  iptnca(„)= component label (labelling starts at 0)
00055 c                             1st dim: partition level index
00056 c                             2nd dim: member partition index in partition
00057 c                             level
00058 c                             3rd dim: component index of member partition
00059 c  output: (l*4)  lptn      = .true.  => partition block present
00060 c                           = .false. => partition block not present
00061 c  output: (c*80) cstrg     = string marking end of partition block
00062 c  output: (i*4)  ncptn_stack= number of text lines in partition block
00063 c  output: (c*80) cptn_stack()=text lines of partition block
00064 c                              1st dim: text line pointer
00065 c
00066 c
00067 c Routines:
00068 c          Routine    Source    Brief description
00069 c          ------------------------------------------------------------
00070 c          I4UNIT     ADAS      Fetch unit number for output of messages
00071 c          XXSLEN     ADAS      Find non-blank characters in string
00072 c          XXWORD     ADAS      Extract position of number in buffer
00073 c
00074 c Author:  H. P. Summers, university of strathclyde
00075 c          JA7.08
00076 c          tel. 0141-548-4196
00077 c
00078 c Date:    25/08/05
00079 c
00080 c Version: 1.1                          Date: 25/08/2005
00081 c Modified: Hugh Summers
00082 c               - First edition.
00083 c
00084 c Version: 1.2                          Date: 28/02/2008
00085 c Modified: Adam Foster
00086 c               - Increased length of strg to 1024
00087 c
00088 c Version: 1.3                          Date: 28/02/2008
00089 c Modified: Allan Whiteford
00090 c               - Added comments for Adam's change
00091 c               - Fixed capitalisation of comments section.
00092 c
00093 c-----------------------------------------------------------------------
00094 c-----------------------------------------------------------------------
00095       integer   idptnl   , idword
00096 c-----------------------------------------------------------------------
00097       parameter (idptnl = 4  , idword = 256 )
00098 c-----------------------------------------------------------------------
00099       integer   iunit     , i4unit
00100       integer   ndptnl    , ndptn      , ndptnc    , ndstack
00101       integer   nptnl
00102       integer   ifirst    , ilast
00103       integer   nchari    , ncharf
00104       integer   nfirst    , iwords     , nwords
00105       integer   nfirstc   , iwordsc    , nwordsc
00106       integer   i         , j          , k         , ic
00107       integer   ncptn_stack
00108 c-----------------------------------------------------------------------
00109       character cstrg*80
00110       Character strg*1024
00111 c-----------------------------------------------------------------------
00112       logical   lresol    , lptn
00113       logical   lptn_in
00114 c-----------------------------------------------------------------------
00115       integer   nptn(ndptnl)          , nptnc(ndptnl,ndptn)
00116       integer   iptnla(ndptnl)        , iptna(ndptnl,ndptn)
00117       integer   iptnca(ndptnl,ndptn,ndptnc)
00118       integer   ifirsta(idword)       , ilasta(idword)
00119       integer   ifirstac(idword)      , ilastac(idword)
00120       integer   ncpta(idptnl)
00121 c-----------------------------------------------------------------------
00122       character cptna(idptnl)*1024
```

```
00123       character cptn_stack(ndstack)*80
00124 c-----------------------------------------------------------------------
00125 c
00126 c-----------------------------------------------------------------------
00127 c  read partition data if present
00128 c-----------------------------------------------------------------------
00129
00130       if(idptnl.ne.ndptnl) then
00131         write(i4unit(-1),1001)'mismatch of internal dimensions'
00132         write(i4unit(-1),1002)
00133         stop
00134       endif
00135
00136       if(idptnl.ne.ndptnl) then
00137         write(i4unit(-1),1001)'mismatch of internal dimensions'
00138         write(i4unit(-1),1002)
00139         stop
00140       endif
00141
00142       do i=1,ndptnl
00143         cptna(i) = ' '
00144       enddo
00145 c
00146       lptn   = .false.
00147       cstrg  = ' '
00148       nptnl  = 0
00149       ncptn_stack = 0
00150
00151    10  read(iunit,'(1a80)') cstrg
00152
00153       if(cstrg(1:3).eq.'//#') then
00154           nchari = 0
00155           nptnl = nptnl + 1
00156           lptn = .true.
00157           lptn_in  = .true.
00158           call xxslen(cstrg,ifirst,ilast)
00159
00160           ncptn_stack=ncptn_stack+1
00161           cptn_stack(ncptn_stack)=' '
00162           cptn_stack(ncptn_stack)(1:ilast-ifirst+1)=cstrg(ifirst:ilast)
00163
00164           ncharf = nchari+ilast-ifirst+1
00165           cptna(nptnl)(nchari+1:ncharf) = cstrg(ifirst:ilast)
00166           nchari=ncharf
00167       elseif((cstrg(1:3).eq.'   ').and.lptn_in) then
00168           call xxslen(cstrg,ifirst,ilast)
00169
00170           ncptn_stack=ncptn_stack+1
00171           cptn_stack(ncptn_stack)=' '
00172           cptn_stack(ncptn_stack)(1:ilast-ifirst+7)=
00173      &                    '      '//cstrg(ifirst:ilast)
00174           ncharf=nchari+ilast-ifirst+2
00175           cptna(nptnl)(nchari+1: ncharf) = ' '//cstrg(ifirst:ilast)
00176
00177           nchari = ncharf
00178       elseif(cstrg(1:3).eq.'---') then
00179           lptn_in = .false.
00180           go to 15
00181       else
00182           write(i4unit(-1),1001)
00183           write(i4unit(-1),1002)
00184           stop
00185       endif
00186       go to 10
00187    15  continue
00188
00189 c
00190 c-----------------------------------------------------------------------
00191 c  preliminary checks
00192 c-----------------------------------------------------------------------
00193 c
00194       if(lptn) then
00195           do i = 1,nptnl
00196             read(cptna(i)(4:5),'(i2)')iptnla(i)
00197 c            write(i4unit(-1),*)'xxrptn: i,iptnla=',i,iptnla(i)
00198           enddo
00199           if((iptnla(1).gt.2).and.(nptnl.eq.1).and.(.not.lresol)) then
00200               write(i4unit(-1),1001) 'partition level',i,' in error'
00201               write(i4unit(-1),1002)
00202               stop
00203           endif
00204           do i = 1,nptnl
00205             call xxslen(cptna(i),ifirst,ilast)
00206             if(cptna(i)(ilast:ilast).ne.'/') then
00207                 write(i4unit(-1),1001) 'partition level',i,
00208      &                                 ' not terminated'
00209                 write(i4unit(-1),1002)
```

```
00210                stop
00211              endif
00212            enddo
00213
00214          endif
00215 c
00216 c-----------------------------------------------------------------------
00217 c  analyse partitions
00218 c-----------------------------------------------------------------------
00219 c
00220         do i=1,nptnl
00221            nfirst = 1
00222            iwords = idword
00223            call xxslen(cptna(i),ifirst,ilast)
00224            call xxword( cptna(i)(ifirst+5:ilast) , '/'    , nfirst ,
00225      &                  iwords  ,
00226      &                  ifirsta , ilasta , nwords
00227      &                )
00228            if((nwords-2*int(nwords/2)).ne.0) then
00229                  write(i4unit(-1),1001) 'partition level',i,
00230      &                                  ' partition count wrong'
00231                  write(i4unit(-1),1002)
00232                  stop
00233            endif
00234
00235            nptn(i)=nwords/2
00236            ic = 0
00237 c           write(i4unit(-1),*)'xxrptn: i,nwords,nptn(i)=',
00238 c     &                                 i,nwords,nptn(i)
00239            do j=1,nptn(i)
00240               strg=' '
00241               strg=cptna(i)(ifirsta(2*j-1)+5:ilasta(2*j-1)+5)
00242 c              write(i4unit(-1),*)'xxrptn-xxslen:strg=',strg
00243               call xxslen(strg,ifirst,ilast)
00244               read(strg(ifirst+1:ilast),'(i2)')iptna(i,j)
00245               strg = ' '
00246
00247
00248               strg=cptna(i)(ifirsta(2*j)+5:ilasta(2*j)+5)
00249
00250               iwordsc = idword
00251               nfirstc = 1
00252 c              write(i4unit(-1),*)'xxrptn-xxword:strg=',strg
00253 c              write(i4unit(-1),*)'xxrptn-xxword:nfirstc=',nfirstc
00254               call xxword( strg , ' '    , nfirstc ,
00255      &                     iwordsc  ,
00256      &                     ifirstac , ilastac , nwordsc
00257      &                   )
00258 c              write(i4unit(-1),*)'xxrptn:nwordsc=',nwordsc
00259               nptnc(i,j)=nwordsc
00260 c              write(i4unit(-1),*)'xxrptn: i,j,nptnc(i,j)=',
00261 c     &                                    i,j,nptnc(i,j)
00262 c              write(i4unit(-1),*)'xxrptn: ifirstac(1),ilastac(1)=',
00263 c     &                                    ifirstac(1),ilastac(1)
00264 c             write(i4unit(-1),*)'xxrptn:',strg(ifirstac(1):ilastac(1))
00265
00266
00267               do k=1,nptnc(i,j)
00268
00269                 read(strg(ifirstac(k):ilastac(k)),*)iptnca(i,j,k)
00270 c-----------------------------------------------
00271 c  components should be sequential and contiguous.
00272 c-----------------------------------------------
00273                 if(iptnca(i,j,k).ne.ic) then
00274                      write(i4unit(-1),1001) 'partition count',ic,
00275      &                                  ' out of sequence'
00276                      write(i4unit(-1),1002)
00277                      stop
00278                 else
00279                      ic=ic+1
00280                 endif
00281
00282               enddo
00283            enddo
00284
00285            ncpta(i)=ic
00286 c          write(i4unit(-1),*)'i,ncpta(i)=',i,ncpta(i)
00287
00288         enddo
00289 c
00290 c-----------------------------------------------------------------------
00291 c  final checks. level i+1 partition components should span
00292 c  partition i p-values
00293 c-----------------------------------------------------------------------
00294 c
00295         if (nptnl.ge.2)then
00296            do i=2,nptnl
```

```
00297              if(ncpta(i-1).ne.nptn(i))then
00298                  write(i4unit(-1),1001) 'partition level',i,
00299      &                                  ' count incorrect'
00300                  write(i4unit(-1),1002)
00301                  stop
00302              endif
00303            enddo
00304         endif
00305
00306         return
00307 c
00308 c-----------------------------------------------------------------------
00309 c
00310  1001 format(1x,32('*'),' xxrptn error ',32('*')//
00311     &       1x,'fault in input data file: ',a,i3,a)
00312  1002 format(/1x,29('*'),' program terminated ',29('*'))
00313
00314       end
```

16.72 src/amns_driver/xxslen.f File Reference

Functions/Subroutines

- subroutine xxslen (CSTRNG, IFIRST, ILAST)

16.72.1 Function/Subroutine Documentation

16.72.1.1 xxslen()

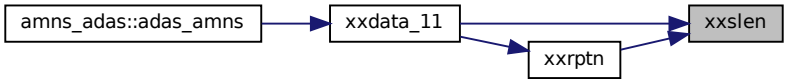
```
subroutine xxslen (
             character, dimension(*) CSTRNG,
             integer IFIRST,
             integer ILAST )
```

Definition at line 3 of file xxslen.f.

```
00004       IMPLICIT NONE
00005 C-----------------------------------------------------------------------
00006 C
00007 C  ****************** FORTRAN77 SUBROUTINE: XXSLEN *********************
00008 C
00009 C  PURPOSE: TO IDENTIFY THE FIRST AND LAST NON-BLANK CHARACTER IN A
00010 C           STRING. (IF INPUT STRING IS BLANK IFIRST=ILAST=0)
00011 C
00012 C  CALLING PROGRAM: GENERAL USE
00013 C
00014 C  SUBROUTINE:
00015 C
00016 C  INPUT : (C*(*)) CSTRNG   = INPUT STRING FOR INTERROGATION
00017 C
00018 C  OUTPUT: (I*4)   IFIRST   = BYTE POSITION OF FIRST NON-BLANK CHARACTER
00019 C                             IN INPUT STRING.
00020 C  OUTPUT: (I*4)   ILAST    = BYTE POSITION OF LAST  NON-BLANK CHARACTER
00021 C                             IN INPUT STRING.
00022 C
00023 C          (I*4)   I        = GENERAL USE
00024 C          (I*4)   ILEN     = LENGTH OF 'CSTRNG' STRING IN BYTES
00025 C
00026 C ROUTINES: NONE
00027 C
00028 C NOTE:
00029 C
00030 C
00031 C AUTHOR:  PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00032 C          K1/0/37
00033 C          JET EXT. 6023
00034 C
00035 C DATE  :  06/07/93
00036 C
00037 C-----------------------------------------------------------------------
00038 C-----------------------------------------------------------------------
00039 C
00040 C-----------------------------------------------------------------------
00041       INTEGER     IFIRST     , ILAST    , ILEN    , I
00042 C-----------------------------------------------------------------------
00043       CHARACTER   CSTRNG*(*)
00044 C-----------------------------------------------------------------------
00045 C-----------------------------------------------------------------------
```

```
00046 C
00047 C-----------------------------------------------------------------------
00048 C
00049       ilen   = len(cstrng)
00050 C-----------------------------------------------------------------------
00051       ifirst = 0
00052       ilast  = 0
00053 C-----------------------------------------------------------------------
00054 C
00055          DO 1 i=1,ilen
00056 C
00057             IF (cstrng(i:i).NE.' ') THEN
00058                IF (ifirst.EQ.0) ifirst = i
00059                ilast = i
00060             ENDIF
00061 C
00062     1    CONTINUE
00063 C
00064 C-----------------------------------------------------------------------
00065 C
00066        RETURN
```

Here is the caller graph for this function:



16.73 xxslen.f

```
00001 CX UNIX PORT - SCCS Info : Module @(#)$Header: /home/adascvs/fortran/adaslib/utility/xxslen.for,v 1.1
       2004/07/06 15:39:12 whitefor Exp $ Date $Date: 2004/07/06 15:39:12 $
00002 CX
00003       SUBROUTINE xxslen( CSTRNG , IFIRST , ILAST )
00004       IMPLICIT NONE
00005 C-----------------------------------------------------------------------
00006 C
00007 C  ****************** FORTRAN77 SUBROUTINE: XXSLEN *********************
00008 C
00009 C  PURPOSE: TO IDENTIFY THE FIRST AND LAST NON-BLANK CHARACTER IN A
00010 C           STRING. (IF INPUT STRING IS BLANK IFIRST=ILAST=0)
00011 C
00012 C  CALLING PROGRAM: GENERAL USE
00013 C
00014 C  SUBROUTINE:
00015 C
00016 C  INPUT : (C*(*)) CSTRNG   = INPUT STRING FOR INTERROGATION
00017 C
00018 C  OUTPUT: (I*4)   IFIRST   = BYTE POSITION OF FIRST NON-BLANK CHARACTER
00019 C                            IN INPUT STRING.
00020 C  OUTPUT: (I*4)   ILAST    = BYTE POSITION OF LAST  NON-BLANK CHARACTER
00021 C                            IN INPUT STRING.
00022 C
00023 C          (I*4)   I        = GENERAL USE
00024 C          (I*4)   ILEN     = LENGTH OF 'CSTRNG' STRING IN BYTES
00025 C
00026 C ROUTINES: NONE
00027 C
00028 C NOTE:
00029 C
00030 C
00031 C AUTHOR:  PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00032 C          K1/0/37
00033 C          JET EXT. 6023
00034 C
00035 C DATE  :  06/07/93
00036 C
00037 C-----------------------------------------------------------------------
00038 C-----------------------------------------------------------------------
00039 C
00040 C-----------------------------------------------------------------------
00041       INTEGER     IFIRST     , ILAST    , ILEN    , I
00042 C-----------------------------------------------------------------------
00043       CHARACTER   CSTRNG*(*)
00044 C-----------------------------------------------------------------------
```

```
00045 C-----------------------------------------------------------------------
00046 C
00047 C-----------------------------------------------------------------------
00048 C
00049       ilen   = len(cstrng)
00050 C-----------------------------------------------------------------------
00051       ifirst = 0
00052       ilast  = 0
00053 C-----------------------------------------------------------------------
00054 C
00055          DO 1 i=1,ilen
00056 C
00057             IF (cstrng(i:i).NE.' ') THEN
00058                IF (ifirst.EQ.0) ifirst = i
00059                ilast = i
00060             ENDIF
00061 C
00062     1    CONTINUE
00063 C
00064 C-----------------------------------------------------------------------
00065 C
00066        RETURN
00067        END
```

16.74 src/amns_driver/xxword.f File Reference

Functions/Subroutines

- subroutine xxword (CTEXT, CDELIM, NFIRST, IWORDS, IFIRST, ILAST, NWORDS)

16.74.1 Function/Subroutine Documentation

16.74.1.1 xxword()

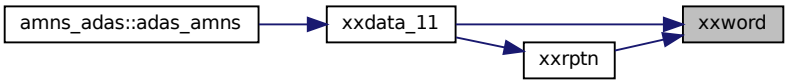
```
subroutine xxword (
             character, dimension(*) CTEXT,
             character, dimension(*) CDELIM,
             integer NFIRST,
             integer IWORDS,
             integer, dimension(iwords) IFIRST,
             integer, dimension(iwords) ILAST,
             integer NWORDS )
```

Definition at line 3 of file xxword.f.

```
00007       IMPLICIT NONE
00008 C-----------------------------------------------------------------------
00009 C
00010 C  ****************** FORTRAN77 SUBROUTINE: XXWORD *********************
00011 C
00012 C  PURPOSE: TO EXTRACT THE Nfirst to (Nfirst+IWORDS-1) WORDS FROM AN
00013 C           INPUT STRING. OUTPUTS THE FIRST AND LAST BYTE INDEXES OF
00014 C           EACH WORD AS WELL AS THE TOTAL NUMBER OF WORDS FOUND.
00015 C
00016 C           A WORD = A STRING OF CHARACTERS SEPARATED BY ANY CHARACTER
00017 C                    CONTAINED IN THE INPUT STRING CDELIM.
00018 C
00019 C  CALLING PROGRAM: GENERAL USE
00020 C
00021 C  SUBROUTINE:
00022 C
00023 C  INPUT : (C*(*)) CTEXT   = INPUT TEXT LINE CONTAINING STRING
00024 C  INPUT : (C*(*)) CDELIM  = INPUT STRING CONTAINING DELIMITER CHARS.
00025 C  INPUT : (I*4)   NFIRST  = THE INDEX NO. OF THE FIRST WORD TO EXTRACT.
00026 C
00027 C  I/O   : (I*4)   IWORDS  = INPUT : SIZE OF IFIRST, ILAST(ARRAYS)
00028 C                                    (I.E. NUMBER OF WORDS TO EXTRACT)
00029 C                           = OUTPUT: NUMBER OF REQUESTED WORDS FOUND
00030 C
00031 C  OUTPUT: (I*4)   IFIRST()= INDEX OF FIRST BYTE OF THE Nth WORD
00032 C  OUTPUT: (I*4)   ILAST()  = INDEX OF LAST  BYTE OF THE Nth WORD
00033 C  OUTPUT: (I*4)   NWORDS  = THE TOTAL NUMBER OF WORDS FOUND IN CTEXT
00034 C
00035 C           (I*4)   LENTXT  = LENGTH IN BYTES OF 'CTEXT' STRING
00036 C           (I*4)   IDELIM  = 0 => CTEXT CHARACTER IS NOT A DELIMITER
00037 C                           > 0 => CTEXT CHARACTER IS A DELIMITER
```

```
00038 C           (I*4)    ITOTAL  = NUMBER OF WORDS FOUND SO FAR
00039 C           (I*4)    IINDEX  = IFIRST()/ILAST() INDEX OF CURRENT WORD
00040 C           (I*4)    NLAST   = THE INDEX NO. OF THE LAST WORD TO EXTRACT
00041 C           (I*4)    I       = GENERAL USE INDEX
00042 C
00043 C           (L*4)    LWORD   = .TRUE.  - PROCESSING AN IDENTIFIED WORD
00044 C                              .FALSE. - PROCESSING SPACE BETWEEN WORDS
00045 C
00046 C ROUTINES: NONE
00047 C
00048 C NOTES:     IF THERE IS NO Nfirst WORD OR NO WORDS ARE FOUND
00049 C            (I.E. INPUT STRING IS BLANK) THEN IWORDS=0
00050 C
00051 C AUTHOR:  PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00052 C          K1/0/37
00053 C          JET EXT. 5023
00054 C
00055 C DATE:    20/05/93
00056 C
00057 C-----------------------------------------------------------------------
00058       INTEGER    NFIRST          , IWORDS           , NWORDS      ,
00059      &           LENTXT          , IDELIM           , ITOTAL      ,
00060      &           IINDEX          , NLAST            , I
00061 C-----------------------------------------------------------------------
00062       LOGICAL    LWORD
00063 C-----------------------------------------------------------------------
00064       CHARACTER  CTEXT*(*)       , CDELIM*(*)
00065 C-----------------------------------------------------------------------
00066       INTEGER    IFIRST(IWORDS)  , ILAST(IWORDS)
00067 C-----------------------------------------------------------------------
00068 C
00069 C-----------------------------------------------------------------------
00070 C
00071       lentxt = len(ctext)
00072       nlast  = iwords + nfirst - 1
00073 C
00074 C-----------------------------------------------------------------------
00075 C FIND THE REQUIRED WORDS
00076 C-----------------------------------------------------------------------
00077 C
00078          DO 1 i = 1,iwords
00079             ifirst(i) = 0
00080             ilast(i)  = 0
00081     1    CONTINUE
00082 C
00083       itotal = 0
00084       lword  = .false.
00085       iindex = 0
00086 C
00087          DO 2 i = 1,lentxt
00088
00089             idelim = index( cdelim , ctext(i:i) )
00090
00091                IF (lword) THEN
00092
00093                   IF (idelim.GT.0) THEN
00094                      lword  = .false.
00095                      IF ((itotal.GE.nfirst).AND.(itotal.LE.nlast)) THEN
00096                         ilast(iindex)  = i - 1
00097                      ENDIF
00098                   ENDIF
00099
00100                ELSE
00101
00102                   IF (idelim.EQ.0) THEN
00103                      lword  = .true.
00104                      itotal = itotal + 1
00105                      IF ((itotal.GE.nfirst).AND.(itotal.LE.nlast)) THEN
00106                         iindex         = iindex + 1
00107                         ifirst(iindex) = i
00108                         ilast(iindex)  = lentxt
00109                      ENDIF
00110                   ENDIF
00111
00112                ENDIF
00113
00114     2    CONTINUE
00115 C
00116       iwords = iindex
00117       nwords = itotal
00118 C
00119 C-----------------------------------------------------------------------
00120 C
00121       RETURN
```

Here is the caller graph for this function:



16.75 xxword.f

```
00001 CX UNIX PORT - SCCS Info : Module @(#)$Header: /home/adascvs/fortran/adaslib/utility/xxword.for,v 1.1
       2004/07/06 15:40:43 whitefor Exp $ Date $Date: 2004/07/06 15:40:43 $
00002 CX
00003       SUBROUTINE xxword( CTEXT  , CDELIM , NFIRST ,
00004      &                   IWORDS ,
00005      &                   IFIRST , ILAST  , NWORDS
00006      &                 )
00007       IMPLICIT NONE
00008 C-----------------------------------------------------------------------
00009 C
00010 C  ****************** FORTRAN77 SUBROUTINE: XXWORD *********************
00011 C
00012 C  PURPOSE: TO EXTRACT THE Nfirst to (Nfirst+IWORDS-1) WORDS FROM AN
00013 C           INPUT STRING. OUTPUTS THE FIRST AND LAST BYTE INDEXES OF
00014 C           EACH WORD AS WELL AS THE TOTAL NUMBER OF WORDS FOUND.
00015 C
00016 C           A WORD = A STRING OF CHARACTERS SEPARATED BY ANY CHARACTER
00017 C                    CONTAINED IN THE INPUT STRING CDELIM.
00018 C
00019 C  CALLING PROGRAM: GENERAL USE
00020 C
00021 C  SUBROUTINE:
00022 C
00023 C  INPUT : (C*(*)) CTEXT   = INPUT TEXT LINE CONTAINING STRING
00024 C  INPUT : (C*(*)) CDELIM  = INPUT STRING CONTAINING DELIMITER CHARS.
00025 C  INPUT : (I*4)   NFIRST  = THE INDEX NO. OF THE FIRST WORD TO EXTRACT.
00026 C
00027 C  I/O   : (I*4)   IWORDS  = INPUT : SIZE OF IFIRST, ILAST(ARRAYS)
00028 C                                    (I.E. NUMBER OF WORDS TO EXTRACT)
00029 C                           = OUTPUT: NUMBER OF REQUESTED WORDS FOUND
00030 C
00031 C  OUTPUT: (I*4)   IFIRST()= INDEX OF FIRST BYTE OF THE Nth WORD
00032 C  OUTPUT: (I*4)   ILAST() = INDEX OF LAST  BYTE OF THE Nth WORD
00033 C  OUTPUT: (I*4)   NWORDS  = THE TOTAL NUMBER OF WORDS FOUND IN CTEXT
00034 C
00035 C          (I*4)   LENTXT  = LENGTH IN BYTES OF 'CTEXT' STRING
00036 C          (I*4)   IDELIM  = 0 => CTEXT CHARACTER IS NOT A DELIMITER
00037 C                          > 0 => CTEXT CHARACTER IS A DELIMITER
00038 C          (I*4)   ITOTAL  = NUMBER OF WORDS FOUND SO FAR
00039 C          (I*4)   IINDEX  = IFIRST()/ILAST() INDEX OF CURRENT WORD
00040 C          (I*4)   NLAST   = THE INDEX NO. OF THE LAST WORD TO EXTRACT
00041 C          (I*4)   I       = GENERAL USE INDEX
00042 C
00043 C          (L*4)   LWORD   = .TRUE.  - PROCESSING AN IDENTIFIED WORD
00044 C                            .FALSE. - PROCESSING SPACE BETWEEN WORDS
00045 C
00046 C ROUTINES: NONE
00047 C
00048 C NOTES:    IF THERE IS NO Nfirst WORD OR NO WORDS ARE FOUND
00049 C           (I.E. INPUT STRING IS BLANK) THEN IWORDS=0
00050 C
00051 C AUTHOR:  PAUL E. BRIDEN (TESSELLA SUPPORT SERVICES PLC)
00052 C          K1/0/37
00053 C          JET EXT. 5023
00054 C
00055 C DATE:    20/05/93
00056 C
00057 C-----------------------------------------------------------------------
00058       INTEGER    NFIRST          , IWORDS          , NWORDS      ,
00059      &           LENTXT          , IDELIM          , ITOTAL      ,
00060      &           IINDEX          , NLAST           , I
00061 C-----------------------------------------------------------------------
00062       LOGICAL    LWORD
00063 C-----------------------------------------------------------------------
00064       CHARACTER  CTEXT*(*)       , CDELIM*(*)
00065 C-----------------------------------------------------------------------
00066       INTEGER    IFIRST(IWORDS)  , ILAST(IWORDS)
```

```
00067 C-----------------------------------------------------------------------
00068 C
00069 C-----------------------------------------------------------------------
00070 C
00071       lentxt = len(ctext)
00072       nlast  = iwords + nfirst - 1
00073 C
00074 C-----------------------------------------------------------------------
00075 C FIND THE REQUIRED WORDS
00076 C-----------------------------------------------------------------------
00077 C
00078          DO 1 i = 1,iwords
00079             ifirst(i) = 0
00080             ilast(i)  = 0
00081     1    CONTINUE
00082 C
00083       itotal = 0
00084       lword  = .false.
00085       iindex = 0
00086 C
00087          DO 2 i = 1,lentxt
00088
00089             idelim = index( cdelim , ctext(i:i) )
00090
00091                IF (lword) THEN
00092
00093                   IF (idelim.GT.0) THEN
00094                      lword  = .false.
00095                      IF ((itotal.GE.nfirst).AND.(itotal.LE.nlast)) THEN
00096                         ilast(iindex)  = i - 1
00097                      ENDIF
00098                   ENDIF
00099
00100                ELSE
00101
00102                   IF (idelim.EQ.0) THEN
00103                      lword  = .true.
00104                      itotal = itotal + 1
00105                      IF ((itotal.GE.nfirst).AND.(itotal.LE.nlast)) THEN
00106                         iindex         = iindex + 1
00107                         ifirst(iindex) = i
00108                         ilast(iindex)  = lentxt
00109                      ENDIF
00110                   ENDIF
00111
00112                ENDIF
00113
00114     2    CONTINUE
00115 C
00116       iwords = iindex
00117       nwords = itotal
00118 C
00119 C-----------------------------------------------------------------------
00120 C
00121       RETURN
00122       END
```

16.76 src/java/src/amns/Amns.java File Reference

Classes

- class amns.Amns

Packages

- amns

16.77 Amns.java

```
00001 package amns;
00002
00003 import amns.type.*;
00004 import java.io.*;
00005
00014 public class Amns {
00025     public native long ImasAmnsCcSetupReactants(int idx);
00026
00038     public native long ImasAmnsCcSetupReactantsNumber(int idx, int n_reactants);
00039
```

```
00047     public native void ImasAmnsCCSetReactant(long ptr, AmnsReactantType settings);
00048
00056     public native void ImasAmnsCCSetReactantIdx(long ptr, int reactant_idx, AmnsReactantType
      settings);
00057
00064     public native long ImasAmnsCCSetup(AmnsErrorType error);
00065
00072     public native void ImasAmnsCCGetReactant(long ptr, AmnsReactantType species);
00073
00084     public native double ImasAmnsCCRX0B(long ptr, double arg1, double arg2, AmnsErrorType error);
00085
00095     public native double[] ImasAmnsCCRX1A(long ptr, int nx, double [] arg1, AmnsErrorType error);
00096
00107     public native double[] ImasAmnsCCRX1B(long ptr, int nx, double [] arg1, double [] arg2,
      AmnsErrorType error);
00108
00120     public native double[] ImasAmnsCCRX1C(long ptr, int nx, double [] arg1, double [] arg2, double []
      arg3, AmnsErrorType error);
00121
00130     public native long ImasAmnsCCFinishTable(long ptr, AmnsErrorType error);
00131
00138     public native long ImasAmnsCCFinishReactants(long ptr);
00139
00147     public native long ImasAmnsCCFinish(long ptr, AmnsErrorType error);
00148
00158     public native long ImasAmnsCCSetupTable(long ptrAmns, AmnsReactionType reaction, long
      ptrReactants, AmnsErrorType error);
00159
00168     public native void ImasAmnsCCSet(long ptrAmns, AmnsSetType setType, AmnsErrorType error);
00169
00178     public native void ImasAmnsCCSetTable(long ptr, AmnsSetType setType, AmnsErrorType error);
00179
00189     public native void ImasAmnsCCQuery(long ptrAmns, AmnsQueryType query, AmnsAnswerType answer,
      AmnsErrorType error);
00190
00200     public native void ImasAmnsCCQueryTable(long ptrTable, AmnsQueryType query, AmnsAnswerType answer,
      AmnsErrorType error);
00201
00203     static {
00204         System.loadLibrary("amnsjni"); // Load native library libamnsjni.so
00205     }
00206
00213     public static void printErrorCode(AmnsErrorType error, String message) {
00214
00215         if(message != null && message.length() != 0 ) {
00216             System.out.println("Error result for: " + message);
00217         }
00218         System.out.println("[AMNS error code] message: " + error.string + " flag: " + error.flag);
00219
00220     }
00221
00230     public static double [] reshape2DTo1D(double [][] input, int nx, int ny) {
00231         double [] result = new double[nx * ny];
00232
00233         for( int i=0; i<nx; i++) {
00234             for( int p=0; p<ny; p++) {
00235                 result[ (i * ny) + p] = input[i][p];
00236             }
00237         }
00238
00239         return result;
00240     }
00241
00250     public static double [][] reshape1DTo2D(double [] input, int nx, int ny) {
00251
00252         double [][] result = new double[nx][ny];
00253
00254         for( int i=0; i<nx; i++) {
00255             for( int p=0; p<ny; p++) {
00256                 result[i][p] = input[ (i * ny) + p];
00257             }
00258         }
00259         return result;
00260     }
00261 }
00262
```

16.78 src/java/src/amns/type/AmnsAnswerType.java File Reference

Classes

- class amns.type.AmnsAnswerType

Packages

- package amns.type

16.79 AmnsAnswerType.java

```
00001 package amns.type;
00002
00012 public class AmnsAnswerType {
00014     public String string;
00016     public int number;
00017 }
```

16.80 src/java/src/amns/type/AmnsErrorType.java File Reference

Classes

- class amns.type.AmnsErrorType

Packages

- package amns.type

16.81 AmnsErrorType.java

```
00001 package amns.type;
00002
00012 public class AmnsErrorType {
00014     public boolean flag;
00016     public String string;
00017 }
00018
```

16.82 src/java/src/amns/type/AmnsQueryType.java File Reference

Classes

- class amns.type.AmnsQueryType

Packages

- package amns.type

16.83 AmnsQueryType.java

```
00001 package amns.type;
00002
00012 public class AmnsQueryType {
00022     public String string;
00023 }
```

16.84 src/java/src/amns/type/AmnsReactantType.java File Reference

Classes

- class amns.type.AmnsReactantType

Packages

- package amns.type

16.85 AmnsReactantType.java

```
00001 package amns.type;
00002
00010 public class AmnsReactantType {
00012     public double ZN;
00014     public double ZA;
00016     public double MI;
00018     public int LR;
00020     public double real_specifier;
00022     public int int_specifier;
00023 }
```

16.86 src/java/src/amns/type/AmnsReactionType.java File Reference

Classes

- class amns.type.AmnsReactionType

Packages

- package amns.type

16.87 AmnsReactionType.java

```
00001 package amns.type;
00002
00011 public class AmnsReactionType {
00012     public String string;
00013     public int isotopeResolved;
00014 }
00015
00016
```

16.88 src/java/src/amns/type/AmnsSetType.java File Reference

Classes

- class amns.type.AmnsSetType

Packages

- package amns.type

16.89 AmnsSetType.java

```
00001 package amns.type;
00002
00010 public class AmnsSetType {
00016     public String string;
00017 }
```

16.90 src/libamns/amns.dox File Reference

16.91 src/libamns/amns_external_functions.f90 File Reference

Data Types

- type amns_external_functions::fun_err_t

Modules

- module amns_external_functions

Functions/Subroutines

- subroutine amns_external_functions::nuclear_data_1001 (function_parameters, x, f, with_warnings, fun_err)

  AMNS External function ...
- subroutine amns_external_functions::nuclear_data_1002 (function_parameters, Tin, f, with_warnings, fun↩ _err)

  AMNS External function ...
- subroutine amns_external_functions::rct_data_1003 (function_parameters, Tin, f, with_warnings, fun_err)

  AMNS External function ...
- subroutine amns_external_functions::sputter_data_1004 (function_parameters, energy_arr, angle_arr, yield_arr, with_warnings, fun_err)

  AMNS External function ...
- subroutine amns_external_functions::reflect_data_1005 (function_parameters, energy_arr, angle_arr, refl_↩ arr, with_warnings, fun_err)

  AMNS External function ...
- subroutine amns_external_functions::nuclear_data_1006 (function_parameters, x, f, with_warnings, fun_err)

  AMNS External function ...

16.92 amns_external_functions.f90

```
00001
00007 module amns_external_functions
00008
00009   use ids_types  ! IGNORE
00010   implicit none
00011
00012   type fun_err_t
00013      integer                     :: ierr
00014      character (len=128)         :: cerr
00015   end type fun_err_t
00016
00017 contains
00018
00025   subroutine nuclear_data_1001(function_parameters, x, f, with_warnings, fun_err)
00026     implicit none
00027     real (ids_real),  intent(in)  :: function_parameters(:,:), x(:)
00028     real (ids_real),  intent(out) :: f(:)
00029     logical,           intent(in)  :: with_warnings
00030     type (fun_err_t),  intent(out) :: fun_err
00031     integer :: i, j
00032     real (ids_real) :: e, s
00033
00034     fun_err%ierr = 0
00035     fun_err%cerr = ''
00036
00037     if(size(function_parameters,1).ne.12) then
00038        write(fun_err%cerr,*) "nuclear_data_1001: Expected the length of 'function_parameters' to be 12
       in 'nuclear_data_1001' and not ", &
00039             size(function_parameters)
00040        fun_err%ierr = -1
00041        return
00042     endif
00043     if(size(x).ne.size(f)) then
00044        write(fun_err%cerr,*) "nuclear_data_1001: Expected the input and output vectors to be the same
      size in 'nuclear_data_1001'"
00045        fun_err%ierr = -1
00046        return
00047     endif
00048
00049     do i=1, size(x)
00050        e = x(i) * 1.0e-3_ids_real
00051        j=1
00052        if(e.lt.function_parameters(11,j) .and. with_warnings) then
00053           write(*,*) 'extrapolating  below the desired range'
00054        endif
00055        do while(e.gt.function_parameters(12,j).and. &
00056             j.lt.size(function_parameters,2))
00057           j=j+1
00058        enddo
00059        if(e.gt.function_parameters(12,j)) then
00060           if (with_warnings) then
00061              write(*,*) 'extrapolating  above the desired range'
00062              write(*,*) j, x(i), function_parameters(12,j)
00063              write(*,*) 'Taking the boundary value when calculating S!'
00064           endif
```

```
00065          e = function_parameters(12,j)
00066       endif
00067       s = ( function_parameters(2,j) &
00068            & + e*(function_parameters(3,j) &
00069            & + e*(function_parameters(4,j) &
00070            & + e*( function_parameters(5,j) &
00071            & +e*function_parameters(6,j) ) ) ) ) &
00072            & /( 1 + e*(function_parameters(7,j)&
00073            &  + e*(function_parameters(8,j) &
00074            & + e*( function_parameters(9,j) &
00075            & +e*function_parameters(10,j) ) ) ) )
00076       ! We clamp the energy for S (the nuclear fusion probability).
00077       ! However, there is no need to clamp the energy in denominator.
00078       ! The denominator roughly describes the probability of the reactants to
00079       ! overcome the Coulomb barrier.
00080       e = x(i) * 1.0e-3_ids_real
00081       f(i) = s / (e * exp(function_parameters(1,j)/sqrt(e))) * 1.0e-31_ids_real
00082    enddo
00083  end subroutine nuclear_data_1001
00084
00091  subroutine nuclear_data_1002(function_parameters, Tin, f, with_warnings, fun_err)
00092    implicit none
00093    real (ids_real), intent(in)  :: function_parameters(:,:), Tin(:)
00094    real (ids_real), intent(out) :: f(:)
00095    logical,           intent(in)  :: with_warnings
00096    type (fun_err_t),  intent(out) :: fun_err
00097    integer :: iT ! index for the different energy vaulues
00098    integer :: iR ! index for possible different temperature ranges
00099
00100    real (ids_real) :: T
00101    real (ids_real) :: theta,xi,mrc2,BG
00102    real (ids_real) :: C(1:7)
00103
00104    fun_err%ierr = 0
00105    fun_err%cerr = ”
00106
00107    if(size(function_parameters,1).ne.11) then
00108       write(fun_err%cerr,*) "nuclear_data_1002: Expected the length of ’function_parameters’ to be 11
       in ’nuclear_data_1002’ and not ", &
00109            size(function_parameters)
00110       fun_err%ierr = -1
00111       return
00112    endif
00113    if(size(tin).ne.size(f)) then
00114       write(fun_err%cerr,*) "nuclear_data_1002: Expected the input and output vectors to be the same
       size in ’nuclear_data_1002’"
00115       fun_err%ierr = -1
00116       return
00117    endif
00118
00119    do it=1, size(tin)
00120       t=tin(it)*1e-3 ! temperature in keV
00121       ir=1
00122       if(t.lt.function_parameters(10,ir) .and. with_warnings) then
00123          write(*,*) ’extrapolating  below the desired range’
00124       endif
00125       do while(t.gt.function_parameters(11,ir).and. &
00126            ir.lt.size(function_parameters,2))
00127          ir=ir+1
00128       enddo
00129       if(t.gt.function_parameters(11,ir) .and. with_warnings) then
00130          write(*,*) ’extrapolating  above the desired range’
00131 ! dpc fix: changed 12 to 11
00132          write(*,*) ir, tin(it), function_parameters(11,ir)
00133       endif
00134
00135       bg  =function_parameters(1,   ir)
00136       mrc2=function_parameters(2,   ir)
00137       c   =function_parameters(3:9, ir)
00138       ! (/Tmin,Tmax/) = function_parameters(10:11, iR)
00139
00140       theta=t/(1-t*(c(2)+t*(c(4)+t*c(6)))&
00141           /(1+t*(c(3)+t*(c(5)+t*c(7)))))
00142       xi=(bg**2/(4*theta))**(1.0_ids_real/3)
00143       f(it) = c(1)*theta*sqrt(xi/(mrc2*t**3))&
00144           *exp(-3*xi)*1.e-6 ! from cm^3/s to m^3s/
00145    end do
00146
00147  end subroutine nuclear_data_1002
00148
00156  subroutine rct_data_1003(function_parameters, Tin, f, with_warnings, fun_err)
00157    implicit none
00158    real (ids_real), intent(in)  :: function_parameters(:), Tin(:)
00159    real (ids_real), intent(out) :: f(:)
00160    logical,           intent(in)  :: with_warnings
00161    type (fun_err_t),  intent(out) :: fun_err
00162    integer                :: iT
```

```
00163
00164     fun_err%ierr = 0
00165     fun_err%cerr = ”
00166
00167     if(size(function_parameters,1).ne.2) then
00168        write(fun_err%cerr,*) "rct_data_1003: Expected the length of ’function_parameters’ to be 2 in
          ’rct_data_1003’ and not ", &
00169             size(function_parameters)
00170        fun_err%ierr = -1
00171        return
00172     endif
00173     if(size(tin).ne.size(f)) then
00174        write(fun_err%cerr,*) "rct_data_1003: Expected the input and output vectors to be the same size
          in ’rct_data_1003’"
00175        fun_err%ierr = -1
00176        return
00177     endif
00178
00179     do it=1, size(tin)
00180        f(it) = function_parameters(1)*(1.0_ids_real + function_parameters(2)*
          log(1.0_ids_real/tin(it)))**2*1e-20_ids_real
00181     end do
00182
00183  end subroutine rct_data_1003
00184
00192  subroutine sputter_data_1004(function_parameters, energy_arr, angle_arr, yield_arr, with_warnings,
          fun_err)
00193     use eckstein_yields
00194     implicit none
00195     real (ids_real), intent(in)  :: function_parameters(:,:), energy_arr(:), angle_arr(:)
00196     real (ids_real), intent(out) :: yield_arr(:)
00197     logical,           intent(in)  :: with_warnings
00198     type (fun_err_t),  intent(out) :: fun_err
00199
00200     real (ids_real)              :: energy, angle, yield
00201     real(ids_real)               :: matchanglee0, matchanglee1
00202     real(ids_real)               :: M1, M2, Z1, Z2, q, lambda, mu, ETh, f,b,c,esp
00203     integer                 :: numtab, ianglee0, numpars, boundianglee0, ipars
00204     integer                 :: iD, iA
00205     logical, parameter      :: debug=.true.
00206
00207     integer                 :: have_angle_data
00208
00209     fun_err%ierr = 0
00210     fun_err%cerr = ”
00211
00212     numtab = ubound(function_parameters, 2)
00213     if(debug .and. with_warnings) then
00214        write(*,*) numtab
00215     endif
00216     do id = lbound(energy_arr,1), ubound(energy_arr,1)
00217
00218        energy = energy_arr(id)
00219        angle  = angle_arr(id)
00220
00221        if(debug .and. with_warnings) then
00222           write(*,*) energy, angle
00223        endif
00224
00225        matchanglee0 = 0.0
00226        boundianglee0 = -1
00227        have_angle_data = -1
00228        do ia = lbound(function_parameters, 2), ubound(function_parameters, 2)
00229           matchanglee1 = function_parameters(3, ia)
00230           if(debug .and. with_warnings) then
00231              write(*,*) matchanglee0, energy, matchanglee1
00232           endif
00233           if((matchanglee0.le.energy).and.(matchanglee1.ge.energy)) then
00234              boundianglee0 = ia
00235              have_angle_data = 1
00236              exit
00237           end if
00238           if (matchanglee1 < matchanglee0)  then
00239              if (matchanglee1 < matchanglee0)  then
00240                 if(debug .and. with_warnings) then
00241                    write(*, *) ’WARNING::sputteryield-> no angular reflection yield data available,
                   returning value for perpendicular impact’
00242                 endif
00243                 have_angle_data = -1
00244                 exit
00245              else
00246                 write(fun_err%cerr,*) ’sputter_data_1004: sputteryield-> angular dependence energies
                   are not sorted, rebuild the database’
00247                 fun_err%ierr = -1
00248                 return
00249              endif
00250           end if
```

```
00251          matchanglee0 = matchanglee1
00252       end do
00253
00254       if(boundianglee0.lt.0) then
00255          if (debug .and. with_warnings) then
00256             write(*,*) 'sputter_data_1004: No Bound found for: ', energy, ' angle: ', angle, 'using
    boundianglee0 = ', numtab
00257          end if
00258          boundianglee0 = ubound(function_parameters, 2)
00259       end if
00260
00261       m1     = function_parameters( 4, boundianglee0)
00262       m2     = function_parameters( 5, boundianglee0)
00263       z1     = function_parameters( 1, boundianglee0)
00264       z2     = function_parameters( 2, boundianglee0)
00265       q      = function_parameters( 6, boundianglee0)
00266       lambda = function_parameters( 9, boundianglee0)
00267       eth    = function_parameters( 8, boundianglee0)
00268       mu     = function_parameters( 7, boundianglee0)
00269       f      = function_parameters(10, boundianglee0)
00270       b      = function_parameters(11, boundianglee0)
00271       c      = function_parameters(12, boundianglee0)
00272       esp    = function_parameters(13, boundianglee0)
00273
00274       yield = 0.0
00275       if (energy .gt. eth) then
00276          if (have_angle_data .gt. 0) then
00277             yield = seyield(energy, m1, m2, z1, z2, q, lambda, mu, eth) * sayield(energy, angle, f,
    b, c, esp)
00278          else
00279             yield = seyield(energy, m1, m2, z1, z2, q, lambda, mu, eth)
00280          endif
00281       end if
00282
00283       yield_arr(id) = yield
00284
00285    enddo
00286
00287  end subroutine sputter_data_1004
00288
00296  subroutine reflect_data_1005(function_parameters, energy_arr, angle_arr, refl_arr, with_warnings,
    fun_err)
00297    use eckstein_yields
00298    implicit none
00299    real (ids_real), intent(in)  :: function_parameters(:,:), energy_arr(:), angle_arr(:)
00300    real (ids_real), intent(out) :: refl_arr(:)
00301    logical,           intent(in)  :: with_warnings
00302    type (fun_err_t),  intent(out) :: fun_err
00303
00304    real (ids_real)              :: energy, angle, refl
00305    real(ids_real)               :: M1, M2, Z1, Z2, A1, A2, A3, A4, E1, ESP, ANGLEE0, ANGLEE1, C1, C2,
    C3, C4
00306    real(ids_real)               :: C1FROM, C2FROM, C3FROM, C4FROM
00307    real(ids_real)               :: C1TO, C2TO, C3TO, C4TO
00308    real(ids_real)               :: matchanglee0, matchanglee1
00309    integer                :: numtab, ianglee0, numpars, boundianglee0, boundianglee1, ipars
00310    integer                :: iD, iA
00311    logical, parameter     :: debug=.true.
00312
00313    integer                :: have_angle_data
00314
00315    fun_err%ierr = 0
00316    fun_err%cerr = ''
00317
00318    numtab = ubound(function_parameters, 2)
00319    do id = lbound(energy_arr,1), ubound(energy_arr,1)
00320
00321       energy = energy_arr(id)
00322       angle  = angle_arr(id)
00323
00324       matchanglee0 = 0.0
00325       boundianglee0 = -1
00326       boundianglee1 = -1
00327       have_angle_data = -1
00328       do ia = lbound(function_parameters, 2), ubound(function_parameters, 2)
00329          matchanglee1 = function_parameters(11, ia)
00330          if(debug .and. with_warnings) then
00331             write(*,*) matchanglee0, energy, matchanglee1
00332          endif
00333          if((matchanglee0.le.energy).and.(matchanglee1.ge.energy)) then
00334             if (ia .gt. 1) then
00335                if(ia .lt. numtab) then
00336                   boundianglee0 = ia - 1
00337                   boundianglee1 = ia
00338                else
00339                   boundianglee0 = numtab
00340                   boundianglee1 = numtab
```

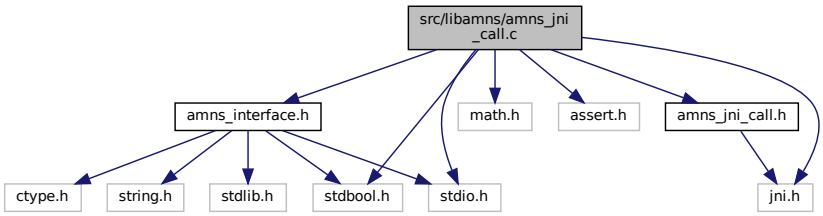
```
00341                end if
00342             else
00343                boundianglee0 = 1
00344                boundianglee1 = 1
00345             end if
00346             have_angle_data = 1
00347             exit
00348          end if
00349          if (matchanglee1 < matchanglee0)  then
00350             if (matchanglee1 .lt. 0) then
00351                if(debug .and. with_warnings) then
00352                   write(*,*) 'WARNING::reflyield-> no angular reflection yield data available,
       returning value for perpendicular impact'
00353                endif
00354                have_angle_data = -1
00355                exit
00356             else
00357                write(fun_err%cerr,*) 'reflect_data_1005: reflyield-> angular dependence energies are
       not sorted, rebuild the database'
00358                fun_err%ierr = -1
00359                return
00360             end if
00361          endif
00362          matchanglee0 = matchanglee1
00363       end do
00364
00365       if((boundianglee0.lt.0).or.(boundianglee1.lt.0)) then
00366          if (matchanglee1.lt.energy) then
00367             boundianglee0 = numtab
00368             boundianglee1 = numtab
00369          else
00370             boundianglee0 = 1
00371             boundianglee1 = 1
00372          end if
00373          if (debug .and. with_warnings) then
00374             write(*,*) 'reflect_data_1005: No Bound found for: ', energy, ' angle: ', angle, 'using
       boundianglee0 = ', boundianglee0, 'and boundianglee1 = ',boundianglee1
00375          end if
00376       end if
00377
00378       z1  = function_parameters( 1, boundianglee0)
00379       z2  = function_parameters( 2, boundianglee0)
00380       m1  = function_parameters( 3, boundianglee0)
00381       m2  = function_parameters( 4, boundianglee0)
00382       a1  = function_parameters( 5, boundianglee0)
00383       a2  = function_parameters( 6, boundianglee0)
00384       a3  = function_parameters( 7, boundianglee0)
00385       a4  = function_parameters( 8, boundianglee0)
00386       e1  = function_parameters( 9, boundianglee0)
00387       esp = function_parameters(10, boundianglee0)
00388
00389       if ((angle.lt.1.0).or.(have_angle_data.eq.-1)) then
00390          if (debug .and. with_warnings) then
00391             write(*,*) 'angle = ',angle, ' is small or no angular data available: using perpendicular
       impact model with energy dependence'
00392          end if
00393          refl = reyield(energy, m1, m2, z1, z2, a1, a2, a3, a4)
00394       else
00395          if (debug .and. with_warnings) then
00396             write(*,*) 'angle = ',angle, ' is large using angular dependent model and interpolation
       of energy dependence'
00397          end if
00398          ! Interplolate the bounding table entries
00399          if(boundianglee0.lt.boundianglee1) then
00400             anglee0 = function_parameters(11, boundianglee0)
00401             anglee1 = function_parameters(11, boundianglee1)
00402             if(debug .and. with_warnings) then
00403                write(*,*) 'Interpolating angular dependent reflection yield for energy' ,energy,'
       between values for energies ', anglee0, ' and ', anglee1
00404             end if
00405             c1from = function_parameters(12, boundianglee0)
00406             c2from = function_parameters(13, boundianglee0)
00407             c3from = function_parameters(14, boundianglee0)
00408             c4from = function_parameters(15, boundianglee0)
00409
00410             c1to = function_parameters(12, boundianglee1)
00411             c2to = function_parameters(13, boundianglee1)
00412             c3to = function_parameters(14, boundianglee1)
00413             c4to = function_parameters(15, boundianglee1)
00414
00415             c1 = c1from + ((energy - anglee0) * (c1to - c1from) / (anglee1 - anglee0))
00416             c2 = c2from + ((energy - anglee0) * (c2to - c2from) / (anglee1 - anglee0))
00417             c3 = c3from + ((energy - anglee0) * (c3to - c3from) / (anglee1 - anglee0))
00418             c4 = c4from + ((energy - anglee0) * (c4to - c4from) / (anglee1 - anglee0))
00419          else
00420             anglee0 = function_parameters(11, boundianglee0)
00421             if(debug .and. with_warnings) then
```

```
00422                write(*,*) 'Energy: ',energy,'  for requested angle: ',angle,' not within db bounds
      using anglular dependent value at energy = ', anglee0
00423             end if
00424             c1 = function_parameters(12, boundianglee0)
00425             c2 = function_parameters(13, boundianglee0)
00426             c3 = function_parameters(14, boundianglee0)
00427             c4 = function_parameters(15, boundianglee0)
00428          end if
00429          refl = rayield(angle, c1, c2, c3, c4)
00430       end if
00431
00432       refl_arr(id) = min(1.0_ids_real, refl)
00433
00434    enddo
00435
00436  end subroutine reflect_data_1005
00437
00443  subroutine nuclear_data_1006(function_parameters, x, f, with_warnings, fun_err)
00444    implicit none
00445    real (ids_real),   intent(in)  :: function_parameters(:,:), x(:)
00446    real (ids_real),   intent(out) :: f(:)
00447    logical,           intent(in)  :: with_warnings
00448    type (fun_err_t),  intent(out) :: fun_err
00449    integer :: i, j
00450    real (ids_real) :: e, s
00451
00452    fun_err%ierr = 0
00453    fun_err%cerr = ''
00454
00455    if(size(function_parameters,1).ne.14) then
00456       write(fun_err%cerr,*) "nuclear_data_1006: Expected the length of 'function_parameters' to be 14
      in 'nuclear_data_1006' and not ", &
00457            size(function_parameters)
00458       fun_err%ierr = -1
00459       return
00460    endif
00461    if(size(x).ne.size(f)) then
00462       write(fun_err%cerr,*) "nuclear_data_1006: Expected the input and output vectors to be the same
      size in 'nuclear_data_1006'"
00463       fun_err%ierr = -1
00464       return
00465    endif
00466
00467    do i=1, size(x)
00468       e = x(i) * 1.0e-3_ids_real
00469       j=1
00470       if(e.lt.function_parameters(13,j) .and. with_warnings) then
00471          write(*,*) 'extrapolating  below the desired range'
00472       endif
00473       do while(e.gt.function_parameters(14,j).and. &
00474            j.lt.size(function_parameters,2))
00475          j=j+1
00476       enddo
00477       if(e.gt.function_parameters(14,j)) then
00478          if (with_warnings) then
00479             write(*,*) 'extrapolating  above the desired range'
00480             write(*,*) j, x(i), function_parameters(12,j)
00481             write(*,*) 'Taking the boundary value when calculating S!'
00482          endif
00483          e = function_parameters(14,j)
00484       endif
00485       s = ( function_parameters(2,j) &
00486            & + e*(function_parameters(3,j) &
00487            & + e*(function_parameters(4,j) &
00488            & + e*(function_parameters(5,j) &
00489            & + e*(function_parameters(6,j) &
00490            & + e* function_parameters(7,j) ) ) ) ) ) &
00491            & /( 1.0_ids_real &
00492            & + e*(function_parameters(8,j) &
00493            & + e*(function_parameters(9,j) &
00494            & + e*(function_parameters(10,j) &
00495            & + e*(function_parameters(11,j) &
00496            & + e* function_parameters(12,j) ) ) ) ) )
00497       ! We clamp the energy for S (the nuclear fusion probability).
00498       ! However, there is no need to clamp the energy in denominator.
00499       ! The denominator roughly describes the probability of the reactants to
00500       ! overcome the Coulomb barrier.
00501       e = x(i) * 1.0e-3_ids_real
00502       f(i) = s / (e * exp(function_parameters(1,j)/sqrt(e))) * 1.0e-31_ids_real
00503    enddo
00504  end subroutine nuclear_data_1006
00505
00506 end module amns_external_functions
```

16.93 src/libamns/amns_jni_call.c File Reference

```
#include "amns_interface.h"
#include <stdbool.h>
#include <math.h>
#include <assert.h>
#include <stdio.h>
#include "amns_jni_call.h"
#include <jni.h>
```

Include dependency graph for amns_jni_call.c:



Functions

- void copyCError2JavaError (amns_c_error_type *error_stat, JNIEnv *env, jobject error_structure)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactants (JNIEnv *env, jobject obj, jint idx)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactantsNumber (JNIEnv *env, jobject obj, jint idx, jint number)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactant (JNIEnv *env, jobject obj, jlong ptr, jobject acrt)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactantIdx (JNIEnv *env, jobject obj, jlong ptr, jint idx, jobject acrt)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCGetReactant (JNIEnv *env, jobject obj, jlong ptr, jobject jSpecies)
- JNIEXPORT long JNICALL Java_amns_Amns_ImasAmnsCCSetup (JNIEnv *env, jobject obj, jobject error_structure)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetupTable (JNIEnv *env, jobject obj, jlong ptr_amns_handle, jobject j_reaction_type, jlong ptr_reactants_handle, jobject j_error_state)
- JNIEXPORT jdouble JNICALL Java_amns_Amns_ImasAmnsCCRX0B (JNIEnv *env, jobject obj, jlong ptrHandleRx, double j_arg1, double j_arg2, jobject j_error_state)
- JNIEXPORT long JNICALL Java_amns_Amns_ImasAmnsCCFinishTable (JNIEnv *env, jobject obj, jlong ptrHandleRX, jobject j_error_state)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishReactants (JNIEnv *env, jobject obj, jlong ptr_reactants_handle)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinish (JNIEnv *env, jobject obj, jlong ptr_amns_handle, jobject j_error_state)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSet (JNIEnv *env, jobject obj, jlong ptrAmnsHandle, jobject j_amns_set_type, jobject j_error_state)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQuery (JNIEnv *env, jobject obj, jlong ptrAmnsHandle, jobject j_amns_query, jobject j_amns_answer, jobject j_error_state)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQueryTable (JNIEnv *env, jobject obj, jlong ptrTableHandle, jobject j_amns_query, jobject j_amns_answer, jobject j_error_state)
- JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1A (JNIEnv *env, jobject obj, jlong ptrHandleRx, jint nx, jdoubleArray j_arg1, jobject j_error_state)

- JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1B (JNIEnv ∗env, jobject obj, jlong ptrHandleRx, jint nx, jdoubleArray j_arg1, jdoubleArray j_arg2, jobject j_error_state)

- JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1C (JNIEnv ∗env, jobject obj, jlong ptrHandleRx, jint nx, jdoubleArray j_arg1, jdoubleArray j_arg2, jdoubleArray j_arg3, jobject j_error_state)

- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetTable (JNIEnv ∗env, jobject obj, jlong ptr↩ AmnsCxHandle, jobject j_amns_set_type, jobject j_error_state)

16.93.1 Function Documentation

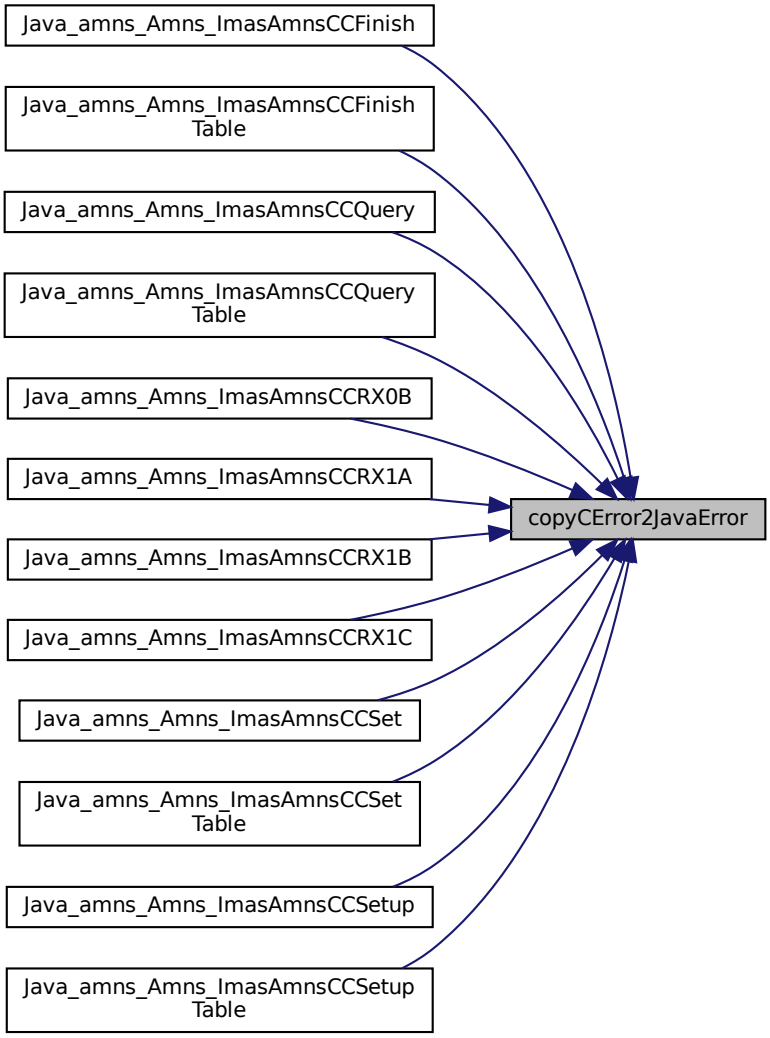
16.93.1.1 copyCError2JavaError()

```
void copyCError2JavaError (
            amns_c_error_type * error_stat,
            JNIEnv * env,
            jobject error_structure )
```

Definition at line 9 of file amns_jni_call.c.

```
00009                                                                                                                {
00010
00011   // we have to return values into Java
00012   // note that we are playing here with pointers - they should be preserved accross calls
00013   // but will not be garbage collected from java
00014
00015   jclass cls = (*env)->GetObjectClass(env, error_structure);
00016   jfieldID paramFlag = (*env)->GetFieldID(env, cls, "flag", "Z");
00017   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00018
00019   jstring ret_message;
00020
00021   if(error_stat->string != NULL) {
00022     ret_message = (*env)->NewStringUTF(env,error_stat->string);
00023   } else {
00024     ret_message = (*env)->NewStringUTF(env,"");
00025   }
00026   if(ret_message == NULL) {
00027     (*env)->SetObjectField(env, error_structure, paramString, "");
00028   } else {
00029     (*env)->SetObjectField(env, error_structure, paramString, ret_message);
00030   }
00031
00032   jboolean ret_flag = error_stat->flag;
00033   (*env)->SetBooleanField(env, error_structure, paramFlag, ret_flag);
00034 }
```

Here is the caller graph for this function:



16.93.1.2 Java_amns_Amns_ImasAmnsCCFinish()

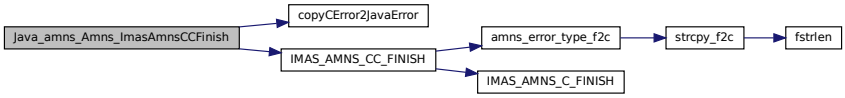
```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinish (
            JNIEnv * env,
            jobject obj,
            jlong ptr_amns_handle,
            jobject j_error_state )
```

Definition at line 284 of file amns_jni_call.c.

```
00288                                                                    {
00289
00290   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00291   void* amns_handle = (void*) ptr_amns_handle;
00292
00293   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH\n");
00294   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptr_amns_handle);
00295   IMAS_AMNS_CC_FINISH(&amns_handle, &error_stat);
00296   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH\n");
00297
00298   // we have to make sure that error_stat is passed back inside object
```

```
00299
00300   copyCError2JavaError( &error_stat, env, j_error_state);
00301
00302   return (long) amns_handle;
00303
00304 }
```

Here is the call graph for this function:



16.93.1.3 Java_amns_Amns_ImasAmnsCCFinishReactants()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishReactants (
            JNIEnv * env,
            jobject obj,
            jlong ptr_reactants_handle )
```

Definition at line 268 of file amns_jni_call.c.

```
00271                                                                                                    {
00272
00273   void* reactants_handle = (void *) ptr_reactants_handle;
00274
00275   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH_REACTANTS\n");
00276   // DEBUG printf("[JNI] reactants_handle: %lu\n", (long)ptr_reactants_handle);
00277   IMAS_AMNS_CC_FINISH_REACTANTS(&reactants_handle);
00278   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH_REACTANTS\n");
00279
00280   return (long) reactants_handle;
00281 }
```

Here is the call graph for this function:

16.93.1.4 Java_amns_Amns_ImasAmnsCCFinishTable()

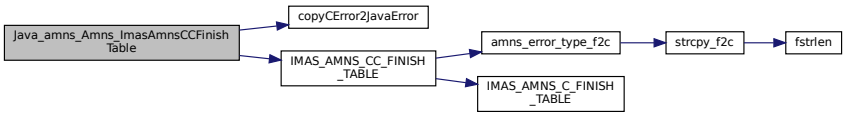
```
JNIEXPORT long JNICALL Java_amns_Amns_ImasAmnsCCFinishTable (
            JNIEnv * env,
            jobject obj,
            jlong ptrHandleRX,
            jobject j_error_state )
```

Definition at line 246 of file amns_jni_call.c.

```
00250                                                                                                {
00251
00252   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00253   void* amns_cx_handle = (void *) ptrHandleRX;
00254
00255   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH_TABLE\n");
00256   // DEBUG printf("[JNI] cx_handle: %lu\n", (long)ptrHandleRX);
00257   IMAS_AMNS_CC_FINISH_TABLE(&amns_cx_handle, &error_stat);
00258   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH_TABLE\n");
00259
00260   // we have to make sure that error_stat is passed back inside object
00261
00262   copyCError2JavaError( &error_stat, env, j_error_state );
00263
```

```
00264   return (jlong) amns_cx_handle;
00265
00266 }
```

Here is the call graph for this function:



16.93.1.5 Java_amns_Amns_ImasAmnsCCGetReactant()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCGetReactant (
            JNIEnv * env,
            jobject obj,
            jlong ptr,
            jobject jSpecies )
```

Definition at line 116 of file amns_jni_call.c.

```
00116
              {
00117
00118   void *reactant_handle = (void*) ptr;
00119
00120   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00121
00122   // at first, get all elements of the species
00123   jclass cls = (*env)->GetObjectClass(env, jSpecies);
00124   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00125   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00126   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00127   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00128   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00129   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00130
00131   // retrieve data into C structure
00132   IMAS_AMNS_CC_GET_REACTANT(reactant_handle, 1, &species);
00133   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_GET_REACTANTS\n");
00134
00135   // copy data back to Java
00136   (*env)->SetDoubleField(env, jSpecies, paramZN, (jdouble)species.ZN );
00137   (*env)->SetDoubleField(env, jSpecies, paramZA, species.ZA );
00138   (*env)->SetDoubleField(env, jSpecies, paramMI, species.MI );
00139   (*env)->SetIntField(env, jSpecies, paramLR, species.LR );
00140   (*env)->SetDoubleField(env, jSpecies, paramRealSpecifier, species.real_specifier);
00141   (*env)->SetIntField(env, jSpecies, paramIntSpecifier, species.int_specifier);
00142 }
```

Here is the call graph for this function:

16.93.1.6 Java_amns_Amns_ImasAmnsCCQuery()

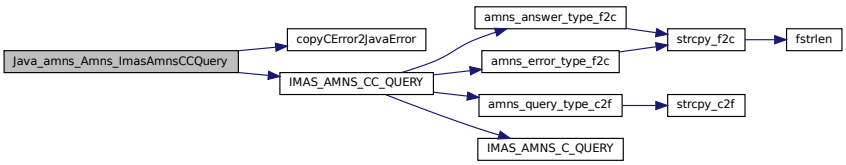
```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQuery (
            JNIEnv * env,
            jobject obj,
            jlong ptrAmnsHandle,
            jobject j_amns_query,
```

```
            jobject j_amns_answer,
            jobject j_error_state )
```

Definition at line 337 of file amns_jni_call.c.

```
00343                                                                                {
00344
00345   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00346   amns_c_query_type query = DEFAULT_AMNS_C_QUERY_TYPE;
00347   amns_c_answer_type answer = DEFAULT_AMNS_C_ANSWER_TYPE;
00348   void *amns_handler = (void *) ptrAmnsHandle;
00349
00350   // we have to retrieve the query and pass it to function
00351
00352   jclass cls = (*env)->GetObjectClass(env, j_amns_query);
00353   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00354
00355   jstring j_query_string = (*env)->GetObjectField(env, j_amns_query, paramString);
00356
00357   char *query_string = (char*)(*env)->GetStringUTFChars(env, j_query_string, NULL);
00358
00359   query.string = query_string;
00360
00361   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_QUERY\n");
00362   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptrAmnsHandle);
00363   IMAS_AMNS_CC_QUERY(amns_handler, &query, &answer, &error_stat);
00364   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_QUERY\n");
00365
00366   // copy error to Java
00367   copyCError2JavaError( &error_stat, env, j_error_state);
00368
00369   // copy answer to Java
00370
00371   jclass cls_answer = (*env)->GetObjectClass(env, j_amns_answer);
00372   jfieldID answerNumber = (*env)->GetFieldID(env, cls_answer, "number", "I");
00373   jfieldID answerString = (*env)->GetFieldID(env, cls_answer, "string", "Ljava/lang/String;");
00374
00375   jstring answer_value;
00376
00377   if(answer.string != NULL) {
00378     answer_value = (*env)->NewStringUTF(env,answer.string);
00379   } else {
00380     answer_value = (*env)->NewStringUTF(env,"");
00381   }
00382   if(answer_value == NULL) {
00383     (*env)->SetObjectField(env, j_amns_answer, answerString, "");
00384   } else {
00385     (*env)->SetObjectField(env, j_amns_answer, answerString, answer_value);
00386   }
00387
00388   jint answer_number = answer.number;
00389   (*env)->SetIntField(env, j_amns_answer, answerNumber, answer_number);
00390 }
```

Here is the call graph for this function:



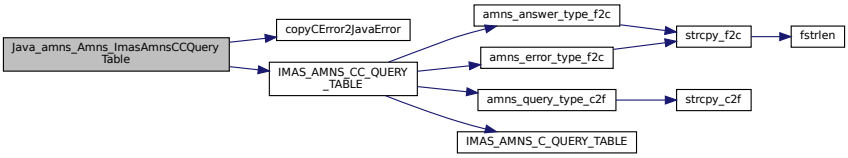
16.93.1.7 Java_amns_Amns_ImasAmnsCCQueryTable()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQueryTable (
            JNIEnv * env,
            jobject obj,
            jlong ptrTableHandle,
            jobject j_amns_query,
            jobject j_amns_answer,
            jobject j_error_state )
```

Definition at line 392 of file amns_jni_call.c.

```
00398                                                                                {
00399
00400   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00401   amns_c_query_type query = DEFAULT_AMNS_C_QUERY_TYPE;
00402   amns_c_answer_type answer = DEFAULT_AMNS_C_ANSWER_TYPE;
00403   void *table_handler = (void *) ptrTableHandle;
00404
00405   // we have to retrieve the query and pass it to function
00406
00407   jclass cls = (*env)->GetObjectClass(env, j_amns_query);
00408   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00409
00410   jstring j_query_string = (*env)->GetObjectField(env, j_amns_query, paramString);
00411
00412   char *query_string = (char*)(*env)->GetStringUTFChars(env, j_query_string, NULL);
00413
00414   query.string = query_string;
00415
00416   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_QUERY_TABLE\n");
00417   // DEBUG printf("[JNI] table_handle: %lu\n", (long)ptrTableHandle);
00418   IMAS_AMNS_CC_QUERY_TABLE(table_handler, &query, &answer, &error_stat);
00419   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_QUERY_TABLE\n");
00420
00421   // copy error to Java
00422   copyCError2JavaError( &error_stat, env, j_error_state);
00423
00424   // copy answer to Java
00425
00426   jclass cls_answer = (*env)->GetObjectClass(env, j_amns_answer);
00427   jfieldID answerNumber = (*env)->GetFieldID(env, cls_answer, "number", "I");
00428   jfieldID answerString = (*env)->GetFieldID(env, cls_answer, "string", "Ljava/lang/String;");
00429
00430   jstring answer_value;
00431
00432   if(answer.string != NULL) {
00433     answer_value = (*env)->NewStringUTF(env,answer.string);
00434   } else {
00435     answer_value = (*env)->NewStringUTF(env,"");
00436   }
00437   if(answer_value == NULL) {
00438     (*env)->SetObjectField(env, j_amns_answer, answerString, "");
00439   } else {
00440     (*env)->SetObjectField(env, j_amns_answer, answerString, answer_value);
00441   }
00442
00443   jint answer_number = answer.number;
00444   (*env)->SetIntField(env, j_amns_answer, answerNumber, answer_number);
00445 }
```

Here is the call graph for this function:



16.93.1.8 Java_amns_Amns_ImasAmnsCCRX0B()

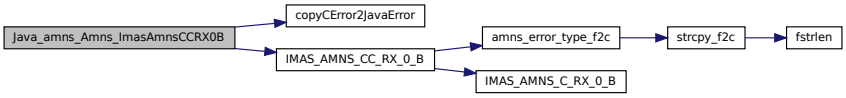
```
JNIEXPORT jdouble JNICALL Java_amns_Amns_ImasAmnsCCRX0B (
            JNIEnv * env,
            jobject obj,
            jlong ptrHandleRx,
            double j_arg1,
            double j_arg2,
            jobject j_error_state )
```

Definition at line 218 of file amns_jni_call.c.

```
00224                                                                                {
00225
00226   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00227   void* amns_cx_handle = (void *) ptrHandleRx;
```

```
00228   double arg1 = (double) j_arg1;
00229   double arg2 = (double) j_arg2;
00230   double rate;
00231
00232   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_0_B\n");
00233   // DEBUG printf("[JNI] cx_handle: %x, arg1: %e, arg2: %e\n", ptrHandleRx, arg1, arg2);
00234   IMAS_AMNS_CC_RX_0_B(amns_cx_handle, &rate, arg1, arg2, &error_stat);
00235   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_0_B\n");
00236   // DEBUG printf("[JNI] rate: %e\n", rate);
00237
00238   // we have to make sure that error_stat is passed back inside object
00239
00240   copyCError2JavaError( &error_stat, env, j_error_state);
00241
00242   return (jdouble) rate;
00243
00244 }
```

Here is the call graph for this function:



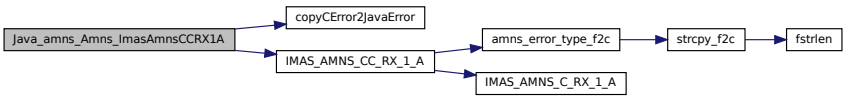
16.93.1.9 Java_amns_Amns_ImasAmnsCCRX1A()

```
JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1A (
            JNIEnv * env,
            jobject obj,
            jlong ptrHandleRx,
            jint nx,
            jdoubleArray j_arg1,
            jobject j_error_state )
```

Definition at line 447 of file amns_jni_call.c.

```
00453                                                                                {
00454
00455   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00456   void *handle_rx_in = (void *) ptrHandleRx;
00457
00458   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00459   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00460
00461   double *outputArray = (double *) malloc(nx * sizeof(double));
00462
00463   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_A\n");
00464   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00465   IMAS_AMNS_CC_RX_1_A(handle_rx_in, nx, outputArray, arg1, &error_stat);
00466   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_A\n");
00467
00468   // copy error to Java
00469   copyCError2JavaError( &error_stat, env, j_error_state);
00470
00471   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00472   if (NULL == outJNIArray) {
00473     return NULL;
00474   }
00475   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00476
00477   return outJNIArray;
00478 }
```

Here is the call graph for this function:



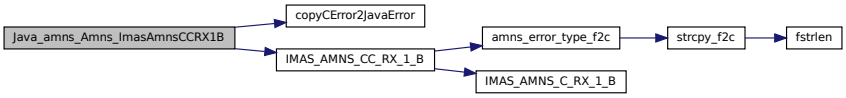
16.93.1.10 Java_amns_Amns_ImasAmnsCCRX1B()

```
JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1B (
            JNIEnv * env,
            jobject obj,
            jlong ptrHandleRx,
            jint nx,
            jdoubleArray j_arg1,
            jdoubleArray j_arg2,
            jobject j_error_state )
```

Definition at line 480 of file amns_jni_call.c.

```
00487                                                                                {
00488
00489   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00490   void *handle_rx_in = (void *) ptrHandleRx;
00491
00492   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00493   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00494
00495   jsize len_arr2 = (*env)->GetArrayLength(env, j_arg2);
00496   jdouble *arg2 = (*env)->GetDoubleArrayElements(env, j_arg2, 0);
00497
00498   double *outputArray = (double *) malloc(nx * sizeof(double));
00499
00500   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_B\n");
00501   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00502
00503   /*printf("--- JNI CODE - arg1 - start ---\n");
00504     int ii=0;
00505     for(ii=0; ii<nx; ii++ ) {
00506     printf("%1.6f ", arg1[ii]);
00507     }
00508     printf("\n");
00509     printf("--- JNI CODE - end ---\n");
00510     fflush(stdout);
00511     printf("--- JNI CODE - arg2 - start ---\n");
00512     for(ii=0; ii<nx; ii++ ) {
00513     printf("%1.6f ", arg2[ii]);
00514     }
00515     printf("\n");
00516     printf("--- JNI CODE - end ---\n");
00517     fflush(stdout);*/
00518
00519   IMAS_AMNS_CC_RX_1_B(handle_rx_in, nx, outputArray, arg1, arg2, &error_stat);
00520   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_B\n");
00521
00522   // copy error to Java
00523   copyCError2JavaError( &error_stat, env, j_error_state);
00524
00525   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00526   if (NULL == outJNIArray) {
00527     return NULL;
00528   }
00529   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00530
00531   return outJNIArray;
00532
00533 }
```

Here is the call graph for this function:



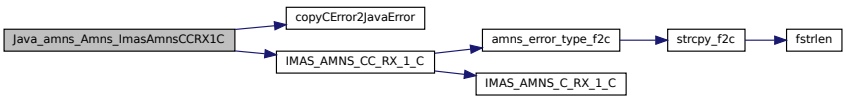
16.93.1.11 Java_amns_Amns_ImasAmnsCCRX1C()

```
JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1C (
            JNIEnv * env,
            jobject obj,
            jlong ptrHandleRx,
            jint nx,
            jdoubleArray j_arg1,
            jdoubleArray j_arg2,
            jdoubleArray j_arg3,
            jobject j_error_state )
```

Definition at line 535 of file amns_jni_call.c.

```
00543                                                                                {
00544
00545   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00546   void *handle_rx_in = (void *) ptrHandleRx;
00547
00548
00549   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00550   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00551
00552   jsize len_arr2 = (*env)->GetArrayLength(env, j_arg2);
00553   jdouble *arg2 = (*env)->GetDoubleArrayElements(env, j_arg2, 0);
00554
00555   jsize len_arr3 = (*env)->GetArrayLength(env, j_arg3);
00556   jdouble *arg3 = (*env)->GetDoubleArrayElements(env, j_arg3, 0);
00557
00558   double *outputArray = (double *) malloc(nx * sizeof(double));
00559
00560   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_C\n");
00561   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00562   IMAS_AMNS_CC_RX_1_C(handle_rx_in, nx, outputArray, arg1, arg2, arg3, &error_stat);
00563   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_C\n");
00564
00565   // copy error to Java
00566   copyCError2JavaError( &error_stat, env, j_error_state);
00567
00568   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00569   if (NULL == outJNIArray) {
00570     return NULL;
00571   }
00572   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00573
00574   return outJNIArray;
00575
00576 }
```

Here is the call graph for this function:



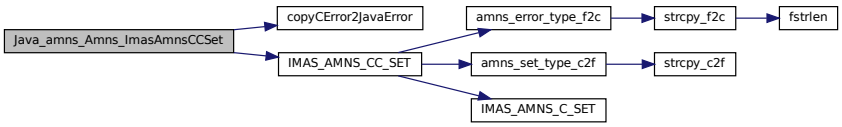
16.93.1.12 Java_amns_Amns_ImasAmnsCCSet()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSet (
            JNIEnv * env,
            jobject obj,
            jlong ptrAmnsHandle,
            jobject j_amns_set_type,
            jobject j_error_state )
```

Definition at line 306 of file amns_jni_call.c.

```
00311                                                                    {
00312
00313   amns_c_set_type set = DEFAULT_AMNS_C_SET_TYPE;
00314   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00315   void *amns_handler = (void *) ptrAmnsHandle;
00316
00317   jclass cls = (*env)->GetObjectClass(env, j_amns_set_type);
00318   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00319
00320   jstring j_set_type_string = (*env)->GetObjectField(env, j_amns_set_type, paramString);
00321
00322   char *set_type_string = (char*)(*env)->GetStringUTFChars(env, j_set_type_string, NULL);
00323
00324   set.string = set_type_string;
00325
00326   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SET\n");
00327   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptrAmnsHandle);
00328   IMAS_AMNS_CC_SET(amns_handler, &set, &error_stat);
00329   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET\n");
00330
00331   copyCError2JavaError( &error_stat, env, j_error_state);
00332
00333   (*env)->ReleaseStringUTFChars(env, j_set_type_string, set_type_string);
00334
00335 }
```

Here is the call graph for this function:



16.93.1.13 Java_amns_Amns_ImasAmnsCCSetReactant()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactant (
            JNIEnv * env,
            jobject obj,
            jlong ptr,
            jobject acrt )
```

Definition at line 54 of file amns_jni_call.c.

```
00054
            {
00055
00056   void *reactant_handle = (void*) ptr;
00057
00058   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00059
00060   jclass cls = (*env)->GetObjectClass(env, acrt);
00061   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00062   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00063   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00064   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00065   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00066   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00067
00068   jdouble zn = (*env)->GetDoubleField(env, acrt, paramZN);
00069   jdouble za = (*env)->GetDoubleField(env, acrt, paramZA);
00070   jdouble mi = (*env)->GetDoubleField(env, acrt, paramMI);
00071   jint lr = (*env)->GetIntField(env, acrt, paramLR);
```

```
00072   jdouble rs = (*env)->GetDoubleField(env, acrt, paramRealSpecifier);
00073   jint int_spec = (*env)->GetIntField(env, acrt, paramIntSpecifier);
00074
00075   // DEBUG printf("[JNI] Values from Java: %f:%f:%f:%i:%f:%i\n", zn, za, mi, lr, rs, int_spec);
00076
00077   species.ZN=zn; species.ZA=za; species.MI=mi; species.LR=lr; species.real_specifier=rs;
      species.int_specifier=int_spec;
00078   IMAS_AMNS_CC_SET_REACTANT(reactant_handle, 1, &species);
00079   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_REACTANT\n");
00080 }
```

Here is the call graph for this function:

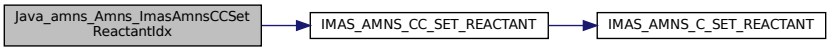
16.93.1.14 Java_amns_Amns_ImasAmnsCCSetReactantIdx()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactantIdx (
            JNIEnv * env,
            jobject obj,
            jlong ptr,
            jint idx,
            jobject acrt )
```

Definition at line 82 of file amns_jni_call.c.

```
00087                                                                                {
00088
00089   void *reactant_handle = (void*) ptr;
00090
00091   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00092
00093   jclass cls = (*env)->GetObjectClass(env, acrt);
00094   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00095   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00096   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00097   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00098   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00099   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00100
00101   jdouble zn = (*env)->GetDoubleField(env, acrt, paramZN);
00102   jdouble za = (*env)->GetDoubleField(env, acrt, paramZA);
00103   jdouble mi = (*env)->GetDoubleField(env, acrt, paramMI);
00104   jint lr = (*env)->GetIntField(env, acrt, paramLR);
00105   jdouble rs = (*env)->GetDoubleField(env, acrt, paramRealSpecifier);
00106   jint int_spec = (*env)->GetIntField(env, acrt, paramIntSpecifier);
00107
00108   // DEBUG printf("[JNI] Values from Java: %f:%f:%f:%i:%f:%i\n", zn, za, mi, lr, rs, int_spec);
00109
00110   species.ZN=zn; species.ZA=za; species.MI=mi; species.LR=lr; species.real_specifier=rs;
      species.int_specifier=int_spec;
00111   // DEBUG printf("[JNI] Calling IMAS_AMNS_CC_SET_REACTANT with index: %d\n", idx);
00112   IMAS_AMNS_CC_SET_REACTANT(reactant_handle, idx, &species);
00113   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_REACTANT\n");
00114 }
```

Here is the call graph for this function:



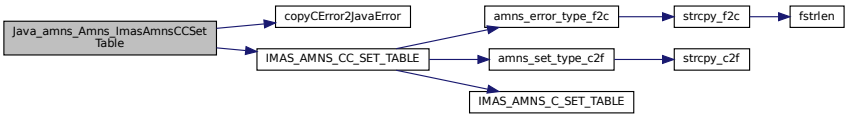
16.93.1.15 Java_amns_Amns_ImasAmnsCCSetTable()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetTable (
            JNIEnv * env,
            jobject obj,
            jlong ptrAmnsCxHandle,
            jobject j_amns_set_type,
            jobject j_error_state )
```

Definition at line 578 of file amns_jni_call.c.

```
00583                                                                                {
00584   amns_c_set_type set = DEFAULT_AMNS_C_SET_TYPE;
00585   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00586   void *amns_cx_handler = (void *) ptrAmnsCxHandle;
00587
00588   jclass cls = (*env)->GetObjectClass(env, j_amns_set_type);
00589   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00590
00591   jstring j_set_type_string = (*env)->GetObjectField(env, j_amns_set_type, paramString);
00592
00593   char *set_type_string = (char*)(*env)->GetStringUTFChars(env, j_set_type_string, NULL);
00594
00595   set.string = set_type_string;
00596
00597   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SET_TABLE\n");
00598   // DEBUG printf("[JNI] amns_cx_handle: %lu\n", (long)ptrAmnsCxHandle);
00599   IMAS_AMNS_CC_SET_TABLE(amns_cx_handler, &set, &error_stat);
00600   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_TABLE\n");
00601
00602   copyCError2JavaError( &error_stat, env, j_error_state);
00603
00604   (*env)->ReleaseStringUTFChars(env, j_set_type_string, set_type_string);
00605 }
```

Here is the call graph for this function:



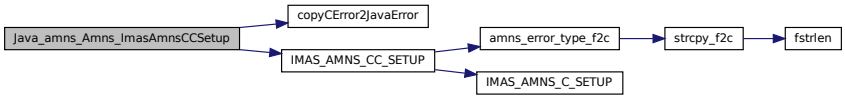
16.93.1.16 Java_amns_Amns_ImasAmnsCCSetup()

```
JNIEXPORT long JNICALL Java_amns_Amns_ImasAmnsCCSetup (
            JNIEnv * env,
            jobject obj,
            jobject error_structure )
```

Definition at line 152 of file amns_jni_call.c.

```
00152
        {
00153   void* amns_handle = NULL;
00154   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00155
00156   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SETUP\n");
00157   IMAS_AMNS_CC_SETUP(&amns_handle, &error_stat);
00158   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP\n");
00159   // DEBUG printf("[JNI] Pointer: %lu\n", (long)amns_handle);
00160
00161   copyCError2JavaError( &error_stat, env, error_structure );
00162
00163   return (long) amns_handle;
00164 }
```

Here is the call graph for this function:



16.93.1.17 Java_amns_Amns_ImasAmnsCcSetupReactants()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactants (
            JNIEnv * env,
            jobject obj,
            jint idx )
```

Definition at line 36 of file amns_jni_call.c.

```
00036                                                                                                {
00037   void *reactant_handle;
00038   IMAS_AMNS_CC_SETUP_REACTANTS(&reactant_handle, "", idx, 1);
00039   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_REACTANTS\n");
00040   // DEBUG printf("[JNI] Pointer: %lu\n", (long)reactant_handle);
00041   return (long)reactant_handle;
00042 }
```

Here is the call graph for this function:

16.93.1.18 Java_amns_Amns_ImasAmnsCcSetupReactantsNumber()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactantsNumber (
            JNIEnv * env,
            jobject obj,
            jint idx,
            jint number )
```

Definition at line 44 of file amns_jni_call.c.

```
00044
                    {
00045   void *reactant_handle;
00046   // DEBUG printf("[JNI] Calling IMAS_AMNS_CC_SETUP_REACTANTS with number of reactions: %d\n",
        number);
00047   IMAS_AMNS_CC_SETUP_REACTANTS(&reactant_handle, "", idx, number);
00048   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_REACTANTS\n");
00049   // DEBUG printf("[JNI] Pointer: %lu\n", (long)reactant_handle);
00050   return (long)reactant_handle;
00051 }
```

Here is the call graph for this function:

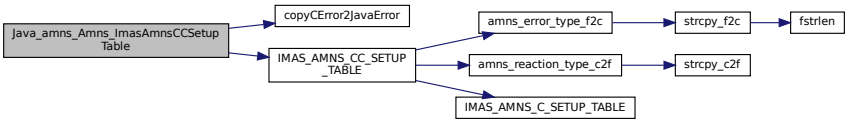
16.93.1.19 Java_amns_Amns_ImasAmnsCCSetupTable()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetupTable (
            JNIEnv * env,
            jobject obj,
            jlong ptr_amns_handle,
            jobject j_reaction_type,
            jlong ptr_reactants_handle,
            jobject j_error_state )
```

Definition at line 166 of file amns_jni_call.c.

```
00172                                                                                {
00173   // declare structures used in call
00174   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00175   void* amns_cx_handle;
00176   amns_c_reaction_type xx_rx;
00177
00178   // get pointers
00179   void *amns_handle = (void*) ptr_amns_handle;
00180   void *reactants_handle = (void*) ptr_reactants_handle;
00181
00182   // retrieve and copy values from reaction_type object
00183   jclass cls = (*env)->GetObjectClass(env, j_reaction_type);
00184   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00185   jfieldID paramIsotope = (*env)->GetFieldID(env, cls, "isotopeResolved", "I");
00186
00187   jstring j_reaction_string = (*env)->GetObjectField(env, j_reaction_type, paramString);
00188   jint j_reaction_isotope = (*env)->GetIntField(env, j_reaction_type, paramIsotope);
00189
00190   char *reaction_string = (char*) (*env)->GetStringUTFChars(env, j_reaction_string, NULL);
00191   int reaction_isotope = (int) j_reaction_isotope;
00192
00193   xx_rx.string = reaction_string;
00194   xx_rx.isotope_resolved = reaction_isotope;
00195
00196   // DEBUG printf("[JNI] Reaction type = %s, %u\n", xx_rx.string, xx_rx.isotope_resolved);
00197
00198   // we can now call C function
00199   // note that we will return amns_cx_handle as return value from function
00200   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SETUP_TABLE\n");
00201   // DEBUG printf("[JNI] amns_handle pointer: %lu\n", (long)amns_handle);
00202   // DEBUG printf("[JNI] reactants_handle pointer: %lu\n", (long)reactants_handle);
00203   IMAS_AMNS_CC_SETUP_TABLE(amns_handle, &xx_rx, reactants_handle, &amns_cx_handle, &error_stat);
00204
00205   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_TABLE\n");
00206   // DEBUG printf("[JNI] reaction_handle pointer: %lu\n", (long)amns_cx_handle);
00207
00208   // we have to make sure that error_stat is passed back inside object
00209
00210   copyCError2JavaError( &error_stat, env, j_error_state);
00211
00212   // remember to release string
00213   (*env)->ReleaseStringUTFChars(env, j_reaction_string, reaction_string);
00214
00215   return (long) amns_cx_handle;
00216 }
```

Here is the call graph for this function:



16.94 amns_jni_call.c

```
00001 #include "amns_interface.h"
00002 #include <stdbool.h>
00003 #include <math.h>
00004 #include <assert.h>
00005 #include <stdio.h>
```

```
00006 #include "amns_jni_call.h"
00007 #include <jni.h>
00008
00009 void copyCError2JavaError( amns_c_error_type *error_stat, JNIEnv *env, jobject error_structure) {
00010
00011   // we have to return values into Java
00012   // note that we are playing here with pointers - they should be preserved accross calls
00013   // but will not be garbage collected from java
00014
00015   jclass cls = (*env)->GetObjectClass(env, error_structure);
00016   jfieldID paramFlag = (*env)->GetFieldID(env, cls, "flag", "Z");
00017   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00018
00019   jstring ret_message;
00020
00021   if(error_stat->string != NULL) {
00022     ret_message = (*env)->NewStringUTF(env,error_stat->string);
00023   } else {
00024     ret_message = (*env)->NewStringUTF(env,"");
00025   }
00026   if(ret_message == NULL) {
00027     (*env)->SetObjectField(env, error_structure, paramString, "");
00028   } else {
00029     (*env)->SetObjectField(env, error_structure, paramString, ret_message);
00030   }
00031
00032   jboolean ret_flag = error_stat->flag;
00033   (*env)->SetBooleanField(env, error_structure, paramFlag, ret_flag);
00034 }
00035
00036 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactants(JNIEnv *env, jobject obj, jint idx) {
00037   void *reactant_handle;
00038   IMAS_AMNS_CC_SETUP_REACTANTS(&reactant_handle, "", idx, 1);
00039   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_REACTANTS\n");
00040   // DEBUG printf("[JNI] Pointer: %lu\n", (long)reactant_handle);
00041   return (long)reactant_handle;
00042 }
00043
00044 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactantsNumber(JNIEnv *env, jobject obj, jint
       idx, jint number) {
00045   void *reactant_handle;
00046   // DEBUG printf("[JNI] Calling IMAS_AMNS_CC_SETUP_REACTANTS with number of reactions: %d\n",
      number);
00047   IMAS_AMNS_CC_SETUP_REACTANTS(&reactant_handle, "", idx, number);
00048   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_REACTANTS\n");
00049   // DEBUG printf("[JNI] Pointer: %lu\n", (long)reactant_handle);
00050   return (long)reactant_handle;
00051 }
00052
00053
00054 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactant(JNIEnv *env, jobject obj, jlong ptr,
       jobject acrt) {
00055
00056   void *reactant_handle = (void*) ptr;
00057
00058   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00059
00060   jclass cls = (*env)->GetObjectClass(env, acrt);
00061   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00062   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00063   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00064   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00065   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00066   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00067
00068   jdouble zn = (*env)->GetDoubleField(env, acrt, paramZN);
00069   jdouble za = (*env)->GetDoubleField(env, acrt, paramZA);
00070   jdouble mi = (*env)->GetDoubleField(env, acrt, paramMI);
00071   jint lr = (*env)->GetIntField(env, acrt, paramLR);
00072   jdouble rs = (*env)->GetDoubleField(env, acrt, paramRealSpecifier);
00073   jint int_spec = (*env)->GetIntField(env, acrt, paramIntSpecifier);
00074
00075   // DEBUG printf("[JNI] Values from Java: %f:%f:%f:%i:%f:%i\n", zn, za, mi, lr, rs, int_spec);
00076
00077   species.ZN=zn; species.ZA=za; species.MI=mi; species.LR=lr; species.real_specifier=rs;
      species.int_specifier=int_spec;
00078   IMAS_AMNS_CC_SET_REACTANT(reactant_handle, 1, &species);
00079   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_REACTANT\n");
00080 }
00081
00082 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactantIdx(
00083                                                                JNIEnv *env,
00084                                                                jobject obj,
00085                                                                jlong ptr,
00086                                                                jint idx,
00087                                                                jobject acrt) {
00088
```

```
00089   void *reactant_handle = (void*) ptr;
00090
00091   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00092
00093   jclass cls = (*env)->GetObjectClass(env, acrt);
00094   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00095   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00096   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00097   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00098   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00099   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00100
00101   jdouble zn = (*env)->GetDoubleField(env, acrt, paramZN);
00102   jdouble za = (*env)->GetDoubleField(env, acrt, paramZA);
00103   jdouble mi = (*env)->GetDoubleField(env, acrt, paramMI);
00104   jint lr = (*env)->GetIntField(env, acrt, paramLR);
00105   jdouble rs = (*env)->GetDoubleField(env, acrt, paramRealSpecifier);
00106   jint int_spec = (*env)->GetIntField(env, acrt, paramIntSpecifier);
00107
00108   // DEBUG printf("[JNI] Values from Java: %f:%f:%f:%i:%f:%i\n", zn, za, mi, lr, rs, int_spec);
00109
00110   species.ZN=zn; species.ZA=za; species.MI=mi; species.LR=lr; species.real_specifier=rs;
      species.int_specifier=int_spec;
00111   // DEBUG printf("[JNI] Calling IMAS_AMNS_CC_SET_REACTANT with index: %d\n", idx);
00112   IMAS_AMNS_CC_SET_REACTANT(reactant_handle, idx, &species);
00113   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_REACTANT\n");
00114 }
00115
00116 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCGetReactant(JNIEnv *env, jobject obj, jlong ptr,
      jobject jSpecies) {
00117
00118   void *reactant_handle = (void*) ptr;
00119
00120   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00121
00122   // at first, get all elements of the species
00123   jclass cls = (*env)->GetObjectClass(env, jSpecies);
00124   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00125   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00126   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00127   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00128   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00129   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00130
00131   // retrieve data into C structure
00132   IMAS_AMNS_CC_GET_REACTANT(reactant_handle, 1, &species);
00133   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_GET_REACTANTS\n");
00134
00135   // copy data back to Java
00136   (*env)->SetDoubleField(env, jSpecies, paramZN, (jdouble)species.ZN );
00137   (*env)->SetDoubleField(env, jSpecies, paramZA, species.ZA );
00138   (*env)->SetDoubleField(env, jSpecies, paramMI, species.MI );
00139   (*env)->SetIntField(env, jSpecies, paramLR, species.LR );
00140   (*env)->SetDoubleField(env, jSpecies, paramRealSpecifier, species.real_specifier);
00141   (*env)->SetIntField(env, jSpecies, paramIntSpecifier, species.int_specifier);
00142 }
00143
00144
00145 /*
00146
00147 This function calls IMAS_AMNS_CC_SETUP
00148
00149 it returns pointer to amns_handle structure and error_stat structure
00150
00151 */
00152 JNIEXPORT long JNICALL Java_amns_Amns_ImasAmnsCCSetup(JNIEnv *env, jobject obj, jobject
      error_structure) {
00153   void* amns_handle = NULL;
00154   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00155
00156   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SETUP\n");
00157   IMAS_AMNS_CC_SETUP(&amns_handle, &error_stat);
00158   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP\n");
00159   // DEBUG printf("[JNI] Pointer: %lu\n", (long)amns_handle);
00160
00161   copyCError2JavaError( &error_stat, env, error_structure );
00162
00163   return (long) amns_handle;
00164 }
00165
00166 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetupTable(
00167                                                             JNIEnv *env
00168                                                             , jobject obj
00169                                                             , jlong ptr_amns_handle
00170                                                             , jobject j_reaction_type
00171                                                             , jlong ptr_reactants_handle
00172                                                             , jobject j_error_state) {
```

```
00173   // declare structures used in call
00174   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00175   void* amns_cx_handle;
00176   amns_c_reaction_type xx_rx;
00177
00178   // get pointers
00179   void *amns_handle = (void*) ptr_amns_handle;
00180   void *reactants_handle = (void*) ptr_reactants_handle;
00181
00182   // retrieve and copy values from reaction_type object
00183   jclass cls = (*env)->GetObjectClass(env, j_reaction_type);
00184   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00185   jfieldID paramIsotope = (*env)->GetFieldID(env, cls, "isotopeResolved", "I");
00186
00187   jstring j_reaction_string = (*env)->GetObjectField(env, j_reaction_type, paramString);
00188   jint j_reaction_isotope = (*env)->GetIntField(env, j_reaction_type, paramIsotope);
00189
00190   char *reaction_string = (char*) (*env)->GetStringUTFChars(env, j_reaction_string, NULL);
00191   int reaction_isotope = (int) j_reaction_isotope;
00192
00193   xx_rx.string = reaction_string;
00194   xx_rx.isotope_resolved = reaction_isotope;
00195
00196   // DEBUG printf("[JNI] Reaction type = %s, %u\n", xx_rx.string, xx_rx.isotope_resolved);
00197
00198   // we can now call C function
00199   // note that we will return amns_cx_handle as return value from function
00200   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SETUP_TABLE\n");
00201   // DEBUG printf("[JNI] amns_handle pointer: %lu\n", (long)amns_handle);
00202   // DEBUG printf("[JNI] reactants_handle pointer: %lu\n", (long)reactants_handle);
00203   IMAS_AMNS_CC_SETUP_TABLE(amns_handle, &xx_rx, reactants_handle, &amns_cx_handle, &error_stat);
00204
00205   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_TABLE\n");
00206   // DEBUG printf("[JNI] reaction_handle pointer: %lu\n", (long)amns_cx_handle);
00207
00208   // we have to make sure that error_stat is passed back inside object
00209
00210   copyCError2JavaError( &error_stat, env, j_error_state);
00211
00212   // remember to release string
00213   (*env)->ReleaseStringUTFChars(env, j_reaction_string, reaction_string);
00214
00215   return (long) amns_cx_handle;
00216 }
00217
00218 JNIEXPORT jdouble JNICALL Java_amns_Amns_ImasAmnsCCRX0B(
00219                                                         JNIEnv *env
00220                                                         , jobject obj
00221                                                         , jlong ptrHandleRx
00222                                                         , double j_arg1
00223                                                         , double j_arg2
00224                                                         , jobject j_error_state ) {
00225
00226   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00227   void* amns_cx_handle = (void *) ptrHandleRx;
00228   double arg1 = (double) j_arg1;
00229   double arg2 = (double) j_arg2;
00230   double rate;
00231
00232   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_0_B\n");
00233   // DEBUG printf("[JNI] cx_handle: %x, arg1: %e, arg2: %e\n", ptrHandleRx, arg1, arg2);
00234   IMAS_AMNS_CC_RX_0_B(amns_cx_handle, &rate, arg1, arg2, &error_stat);
00235   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_0_B\n");
00236   // DEBUG printf("[JNI] rate: %e\n", rate);
00237
00238   // we have to make sure that error_stat is passed back inside object
00239
00240   copyCError2JavaError( &error_stat, env, j_error_state);
00241
00242   return (jdouble) rate;
00243
00244 }
00245
00246 JNIEXPORT long JNICALL Java_amns_Amns_ImasAmnsCCFinishTable(
00247                                                           JNIEnv *env
00248                                                           , jobject obj
00249                                                           , jlong ptrHandleRX
00250                                                           , jobject j_error_state) {
00251
00252   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00253   void* amns_cx_handle = (void *) ptrHandleRX;
00254
00255   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH_TABLE\n");
00256   // DEBUG printf("[JNI] cx_handle: %lu\n", (long)ptrHandleRX);
00257   IMAS_AMNS_CC_FINISH_TABLE(&amns_cx_handle, &error_stat);
00258   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH_TABLE\n");
00259
```

```
00260   // we have to make sure that error_stat is passed back inside object
00261
00262   copyCError2JavaError( &error_stat, env, j_error_state );
00263
00264   return (jlong) amns_cx_handle;
00265
00266 }
00267
00268 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishReactants(
00269                                                                  JNIEnv *env
00270                                                                  , jobject obj
00271                                                                  , jlong ptr_reactants_handle) {
00272
00273   void* reactants_handle = (void *) ptr_reactants_handle;
00274
00275   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH_REACTANTS\n");
00276   // DEBUG printf("[JNI] reactants_handle: %lu\n", (long)ptr_reactants_handle);
00277   IMAS_AMNS_CC_FINISH_REACTANTS(&reactants_handle);
00278   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH_REACTANTS\n");
00279
00280   return (long) reactants_handle;
00281 }
00282
00283
00284 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinish(
00285                                                         JNIEnv *env
00286                                                         , jobject obj
00287                                                         , jlong ptr_amns_handle
00288                                                         , jobject j_error_state) {
00289
00290   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00291   void* amns_handle = (void*) ptr_amns_handle;
00292
00293   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH\n");
00294   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptr_amns_handle);
00295   IMAS_AMNS_CC_FINISH(&amns_handle, &error_stat);
00296   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH\n");
00297
00298   // we have to make sure that error_stat is passed back inside object
00299
00300   copyCError2JavaError( &error_stat, env, j_error_state);
00301
00302   return (long) amns_handle;
00303
00304 }
00305
00306 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSet(
00307                                                    JNIEnv *env
00308                                                    , jobject obj
00309                                                    , jlong ptrAmnsHandle
00310                                                    , jobject j_amns_set_type
00311                                                    , jobject j_error_state) {
00312
00313   amns_c_set_type set = DEFAULT_AMNS_C_SET_TYPE;
00314   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00315   void *amns_handler = (void *) ptrAmnsHandle;
00316
00317   jclass cls = (*env)->GetObjectClass(env, j_amns_set_type);
00318   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00319
00320   jstring j_set_type_string = (*env)->GetObjectField(env, j_amns_set_type, paramString);
00321
00322   char *set_type_string = (char*)(*env)->GetStringUTFChars(env, j_set_type_string, NULL);
00323
00324   set.string = set_type_string;
00325
00326   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SET\n");
00327   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptrAmnsHandle);
00328   IMAS_AMNS_CC_SET(amns_handler, &set, &error_stat);
00329   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET\n");
00330
00331   copyCError2JavaError( &error_stat, env, j_error_state);
00332
00333   (*env)->ReleaseStringUTFChars(env, j_set_type_string, set_type_string);
00334
00335 }
00336
00337 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQuery(
00338                                                      JNIEnv *env
00339                                                      , jobject obj
00340                                                      , jlong ptrAmnsHandle
00341                                                      , jobject j_amns_query
00342                                                      , jobject j_amns_answer
00343                                                      , jobject j_error_state) {
00344
00345   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00346   amns_c_query_type query = DEFAULT_AMNS_C_QUERY_TYPE;
```

```
00347   amns_c_answer_type answer = DEFAULT_AMNS_C_ANSWER_TYPE;
00348   void *amns_handler = (void *) ptrAmnsHandle;
00349
00350   // we have to retrieve the query and pass it to function
00351
00352   jclass cls = (*env)->GetObjectClass(env, j_amns_query);
00353   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00354
00355   jstring j_query_string = (*env)->GetObjectField(env, j_amns_query, paramString);
00356
00357   char *query_string = (char*)(*env)->GetStringUTFChars(env, j_query_string, NULL);
00358
00359   query.string = query_string;
00360
00361   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_QUERY\n");
00362   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptrAmnsHandle);
00363   IMAS_AMNS_CC_QUERY(amns_handler, &query, &answer, &error_stat);
00364   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_QUERY\n");
00365
00366   // copy error to Java
00367   copyCError2JavaError( &error_stat, env, j_error_state);
00368
00369   // copy answer to Java
00370
00371   jclass cls_answer = (*env)->GetObjectClass(env, j_amns_answer);
00372   jfieldID answerNumber = (*env)->GetFieldID(env, cls_answer, "number", "I");
00373   jfieldID answerString = (*env)->GetFieldID(env, cls_answer, "string", "Ljava/lang/String;");
00374
00375   jstring answer_value;
00376
00377   if(answer.string != NULL) {
00378     answer_value = (*env)->NewStringUTF(env,answer.string);
00379   } else {
00380     answer_value = (*env)->NewStringUTF(env,"");
00381   }
00382   if(answer_value == NULL) {
00383     (*env)->SetObjectField(env, j_amns_answer, answerString, "");
00384   } else {
00385     (*env)->SetObjectField(env, j_amns_answer, answerString, answer_value);
00386   }
00387
00388   jint answer_number = answer.number;
00389   (*env)->SetIntField(env, j_amns_answer, answerNumber, answer_number);
00390 }
00391
00392 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQueryTable(
00393                                                           JNIEnv *env
00394                                                           , jobject obj
00395                                                           , jlong ptrTableHandle
00396                                                           , jobject j_amns_query
00397                                                           , jobject j_amns_answer
00398                                                           , jobject j_error_state) {
00399
00400   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00401   amns_c_query_type query = DEFAULT_AMNS_C_QUERY_TYPE;
00402   amns_c_answer_type answer = DEFAULT_AMNS_C_ANSWER_TYPE;
00403   void *table_handler = (void *) ptrTableHandle;
00404
00405   // we have to retrieve the query and pass it to function
00406
00407   jclass cls = (*env)->GetObjectClass(env, j_amns_query);
00408   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00409
00410   jstring j_query_string = (*env)->GetObjectField(env, j_amns_query, paramString);
00411
00412   char *query_string = (char*)(*env)->GetStringUTFChars(env, j_query_string, NULL);
00413
00414   query.string = query_string;
00415
00416   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_QUERY_TABLE\n");
00417   // DEBUG printf("[JNI] table_handle: %lu\n", (long)ptrTableHandle);
00418   IMAS_AMNS_CC_QUERY_TABLE(table_handler, &query, &answer, &error_stat);
00419   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_QUERY_TABLE\n");
00420
00421   // copy error to Java
00422   copyCError2JavaError( &error_stat, env, j_error_state);
00423
00424   // copy answer to Java
00425
00426   jclass cls_answer = (*env)->GetObjectClass(env, j_amns_answer);
00427   jfieldID answerNumber = (*env)->GetFieldID(env, cls_answer, "number", "I");
00428   jfieldID answerString = (*env)->GetFieldID(env, cls_answer, "string", "Ljava/lang/String;");
00429
00430   jstring answer_value;
00431
00432   if(answer.string != NULL) {
00433     answer_value = (*env)->NewStringUTF(env,answer.string);
```

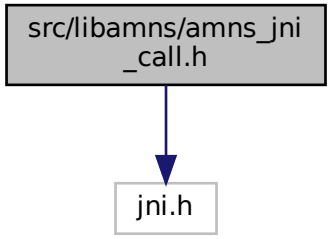
```
00434   } else {
00435     answer_value = (*env)->NewStringUTF(env,"");
00436   }
00437   if(answer_value == NULL) {
00438     (*env)->SetObjectField(env, j_amns_answer, answerString, "");
00439   } else {
00440     (*env)->SetObjectField(env, j_amns_answer, answerString, answer_value);
00441   }
00442
00443   jint answer_number = answer.number;
00444   (*env)->SetIntField(env, j_amns_answer, answerNumber, answer_number);
00445 }
00446
00447 JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1A(
00448                                                             JNIEnv *env,
00449                                                             jobject obj,
00450                                                             jlong ptrHandleRx,
00451                                                             jint nx,
00452                                                             jdoubleArray j_arg1,
00453                                                             jobject j_error_state) {
00454
00455   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00456   void *handle_rx_in = (void *) ptrHandleRx;
00457
00458   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00459   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00460
00461   double *outputArray = (double *) malloc(nx * sizeof(double));
00462
00463   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_A\n");
00464   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00465   IMAS_AMNS_CC_RX_1_A(handle_rx_in, nx, outputArray, arg1, &error_stat);
00466   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_A\n");
00467
00468   // copy error to Java
00469   copyCError2JavaError( &error_stat, env, j_error_state);
00470
00471   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00472   if (NULL == outJNIArray) {
00473     return NULL;
00474   }
00475   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00476
00477   return outJNIArray;
00478 }
00479
00480 JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1B(
00481                                                             JNIEnv *env,
00482                                                             jobject obj,
00483                                                             jlong ptrHandleRx,
00484                                                             jint nx,
00485                                                             jdoubleArray j_arg1,
00486                                                             jdoubleArray j_arg2,
00487                                                             jobject j_error_state) {
00488
00489   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00490   void *handle_rx_in = (void *) ptrHandleRx;
00491
00492   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00493   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00494
00495   jsize len_arr2 = (*env)->GetArrayLength(env, j_arg2);
00496   jdouble *arg2 = (*env)->GetDoubleArrayElements(env, j_arg2, 0);
00497
00498   double *outputArray = (double *) malloc(nx * sizeof(double));
00499
00500   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_B\n");
00501   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00502
00503   /*printf("--- JNI CODE - arg1 - start ---\n");
00504     int ii=0;
00505     for(ii=0; ii<nx; ii++ ) {
00506     printf("%1.6f ", arg1[ii]);
00507     }
00508     printf("\n");
00509     printf("--- JNI CODE - end ---\n");
00510     fflush(stdout);
00511     printf("--- JNI CODE - arg2 - start ---\n");
00512     for(ii=0; ii<nx; ii++ ) {
00513     printf("%1.6f ", arg2[ii]);
00514     }
00515     printf("\n");
00516     printf("--- JNI CODE - end ---\n");
00517     fflush(stdout);*/
00518
00519   IMAS_AMNS_CC_RX_1_B(handle_rx_in, nx, outputArray, arg1, arg2, &error_stat);
00520   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_B\n");
```

```
00521
00522   // copy error to Java
00523   copyCError2JavaError( &error_stat, env, j_error_state);
00524
00525   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00526   if (NULL == outJNIArray) {
00527     return NULL;
00528   }
00529   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00530
00531   return outJNIArray;
00532
00533 }
00534
00535 JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1C(
00536                                                             JNIEnv *env,
00537                                                             jobject obj,
00538                                                             jlong ptrHandleRx,
00539                                                             jint nx,
00540                                                             jdoubleArray j_arg1,
00541                                                             jdoubleArray j_arg2,
00542                                                             jdoubleArray j_arg3,
00543                                                             jobject j_error_state) {
00544
00545   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00546   void *handle_rx_in = (void *) ptrHandleRx;
00547
00548
00549   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00550   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00551
00552   jsize len_arr2 = (*env)->GetArrayLength(env, j_arg2);
00553   jdouble *arg2 = (*env)->GetDoubleArrayElements(env, j_arg2, 0);
00554
00555   jsize len_arr3 = (*env)->GetArrayLength(env, j_arg3);
00556   jdouble *arg3 = (*env)->GetDoubleArrayElements(env, j_arg3, 0);
00557
00558   double *outputArray = (double *) malloc(nx * sizeof(double));
00559
00560   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_C\n");
00561   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00562   IMAS_AMNS_CC_RX_1_C(handle_rx_in, nx, outputArray, arg1, arg2, arg3, &error_stat);
00563   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_C\n");
00564
00565   // copy error to Java
00566   copyCError2JavaError( &error_stat, env, j_error_state);
00567
00568   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00569   if (NULL == outJNIArray) {
00570     return NULL;
00571   }
00572   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00573
00574   return outJNIArray;
00575
00576 }
00577
00578 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetTable(
00579                                                         JNIEnv *env,
00580                                                         jobject obj,
00581                                                         jlong ptrAmnsCxHandle,
00582                                                         jobject j_amns_set_type,
00583                                                         jobject j_error_state) {
00584   amns_c_set_type set = DEFAULT_AMNS_C_SET_TYPE;
00585   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00586   void *amns_cx_handler = (void *) ptrAmnsCxHandle;
00587
00588   jclass cls = (*env)->GetObjectClass(env, j_amns_set_type);
00589   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00590
00591   jstring j_set_type_string = (*env)->GetObjectField(env, j_amns_set_type, paramString);
00592
00593   char *set_type_string = (char*)(*env)->GetStringUTFChars(env, j_set_type_string, NULL);
00594
00595   set.string = set_type_string;
00596
00597   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SET_TABLE\n");
00598   // DEBUG printf("[JNI] amns_cx_handle: %lu\n", (long)ptrAmnsCxHandle);
00599   IMAS_AMNS_CC_SET_TABLE(amns_cx_handler, &set, &error_stat);
00600   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_TABLE\n");
00601
00602   copyCError2JavaError( &error_stat, env, j_error_state);
00603
00604   (*env)->ReleaseStringUTFChars(env, j_set_type_string, set_type_string);
00605 }
```

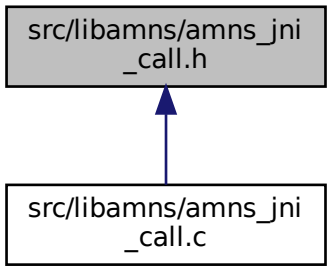
16.95 src/libamns/amns_jni_call.h File Reference

```
#include <jni.h>
```

Include dependency graph for amns_jni_call.h:



This graph shows which files directly or indirectly include this file:



Functions

- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactants (JNIEnv ∗, jobject, jint)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactantsNumber (JNIEnv ∗, jobject, jint, jint)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactant (JNIEnv ∗, jobject, jlong, jobject)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactantIdx (JNIEnv ∗, jobject, jlong, jint, jobject)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetup (JNIEnv ∗, jobject, jobject)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCGetReactant (JNIEnv ∗, jobject, jlong, jobject)
- JNIEXPORT jdouble JNICALL Java_amns_Amns_ImasAmnsCCRX0B (JNIEnv ∗, jobject, jlong, jdouble, jdouble, jobject)
- JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1A (JNIEnv ∗, jobject, jlong, jint, jdoubleArray, jobject)
- JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1B (JNIEnv ∗, jobject, jlong, jint, jdoubleArray, jdoubleArray, jobject)
- JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1C (JNIEnv ∗, jobject, jlong, jint, jdoubleArray, jdoubleArray, jdoubleArray, jobject)

- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishTable (JNIEnv ∗, jobject, jlong, jobject)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishReactants (JNIEnv ∗, jobject, jlong)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinish (JNIEnv ∗, jobject, jlong, jobject)
- JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetupTable (JNIEnv ∗, jobject, jlong, jobject, jlong, jobject)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSet (JNIEnv ∗, jobject, jlong, jobject, jobject)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetTable (JNIEnv ∗, jobject, jlong, jobject, jobject)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQuery (JNIEnv ∗, jobject, jlong, jobject, jobject, jobject)
- JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQueryTable (JNIEnv ∗, jobject, jlong, jobject, jobject, jobject)

16.95.1 Function Documentation

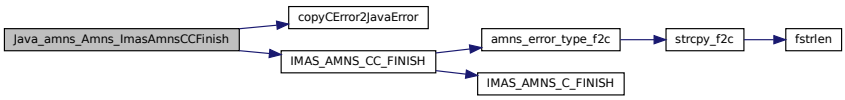
16.95.1.1 Java_amns_Amns_ImasAmnsCCFinish()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinish (
              JNIEnv * ,
              jobject ,
              jlong ,
              jobject  )
```

Definition at line 284 of file amns_jni_call.c.

```
00288                                                                                {
00289
00290   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00291   void* amns_handle = (void*) ptr_amns_handle;
00292
00293   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH\n");
00294   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptr_amns_handle);
00295   IMAS_AMNS_CC_FINISH(&amns_handle, &error_stat);
00296   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH\n");
00297
00298   // we have to make sure that error_stat is passed back inside object
00299
00300   copyCError2JavaError( &error_stat, env, j_error_state);
00301
00302   return (long) amns_handle;
00303
00304 }
```

Here is the call graph for this function:



16.95.1.2 Java_amns_Amns_ImasAmnsCCFinishReactants()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishReactants (
              JNIEnv * ,
              jobject ,
              jlong  )
```

Definition at line 268 of file amns_jni_call.c.

```
00271                                                                                {
00272
00273   void* reactants_handle = (void *) ptr_reactants_handle;
00274
00275   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH_REACTANTS\n");
```

```
00276   // DEBUG printf("[JNI] reactants_handle: %lu\n", (long)ptr_reactants_handle);
00277   IMAS_AMNS_CC_FINISH_REACTANTS(&reactants_handle);
00278   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH_REACTANTS\n");
00279
00280   return (long) reactants_handle;
00281 }
```

Here is the call graph for this function:

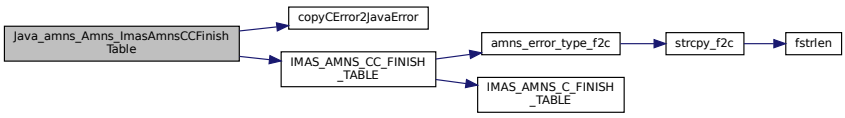
16.95.1.3 Java_amns_Amns_ImasAmnsCCFinishTable()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishTable (
              JNIEnv * ,
              jobject ,
              jlong ,
              jobject  )
```

Definition at line 246 of file amns_jni_call.c.

```
00250                                                                                                {
00251
00252   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00253   void* amns_cx_handle = (void *) ptrHandleRX;
00254
00255   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_FINISH_TABLE\n");
00256   // DEBUG printf("[JNI] cx_handle: %lu\n", (long)ptrHandleRX);
00257   IMAS_AMNS_CC_FINISH_TABLE(&amns_cx_handle, &error_stat);
00258   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_FINISH_TABLE\n");
00259
00260   // we have to make sure that error_stat is passed back inside object
00261
00262   copyCError2JavaError( &error_stat, env, j_error_state );
00263
00264   return (jlong) amns_cx_handle;
00265
00266 }
```

Here is the call graph for this function:



16.95.1.4 Java_amns_Amns_ImasAmnsCCGetReactant()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCGetReactant (
              JNIEnv * ,
              jobject ,
              jlong ,
              jobject  )
```

Definition at line 116 of file amns_jni_call.c.

```
00116                                                                                                {
00117
00118   void *reactant_handle = (void*) ptr;
00119
```

```
00120   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00121
00122   // at first, get all elements of the species
00123   jclass cls = (*env)->GetObjectClass(env, jSpecies);
00124   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00125   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00126   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00127   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00128   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00129   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00130
00131   // retrieve data into C structure
00132   IMAS_AMNS_CC_GET_REACTANT(reactant_handle, 1, &species);
00133   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_GET_REACTANTS\n");
00134
00135   // copy data back to Java
00136   (*env)->SetDoubleField(env, jSpecies, paramZN, (jdouble)species.ZN );
00137   (*env)->SetDoubleField(env, jSpecies, paramZA, species.ZA );
00138   (*env)->SetDoubleField(env, jSpecies, paramMI, species.MI );
00139   (*env)->SetIntField(env, jSpecies, paramLR, species.LR );
00140   (*env)->SetDoubleField(env, jSpecies, paramRealSpecifier, species.real_specifier);
00141   (*env)->SetIntField(env, jSpecies, paramIntSpecifier, species.int_specifier);
00142 }
```

Here is the call graph for this function:

16.95.1.5 Java_amns_Amns_ImasAmnsCCQuery()

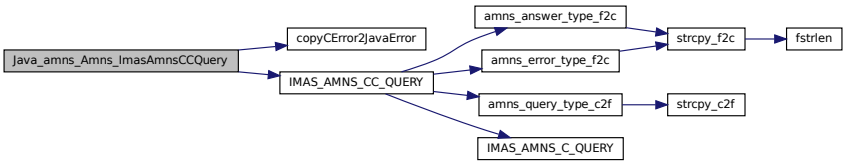
```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQuery (
            JNIEnv * ,
            jobject ,
            jlong ,
            jobject ,
            jobject ,
            jobject  )
```

Definition at line 337 of file amns_jni_call.c.

```
00343                                                                                {
00344
00345   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00346   amns_c_query_type query = DEFAULT_AMNS_C_QUERY_TYPE;
00347   amns_c_answer_type answer = DEFAULT_AMNS_C_ANSWER_TYPE;
00348   void *amns_handler = (void *) ptrAmnsHandle;
00349
00350   // we have to retrieve the query and pass it to function
00351
00352   jclass cls = (*env)->GetObjectClass(env, j_amns_query);
00353   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00354
00355   jstring j_query_string = (*env)->GetObjectField(env, j_amns_query, paramString);
00356
00357   char *query_string = (char*)(*env)->GetStringUTFChars(env, j_query_string, NULL);
00358
00359   query.string = query_string;
00360
00361   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_QUERY\n");
00362   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptrAmnsHandle);
00363   IMAS_AMNS_CC_QUERY(amns_handler, &query, &answer, &error_stat);
00364   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_QUERY\n");
00365
00366   // copy error to Java
00367   copyCError2JavaError( &error_stat, env, j_error_state);
00368
00369   // copy answer to Java
00370
00371   jclass cls_answer = (*env)->GetObjectClass(env, j_amns_answer);
00372   jfieldID answerNumber = (*env)->GetFieldID(env, cls_answer, "number", "I");
00373   jfieldID answerString = (*env)->GetFieldID(env, cls_answer, "string", "Ljava/lang/String;");
00374
00375   jstring answer_value;
```

```
00376
00377   if(answer.string != NULL) {
00378     answer_value = (*env)->NewStringUTF(env,answer.string);
00379   } else {
00380     answer_value = (*env)->NewStringUTF(env,"");
00381   }
00382   if(answer_value == NULL) {
00383     (*env)->SetObjectField(env, j_amns_answer, answerString, "");
00384   } else {
00385     (*env)->SetObjectField(env, j_amns_answer, answerString, answer_value);
00386   }
00387
00388   jint answer_number = answer.number;
00389   (*env)->SetIntField(env, j_amns_answer, answerNumber, answer_number);
00390 }
```

Here is the call graph for this function:



16.95.1.6 Java_amns_Amns_ImasAmnsCCQueryTable()

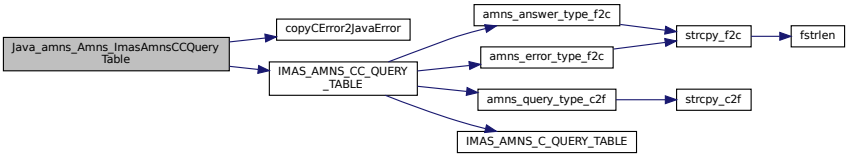
```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQueryTable (
            JNIEnv * ,
            jobject ,
            jlong ,
            jobject ,
            jobject ,
            jobject  )
```

Definition at line 392 of file amns_jni_call.c.

```
00398                                                                                {
00399
00400   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00401   amns_c_query_type query = DEFAULT_AMNS_C_QUERY_TYPE;
00402   amns_c_answer_type answer = DEFAULT_AMNS_C_ANSWER_TYPE;
00403   void *table_handler = (void *) ptrTableHandle;
00404
00405   // we have to retrieve the query and pass it to function
00406
00407   jclass cls = (*env)->GetObjectClass(env, j_amns_query);
00408   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00409
00410   jstring j_query_string = (*env)->GetObjectField(env, j_amns_query, paramString);
00411
00412   char *query_string = (char*)(*env)->GetStringUTFChars(env, j_query_string, NULL);
00413
00414   query.string = query_string;
00415
00416   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_QUERY_TABLE\n");
00417   // DEBUG printf("[JNI] table_handle: %lu\n", (long)ptrTableHandle);
00418   IMAS_AMNS_CC_QUERY_TABLE(table_handler, &query, &answer, &error_stat);
00419   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_QUERY_TABLE\n");
00420
00421   // copy error to Java
00422   copyCError2JavaError( &error_stat, env, j_error_state);
00423
00424   // copy answer to Java
00425
00426   jclass cls_answer = (*env)->GetObjectClass(env, j_amns_answer);
00427   jfieldID answerNumber = (*env)->GetFieldID(env, cls_answer, "number", "I");
00428   jfieldID answerString = (*env)->GetFieldID(env, cls_answer, "string", "Ljava/lang/String;");
00429
00430   jstring answer_value;
00431
00432   if(answer.string != NULL) {
00433     answer_value = (*env)->NewStringUTF(env,answer.string);
00434   } else {
```

```
00435     answer_value = (*env)->NewStringUTF(env,"");
00436   }
00437   if(answer_value == NULL) {
00438     (*env)->SetObjectField(env, j_amns_answer, answerString, "");
00439   } else {
00440     (*env)->SetObjectField(env, j_amns_answer, answerString, answer_value);
00441   }
00442
00443   jint answer_number = answer.number;
00444   (*env)->SetIntField(env, j_amns_answer, answerNumber, answer_number);
00445 }
```

Here is the call graph for this function:



16.95.1.7 Java_amns_Amns_ImasAmnsCCRX0B()

```
JNIEXPORT jdouble JNICALL Java_amns_Amns_ImasAmnsCCRX0B (
            JNIEnv * ,
            jobject ,
            jlong ,
            jdouble ,
            jdouble ,
            jobject  )
```

16.95.1.8 Java_amns_Amns_ImasAmnsCCRX1A()

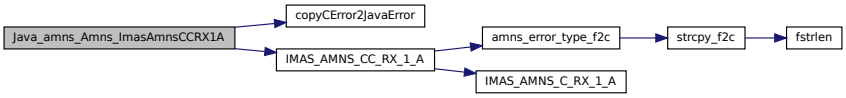
```
JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1A (
            JNIEnv * ,
            jobject ,
            jlong ,
            jint ,
            jdoubleArray ,
            jobject  )
```

Definition at line 447 of file amns_jni_call.c.

```
00453                                                                                {
00454
00455   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00456   void *handle_rx_in = (void *) ptrHandleRx;
00457
00458   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00459   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00460
00461   double *outputArray = (double *) malloc(nx * sizeof(double));
00462
00463   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_A\n");
00464   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00465   IMAS_AMNS_CC_RX_1_A(handle_rx_in, nx, outputArray, arg1, &error_stat);
00466   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_A\n");
00467
00468   // copy error to Java
00469   copyCError2JavaError( &error_stat, env, j_error_state);
00470
00471   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00472   if (NULL == outJNIArray) {
00473     return NULL;
00474   }
00475   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00476
00477   return outJNIArray;
```

```
00478 }
```

Here is the call graph for this function:



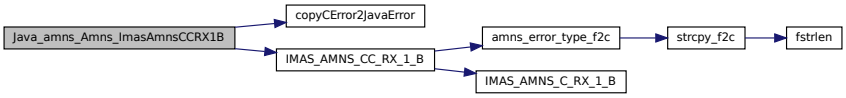
16.95.1.9 Java_amns_Amns_ImasAmnsCCRX1B()

```
JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1B (
            JNIEnv * ,
            jobject ,
            jlong ,
            jint ,
            jdoubleArray ,
            jdoubleArray ,
            jobject  )
```

Definition at line 480 of file amns_jni_call.c.

```
00487                                                                                {
00488
00489   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00490   void *handle_rx_in = (void *) ptrHandleRx;
00491
00492   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00493   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00494
00495   jsize len_arr2 = (*env)->GetArrayLength(env, j_arg2);
00496   jdouble *arg2 = (*env)->GetDoubleArrayElements(env, j_arg2, 0);
00497
00498   double *outputArray = (double *) malloc(nx * sizeof(double));
00499
00500   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_B\n");
00501   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00502
00503   /*printf("--- JNI CODE - arg1 - start ---\n");
00504     int ii=0;
00505     for(ii=0; ii<nx; ii++ ) {
00506     printf("%1.6f ", arg1[ii]);
00507     }
00508     printf("\n");
00509     printf("--- JNI CODE - end ---\n");
00510     fflush(stdout);
00511     printf("--- JNI CODE - arg2 - start ---\n");
00512     for(ii=0; ii<nx; ii++ ) {
00513     printf("%1.6f ", arg2[ii]);
00514     }
00515     printf("\n");
00516     printf("--- JNI CODE - end ---\n");
00517     fflush(stdout);*/
00518
00519   IMAS_AMNS_CC_RX_1_B(handle_rx_in, nx, outputArray, arg1, arg2, &error_stat);
00520   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_B\n");
00521
00522   // copy error to Java
00523   copyCError2JavaError( &error_stat, env, j_error_state);
00524
00525   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00526   if (NULL == outJNIArray) {
00527     return NULL;
00528   }
00529   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00530
00531   return outJNIArray;
00532
00533 }
```

Here is the call graph for this function:



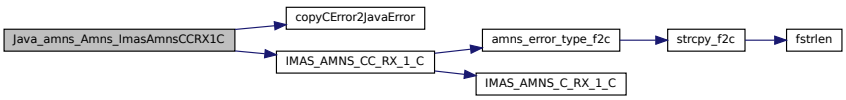
16.95.1.10 Java_amns_Amns_ImasAmnsCCRX1C()

```
JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1C (
            JNIEnv * ,
            jobject ,
            jlong ,
            jint ,
            jdoubleArray ,
            jdoubleArray ,
            jdoubleArray ,
            jobject  )
```

Definition at line 535 of file amns_jni_call.c.

```
00543                                                                                {
00544
00545   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00546   void *handle_rx_in = (void *) ptrHandleRx;
00547
00548
00549   jsize len_arr1 = (*env)->GetArrayLength(env, j_arg1);
00550   jdouble *arg1 = (*env)->GetDoubleArrayElements(env, j_arg1, 0);
00551
00552   jsize len_arr2 = (*env)->GetArrayLength(env, j_arg2);
00553   jdouble *arg2 = (*env)->GetDoubleArrayElements(env, j_arg2, 0);
00554
00555   jsize len_arr3 = (*env)->GetArrayLength(env, j_arg3);
00556   jdouble *arg3 = (*env)->GetDoubleArrayElements(env, j_arg3, 0);
00557
00558   double *outputArray = (double *) malloc(nx * sizeof(double));
00559
00560   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_RX_1_C\n");
00561   // DEBUG printf("[JNI] table_handle: %lu\n", (long)handle_rx_in);
00562   IMAS_AMNS_CC_RX_1_C(handle_rx_in, nx, outputArray, arg1, arg2, arg3, &error_stat);
00563   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_RX_1_C\n");
00564
00565   // copy error to Java
00566   copyCError2JavaError( &error_stat, env, j_error_state);
00567
00568   jdoubleArray outJNIArray = (*env)->NewDoubleArray(env, nx);
00569   if (NULL == outJNIArray) {
00570     return NULL;
00571   }
00572   (*env)->SetDoubleArrayRegion(env, outJNIArray, 0 , nx, outputArray);
00573
00574   return outJNIArray;
00575
00576 }
```

Here is the call graph for this function:



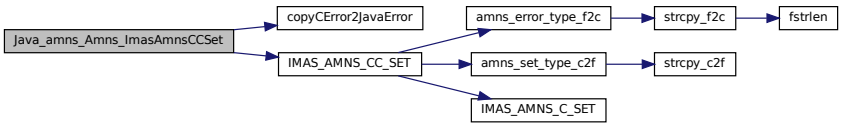
16.95.1.11 Java_amns_Amns_ImasAmnsCCSet()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSet (
            JNIEnv * ,
            jobject ,
            jlong ,
            jobject ,
            jobject  )
```

Definition at line 306 of file amns_jni_call.c.

```
00311                                                                    {
00312
00313   amns_c_set_type set = DEFAULT_AMNS_C_SET_TYPE;
00314   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00315   void *amns_handler = (void *) ptrAmnsHandle;
00316
00317   jclass cls = (*env)->GetObjectClass(env, j_amns_set_type);
00318   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00319
00320   jstring j_set_type_string = (*env)->GetObjectField(env, j_amns_set_type, paramString);
00321
00322   char *set_type_string = (char*)(*env)->GetStringUTFChars(env, j_set_type_string, NULL);
00323
00324   set.string = set_type_string;
00325
00326   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SET\n");
00327   // DEBUG printf("[JNI] amns_handle: %lu\n", (long)ptrAmnsHandle);
00328   IMAS_AMNS_CC_SET(amns_handler, &set, &error_stat);
00329   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET\n");
00330
00331   copyCError2JavaError( &error_stat, env, j_error_state);
00332
00333   (*env)->ReleaseStringUTFChars(env, j_set_type_string, set_type_string);
00334
00335 }
```

Here is the call graph for this function:



16.95.1.12 Java_amns_Amns_ImasAmnsCCSetReactant()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactant (
            JNIEnv * ,
            jobject ,
            jlong ,
            jobject  )
```

Definition at line 54 of file amns_jni_call.c.

```
00054
            {
00055
00056   void *reactant_handle = (void*) ptr;
00057
00058   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00059
00060   jclass cls = (*env)->GetObjectClass(env, acrt);
00061   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00062   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00063   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00064   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00065   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00066   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00067
00068   jdouble zn = (*env)->GetDoubleField(env, acrt, paramZN);
00069   jdouble za = (*env)->GetDoubleField(env, acrt, paramZA);
00070   jdouble mi = (*env)->GetDoubleField(env, acrt, paramMI);
00071   jint lr = (*env)->GetIntField(env, acrt, paramLR);
```

```
00072   jdouble rs = (*env)->GetDoubleField(env, acrt, paramRealSpecifier);
00073   jint int_spec = (*env)->GetIntField(env, acrt, paramIntSpecifier);
00074
00075   // DEBUG printf("[JNI] Values from Java: %f:%f:%f:%i:%f:%i\n", zn, za, mi, lr, rs, int_spec);
00076
00077   species.ZN=zn; species.ZA=za; species.MI=mi; species.LR=lr; species.real_specifier=rs;
      species.int_specifier=int_spec;
00078   IMAS_AMNS_CC_SET_REACTANT(reactant_handle, 1, &species);
00079   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_REACTANT\n");
00080 }
```

Here is the call graph for this function:

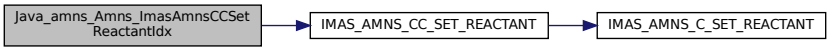
16.95.1.13 Java_amns_Amns_ImasAmnsCCSetReactantIdx()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactantIdx (
            JNIEnv * ,
            jobject ,
            jlong ,
            jint ,
            jobject  )
```

Definition at line 82 of file amns_jni_call.c.

```
00087                                                                                {
00088
00089   void *reactant_handle = (void*) ptr;
00090
00091   amns_c_reactant_type species  = DEFAULT_AMNS_C_REACTANT_TYPE;
00092
00093   jclass cls = (*env)->GetObjectClass(env, acrt);
00094   jfieldID paramZN = (*env)->GetFieldID(env, cls, "ZN", "D");
00095   jfieldID paramZA = (*env)->GetFieldID(env, cls, "ZA", "D");
00096   jfieldID paramMI = (*env)->GetFieldID(env, cls, "MI", "D");
00097   jfieldID paramLR = (*env)->GetFieldID(env, cls, "LR", "I");
00098   jfieldID paramRealSpecifier = (*env)->GetFieldID(env, cls, "real_specifier", "D");
00099   jfieldID paramIntSpecifier = (*env)->GetFieldID(env, cls, "int_specifier", "I");
00100
00101   jdouble zn = (*env)->GetDoubleField(env, acrt, paramZN);
00102   jdouble za = (*env)->GetDoubleField(env, acrt, paramZA);
00103   jdouble mi = (*env)->GetDoubleField(env, acrt, paramMI);
00104   jint lr = (*env)->GetIntField(env, acrt, paramLR);
00105   jdouble rs = (*env)->GetDoubleField(env, acrt, paramRealSpecifier);
00106   jint int_spec = (*env)->GetIntField(env, acrt, paramIntSpecifier);
00107
00108   // DEBUG printf("[JNI] Values from Java: %f:%f:%f:%i:%f:%i\n", zn, za, mi, lr, rs, int_spec);
00109
00110   species.ZN=zn; species.ZA=za; species.MI=mi; species.LR=lr; species.real_specifier=rs;
      species.int_specifier=int_spec;
00111   // DEBUG printf("[JNI] Calling IMAS_AMNS_CC_SET_REACTANT with index: %d\n", idx);
00112   IMAS_AMNS_CC_SET_REACTANT(reactant_handle, idx, &species);
00113   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_REACTANT\n");
00114 }
```

Here is the call graph for this function:



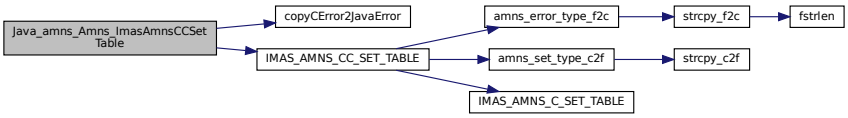
16.95.1.14 Java_amns_Amns_ImasAmnsCCSetTable()

```
JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetTable (
            JNIEnv * ,
            jobject ,
            jlong ,
            jobject ,
            jobject  )
```

Definition at line 578 of file amns_jni_call.c.

```
00583                                                                                {
00584   amns_c_set_type set = DEFAULT_AMNS_C_SET_TYPE;
00585   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00586   void *amns_cx_handler = (void *) ptrAmnsCxHandle;
00587
00588   jclass cls = (*env)->GetObjectClass(env, j_amns_set_type);
00589   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00590
00591   jstring j_set_type_string = (*env)->GetObjectField(env, j_amns_set_type, paramString);
00592
00593   char *set_type_string = (char*)(*env)->GetStringUTFChars(env, j_set_type_string, NULL);
00594
00595   set.string = set_type_string;
00596
00597   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SET_TABLE\n");
00598   // DEBUG printf("[JNI] amns_cx_handle: %lu\n", (long)ptrAmnsCxHandle);
00599   IMAS_AMNS_CC_SET_TABLE(amns_cx_handler, &set, &error_stat);
00600   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SET_TABLE\n");
00601
00602   copyCError2JavaError( &error_stat, env, j_error_state);
00603
00604   (*env)->ReleaseStringUTFChars(env, j_set_type_string, set_type_string);
00605 }
```

Here is the call graph for this function:



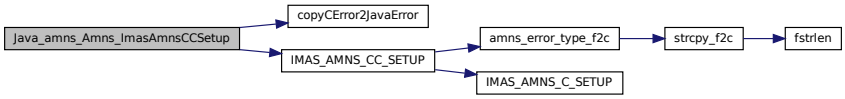
16.95.1.15 Java_amns_Amns_ImasAmnsCCSetup()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetup (
            JNIEnv * ,
            jobject ,
            jobject  )
```

Definition at line 152 of file amns_jni_call.c.

```
00152
        {
00153   void* amns_handle = NULL;
00154   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00155
00156   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SETUP\n");
00157   IMAS_AMNS_CC_SETUP(&amns_handle, &error_stat);
00158   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP\n");
00159   // DEBUG printf("[JNI] Pointer: %lu\n", (long)amns_handle);
00160
00161   copyCError2JavaError( &error_stat, env, error_structure );
00162
00163   return (long) amns_handle;
00164 }
```

Here is the call graph for this function:



16.95.1.16 Java_amns_Amns_ImasAmnsCcSetupReactants()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactants (
            JNIEnv * ,
            jobject ,
            jint  )
```

Definition at line 36 of file amns_jni_call.c.

```
00036                                                                                                {
00037   void *reactant_handle;
00038   IMAS_AMNS_CC_SETUP_REACTANTS(&reactant_handle, "", idx, 1);
00039   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_REACTANTS\n");
00040   // DEBUG printf("[JNI] Pointer: %lu\n", (long)reactant_handle);
00041   return (long)reactant_handle;
00042 }
```

Here is the call graph for this function:

16.95.1.17 Java_amns_Amns_ImasAmnsCcSetupReactantsNumber()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactantsNumber (
            JNIEnv * ,
            jobject ,
            jint ,
            jint  )
```

Definition at line 44 of file amns_jni_call.c.

```
00044
                    {
00045   void *reactant_handle;
00046   // DEBUG printf("[JNI] Calling IMAS_AMNS_CC_SETUP_REACTANTS with number of reactions: %d\n",
        number);
00047   IMAS_AMNS_CC_SETUP_REACTANTS(&reactant_handle, "", idx, number);
00048   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_REACTANTS\n");
00049   // DEBUG printf("[JNI] Pointer: %lu\n", (long)reactant_handle);
00050   return (long)reactant_handle;
00051 }
```

Here is the call graph for this function:

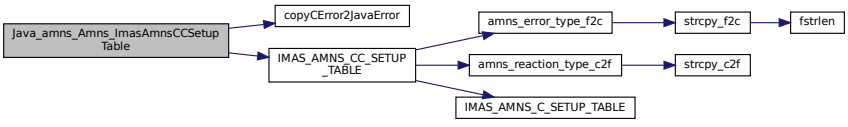
16.95.1.18 Java_amns_Amns_ImasAmnsCCSetupTable()

```
JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetupTable (
            JNIEnv * ,
            jobject ,
            jlong ,
            jobject ,
            jlong ,
            jobject  )
```

Definition at line 166 of file amns_jni_call.c.

```
00172                                                                                {
00173   // declare structures used in call
00174   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00175   void* amns_cx_handle;
00176   amns_c_reaction_type xx_rx;
00177
00178   // get pointers
00179   void *amns_handle = (void*) ptr_amns_handle;
00180   void *reactants_handle = (void*) ptr_reactants_handle;
00181
00182   // retrieve and copy values from reaction_type object
00183   jclass cls = (*env)->GetObjectClass(env, j_reaction_type);
00184   jfieldID paramString = (*env)->GetFieldID(env, cls, "string", "Ljava/lang/String;");
00185   jfieldID paramIsotope = (*env)->GetFieldID(env, cls, "isotopeResolved", "I");
00186
00187   jstring j_reaction_string = (*env)->GetObjectField(env, j_reaction_type, paramString);
00188   jint j_reaction_isotope = (*env)->GetIntField(env, j_reaction_type, paramIsotope);
00189
00190   char *reaction_string = (char*) (*env)->GetStringUTFChars(env, j_reaction_string, NULL);
00191   int reaction_isotope = (int) j_reaction_isotope;
00192
00193   xx_rx.string = reaction_string;
00194   xx_rx.isotope_resolved = reaction_isotope;
00195
00196   // DEBUG printf("[JNI] Reaction type = %s, %u\n", xx_rx.string, xx_rx.isotope_resolved);
00197
00198   // we can now call C function
00199   // note that we will return amns_cx_handle as return value from function
00200   // DEBUG printf("[JNI] Before calling IMAS_AMNS_CC_SETUP_TABLE\n");
00201   // DEBUG printf("[JNI] amns_handle pointer: %lu\n", (long)amns_handle);
00202   // DEBUG printf("[JNI] reactants_handle pointer: %lu\n", (long)reactants_handle);
00203   IMAS_AMNS_CC_SETUP_TABLE(amns_handle, &xx_rx, reactants_handle, &amns_cx_handle, &error_stat);
00204
00205   // DEBUG printf("[JNI] After calling IMAS_AMNS_CC_SETUP_TABLE\n");
00206   // DEBUG printf("[JNI] reaction_handle pointer: %lu\n", (long)amns_cx_handle);
00207
00208   // we have to make sure that error_stat is passed back inside object
00209
00210   copyCError2JavaError( &error_stat, env, j_error_state);
00211
00212   // remember to release string
00213   (*env)->ReleaseStringUTFChars(env, j_reaction_string, reaction_string);
00214
00215   return (long) amns_cx_handle;
00216 }
```

Here is the call graph for this function:



16.96 amns_jni_call.h

```
00001 /* DO NOT EDIT THIS FILE - it is machine generated */
00002 #include <jni.h>
00003 /* Header for class amns_Amns */
00004
00005 #ifndef _Included_amns_Amns
```

```
00006 #define _Included_amns_Amns
00007 #ifdef __cplusplus
00008 extern "C" {
00009 #endif
00010 /*
00011  * Class:     amns_Amns
00012  * Method:    ImasAmnsCcSetupReactants
00013  * Signature: (I)J
00014  */
00015 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactants
00016   (JNIEnv *, jobject, jint);
00017
00018 /*
00019  * Class:     amns_Amns
00020  * Method:    ImasAmnsCcSetupReactantsNumber
00021  * Signature: (II)J
00022  */
00023 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCcSetupReactantsNumber
00024   (JNIEnv *, jobject, jint, jint);
00025
00026 /*
00027  * Class:     amns_Amns
00028  * Method:    ImasAmnsCCSetReactant
00029  * Signature: (JLamns/type/AmnsReactantType;)V
00030  */
00031 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactant
00032   (JNIEnv *, jobject, jlong, jobject);
00033
00034 /*
00035  * Class:     amns_Amns
00036  * Method:    ImasAmnsCCSetReactantIdx
00037  * Signature: (JILamns/type/AmnsReactantType;)V
00038  */
00039 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetReactantIdx
00040   (JNIEnv *, jobject, jlong, jint, jobject);
00041
00042 /*
00043  * Class:     amns_Amns
00044  * Method:    ImasAmnsCCSetup
00045  * Signature: (Lamns/type/AmnsErrorType;)J
00046  */
00047 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetup
00048   (JNIEnv *, jobject, jobject);
00049
00050 /*
00051  * Class:     amns_Amns
00052  * Method:    ImasAmnsCCGetReactant
00053  * Signature: (JLamns/type/AmnsReactantType;)V
00054  */
00055 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCGetReactant
00056   (JNIEnv *, jobject, jlong, jobject);
00057
00058 /*
00059  * Class:     amns_Amns
00060  * Method:    ImasAmnsCCRX0B
00061  * Signature: (JDDLamns/type/AmnsErrorType;)D
00062  */
00063 JNIEXPORT jdouble JNICALL Java_amns_Amns_ImasAmnsCCRX0B
00064   (JNIEnv *, jobject, jlong, jdouble, jdouble, jobject);
00065
00066 /*
00067  * Class:     amns_Amns
00068  * Method:    ImasAmnsCCRX1A
00069  * Signature: (JI[DLamns/type/AmnsErrorType;)[D
00070  */
00071 JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1A
00072   (JNIEnv *, jobject, jlong, jint, jdoubleArray, jobject);
00073
00074 /*
00075  * Class:     amns_Amns
00076  * Method:    ImasAmnsCCRX1B
00077  * Signature: (JI[D[DLamns/type/AmnsErrorType;)[D
00078  */
00079 JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1B
00080   (JNIEnv *, jobject, jlong, jint, jdoubleArray, jdoubleArray, jobject);
00081
00082 /*
00083  * Class:     amns_Amns
00084  * Method:    ImasAmnsCCRX1C
00085  * Signature: (JI[D[D[DLamns/type/AmnsErrorType;)[D
00086  */
00087 JNIEXPORT jdoubleArray JNICALL Java_amns_Amns_ImasAmnsCCRX1C
00088   (JNIEnv *, jobject, jlong, jint, jdoubleArray, jdoubleArray, jdoubleArray, jobject);
00089
00090 /*
00091  * Class:     amns_Amns
00092  * Method:    ImasAmnsCCFinishTable
```

```
00093  * Signature: (JLamns/type/AmnsErrorType;)J
00094  */
00095 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishTable
00096   (JNIEnv *, jobject, jlong, jobject);
00097
00098 /*
00099  * Class:     amns_Amns
00100  * Method:    ImasAmnsCCFinishReactants
00101  * Signature: (J)J
00102  */
00103 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinishReactants
00104   (JNIEnv *, jobject, jlong);
00105
00106 /*
00107  * Class:     amns_Amns
00108  * Method:    ImasAmnsCCFinish
00109  * Signature: (JLamns/type/AmnsErrorType;)J
00110  */
00111 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCFinish
00112   (JNIEnv *, jobject, jlong, jobject);
00113
00114 /*
00115  * Class:     amns_Amns
00116  * Method:    ImasAmnsCCSetupTable
00117  * Signature: (JLamns/type/AmnsReactionType;JLamns/type/AmnsErrorType;)J
00118  */
00119 JNIEXPORT jlong JNICALL Java_amns_Amns_ImasAmnsCCSetupTable
00120   (JNIEnv *, jobject, jlong, jobject, jlong, jobject);
00121
00122 /*
00123  * Class:     amns_Amns
00124  * Method:    ImasAmnsCCSet
00125  * Signature: (JLamns/type/AmnsSetType;Lamns/type/AmnsErrorType;)V
00126  */
00127 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSet
00128   (JNIEnv *, jobject, jlong, jobject, jobject);
00129
00130 /*
00131  * Class:     amns_Amns
00132  * Method:    ImasAmnsCCSetTable
00133  * Signature: (JLamns/type/AmnsSetType;Lamns/type/AmnsErrorType;)V
00134  */
00135 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCSetTable
00136   (JNIEnv *, jobject, jlong, jobject, jobject);
00137
00138 /*
00139  * Class:     amns_Amns
00140  * Method:    ImasAmnsCCQuery
00141  * Signature: (JLamns/type/AmnsQueryType;Lamns/type/AmnsAnswerType;Lamns/type/AmnsErrorType;)V
00142  */
00143 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQuery
00144   (JNIEnv *, jobject, jlong, jobject, jobject, jobject);
00145
00146 /*
00147  * Class:     amns_Amns
00148  * Method:    ImasAmnsCCQueryTable
00149  * Signature: (JLamns/type/AmnsQueryType;Lamns/type/AmnsAnswerType;Lamns/type/AmnsErrorType;)V
00150  */
00151 JNIEXPORT void JNICALL Java_amns_Amns_ImasAmnsCCQueryTable
00152   (JNIEnv *, jobject, jlong, jobject, jobject, jobject);
00153
00154 #ifdef __cplusplus
00155 }
00156 #endif
00157 #endif
```

16.97 src/libamns/amns_module.f90 File Reference

Data Types

- interface amns_module::imas_amns_rx

  get the rates associated with the input args for a particular reaction (generic interface for 1d, 2d & 3d arrays)

Modules

- module amns_module

Functions/Subroutines

- subroutine amns_module::errorstop (error_message)
- subroutine amns_module::imas_amns_setup (handle, version, error_status)

  initialization call for the AMNS package
- subroutine amns_module::imas_amns_setup_table (handle, reaction_type, reactants, handle_rx, error_↩ status)

  initialization call for a particular reaction
- subroutine amns_module::imas_amns_finish (handle, error_status)

  finalization call for the AMNS package
- subroutine amns_module::imas_amns_finish_table (handle_rx, error_status)

  finalization call for a particular reaction
- subroutine amns_module::imas_amns_query (handle, query, answer, error_status)

  query routine for the AMNS package
- subroutine amns_module::imas_amns_query_table (handle_rx, query, answer, error_status)

  query routine for a particular reaction
- subroutine amns_module::imas_amns_set (handle, set, error_status)

  set a parameter for the AMNS package
- subroutine amns_module::imas_amns_set_table (handle_rx, set, error_status)

  set a parameter for a particular reaction
- subroutine amns_module::imas_amns_rx_0 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (0d) args for a particular reaction
- subroutine amns_module::imas_amns_rx_1 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (1d) args for a particular reaction
- subroutine amns_module::imas_amns_rx_2 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (2d) args for a particular reaction
- subroutine amns_module::imas_amns_rx_3 (handle_rx, out, arg1, arg2, arg3, arg4, error_status)

  get the rates associated with the input (3d) args for a particular reaction

Variables

- integer, save amns_module::version_no
- character(len=256), save amns_module::user
- character(len=32), save amns_module::ds_version ='3'
- character(len=32), save amns_module::backend ='mdsplus'
- type(ids_amns_data), save amns_module::amns00

16.98 amns_module.f90

```
00001
00008
00009 module amns_module
00010
00011   use ids_schemas  ! IGNORE
00012   use ids_routines  ! IGNORE
00013   use amns_types
00014
00015   implicit none
00016
00028   interface imas_amns_rx
00029
00030      module procedure imas_amns_rx_0, imas_amns_rx_1, imas_amns_rx_2, imas_amns_rx_3
00031
00032   end interface
00033
00034   integer, save                  :: version_no
00035   character (len=256), save      :: user
00036   character (len=32), save       :: ds_version='3', backend='mdsplus'
00037   type (ids_amns_data), save     :: amns00
00038
00039 contains
00040
```

```
00041   subroutine errorstop(error_message)
00042     character (len=*) :: error_message
00043     write(0,*) 'Error - ' // error_message
00044     stop 1
00045   end subroutine errorstop
00046
00047   !
      ==========================================================================================================
      !
00053   subroutine imas_amns_setup(handle, version, error_status)
00054     use ids_types  ! IGNORE
00055 !    use amns_types
00056     use amns_utility
00057     use ids_schemas   ! IGNORE
00058     use ids_routines  ! IGNORE
00059     use interface_to_amns
00060 !    use read_structures ! IGNORE
00061     implicit none
00062     optional version, error_status
00063     type(amns_handle_type), intent(out) :: handle
00064     type(amns_version_type), intent(in) :: version
00065     type(amns_error_type), intent(out)  :: error_status
00066
00067     type(amns_version_type) :: default_version
00068
00069 #ifdef AL
00070 !NLL    integer :: imas_open_env, imas_open_hdf5, imas_create_env, imas_create_hdf5
00071     integer :: imas_status
00072 #endif
00073     integer :: idx
00074     integer :: shot, run
00075     character (len=256) :: file
00076     character(len=3)    :: treename = 'ids'
00077     character(len=9)    :: amnspath = 'amns_data'
00078     character(STRMAXLEN) :: uri
00079     character(len=32) :: amns_debug_env
00080     logical :: amns_debug
00081
00082     write(*,'(a)') 'AMNS Library code version "' // trim(git_version_amns) // '"'
00083
00084     if(present(error_status)) then
00085         error_status%flag=.false.
00086         error_status%string="No error"
00087     end if
00088
00089     call getenv('IMAS_AMNS_DEBUG', amns_debug_env)
00090     amns_debug = amns_debug_env.eq."yes" .or. amns_debug_env.eq."YES" .or. amns_debug_env.eq."Yes"
00091     if (amns_debug) handle%debug = .true.
00092     amns_debug = amns_debug_env.eq."no" .or. amns_debug_env.eq."NO" .or. amns_debug_env.eq."No"
00093     if (amns_debug) handle%debug = .false.
00094
00095     if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP start'
00096
00097     ! by default, we get all data from system
00098     ! then, we can overwrite it using values passed as arguments
00099     default_version%string='DEFAULT'
00100     if(handle%debug) call getenv('USER', user)
00101     default_version%user = user
00102     call getenv('IMAS_VERSION', ds_version)
00103     if(handle%debug) write(*,*) 'Using DATAVERSION ', trim(ds_version)
00104
00105     if(present(version)) then
00106        if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP: requested database = ',trim(version%string),
      version%number
00107        handle%version=version
00108        version_no = version%number
00109        if(version%user .eq. '') then
00110           handle%version%USER = user
00111           if(handle%debug) write(*,*) 'Reset USER to ', trim(user)
00112        endif
00113     else
00114        handle%version=default_version
00115        version_no = -1
00116     endif
00117     if(handle%version%backend .eq. '') then
00118        handle%version%backend = backend
00119     endif
00120
00121 ! get the index information from 0/1
00122     shot=0
00123     run=1
00124     if(handle%version%backend.eq.'ascii') then
00125 !       write(file,'(a,"_",i6.6,"_",i6.6,".IDS")') 'amns',shot,run
00126 !       write(*,*) 'Reading data from ', trim(file), ' for ', shot, run
00127 !       call open_read_file(1, trim(file))
00128 !       call read_ids(amns00, 'amns')
00129 !       call close_read_file
```

```
00130        call errorstop('Not supported in the IDS version')
00131 #ifdef AL
00132     elseif(handle%version%backend.eq.'hdf5') then
00133 !       imas_status = imas_open_hdf5(treename, shot, run, idx)
00134 !       write(*,*) 'IMAS_STATUS, IDX = ', imas_status, idx
00135 !       call ids_get(idx, amnspath, amns00)
00136 !       call imas_close(idx)
00137        call errorstop('HDF5 not supported in the IDS version')
00138     elseif(handle%version%backend.eq.'mdsplus') then
00139        if(handle%debug) write(*,*) 'Attempting to access data from ',  shot, run,  &
00140             trim(handle%version%USER), ' amns ', trim(ds_version)
00141 !NLL       imas_status = imas_open_env(treename, shot, run, idx, &
00142 !NLL            trim(handle%version%USER), 'amns', trim(ds_version))
00143        !call ual_begin_pulse_action(MDSPLUS_BACKEND, shot, run, &
00144        !     trim(handle%version%USER), 'amns', trim(ds_version), idx)
00145        !call ual_open_pulse(idx, OPEN_PULSE, '', imas_status)
00146        call al_build_uri_from_legacy_parameters(mdsplus_backend, shot, run, &
00147             trim(handle%version%USER), 'amns', trim(ds_version), '', uri, imas_status)
00148        call al_begin_dataentry_action(uri, open_pulse, idx, imas_status)
00149        if(handle%debug) write(*,*) 'IMAS_STATUS, IDX = ', imas_status, idx
00150        if(imas_status.eq.-1) then
00151           if(handle%debug) write(0,*) 'Failure opening IDS!'
00152           ! stop 1
00153        endif
00154        if(imas_status.ne.0) then
00155           handle%version%USER = 'public'
00156           if(handle%debug) write(*,*) 'Attempting to access data from ',  shot, run,  &
00157                trim(handle%version%USER), ' amns ', trim(ds_version)
00158 !NLL          imas_status = imas_open_env(treename, shot, run, idx, &
00159 !NLL               trim(handle%version%USER), 'amns', trim(ds_version))
00160           !call ual_begin_pulse_action(MDSPLUS_BACKEND, shot, run, &
00161           !     trim(handle%version%USER), 'amns', trim(ds_version), idx)
00162           !call ual_open_pulse(idx, OPEN_PULSE, '', imas_status)
00163           call al_build_uri_from_legacy_parameters(mdsplus_backend, shot, run, &
00164                trim(handle%version%USER), 'amns', trim(ds_version), '', uri, imas_status)
00165           call al_begin_dataentry_action(uri, open_pulse, idx, imas_status)
00166
00167           if(handle%debug) write(*,*) 'IMAS_STATUS, IDX = ', imas_status, idx
00168           if(imas_status.ne.0) then
00169              if(present(error_status)) then
00170                 error_status%flag=.true.
00171                 error_status%string='Could not find any AMNS data, even under "public"'
00172                 return
00173              else
00174                 call errorstop('Could not find any AMNS data, even under "public"')
00175              end if
00176           endif
00177        endif
00178        call ids_get(idx, amnspath, amns00)
00179        call imas_close(idx)
00180 #else
00181     else
00182        write(*,*) 'The version of the AMNS routines without the AL was compiled'
00183        write(*,*) 'Use the ascii backend'
00184        if(present(error_status)) then
00185           error_status%flag=.true.
00186           error_status%string='UAL called in a non-AL version of the code'
00187           return
00188        else
00189           call errorstop('UAL called in a non-AL version of the code')
00190        end if
00191 #endif
00192     endif
00193     handle%properties_comment = assign_if_associated(amns00%ids_properties%comment, "Not specified in
       the AMNS IDS")
00194     handle%properties_source = assign_if_associated(amns00%ids_properties%source, "Not specified in
       the AMNS IDS")
00195     handle%properties_provider = assign_if_associated(amns00%ids_properties%provider, "Not specified
       in the AMNS IDS")
00196     handle%properties_creation_date = assign_if_associated(amns00%ids_properties%creation_date, "Not
       specified in the AMNS IDS")
00197     handle%code_name = assign_if_associated(amns00%code%name, "Not specified in the AMNS IDS")
00198     handle%code_commit = assign_if_associated(amns00%code%commit, "Not specified in the AMNS IDS")
00199     handle%code_version = assign_if_associated(amns00%code%version, "Not specified in the AMNS IDS")
00200     handle%code_repository = assign_if_associated(amns00%code%repository, "Not specified in the AMNS
       IDS")
00201
00202     if(.not.associated(amns00%release)) then
00203        if(present(error_status)) then
00204           error_status%flag=.true.
00205           error_status%string="No version information in INDEX shot"
00206           return
00207        else
00208           call errorstop("No version information in INDEX shot")
00209        end if
00210     endif
00211     if(version_no.le.0) then
```

```
00212        version_no=size(amns00%release)
00213        if(handle%debug) write(*,*) 'Found global version # ', version_no
00214     else
00215        if(version_no.gt.size(amns00%release)) then
00216           write(*,*) 'Requested version out of range ', version_no, ' > ', size(amns00%release)
00217           if(present(error_status)) then
00218              error_status%flag=.true.
00219              error_status%string="Requested version out of range"
00220              return
00221           else
00222              call errorstop('Requested version out of range')
00223           end if
00224        endif
00225     endif
00226     handle%version%string = trim(handle%version%USER) // ': ' // trim(handle%version%backend)
00227     handle%version%number = version_no
00228
00229     if(present(error_status)) then
00230        if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP: requested error_status response'
00231     endif
00232     handle%initialized=.true.
00233     if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP end'
00234
00235   end subroutine imas_amns_setup
00236
00237   !
      ================================================================================================================
      !
00245   subroutine imas_amns_setup_table(handle, reaction_type, reactants, handle_rx, error_status)
00246     use ids_types  ! IGNORE
00247     use amns_types
00248     use amns_utility
00249     use interface_to_amns
00250     ! do not remove the comment!!!
00251     use ids_schemas   ! IGNORE
00252     use ids_routines  ! IGNORE
00253
00254     implicit none
00255     optional error_status
00256     type(amns_handle_type), intent(in) :: handle
00257     type(amns_reaction_type), intent(in) :: reaction_type
00258     type(amns_reactants_type), intent(in) :: reactants
00259     type(amns_handle_rx_type), intent(out) :: handle_rx
00260     type(amns_error_type), intent(out) :: error_status
00261
00262     integer :: no_of_reactants, i, is
00263     integer :: izn, izm, shot, run, ierr
00264     character*128 :: data_file, error_description
00265     integer :: iversion, nrelease, irelease
00266
00267     if(present(error_status)) then
00268        error_status%flag=.false.
00269        error_status%string="No error"
00270     end if
00271
00272     handle_rx%debug= handle%debug
00273     handle_rx%reaction_type= reaction_type%string
00274     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE start'
00275     izn=0
00276     izm=0
00277     if(allocated(reactants%components)) then
00278        no_of_reactants=size(reactants%components)
00279        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE: number of reactants = ',no_of_reactants
00280        do i=1,no_of_reactants
00281 ! only reactants, not products
00282           if(reactants%components(i)%lr .eq. 0 .and. &
00283                (reactants%components(i)%int_specifier .ge. 0 .or.
       reactants%components(i)%int_specifier .eq. ids_int_invalid)) then
00284              if(nint(reactants%components(i)%ZN) .gt. izn) then
00285                 izn=nint(reactants%components(i)%ZN)
00286                 if(reaction_type%isotope_resolved .ne. 0) izm=nint(reactants%components(i)%MI)
00287              else if(nint(reactants%components(i)%ZN) .eq. izn .and. reaction_type%isotope_resolved
       .ne. 0) then
00288                 izm=max(izm,nint(reactants%components(i)%MI))
00289              endif
00290           endif
00291           if(handle_rx%debug) &
00292                write(*,'(a,i3,3(1x,f6.2),i2)') 'IMAS_AMNS_SETUP_TABLE: reactant#, ZN, ZA, MI, LR = ',
       &
00293                i,reactants%components(i)%ZN, &
00294                reactants%components(i)%ZA, &
00295                reactants%components(i)%MI, &
00296                reactants%components(i)%lr
00297        enddo
00298        handle_rx%no_of_reactants=no_of_reactants
00299        allocate(handle_rx%components(no_of_reactants))
00300        handle_rx%components=reactants%components
```

```
00301     else
00302        no_of_reactants=0
00303        handle_rx%no_of_reactants=0
00304     endif
00305     if(allocated(reactants%string)) then
00306        allocate(handle_rx%string(size(reactants%string)))
00307        handle_rx%string=reactants%string
00308     endif
00309     if(present(error_status)) then
00310        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE: requested error_status response'
00311     endif
00312  ! find local version number for shot
00313     shot=izn+1000*izm
00314     if(shot.eq.0) then
00315        if(present(error_status)) then
00316           error_status%flag=.true.
00317           return
00318        else
00319           write(*,*) 'Could not calculate the AMNS shot number'
00320           if(present(error_status)) then
00321              error_status%flag=.true.
00322              error_status%string="Could not calculate the AMNS shot number"
00323              return
00324           else
00325              call errorstop('Could not calculate the AMNS shot number')
00326           end if
00327        endif
00328     endif
00329     run=0
00330     iversion=handle%version%number
00331     if(associated(amns00%release(iversion)%data_entry)) then
00332        nrelease=size(amns00%release(iversion)%data_entry)
00333        do irelease=1, nrelease   ! loop over data in a version
00334           if(amns00%release(iversion)%data_entry(irelease)%shot .eq. shot) then
00335              run=amns00%release(iversion)%data_entry(irelease)%run
00336              exit
00337           endif
00338        enddo
00339     endif
00340     if(run.eq.0) then
00341        if(present(error_status)) then
00342           error_status%flag=.true.
00343           write(error_status%string,*) 'No data found for ', shot, ' in version ', iversion
00344           return
00345        else
00346           write(*,*) 'No data found for ', shot, ' in version ', iversion
00347           call errorstop('No data found')
00348        endif
00349     else
00350        if(handle_rx%debug) write(*,*) 'Found local version ', run, ' for case = ', shot
00351        handle_rx%version%number = run
00352        ! now handle the various reaction possibilities
00353
00354        call get_amns_data(reaction_type, reactants, handle_rx%grid, &
00355             handle_rx%properties_comment, handle_rx%properties_source, &
00356             handle_rx%properties_provider, handle_rx%properties_creation_date, &
00357             handle_rx%code_name, handle_rx%code_commit, &
00358             handle_rx%code_version, handle_rx%code_repository, &
00359             handle_rx%source, handle_rx%provider, handle_rx%citation, &
00360             shot, run, handle%version%backend, handle%version%user, ds_version, &
00361             ierr, error_description, handle_rx%debug)
00362        if(ierr.eq.0) then
00363           handle_rx%filled=.true.
00364        else
00365           if(present(error_status)) then
00366              error_status%flag=.true.
00367              error_status%string="'get_amns_data' returned an error - " // error_description
00368              return
00369           else
00370              stop "'get_amns_data' returned an error"
00371           end if
00372        endif
00373        handle_rx%initialized=.true.
00374        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE end'
00375     endif
00376   end subroutine imas_amns_setup_table
00377
00378   !
      ! ================================================================================================================
      !
00383   subroutine imas_amns_finish(handle, error_status)
00384     use ids_types  ! IGNORE
00385     use amns_types
00386     use amns_utility
00387     use interface_to_amns
00388 !    use deallocate_structures ! IGNORE
00389     implicit none
```

```
00390     optional error_status
00391     type(amns_handle_type), intent(inout) :: handle
00392     type(amns_error_type), intent(out) :: error_status
00393
00394     if(present(error_status)) then
00395         error_status%flag=.false.
00396         error_status%string="No error"
00397     end if
00398
00399     if(handle%debug) write(*,*) 'IMAS_AMNS_FINISH start'
00400     if(handle%initialized) then
00401        handle%initialized=.false.
00402     else
00403        if(handle%debug) write(*,*) 'IMAS_AMNS_FINISH: Attempt to FINISH an uninitialized case'
00404     endif
00405     if(present(error_status)) then
00406        if(handle%debug) write(*,*) 'IMAS_AMNS_SETUP_TABLE: requested error_status response'
00407     endif
00408     call end_amns_data
00409     call ids_deallocate(amns00)
00410     if(handle%debug) write(*,*) 'IMAS_AMNS_FINISH end'
00411
00412   end subroutine imas_amns_finish
00413
00414   !
      ================================================================================================================
      !
00419   subroutine imas_amns_finish_table(handle_rx, error_status)
00420     use ids_types  ! IGNORE
00421     use amns_types
00422     implicit none
00423     optional error_status
00424     type(amns_handle_rx_type), intent(inout) :: handle_rx
00425     type(amns_error_type), intent(out) :: error_status
00426
00427     if(present(error_status)) then
00428         error_status%flag=.false.
00429         error_status%string="No error"
00430     end if
00431
00432     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_FINISH_TABLE start'
00433     if(handle_rx%initialized) then
00434        handle_rx%initialized=.false.
00435        if(allocated(handle_rx%components)) then
00436           deallocate(handle_rx%components)
00437        endif
00438        handle_rx%no_of_reactants=0
00439        if(allocated(handle_rx%string)) then
00440           deallocate(handle_rx%string)
00441        endif
00442        call delete(handle_rx%grid)
00443     else
00444        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_FINISH_TABLE: Attempt to FINISH an uninitialized
      table'
00445     endif
00446     if(present(error_status)) then
00447        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_FINISH_TABLE: requested error_status response'
00448     endif
00449     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_FINISH_TABLE end'
00450
00451   end subroutine imas_amns_finish_table
00452
00453   !
      ================================================================================================================
      !
00470   subroutine imas_amns_query(handle,query,answer,error_status)
00471     use ids_types  ! IGNORE
00472     use amns_types
00473     implicit none
00474     optional error_status
00475     type(amns_handle_type), intent(in) :: handle
00476     type(amns_query_type), intent(in) :: query
00477     type(amns_answer_type), intent(out) :: answer
00478     type(amns_error_type), intent(out) :: error_status
00479
00480     if(present(error_status)) then
00481         error_status%flag=.false.
00482         error_status%string="No error"
00483     end if
00484
00485     if(handle%debug) write(*,*) 'IMAS_AMNS_QUERY start'
00486     if(handle%debug) write(*,*) 'IMAS_AMNS_QUERY: query = ',trim(query%string)
00487 ! The choices are documented in the doxygen block above
00488     select case (query%string)
00489     case ("version")               ! \property version: information about the version
00490        answer%string=handle%version%string
00491        answer%number=handle%version%number
```

```
00492     case ("prop_comment")               ! \property properties_comment: information about the version
00493        answer%string=handle%properties_comment
00494        answer%number=0
00495     case ("prop_source")               ! \property properties_source: information about the version
00496        answer%string=handle%properties_source
00497        answer%number=0
00498     case ("prop_provider")               ! \property properties_provider: information about the
      version
00499        answer%string=handle%properties_provider
00500        answer%number=0
00501     case ("prop_creation")               ! \property properties_creation_date: information about the
      version
00502        answer%string=handle%properties_creation_date
00503        answer%number=0
00504     case ("code_name")               ! \property code_name: information about the version
00505        answer%string=handle%code_name
00506        answer%number=0
00507     case ("code_commit")               ! \property code_commit: information about the version
00508        answer%string=handle%code_commit
00509        answer%number=0
00510     case ("code_version")               ! \property code_version: information about the version
00511        answer%string=handle%code_version
00512        answer%number=0
00513     case ("code_repository")               ! \property code_repository: information about the version
00514        answer%string=handle%code_repository
00515        answer%number=0
00516     case default
00517        answer%string='Not implemented yet'
00518     end select
00519     if(present(error_status)) then
00520        if(handle%debug) write(*,*) 'IMAS_AMNS_QUERY: requested error_status response'
00521     endif
00522     if(handle%debug) write(*,*) 'IMAS_AMNS_QUERY end'
00523
00524   end subroutine imas_amns_query
00525
00526   !
      ================================================================================================================
      !
00557   subroutine imas_amns_query_table(handle_rx,query,answer,error_status)
00558     use ids_types  ! IGNORE
00559     use amns_types
00560     use amns_utility
00561     implicit none
00562     optional error_status
00563     type(amns_handle_rx_type), intent(in) :: handle_rx
00564     type(amns_query_type), intent(in) :: query
00565     type(amns_answer_type), intent(out) :: answer
00566     type(amns_error_type), intent(out) :: error_status
00567     integer ir
00568
00569     if(present(error_status)) then
00570         error_status%flag=.false.
00571         error_status%string="No error"
00572     end if
00573
00574     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_QUERY_TABLE start'
00575     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_QUERY_TABLE: query = ',trim(query%string)
00576 ! The choices are documented in the doxygen block above
00577     select case (query%string)
00578     case ("source")
00579        answer%string=handle_rx%source
00580        answer%number=0
00581     case ("provider")
00582        answer%string=handle_rx%provider
00583        answer%number=0
00584     case ("citation")
00585        answer%string=handle_rx%citation
00586        answer%number=0
00587     case ("no_of_reactants")
00588        answer%string=string(handle_rx%no_of_reactants)
00589        answer%number=handle_rx%no_of_reactants
00590     case ("index")
00591        answer%string=string(handle_rx%index)
00592        answer%number=handle_rx%index
00593     case ("filled")
00594        if(handle_rx%filled) then
00595           answer%string= 'Filled'
00596        else
00597           answer%string= 'Empty'
00598        endif
00599     case ("reaction_type")
00600        answer%string= handle_rx%reaction_type
00601     case ("reactants")
00602        answer%string=""
00603        do ir=1,handle_rx%no_of_reactants
00604           answer%string= trim(answer%string) // " " // &
```

```
00605               trim(string(nint(handle_rx%components(ir)%ZN))) // "/" // &
00606               trim(string(nint(handle_rx%components(ir)%ZA))) // "/" // &
00607               trim(string(nint(handle_rx%components(ir)%MI))) // "/" // &
00608               trim(string(handle_rx%components(ir)%LR))
00609          enddo
00610          answer%string= adjustl(answer%string)
00611       case ("coordinates")
00612          answer%string=""
00613          do ir=1,handle_rx%grid%ndim
00614             answer%string= trim(answer%string) // " " // &
00615               trim(handle_rx%grid%axe(ir)%label) // "/" // &
00616               trim(handle_rx%grid%axe(ir)%units)
00617          enddo
00618          answer%string= adjustl(answer%string)
00619       case ("version")
00620          answer%string=""
00621          answer%number=handle_rx%version%number
00622       case ("state_label")
00623          answer%string=handle_rx%grid%state_label
00624          answer%number=-1
00625       case ("result_unit")
00626          answer%string=handle_rx%grid%result_unit
00627          answer%number=-1
00628       case ("result_label")
00629          answer%string=handle_rx%grid%result_label
00630          answer%number=-1
00631       case ("ndim")
00632          answer%string=string(handle_rx%grid%ndim)
00633          answer%number=handle_rx%grid%ndim
00634       case ("interp_fun")
00635          answer%string=string(handle_rx%grid%interpol_function)
00636          answer%number=handle_rx%grid%ndim
00637       case ("prop_comment")                ! \property properties_comment: information about the version
00638          answer%string=handle_rx%properties_comment
00639          answer%number=0
00640       case ("prop_source")               ! \property properties_source: information about the version
00641          answer%string=handle_rx%properties_source
00642          answer%number=0
00643       case ("prop_provider")               ! \property properties_provider: information about the
    version
00644          answer%string=handle_rx%properties_provider
00645          answer%number=0
00646       case ("prop_creation")               ! \property properties_creation_date: information about the
    version
00647          answer%string=handle_rx%properties_creation_date
00648          answer%number=0
00649       case ("code_name")              ! \property code_name: information about the version
00650          answer%string=handle_rx%code_name
00651          answer%number=0
00652       case ("code_commit")               ! \property code_commit: information about the version
00653          answer%string=handle_rx%code_commit
00654          answer%number=0
00655       case ("code_version")               ! \property code_version: information about the version
00656          answer%string=handle_rx%code_version
00657          answer%number=0
00658       case ("code_repository")               ! \property code_repository: information about the version
00659          answer%string=handle_rx%code_repository
00660          answer%number=0
00661       case default
00662          answer%string='Not implemented yet'
00663       end select
00664       if(present(error_status)) then
00665          if(handle_rx%debug) write(*,*) 'IMAS_AMNS_QUERY_TABLE: requested error_status response'
00666       endif
00667       if(handle_rx%debug) write(*,*) 'IMAS_AMNS_QUERY_TABLE end'
00668
00669    end subroutine imas_amns_query_table
00670
00671    !
    ================================================================================================================
    !
00683    subroutine imas_amns_set(handle,set,error_status)
00684       use ids_types  ! IGNORE
00685       use amns_types
00686       implicit none
00687       optional error_status
00688       type(amns_handle_type), intent(inout) :: handle
00689       type(amns_set_type), intent(in) :: set
00690       type(amns_error_type), intent(out) :: error_status
00691
00692       if(present(error_status)) then
00693           error_status%flag=.false.
00694           error_status%string="No error"
00695       end if
00696
00697       if(handle%debug) write(*,*) 'IMAS_AMNS_SET start'
00698       if(handle%debug) write(*,*) 'IMAS_AMNS_SET: set = ',trim(set%string)
```

```
00699     if(present(error_status)) then
00700        if(handle%debug) write(*,*) 'IMAS_AMNS_SET_TABLE: requested error_status response'
00701     endif
00702 ! The choices are documented in the doxygen block above
00703     select case(set%string)
00704     case ("debug")
00705        handle%debug=.true.
00706     case ("nodebug")
00707        handle%debug=.false.
00708     case ("backend=mdsplus")
00709        handle%version%backend='mdsplus'
00710     case ("backend=hdf5")
00711        handle%version%backend='hdf5'
00712     case ("backend=ascii")
00713        handle%version%backend='ascii'
00714     case default
00715        write(*,*) "IMAS_AMNS_SET: not yet implemented ", trim(set%string)
00716     end select
00717     if(handle%debug) write(*,*) 'IMAS_AMNS_SET end'
00718
00719   end subroutine imas_amns_set
00720
00721   !
      ================================================================================================================
      !
00732   subroutine imas_amns_set_table(handle_rx,set,error_status)
00733     use ids_types  ! IGNORE
00734     use amns_types
00735     implicit none
00736     optional error_status
00737     type(amns_handle_rx_type), intent(inout) :: handle_rx
00738     type(amns_set_type), intent(in) :: set
00739     type(amns_error_type), intent(out) :: error_status
00740
00741     if(present(error_status)) then
00742         error_status%flag=.false.
00743         error_status%string="No error"
00744     end if
00745
00746     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SET_TABLE start'
00747     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SET_TABLE: set = ',trim(set%string)
00748 ! The choices are documented in the doxygen block above
00749     select case (set%string)
00750     case ("warn")
00751        if(handle_rx%filled) then
00752           call set_option(handle_rx%grid,.true.)
00753        else
00754           write(*,*) 'IMAS_AMNS_SET_TABLE: Attempt to set WARN using an unfilled table'
00755        endif
00756     case ("nowarn")
00757        if(handle_rx%filled) then
00758           call set_option(handle_rx%grid,.false.)
00759        else
00760           write(*,*) 'IMAS_AMNS_SET_TABLE: Attempt to set NOWARN using an unfilled table'
00761        endif
00762     case ("debug")
00763        handle_rx%debug=.true.
00764     case ("nodebug")
00765        handle_rx%debug=.false.
00766     case default
00767        write(*,*) 'IMAS_AMNS_SET_TABLE: not implemeted yet '
00768     end select
00769     if(present(error_status)) then
00770        if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SET_TABLE: requested error_status response'
00771     endif
00772     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_SET_TABLE end'
00773
00774   end subroutine imas_amns_set_table
00775
00776   !
      ================================================================================================================
      !
00785   subroutine imas_amns_rx_0(handle_rx,out,arg1,arg2,arg3,arg4,error_status)
00786     use ids_types  ! IGNORE
00787     use amns_types
00788     use data_suport
00789     implicit none
00790     optional arg2,arg3,arg4,error_status
00791     type(amns_handle_rx_type), intent(inout) :: handle_rx
00792     real (kind=ids_real), intent(out) :: out
00793     real (kind=ids_real), intent(in) :: arg1,arg2,arg3,arg4
00794     type(amns_error_type), intent(out) :: error_status
00795
00796     type(data_error_t) :: data_error
00797     real (kind=ids_real) :: out_d(1),arg1_d(1),arg2_d(1),arg3_d(1),arg4_d(1)
00798
00799     if(present(error_status)) then
```

```
00800         error_status%flag=.false.
00801         error_status%string="No error"
00802      end if
00803
00804      if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_0 start'
00805      if(present(arg4)) then
00806         if(present(arg3).and.present(arg2)) then
00807            if(handle_rx%filled) then
00808               arg1_d(1)=arg1
00809               arg2_d(1)=arg2
00810               arg3_d(1)=arg3
00811               arg4_d(1)=arg4
00812               call interpol( &
00813                    reshape(arg1_d,(/size(arg1_d)/)), &
00814                    reshape(arg2_d,(/size(arg2_d)/)), &
00815                    reshape(arg3_d,(/size(arg3_d)/)), &
00816                    reshape(arg4_d,(/size(arg4_d)/)), &
00817                    fd1=out_d, grid=handle_rx%grid, &
00818                    data_error=data_error)
00819               if(present(error_status)) then
00820                  error_status%flag = data_error%ierr .ne. 0
00821                  error_status%string = data_error%cerr
00822               else
00823                  if(data_error%ierr.ne.0) then
00824                     write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
00825                     call errorstop('interpol returned an error')
00826                  endif
00827               endif
00828               out=out_d(1)
00829            else
00830               write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00831            endif
00832         else
00833            write(*,*) 'IMAS_AMNS_RX_0: arg4 present but not arg2/arg3!'
00834         endif
00835      else
00836         if(present(arg3)) then
00837            if(present(arg2)) then
00838               if(handle_rx%filled) then
00839                  arg1_d(1)=arg1
00840                  arg2_d(1)=arg2
00841                  arg3_d(1)=arg3
00842                  call interpol( &
00843                       reshape(arg1_d,(/size(arg1_d)/)), &
00844                       reshape(arg2_d,(/size(arg2_d)/)), &
00845                       reshape(arg3_d,(/size(arg3_d)/)), &
00846                       fd1=out_d, grid=handle_rx%grid, &
00847                       data_error=data_error)
00848                  if(present(error_status)) then
00849                     error_status%flag = data_error%ierr .ne. 0
00850                     error_status%string = data_error%cerr
00851                  else
00852                     if(data_error%ierr.ne.0) then
00853                        write(*,*) 'interpol returned an error: ', data_error%ierr,
    trim(data_error%cerr)
00854                        call errorstop('interpol returned an error')
00855                     endif
00856                  endif
00857                  out=out_d(1)
00858               else
00859                  write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00860               endif
00861            else
00862               write(*,*) 'IMAS_AMNS_RX_0: arg3 present but not arg2!'
00863            endif
00864         else
00865            if(present(arg2)) then
00866               if(handle_rx%filled) then
00867                  arg1_d(1)=arg1
00868                  arg2_d(1)=arg2
00869                  call interpol( &
00870                       reshape(arg1_d,(/size(arg1_d)/)), &
00871                       reshape(arg2_d,(/size(arg2_d)/)), &
00872                       fd1=out_d, grid=handle_rx%grid, &
00873                       data_error=data_error)
00874                  if(present(error_status)) then
00875                     error_status%flag = data_error%ierr .ne. 0
00876                     error_status%string = data_error%cerr
00877                  else
00878                     if(data_error%ierr.ne.0) then
00879                        write(*,*) 'interpol returned an error: ', data_error%ierr,
    trim(data_error%cerr)
00880                        call errorstop('interpol returned an error')
00881                     endif
00882                  endif
00883                  out=out_d(1)
00884               else
```

```
00885                write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00886             endif
00887          else
00888             if(handle_rx%filled) then
00889                arg1_d(1)=arg1
00890                call interpol( &
00891                     reshape(arg1_d,(/size(arg1_d)/)), &
00892                     fd1=out_d, grid=handle_rx%grid, &
00893                     data_error=data_error)
00894                if(present(error_status)) then
00895                   error_status%flag = data_error%ierr .ne. 0
00896                   error_status%string = data_error%cerr
00897                else
00898                   if(data_error%ierr.ne.0) then
00899                      write(*,*) 'interpol returned an error: ', data_error%ierr,
       trim(data_error%cerr)
00900                      call errorstop('interpol returned an error')
00901                   endif
00902                endif
00903                out=out_d(1)
00904             else
00905                write(*,*) 'IMAS_AMNS_RX_0: Attempt to INTERPOLATE using an unfilled table'
00906             endif
00907          endif
00908       endif
00909    endif
00910    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_0 end'
00911
00912  end subroutine imas_amns_rx_0
00913
00914  !
       ================================================================================================================
       !
00923  subroutine imas_amns_rx_1(handle_rx,out,arg1,arg2,arg3,arg4,error_status)
00924    use ids_types  ! IGNORE
00925    use amns_types
00926    use data_suport
00927    implicit none
00928    optional arg2,arg3,arg4,error_status
00929    type(amns_handle_rx_type), intent(inout) :: handle_rx
00930    real (kind=ids_real), intent(out) :: out(:)
00931    real (kind=ids_real), intent(in) :: arg1(:),arg2(:),arg3(:),arg4(:)
00932    type(amns_error_type), intent(out) :: error_status
00933
00934    type(data_error_t) :: data_error
00935
00936    if(present(error_status)) then
00937        error_status%flag=.false.
00938        error_status%string="No error"
00939    end if
00940
00941    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1 start'
00942    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(out) = ',lbound(out),ubound(out)
00943    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg1) = ',lbound(arg1),ubound(arg1)
00944    if(present(arg4)) then
00945       if(present(arg3).and.present(arg2)) then
00946          if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) = ',lbound(arg2),ubound(arg2)
00947          if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg3) = ',lbound(arg3),ubound(arg3)
00948          if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg4) = ',lbound(arg4),ubound(arg4)
00949          if(handle_rx%filled) then
00950             call interpol( &
00951                  reshape(arg1,(/size(arg1)/)), &
00952                  reshape(arg2,(/size(arg2)/)), &
00953                  reshape(arg3,(/size(arg3)/)), &
00954                  reshape(arg4,(/size(arg4)/)), &
00955                  fd1=out, grid=handle_rx%grid, &
00956                  data_error=data_error)
00957             if(present(error_status)) then
00958                error_status%flag = data_error%ierr .ne. 0
00959                error_status%string = data_error%cerr
00960             else
00961                if(data_error%ierr.ne.0) then
00962                   write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
00963                   call errorstop('interpol returned an error')
00964                endif
00965             endif
00966          else
00967             write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
00968          endif
00969       else
00970          write(*,*) 'IMAS_AMNS_RX_1: arg4 present but not arg2/arg3!'
00971       endif
00972    else
00973       if(present(arg3)) then
00974          if(present(arg2)) then
00975             if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) =
       ',lbound(arg2),ubound(arg2)
```

```
00976             if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg3) =
      ',lbound(arg3),ubound(arg3)
00977             if(handle_rx%filled) then
00978                call interpol( &
00979                     reshape(arg1,(/size(arg1)/)), &
00980                     reshape(arg2,(/size(arg2)/)), &
00981                     reshape(arg3,(/size(arg3)/)), &
00982                     fd1=out, grid=handle_rx%grid, &
00983                     data_error=data_error)
00984                if(present(error_status)) then
00985                   error_status%flag = data_error%ierr .ne. 0
00986                   error_status%string = data_error%cerr
00987                else
00988                   if(data_error%ierr.ne.0) then
00989                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
00990                      call errorstop('interpol returned an error')
00991                   endif
00992                endif
00993             else
00994                write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
00995             endif
00996          else
00997             write(*,*) 'IMAS_AMNS_RX_1: arg3 present but not arg2!'
00998          endif
00999       else
01000          if(present(arg2)) then
01001             if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01002             if(handle_rx%filled) then
01003                call interpol( &
01004                     reshape(arg1,(/size(arg1)/)), &
01005                     reshape(arg2,(/size(arg2)/)), &
01006                     fd1=out, grid=handle_rx%grid, &
01007                     data_error=data_error)
01008                if(present(error_status)) then
01009                   error_status%flag = data_error%ierr .ne. 0
01010                   error_status%string = data_error%cerr
01011                else
01012                   if(data_error%ierr.ne.0) then
01013                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01014                      call errorstop('interpol returned an error')
01015                   endif
01016                endif
01017             else
01018                write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
01019             endif
01020          else
01021             if(handle_rx%filled) then
01022                call interpol( &
01023                     reshape(arg1,(/size(arg1)/)), &
01024                     fd1=out, grid=handle_rx%grid, &
01025                     data_error=data_error)
01026                if(present(error_status)) then
01027                   error_status%flag = data_error%ierr .ne. 0
01028                   error_status%string = data_error%cerr
01029                else
01030                   if(data_error%ierr.ne.0) then
01031                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01032                      call errorstop('interpol returned an error')
01033                   endif
01034                endif
01035             else
01036                write(*,*) 'IMAS_AMNS_RX_1: Attempt to INTERPOLATE using an unfilled table'
01037             endif
01038          endif
01039       endif
01040    endif
01041    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_1 end'
01042
01043  end subroutine imas_amns_rx_1
01044
01045  !
      ================================================================================================================
      !
01054  subroutine imas_amns_rx_2(handle_rx,out,arg1,arg2,arg3,arg4,error_status)
01055    use ids_types  ! IGNORE
01056    use amns_types
01057    use data_suport
01058    implicit none
01059    optional arg2,arg3,arg4,error_status
01060    type(amns_handle_rx_type), intent(inout) :: handle_rx
01061    real (kind=ids_real), intent(out) :: out(:,:)
01062    real (kind=ids_real), intent(in) :: arg1(:,:),arg2(:,:),arg3(:,:),arg4(:,:)
01063    type(amns_error_type), intent(out) :: error_status
```

```
01064
01065     real (kind=ids_real) :: tmp_out(size(out))
01066
01067     type(data_error_t) :: data_error
01068
01069     if(present(error_status)) then
01070         error_status%flag=.false.
01071         error_status%string="No error"
01072     end if
01073
01074     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2 start'
01075     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(out) = ',lbound(out),ubound(out)
01076     if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg1) = ',lbound(arg1),ubound(arg1)
01077     if(present(arg4)) then
01078        if(present(arg3).and.present(arg2)) then
01079           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) = ',lbound(arg2),ubound(arg2)
01080           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg3) = ',lbound(arg3),ubound(arg3)
01081           if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg4) = ',lbound(arg4),ubound(arg4)
01082           if(handle_rx%filled) then
01083              call interpol( &
01084                   reshape(arg1,(/size(arg1)/)), &
01085                   reshape(arg2,(/size(arg2)/)), &
01086                   reshape(arg3,(/size(arg3)/)), &
01087                   reshape(arg4,(/size(arg4)/)), &
01088                   fd1=tmp_out, grid=handle_rx%grid, &
01089                   data_error=data_error)
01090              if(present(error_status)) then
01091                 error_status%flag = data_error%ierr .ne. 0
01092                 error_status%string = data_error%cerr
01093              else
01094                 if(data_error%ierr.ne.0) then
01095                    write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
01096                    call errorstop('interpol returned an error')
01097                 endif
01098              endif
01099              out=reshape(tmp_out,shape(out))
01100           else
01101              write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01102           endif
01103        else
01104           write(*,*) 'IMAS_AMNS_RX_2: arg4 present but not arg2/arg3!'
01105        endif
01106     else
01107        if(present(arg3)) then
01108           if(present(arg2)) then
01109              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01110              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg3) =
      ',lbound(arg3),ubound(arg3)
01111              if(handle_rx%filled) then
01112                 call interpol( &
01113                      reshape(arg1,(/size(arg1)/)), &
01114                      reshape(arg2,(/size(arg2)/)), &
01115                      reshape(arg3,(/size(arg3)/)), &
01116                      fd1=tmp_out, grid=handle_rx%grid, &
01117                      data_error=data_error)
01118                 if(present(error_status)) then
01119                    error_status%flag = data_error%ierr .ne. 0
01120                    error_status%string = data_error%cerr
01121                 else
01122                    if(data_error%ierr.ne.0) then
01123                       write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01124                       call errorstop('interpol returned an error')
01125                    endif
01126                 endif
01127                 out=reshape(tmp_out,shape(out))
01128              else
01129                 write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01130              endif
01131           else
01132              write(*,*) 'IMAS_AMNS_RX_2: arg3 present but not arg2!'
01133           endif
01134        else
01135           if(present(arg2)) then
01136              if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2: bounds(arg2) =
      ',lbound(arg2),ubound(arg2)
01137              if(handle_rx%filled) then
01138                 call interpol( &
01139                      reshape(arg1,(/size(arg1)/)), &
01140                      reshape(arg2,(/size(arg2)/)), &
01141                      fd1=tmp_out, grid=handle_rx%grid, &
01142                      data_error=data_error)
01143                 if(present(error_status)) then
01144                    error_status%flag = data_error%ierr .ne. 0
01145                    error_status%string = data_error%cerr
01146                 else
```

```
01147                   if(data_error%ierr.ne.0) then
01148                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01149                      call errorstop('interpol returned an error')
01150                   endif
01151                endif
01152                out=reshape(tmp_out,shape(out))
01153             else
01154                write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01155             endif
01156          else
01157             if(handle_rx%filled) then
01158                call interpol( &
01159                     reshape(arg1,(/size(arg1)/)), &
01160                     fd1=tmp_out, grid=handle_rx%grid, &
01161                     data_error=data_error)
01162                if(present(error_status)) then
01163                   error_status%flag = data_error%ierr .ne. 0
01164                   error_status%string = data_error%cerr
01165                else
01166                   if(data_error%ierr.ne.0) then
01167                      write(*,*) 'interpol returned an error: ', data_error%ierr,
      trim(data_error%cerr)
01168                      call errorstop('interpol returned an error')
01169                   endif
01170                endif
01171                out=reshape(tmp_out,shape(out))
01172             else
01173                write(*,*) 'IMAS_AMNS_RX_2: Attempt to INTERPOLATE using an unfilled table'
01174             endif
01175          endif
01176       endif
01177    endif
01178    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_2 end'
01179
01180  end subroutine imas_amns_rx_2
01181
01182  !
      ================================================================================================================
      !
01191  subroutine imas_amns_rx_3(handle_rx,out,arg1,arg2,arg3,arg4,error_status)
01192    use ids_types  ! IGNORE
01193    use amns_types
01194    use data_suport
01195    implicit none
01196    optional arg2,arg3,arg4,error_status
01197    type(amns_handle_rx_type), intent(inout) :: handle_rx
01198    real (kind=ids_real), intent(out) :: out(:,:,:)
01199    real (kind=ids_real), intent(in) :: arg1(:,:,:),arg2(:,:,:),arg3(:,:,:),arg4(:,:,:)
01200    type(amns_error_type), intent(out) :: error_status
01201
01202    real (kind=ids_real) :: tmp_out(size(out))
01203
01204    type(data_error_t) :: data_error
01205
01206    if(present(error_status)) then
01207        error_status%flag=.false.
01208        error_status%string="No error"
01209    end if
01210
01211    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3 start'
01212    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(out) = ',lbound(out),ubound(out)
01213    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg1) = ',lbound(arg1),ubound(arg1)
01214    if(present(arg4)) then
01215       if(present(arg3).and.present(arg2)) then
01216          if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) = ',lbound(arg2),ubound(arg2)
01217          if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg3) = ',lbound(arg3),ubound(arg3)
01218          if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg4) = ',lbound(arg4),ubound(arg4)
01219          if(handle_rx%filled) then
01220             call interpol( &
01221                  reshape(arg1,(/size(arg1)/)), &
01222                  reshape(arg2,(/size(arg2)/)), &
01223                  reshape(arg3,(/size(arg3)/)), &
01224                  reshape(arg4,(/size(arg4)/)), &
01225                  fd1=tmp_out, grid=handle_rx%grid, &
01226                  data_error=data_error)
01227             if(present(error_status)) then
01228                error_status%flag = data_error%ierr .ne. 0
01229                error_status%string = data_error%cerr
01230             else
01231                if(data_error%ierr.ne.0) then
01232                   write(*,*) 'interpol returned an error: ', data_error%ierr, trim(data_error%cerr)
01233                   call errorstop('interpol returned an error')
01234                endif
01235             endif
01236             out=reshape(tmp_out,shape(out))
01237          else
```

```
01238             write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01239          endif
01240       else
01241          write(*,*) 'IMAS_AMNS_RX_3: arg4 present but not arg2/arg3!'
01242       endif
01243    else
01244       if(present(arg3)) then
01245          if(present(arg2)) then
01246             if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) =
       ',lbound(arg2),ubound(arg2)
01247             if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg3) =
       ',lbound(arg3),ubound(arg3)
01248             if(handle_rx%filled) then
01249                call interpol( &
01250                     reshape(arg1,(/size(arg1)/)), &
01251                     reshape(arg2,(/size(arg2)/)), &
01252                     reshape(arg3,(/size(arg3)/)), &
01253                     fd1=tmp_out, grid=handle_rx%grid, &
01254                     data_error=data_error)
01255                if(present(error_status)) then
01256                   error_status%flag = data_error%ierr .ne. 0
01257                   error_status%string = data_error%cerr
01258                else
01259                   if(data_error%ierr.ne.0) then
01260                      write(*,*) 'interpol returned an error: ', data_error%ierr,
       trim(data_error%cerr)
01261                      call errorstop('interpol returned an error')
01262                   endif
01263                endif
01264                out=reshape(tmp_out,shape(out))
01265             else
01266                write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01267             endif
01268          else
01269             write(*,*) 'IMAS_AMNS_RX_3: arg3 present but not arg2!'
01270          endif
01271       else
01272          if(present(arg2)) then
01273             if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3: bounds(arg2) =
       ',lbound(arg2),ubound(arg2)
01274             if(handle_rx%filled) then
01275                call interpol( &
01276                     reshape(arg1,(/size(arg1)/)), &
01277                     reshape(arg2,(/size(arg2)/)), &
01278                     fd1=tmp_out, grid=handle_rx%grid, &
01279                     data_error=data_error)
01280                if(present(error_status)) then
01281                   error_status%flag = data_error%ierr .ne. 0
01282                   error_status%string = data_error%cerr
01283                else
01284                   if(data_error%ierr.ne.0) then
01285                      write(*,*) 'interpol returned an error: ', data_error%ierr,
       trim(data_error%cerr)
01286                      call errorstop('interpol returned an error')
01287                   endif
01288                endif
01289                out=reshape(tmp_out,shape(out))
01290             else
01291                write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01292             endif
01293          else
01294             if(handle_rx%filled) then
01295                call interpol( &
01296                     reshape(arg1,(/size(arg1)/)), &
01297                     fd1=tmp_out, grid=handle_rx%grid, &
01298                     data_error=data_error)
01299                if(present(error_status)) then
01300                   error_status%flag = data_error%ierr .ne. 0
01301                   error_status%string = data_error%cerr
01302                else
01303                   if(data_error%ierr.ne.0) then
01304                      write(*,*) 'interpol returned an error: ', data_error%ierr,
       trim(data_error%cerr)
01305                      call errorstop('interpol returned an error')
01306                   endif
01307                endif
01308                out=reshape(tmp_out,shape(out))
01309             else
01310                write(*,*) 'IMAS_AMNS_RX_3: Attempt to INTERPOLATE using an unfilled table'
01311             endif
01312          endif
01313       endif
01314    endif
01315    if(handle_rx%debug) write(*,*) 'IMAS_AMNS_RX_3 end'
01316
01317   end subroutine imas_amns_rx_3
01318 end module amns_module
```

16.99 src/libamns/amns_module_isoc.f90 File Reference

Data Types

- interface amns_module_isoc::copy

Modules

- module amns_module_isoc

Functions/Subroutines

- pure character(size(a)) function amns_module_isoc::copy_a2s (a)
- pure character function, dimension(len(s)) amns_module_isoc::copy_s2a (s)
- subroutine amns_module_isoc::imas_amns_c_setup (handle, error_status_fc)

  provides IMAS_AMNS_C_SETUP by calling IMAS_AMNS_SETUP
- subroutine amns_module_isoc::imas_amns_c_setup_version (handle, version_fc, error_status_fc)

  provides IMAS_AMNS_C_SETUP by calling IMAS_AMNS_SETUP with version argument
- subroutine amns_module_isoc::imas_amns_c_finish (handle, error_status_fc)

  provides IMAS_AMNS_C_FINISH by calling IMAS_AMNS_FINISH
- subroutine amns_module_isoc::imas_amns_c_finish_table (handle_rx, error_status_fc)

  provides IMAS_AMNS_C_FINISH_TABLE by calling IMAS_AMNS_FINISH_TABLE
- subroutine amns_module_isoc::imas_amns_c_set (handle, set_fc, error_status_fc)

  provides IMAS_AMNS_C_SET by calling IMAS_AMNS_SET
- subroutine amns_module_isoc::imas_amns_c_query (handle, query_fc, answer_fc, error_status_fc)

  provides IMAS_AMNS_C_QUERY by calling IMAS_AMNS_QUERY
- subroutine amns_module_isoc::imas_amns_c_setup_table (handle, reaction_type_fc, reactant, handle_rx, error_status_fc)

  provides IMAS_AMNS_C_SETUP_TABLE by calling IMAS_AMNS_SETUP_TABLE
- subroutine amns_module_isoc::imas_amns_c_query_table (handle_rx, query_fc, answer_fc, error_status↩_fc)

  provides IMAS_AMNS_C_QUERY_TABLE by calling IMAS_AMNS_QUERY_TABLE
- subroutine amns_module_isoc::imas_amns_c_set_table (handle_rx, set_fc, error_status_fc)

  provides IMAS_AMNS_C_SET_TABLE by calling IMAS_AMNS_SET_TABLE
- subroutine amns_module_isoc::imas_amns_c_rx_0_a (handle_rx, out, arg1, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_0_b (handle_rx, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_0_c (handle_rx, out, arg1, arg2, arg3, error_status_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_0_d (handle_rx, out, arg1, arg2, arg3, arg4, error_status↩_fc)

  get the rates associated with the input (0d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_1_a (handle_rx, nx, out, arg1, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_1_b (handle_rx, nx, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_1_c (handle_rx, nx, out, arg1, arg2, arg3, error_status_fc)

get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.

- subroutine amns_module_isoc::imas_amns_c_rx_1_d (handle_rx, nx, out, arg1, arg2, arg3, arg4, error_↩ status_fc)

  get the rates associated with the input (1d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_2_a (handle_rx, nx, ny, out, arg1, error_status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_2_b (handle_rx, nx, ny, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_2_c (handle_rx, nx, ny, out, arg1, arg2, arg3, error_↩ status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_2_d (handle_rx, nx, ny, out, arg1, arg2, arg3, arg4, error↩ _status_fc)

  get the rates associated with the input (2d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_3_a (handle_rx, nx, ny, nz, out, arg1, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_3_b (handle_rx, nx, ny, nz, out, arg1, arg2, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into three separate subroutine for 1,2 or 3 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_3_c (handle_rx, nx, ny, nz, out, arg1, arg2, arg3, error_↩ status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into three separate subroutine for 1,2 or 3 arguments.
- subroutine amns_module_isoc::imas_amns_c_rx_3_d (handle_rx, nx, ny, nz, out, arg1, arg2, arg3, arg4, error_status_fc)

  get the rates associated with the input (3d) args for a particular reaction Expanded into four separate subroutines for 1, 2, 3 or 4 arguments.
- subroutine amns_module_isoc::imas_amns_c_setup_reactants (reactants_handle, string, index, n_↩ reactants)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.
- subroutine amns_module_isoc::imas_amns_c_set_reactant (reactants_handle, reactant_index, reactant)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.
- subroutine amns_module_isoc::imas_amns_c_get_reactant (reactants_handle, reactant_index, reactant)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.
- subroutine amns_module_isoc::imas_amns_c_finish_reactants (reactants_handle)

  Additional helper routines not contained in amns_module, but required for using the type amns_reactants_type from C.

16.100 amns_module_isoc.f90

```
00001
00007
00008 module amns_module_isoc
00009
00010   use amns_module
00011   use amns_types
00012   use iso_c_binding  ! IGNORE
```

```
00013
00014   implicit none
00015
00016   logical, private, parameter :: IMAS_AMNS_C_DEBUG = .false.
00017
00018   interface copy    ! generic
00019      module procedure copy_a2s, copy_s2a
00020   end interface copy
00021
00022 contains
00023
00024   ! ------------------------
00025   pure function copy_a2s(a)  result (s)    ! copy char array to string
00026     character,intent(in) :: a(:)
00027     character(size(a)) :: s
00028     integer :: i
00029     do i = 1,size(a)
00030        s(i:i) = a(i)
00031     end do
00032   end function copy_a2s
00033
00034   ! ------------------------
00035   pure function copy_s2a(s)  result (a)   ! copy string to char array
00036     character(*),intent(in) :: s
00037     character :: a(len(s))
00038     integer :: i
00039     do i = 1,len(s)
00040        a(i) = s(i:i)
00041     end do
00042   end function copy_s2a
00043
00046   subroutine imas_amns_c_setup(handle, error_status_fc) bind(c, name='IMAS_AMNS_C_SETUP')
00047     type(c_ptr), intent(out) :: handle
00048     type(amns_fc_error_type), intent(out) :: error_status_fc
00049
00050     ! internal
00051     type(amns_handle_type), pointer :: lhandle
00052     type(amns_error_type) :: error_status
00053
00054     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP: enter"
00055     allocate(lhandle)
00056     !lhandle%debug = IMAS_AMNS_C_DEBUG
00057     call imas_amns_setup(lhandle, error_status=error_status)
00058     !lhandle%debug = IMAS_AMNS_C_DEBUG ! this due to issue in IMAS_AMNS_SETUP that lhandle is
      intent(out)
00059     handle = c_loc(lhandle)
00060     error_status_fc%flag = error_status%flag
00061     error_status_fc%string = copy(error_status%string)
00062     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP: return"
00063   end subroutine imas_amns_c_setup
00064
00067   subroutine imas_amns_c_setup_version(handle, version_fc, error_status_fc) bind(c,
      name='IMAS_AMNS_C_SETUP_VERSION')
00068     type(c_ptr), intent(out) :: handle
00069     type(amns_fc_version_type), intent(in) :: version_fc
00070     type(amns_fc_error_type), intent(out) :: error_status_fc
00071
00072     ! internal
00073     type(amns_handle_type), pointer :: lhandle
00074     type(amns_version_type) :: version
00075     type(amns_error_type) :: error_status
00076
00077     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_VERSION: enter"
00078     allocate(lhandle)
00079     !lhandle%debug = IMAS_AMNS_C_DEBUG
00080     version%string = copy(version_fc%string)
00081     version%number = version_fc%number
00082     version%backend = copy(version_fc%backend)
00083     version%user = copy(version_fc%user)
00084     call imas_amns_setup(lhandle, version, error_status=error_status)
00085     !lhandle%debug = IMAS_AMNS_C_DEBUG ! this due to issue in IMAS_AMNS_SETUP that lhandle is
      intent(out)
00086     error_status_fc%flag = error_status%flag
00087     error_status_fc%string = copy(error_status%string)
00088     handle = c_loc(lhandle)
00089     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_VERSION: return"
00090   end subroutine imas_amns_c_setup_version
00091
00094   subroutine imas_amns_c_finish(handle, error_status_fc) bind(c, name='IMAS_AMNS_C_FINISH')
00095     type(c_ptr), intent(inout) :: handle
00096     type(amns_fc_error_type), intent(out) :: error_status_fc
00097
00098     ! internal
00099     type(amns_handle_type), pointer :: lhandle
00100     type(amns_error_type) :: error_status
00101
00102     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH: enter"
```

```
00103     call c_f_pointer(handle, lhandle)
00104     call imas_amns_finish(lhandle, error_status)
00105     deallocate(lhandle)
00106     handle = c_null_ptr
00107     error_status_fc%flag = error_status%flag
00108     error_status_fc%string = copy(error_status%string)
00109     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH: return"
00110   end subroutine imas_amns_c_finish
00111
00114   subroutine imas_amns_c_finish_table(handle_rx, error_status_fc)
      bind(c,name="IMAS_AMNS_C_FINISH_TABLE")
00115     type(c_ptr), intent(inout) :: handle_rx
00116     type(amns_fc_error_type), intent(out) :: error_status_fc
00117
00118     ! internal
00119     type(amns_handle_rx_type), pointer :: lhandle_rx
00120     type(amns_error_type) :: error_status
00121
00122     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH_TABLE: enter"
00123     call c_f_pointer(handle_rx, lhandle_rx)
00124     call imas_amns_finish_table(lhandle_rx, error_status)
00125     deallocate(lhandle_rx)
00126     handle_rx = c_null_ptr
00127     error_status_fc%flag = error_status%flag
00128     error_status_fc%string = copy(error_status%string)
00129     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH_TABLE: return"
00130   end subroutine imas_amns_c_finish_table
00131
00134   subroutine imas_amns_c_set(handle,set_fc,error_status_fc) bind(c,name="IMAS_AMNS_C_SET")
00135     type(c_ptr), intent(in), value :: handle
00136     type(amns_fc_set_type), intent(in) :: set_fc
00137     type(amns_fc_error_type), intent(out) :: error_status_fc
00138
00139     ! internal
00140     type(amns_handle_type), pointer :: lhandle
00141     type(amns_set_type) :: set
00142     type(amns_error_type) :: error_status
00143
00144     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET: enter"
00145     call c_f_pointer(handle, lhandle)
00146     set%string = copy(set_fc%string)
00147     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET: set%string ='"//set%string//"'"
00148     call c_f_pointer(handle, lhandle)
00149     call imas_amns_set(lhandle, set, error_status)
00150     error_status_fc%flag = error_status%flag
00151     error_status_fc%string = copy(error_status%string)
00152     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET: return"
00153   end subroutine imas_amns_c_set
00154
00157   subroutine imas_amns_c_query(handle,query_fc,answer_fc,error_status_fc)
      bind(c,name="IMAS_AMNS_C_QUERY")
00158     type(c_ptr), intent(in), value :: handle
00159     type(amns_fc_query_type), intent(in) :: query_fc
00160     type(amns_fc_answer_type), intent(out) :: answer_fc
00161     type(amns_fc_error_type), intent(out) :: error_status_fc
00162
00163     ! internal
00164     type(amns_handle_type), pointer :: lhandle
00165     type(amns_query_type) :: query
00166     type(amns_answer_type) :: answer
00167     type(amns_error_type) :: error_status
00168
00169     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: enter"
00170     call c_f_pointer(handle, lhandle)
00171     query%string = copy(query_fc%string)
00172     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: query%string ='"//query%string//"'"
00173     call c_f_pointer(handle, lhandle)
00174     call imas_amns_query(lhandle,query,answer,error_status)
00175     answer_fc%string = copy(answer%string)
00176     answer_fc%number = answer%number
00177     error_status_fc%flag = error_status%flag
00178     error_status_fc%string = copy(error_status%string)
00179     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: answer%string ='"//answer%string//"'"
00180     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: answer%number =", answer%number
00181     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY: return"
00182   end subroutine imas_amns_c_query
00183
00186   subroutine imas_amns_c_setup_table(handle, reaction_type_fc, reactant, handle_rx, error_status_fc)
      bind(c, name="IMAS_AMNS_C_SETUP_TABLE")
00187     type(c_ptr), intent(in), value :: handle
00188     type(amns_fc_reaction_type), intent(in) :: reaction_type_fc
00189     type(c_ptr), intent(in), value :: reactant
00190     type(c_ptr), intent(out) :: handle_rx
00191     type(amns_fc_error_type), intent(out) :: error_status_fc
00192
00193     ! internal
00194     type(amns_handle_type), pointer :: lhandle
```

```
00195     type(amns_reactants_type), pointer :: lreactant
00196     type(amns_handle_rx_type), pointer :: lhandle_rx
00197     type(amns_reaction_type) :: reaction_type
00198     type(amns_error_type) :: error_status
00199
00200     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_TABLE: enter"
00201     call c_f_pointer(handle, lhandle)
00202     call c_f_pointer(reactant, lreactant)
00203     allocate(lhandle_rx)
00204     reaction_type%string = copy(reaction_type_fc%string)
00205     reaction_type%isotope_resolved = reaction_type_fc%isotope_resolved
00206     call imas_amns_setup_table(lhandle, reaction_type, lreactant, lhandle_rx, error_status)
00207     handle_rx = c_loc(lhandle_rx)
00208     error_status_fc%flag = error_status%flag
00209     error_status_fc%string = copy(error_status%string)
00210     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_TABLE: return"
00211   end subroutine imas_amns_c_setup_table
00212
00215   subroutine imas_amns_c_query_table(handle_rx,query_fc,answer_fc,error_status_fc)
      bind(c,name="IMAS_AMNS_C_QUERY_TABLE")
00216     type(c_ptr), intent(in), value :: handle_rx
00217     type(amns_fc_query_type), intent(in) :: query_fc
00218     type(amns_fc_answer_type), intent(out) :: answer_fc
00219     type(amns_fc_error_type), intent(out) :: error_status_fc
00220
00221     ! internal
00222     type(amns_handle_rx_type), pointer :: lhandle_rx
00223     type(amns_query_type) :: query
00224     type(amns_answer_type) :: answer
00225     type(amns_error_type) :: error_status
00226
00227     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY_TABLE: enter"
00228     call c_f_pointer(handle_rx, lhandle_rx)
00229     query%string = copy(query_fc%string)
00230     call imas_amns_query_table(lhandle_rx,query,answer,error_status)
00231     answer_fc%string = copy(answer%string)
00232     answer_fc%number = answer%number
00233     error_status_fc%flag = error_status%flag
00234     error_status_fc%string = copy(error_status%string)
00235     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_QUERY_TABLE: return"
00236   end subroutine imas_amns_c_query_table
00237
00240   subroutine imas_amns_c_set_table(handle_rx,set_fc,error_status_fc)
      bind(c,name="IMAS_AMNS_C_SET_TABLE")
00241     type(c_ptr), intent(in), value :: handle_rx
00242     type(amns_fc_set_type), intent(in) :: set_fc
00243     type(amns_fc_error_type), intent(out) :: error_status_fc
00244
00245     ! internal
00246     type(amns_handle_rx_type), pointer :: lhandle_rx
00247     type(amns_set_type) :: set
00248     type(amns_error_type) :: error_status
00249
00250     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET_TABLE: enter"
00251     call c_f_pointer(handle_rx, lhandle_rx)
00252     set%string = copy(set_fc%string)
00253     call imas_amns_set_table(lhandle_rx,set,error_status)
00254     error_status_fc%flag = error_status%flag
00255     error_status_fc%string = copy(error_status%string)
00256     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET_TABLE: return"
00257   end subroutine imas_amns_c_set_table
00258
00259
00263   subroutine imas_amns_c_rx_0_a(handle_rx,out,arg1,error_status_fc) bind(c,name="IMAS_AMNS_C_RX_0_A")
00264     type(c_ptr), intent(in), value :: handle_rx
00265     real (kind=ids_real), intent(out) :: out
00266     real (kind=ids_real), intent(in), value :: arg1
00267     type(amns_fc_error_type), intent(out) :: error_status_fc
00268
00269     ! internal
00270     type(amns_handle_rx_type), pointer :: lhandle_rx
00271     type(amns_error_type) :: error_status
00272
00273     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_A: enter"
00274     call c_f_pointer(handle_rx, lhandle_rx)
00275     call imas_amns_rx_0(lhandle_rx, out, arg1, error_status=error_status)
00276     error_status_fc%flag = error_status%flag
00277     error_status_fc%string = copy(error_status%string)
00278     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_A: return"
00279   end subroutine imas_amns_c_rx_0_a
00280
00284   subroutine imas_amns_c_rx_0_b(handle_rx,out,arg1,arg2,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_0_B")
00285     type(c_ptr), intent(in), value :: handle_rx
00286     real (kind=ids_real), intent(out) :: out
00287     real (kind=ids_real), intent(in), value :: arg1,arg2
00288     type(amns_fc_error_type), intent(out) :: error_status_fc
```

```
00289
00290     ! internal
00291     type(amns_handle_rx_type), pointer :: lhandle_rx
00292     type(amns_error_type) :: error_status
00293
00294     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_B: enter"
00295     call c_f_pointer(handle_rx, lhandle_rx)
00296     call imas_amns_rx_0(lhandle_rx, out, arg1, arg2=arg2, error_status=error_status)
00297     error_status_fc%flag = error_status%flag
00298     error_status_fc%string = copy(error_status%string)
00299     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_B: return"
00300   end subroutine imas_amns_c_rx_0_b
00301
00305   subroutine imas_amns_c_rx_0_c(handle_rx,out,arg1,arg2,arg3,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_0_C")
00306     type(c_ptr), intent(in), value :: handle_rx
00307     real (kind=ids_real), intent(out) :: out
00308     real (kind=ids_real), intent(in), value :: arg1,arg2,arg3
00309     type(amns_fc_error_type), intent(out) :: error_status_fc
00310
00311     ! internal
00312     type(amns_handle_rx_type), pointer :: lhandle_rx
00313     type(amns_error_type) :: error_status
00314
00315     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_C: enter"
00316     call c_f_pointer(handle_rx, lhandle_rx)
00317     call imas_amns_rx_0(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, error_status=error_status)
00318     error_status_fc%flag = error_status%flag
00319     error_status_fc%string = copy(error_status%string)
00320     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_C: return"
00321   end subroutine imas_amns_c_rx_0_c
00322
00326   subroutine imas_amns_c_rx_0_d(handle_rx,out,arg1,arg2,arg3,arg4,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_0_D")
00327     type(c_ptr), intent(in), value :: handle_rx
00328     real (kind=ids_real), intent(out) :: out
00329     real (kind=ids_real), intent(in), value :: arg1,arg2,arg3,arg4
00330     type(amns_fc_error_type), intent(out) :: error_status_fc
00331
00332     ! internal
00333     type(amns_handle_rx_type), pointer :: lhandle_rx
00334     type(amns_error_type) :: error_status
00335
00336     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_D: enter"
00337     call c_f_pointer(handle_rx, lhandle_rx)
00338     call imas_amns_rx_0(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, arg4=arg4,
      error_status=error_status)
00339     error_status_fc%flag = error_status%flag
00340     error_status_fc%string = copy(error_status%string)
00341     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_0_D: return"
00342   end subroutine imas_amns_c_rx_0_d
00343
00344
00348   subroutine imas_amns_c_rx_1_a(handle_rx,nx,out,arg1,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_1_A")
00349     type(c_ptr), intent(in), value :: handle_rx
00350     integer(c_int), intent(in), value :: nx
00351     real (c_double), intent(out) :: out(0:nx-1)
00352     real (c_double), intent(in) :: arg1(0:nx-1)
00353     type(amns_fc_error_type), intent(out) :: error_status_fc
00354
00355     ! internal
00356     type(amns_handle_rx_type), pointer :: lhandle_rx
00357     type(amns_error_type) :: error_status
00358
00359     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_A: enter"
00360     call c_f_pointer(handle_rx, lhandle_rx)
00361     call imas_amns_rx_1(lhandle_rx, out, arg1, error_status=error_status)
00362     error_status_fc%flag = error_status%flag
00363     error_status_fc%string = copy(error_status%string)
00364     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_A: return"
00365   end subroutine imas_amns_c_rx_1_a
00366
00370   subroutine imas_amns_c_rx_1_b(handle_rx,nx,out,arg1,arg2,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_1_B")
00371     type(c_ptr), intent(in), value :: handle_rx
00372     integer(c_int), intent(in), value :: nx
00373     real (c_double), intent(out) :: out(0:nx-1)
00374     real (c_double), intent(in) :: arg1(0:nx-1), arg2(0:nx-1)
00375     type(amns_fc_error_type), intent(out) :: error_status_fc
00376
00377     ! internal
00378     type(amns_handle_rx_type), pointer :: lhandle_rx
00379     type(amns_error_type) :: error_status
00380
00381     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_B: enter"
00382     call c_f_pointer(handle_rx, lhandle_rx)
```

```
00383     call imas_amns_rx_1(lhandle_rx, out, arg1, arg2=arg2, error_status=error_status)
00384     error_status_fc%flag = error_status%flag
00385     error_status_fc%string = copy(error_status%string)
00386     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_B: return"
00387   end subroutine imas_amns_c_rx_1_b
00388
00392   subroutine imas_amns_c_rx_1_c(handle_rx,nx,out,arg1,arg2,arg3,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_1_C")
00393     type(c_ptr), intent(in), value :: handle_rx
00394     integer(c_int), intent(in), value :: nx
00395     real (c_double), intent(out) :: out(0:nx-1)
00396     real (c_double), intent(in) :: arg1(0:nx-1), arg2(0:nx-1), arg3(0:nx-1)
00397     type(amns_fc_error_type), intent(out) :: error_status_fc
00398
00399     ! internal
00400     type(amns_handle_rx_type), pointer :: lhandle_rx
00401     type(amns_error_type) :: error_status
00402
00403     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_C: enter"
00404     call c_f_pointer(handle_rx, lhandle_rx)
00405     call imas_amns_rx_1(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, error_status=error_status)
00406     error_status_fc%flag = error_status%flag
00407     error_status_fc%string = copy(error_status%string)
00408     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_C: return"
00409   end subroutine imas_amns_c_rx_1_c
00410
00414   subroutine imas_amns_c_rx_1_d(handle_rx,nx,out,arg1,arg2,arg3,arg4,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_1_D")
00415     type(c_ptr), intent(in), value :: handle_rx
00416     integer(c_int), intent(in), value :: nx
00417     real (c_double), intent(out) :: out(0:nx-1)
00418     real (c_double), intent(in) :: arg1(0:nx-1), arg2(0:nx-1), arg3(0:nx-1), arg4(0:nx-1)
00419     type(amns_fc_error_type), intent(out) :: error_status_fc
00420
00421     ! internal
00422     type(amns_handle_rx_type), pointer :: lhandle_rx
00423     type(amns_error_type) :: error_status
00424
00425     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_D: enter"
00426     call c_f_pointer(handle_rx, lhandle_rx)
00427     call imas_amns_rx_1(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, arg4=arg4,
      error_status=error_status)
00428     error_status_fc%flag = error_status%flag
00429     error_status_fc%string = copy(error_status%string)
00430     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_1_D: return"
00431   end subroutine imas_amns_c_rx_1_d
00432
00433
00437   subroutine imas_amns_c_rx_2_a(handle_rx,nx,ny,out,arg1,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_2_A")
00438     type(c_ptr), intent(in), value :: handle_rx
00439     integer(c_int), intent(in), value :: nx, ny
00440     real (c_double), intent(out) :: out(0:nx-1,0:ny-1)
00441     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1)
00442     type(amns_fc_error_type), intent(out) :: error_status_fc
00443
00444     ! internal
00445     type(amns_handle_rx_type), pointer :: lhandle_rx
00446     type(amns_error_type) :: error_status
00447
00448     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_A: enter"
00449     call c_f_pointer(handle_rx, lhandle_rx)
00450     call imas_amns_rx_2(lhandle_rx, out, arg1, error_status=error_status)
00451     error_status_fc%flag = error_status%flag
00452     error_status_fc%string = copy(error_status%string)
00453     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_A: return"
00454   end subroutine imas_amns_c_rx_2_a
00455
00459   subroutine imas_amns_c_rx_2_b(handle_rx,nx,ny,out,arg1,arg2,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_2_B")
00460     type(c_ptr), intent(in), value :: handle_rx
00461     integer(c_int), intent(in), value :: nx, ny
00462     real (c_double), intent(out) :: out(0:nx-1,0:ny-1)
00463     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1), arg2(0:nx-1,0:ny-1)
00464     type(amns_fc_error_type), intent(out) :: error_status_fc
00465
00466     ! internal
00467     type(amns_handle_rx_type), pointer :: lhandle_rx
00468     type(amns_error_type) :: error_status
00469
00470     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_B: enter"
00471     call c_f_pointer(handle_rx, lhandle_rx)
00472     call imas_amns_rx_2(lhandle_rx, out, arg1, arg2=arg2, error_status=error_status)
00473     error_status_fc%flag = error_status%flag
00474     error_status_fc%string = copy(error_status%string)
00475     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_B: return"
00476   end subroutine imas_amns_c_rx_2_b
```

```
00477
00481   subroutine imas_amns_c_rx_2_c(handle_rx,nx,ny,out,arg1,arg2,arg3,error_status_fc)
        bind(c,name="IMAS_AMNS_C_RX_2_C")
00482     type(c_ptr), intent(in), value :: handle_rx
00483     integer(c_int), intent(in), value :: nx, ny
00484     real (c_double), intent(out) :: out(0:nx-1,0:ny-1)
00485     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1), arg2(0:nx-1,0:ny-1), arg3(0:nx-1,0:ny-1)
00486     type(amns_fc_error_type), intent(out) :: error_status_fc
00487
00488     ! internal
00489     type(amns_handle_rx_type), pointer :: lhandle_rx
00490     type(amns_error_type) :: error_status
00491
00492     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_C: enter"
00493     call c_f_pointer(handle_rx, lhandle_rx)
00494     call imas_amns_rx_2(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, error_status=error_status)
00495     error_status_fc%flag = error_status%flag
00496     error_status_fc%string = copy(error_status%string)
00497     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_C: return"
00498   end subroutine imas_amns_c_rx_2_c
00499
00503   subroutine imas_amns_c_rx_2_d(handle_rx,nx,ny,out,arg1,arg2,arg3,arg4,error_status_fc)
        bind(c,name="IMAS_AMNS_C_RX_2_D")
00504     type(c_ptr), intent(in), value :: handle_rx
00505     integer(c_int), intent(in), value :: nx, ny
00506     real (c_double), intent(out) :: out(0:nx-1,0:ny-1)
00507     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1), arg2(0:nx-1,0:ny-1), arg3(0:nx-1,0:ny-1),
        arg4(0:nx-1,0:ny-1)
00508     type(amns_fc_error_type), intent(out) :: error_status_fc
00509
00510     ! internal
00511     type(amns_handle_rx_type), pointer :: lhandle_rx
00512     type(amns_error_type) :: error_status
00513
00514     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_D: enter"
00515     call c_f_pointer(handle_rx, lhandle_rx)
00516     call imas_amns_rx_2(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, arg4=arg4,
        error_status=error_status)
00517     error_status_fc%flag = error_status%flag
00518     error_status_fc%string = copy(error_status%string)
00519     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_2_D: return"
00520   end subroutine imas_amns_c_rx_2_d
00521
00522
00526   subroutine imas_amns_c_rx_3_a(handle_rx,nx,ny,nz,out,arg1,error_status_fc)
        bind(c,name="IMAS_AMNS_C_RX_3_A")
00527     type(c_ptr), intent(in), value :: handle_rx
00528     integer(c_int), intent(in), value :: nx, ny, nz
00529     real (c_double), intent(out) :: out(0:nx-1,0:ny-1,0:nz-1)
00530     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1,0:nz-1)
00531     type(amns_fc_error_type), intent(out) :: error_status_fc
00532
00533     ! internal
00534     type(amns_handle_rx_type), pointer :: lhandle_rx
00535     type(amns_error_type) :: error_status
00536
00537     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_A: enter"
00538     call c_f_pointer(handle_rx, lhandle_rx)
00539     call imas_amns_rx_3(lhandle_rx, out, arg1, error_status=error_status)
00540     error_status_fc%flag = error_status%flag
00541     error_status_fc%string = copy(error_status%string)
00542     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_A: return"
00543   end subroutine imas_amns_c_rx_3_a
00544
00548   subroutine imas_amns_c_rx_3_b(handle_rx,nx,ny,nz,out,arg1,arg2,error_status_fc)
        bind(c,name="IMAS_AMNS_C_RX_3_B")
00549     type(c_ptr), intent(in), value :: handle_rx
00550     integer(c_int), intent(in), value :: nx, ny, nz
00551     real (c_double), intent(out) :: out(0:nx-1,0:ny-1,0:nz-1)
00552     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1,0:nz-1), arg2(0:nx-1,0:ny-1,0:nz-1)
00553     type(amns_fc_error_type), intent(out) :: error_status_fc
00554
00555     ! internal
00556     type(amns_handle_rx_type), pointer :: lhandle_rx
00557     type(amns_error_type) :: error_status
00558
00559     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_B: enter"
00560     call c_f_pointer(handle_rx, lhandle_rx)
00561     call imas_amns_rx_3(lhandle_rx, out, arg1, arg2=arg2, error_status=error_status)
00562     error_status_fc%flag = error_status%flag
00563     error_status_fc%string = copy(error_status%string)
00564     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_B: return"
00565   end subroutine imas_amns_c_rx_3_b
00566
00570   subroutine imas_amns_c_rx_3_c(handle_rx,nx,ny,nz,out,arg1,arg2,arg3,error_status_fc)
        bind(c,name="IMAS_AMNS_C_RX_3_C")
00571     type(c_ptr), intent(in), value :: handle_rx
```

```
00572     integer(c_int), intent(in), value :: nx, ny, nz
00573     real (c_double), intent(out) :: out(0:nx-1,0:ny-1,0:nz-1)
00574     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1,0:nz-1), arg2(0:nx-1,0:ny-1,0:nz-1),
      arg3(0:nx-1,0:ny-1,0:nz-1)
00575     type(amns_fc_error_type), intent(out) :: error_status_fc
00576
00577     ! internal
00578     type(amns_handle_rx_type), pointer :: lhandle_rx
00579     type(amns_error_type) :: error_status
00580
00581     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_C: enter"
00582     call c_f_pointer(handle_rx, lhandle_rx)
00583     call imas_amns_rx_3(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, error_status=error_status)
00584     error_status_fc%flag = error_status%flag
00585     error_status_fc%string = copy(error_status%string)
00586     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_C: return"
00587   end subroutine imas_amns_c_rx_3_c
00588
00592   subroutine imas_amns_c_rx_3_d(handle_rx,nx,ny,nz,out,arg1,arg2,arg3,arg4,error_status_fc)
      bind(c,name="IMAS_AMNS_C_RX_3_D")
00593     type(c_ptr), intent(in), value :: handle_rx
00594     integer(c_int), intent(in), value :: nx, ny, nz
00595     real (c_double), intent(out) :: out(0:nx-1,0:ny-1,0:nz-1)
00596     real (c_double), intent(in) :: arg1(0:nx-1,0:ny-1,0:nz-1), arg2(0:nx-1,0:ny-1,0:nz-1),
      arg3(0:nx-1,0:ny-1,0:nz-1), arg4(0:nx-1,0:ny-1,0:nz-1)
00597     type(amns_fc_error_type), intent(out) :: error_status_fc
00598
00599     ! internal
00600     type(amns_handle_rx_type), pointer :: lhandle_rx
00601     type(amns_error_type) :: error_status
00602
00603     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_D: enter"
00604     call c_f_pointer(handle_rx, lhandle_rx)
00605     call imas_amns_rx_3(lhandle_rx, out, arg1, arg2=arg2, arg3=arg3, arg4=arg4,
      error_status=error_status)
00606     error_status_fc%flag = error_status%flag
00607     error_status_fc%string = copy(error_status%string)
00608     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_RX_3_D: return"
00609   end subroutine imas_amns_c_rx_3_d
00610
00611
00615   subroutine imas_amns_c_setup_reactants(reactants_handle, string, index, n_reactants) bind(c,
      name="IMAS_AMNS_C_SETUP_REACTANTS")
00616     type(c_ptr), intent(out) :: reactants_handle
00617     character(kind=c_char), intent(in) :: string(*)
00618     integer(c_int), intent(in), value :: index, n_reactants
00619     integer i
00620
00621     ! internal
00622     type(amns_reactants_type), pointer :: lreactants
00623
00624     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_REACTANTS: enter, string"
00625     allocate(lreactants)
00626     allocate(lreactants%components(n_reactants))
00627     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_REACTANTS: ", lreactants%string
00628     ! FIXME: %string is an array, provide access functions
00629     allocate(lreactants%string(1))
00630     lreactants%string(1) = ''
00631     do i=1, len(lreactants%string(1))
00632        if(string(i) .EQ. c_null_char) exit
00633        lreactants%string(1)(i:i) = string(i)
00634     end do
00635     lreactants%index = index
00636
00637     reactants_handle = c_loc(lreactants)
00638     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SETUP_REACTANTS: return"
00639   end subroutine imas_amns_c_setup_reactants
00640
00644   subroutine imas_amns_c_set_reactant(reactants_handle, reactant_index, reactant) bind(c,
      name="IMAS_AMNS_C_SET_REACTANT")
00645     type(c_ptr), intent(in), value :: reactants_handle
00646     integer(c_int), intent(in), value :: reactant_index
00647     type(amns_reactant_type), intent(in) :: reactant
00648
00649     ! internal
00650     type(amns_reactants_type), pointer :: lreactants
00651
00652     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET_REACTANT: enter"
00653     call c_f_pointer(reactants_handle, lreactants)
00654     lreactants%components(reactant_index) = reactant
00655     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_SET_REACTANT: return"
00656   end subroutine imas_amns_c_set_reactant
00657
00661   subroutine imas_amns_c_get_reactant(reactants_handle, reactant_index, reactant) bind(c,
      name="IMAS_AMNS_C_GET_REACTANT")
00662     type(c_ptr), intent(in), value :: reactants_handle
00663     integer(c_int), intent(in), value :: reactant_index
```

```
00664     type(amns_reactant_type), intent(out) :: reactant
00665
00666     ! internal
00667     type(amns_reactants_type), pointer :: lreactants
00668
00669     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_GET_REACTANT: enter"
00670     call c_f_pointer(reactants_handle, lreactants)
00671     reactant = lreactants%components(reactant_index)
00672     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_GET_REACTANT: return"
00673   end subroutine imas_amns_c_get_reactant
00674
00675
00679   subroutine imas_amns_c_finish_reactants(reactants_handle) bind(c,
       name="IMAS_AMNS_C_FINISH_REACTANTS")
00680     type(c_ptr), intent(inout) :: reactants_handle
00681
00682     ! internal
00683     type(amns_reactants_type), pointer :: lreactants
00684
00685     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH_REACTANTS: enter"
00686     call c_f_pointer(reactants_handle, lreactants)
00687     deallocate(lreactants%components)
00688     deallocate(lreactants)
00689     reactants_handle = c_null_ptr
00690     if (imas_amns_c_debug) write (*,*) "IMAS_AMNS_C_FINISH_REACTANTS: return"
00691   end subroutine imas_amns_c_finish_reactants
00692
00693
00694
00695 end module amns_module_isoc
```

16.101 src/libamns/amns_types.f90 File Reference

Data Types

- type amns_types::amns_version_type

  Type for specifying the AMNS version (not interoperable)
- type amns_types::amns_fc_version_type

  Type for specifying the AMNS version (interoperable with c)
- type amns_types::amns_reactant_type

  Type for indicating a single reactant or product when using the AMNS interface.
- type amns_types::amns_reactants_type

  Type for indicating the reactants when using the AMNS interface NOT interoperable with C.
- type amns_types::amns_handle_type

  type for the AMNS handle (opaque for user codes) NOT interoperable with C.
- type amns_types::amns_handle_rx_type

  Type for the AMNS RX handle (opaque for user codes) NOT interoperable with C.
- type amns_types::amns_error_type

  Type for error returns from the AMNS interface (not interoperable)
- type amns_types::amns_fc_error_type

  Type for error returns from the AMNS interface (interoperable with c)
- type amns_types::amns_reaction_type

  Type used for specifying reactions when using the AMNS interface (not interoperable)
- type amns_types::amns_fc_reaction_type

  Type used for specifying reactions when using the AMNS interface (interoperable with c)
- type amns_types::amns_set_type

  Type for setting parameters in the AMNS package (not interoperable)
- type amns_types::amns_fc_set_type

  Type for setting parameters in the AMNS package (interoperable with c)
- type amns_types::amns_query_type

  Type for querying parameters in the AMNS package (not interoperable)
- type amns_types::amns_fc_query_type

  Type for querying parameters in the AMNS package (interoperable with c)

- type amns_types::amns_answer_type

  Type for answers from queries in the AMNS package (not interoperable)
- type amns_types::amns_fc_answer_type

  Type for answers from queries in the AMNS package (interoperable with c)
- type amns_types::amns_ids_list

  Type for linked list of amns idss NOT interoperable with C.

Modules

- module amns_types

  The derived types defined here are meant to be interoperable with C. The ones for this is not the case are explicitly marked. Note that they are still required when using the AMNS interface from C, but only as opaque handle variables.

Variables

- integer, parameter amns_types::version_length =32

  used to specify the maximum length of the string version number
- integer, parameter amns_types::set_length =32

  maximum length of the string passed by the set calls
- integer, parameter amns_types::reaction_length =16

  maximum length for specifying a reaction type
- integer, parameter amns_types::query_length =16

  maximum length for a query argument
- integer, parameter amns_types::answer_length =128

  maximum length of an answer

16.102 amns_types.f90

```
00001
00007
00014
00015 module amns_types
00016
00017   use ids_types  ! IGNORE
00018   use data_suport
00019   use iso_c_binding  ! IGNORE
00020   use ids_schemas  ! IGNORE
00021
00022   implicit none
00023
00024   include 'git_version_AMNS.h'
00025
00028   integer, parameter :: version_length=32
00031   integer, parameter :: set_length=32
00034   integer, parameter :: reaction_length=16
00037   integer, parameter :: query_length=16
00040   integer, parameter :: answer_length=128
00041
00044   type ::  amns_version_type
00046      character(len=version_length) :: string = ''
00048      integer(c_int) :: number = 0
00050      character(len=version_length) :: backend = 'mdsplus'
00052      character(len=version_length) :: user = ''
00053   end type amns_version_type
00054
00057   type, bind(c) ::  amns_fc_version_type
00059      character(len=1, kind=c_char), dimension(version_length) :: string = ''
00061      integer(c_int) :: number = 0
00063      character(len=1, kind=c_char), dimension(version_length) :: backend = 'mdsplus'
00065      character(len=1, kind=c_char), dimension(version_length) :: user = ''
00066   end type amns_fc_version_type
00067
00070   type, bind(c) ::  amns_reactant_type
00072      real (c_double) :: zn
00074      real (c_double) :: za
00076      real (c_double) :: mi = 0.0_ids_real
00078      integer (c_int) :: lr = 0      ! LR=0 implies LHS; LR=1 implies RHS
00080      real (c_double) :: real_specifier = ids_real_invalid
00082      integer (c_int) :: int_specifier = ids_int_invalid
```

```
00083   end type amns_reactant_type
00084
00088   type :: amns_reactants_type
00090      type (amns_reactant_type), allocatable :: components(:)
00092      character(len=reaction_length), allocatable :: string(:)
00094      integer :: index = 0
00095   end type amns_reactants_type
00096
00100   type :: amns_handle_type
00101      !    private
00102      type (amns_version_type) :: version
00103      integer :: no_of_errors = 0
00104      logical :: debug = .false.
00105      logical :: initialized = .false.
00106      character(len=answer_length) :: &
00107           properties_comment = '', properties_source = '', &
00108           properties_provider = '', properties_creation_date = '',  &
00109           code_name = '', code_commit = '', &
00110           code_version = '', code_repository = ''
00111   end type amns_handle_type
00112
00116   type :: amns_handle_rx_type
00117      !    private
00118      character(len=answer_length) :: &
00119           properties_comment = '', properties_source = '', &
00120           properties_provider = '', properties_creation_date = '',  &
00121           code_name = '', code_commit = '', &
00122           code_version = '', code_repository = ''
00123      character(len=answer_length) :: &
00124           source = '', provider = '', citation = ''
00125      character(len=reaction_length) :: reaction_type = ''
00126      integer :: no_of_reactants = 0
00127      type (amns_version_type) :: version
00128      type (amns_reactant_type), allocatable :: components(:)
00129      character(len=reaction_length), allocatable :: string(:)
00130      integer :: index = 0
00131      logical :: debug, initialized, filled = .false.
00132      type (grid_t) :: grid
00133   end type amns_handle_rx_type
00134
00137   type :: amns_error_type
00139      logical :: flag
00141      character(len=answer_length) :: string
00142   end type amns_error_type
00143
00146   type, bind(c) :: amns_fc_error_type
00148      logical(c_bool) :: flag
00150      character(len=1, kind=c_char), dimension(answer_length) :: string
00151   end type amns_fc_error_type
00152
00155   type :: amns_reaction_type
00157      character(len=reaction_length) :: string
00159      integer(c_int) :: isotope_resolved=0
00160   end type amns_reaction_type
00161
00164   type, bind(c) :: amns_fc_reaction_type
00166      character(len=1, kind=c_char), dimension(reaction_length) :: string
00168      integer(c_int) :: isotope_resolved=0
00169   end type amns_fc_reaction_type
00170
00173   type :: amns_set_type
00175      character(len=set_length) :: string
00176   end type amns_set_type
00177
00180   type, bind(c) :: amns_fc_set_type
00182      character(len=1, kind=c_char), dimension(set_length) :: string
00183   end type amns_fc_set_type
00184
00187   type :: amns_query_type
00189      character(len=query_length) :: string
00190   end type amns_query_type
00191
00194   type, bind(c) :: amns_fc_query_type
00196      character(len=1, kind=c_char), dimension(query_length) :: string
00197   end type amns_fc_query_type
00198
00201   type :: amns_answer_type
00203     character(len=answer_length) :: string
00205     integer(c_int) :: number
00206   end type amns_answer_type
00207
00210   type, bind(c) :: amns_fc_answer_type
00212     character(len=1, kind=c_char), dimension(answer_length) :: string
00214     integer(c_int) :: number
00215  end type amns_fc_answer_type
00216
00219   type :: amns_ids_list
```

```
00220       integer :: shot=0, run = 0
00221       type (ids_amns_data) ::     amns_ids
00222       type (amns_ids_list), pointer :: prev, next => null()
00223    end type amns_ids_list
00224
00225 end module amns_types
```

16.103 src/libamns/amns_utility.f90 File Reference

Data Types

- interface amns_utility::string

Modules

- module amns_utility

  Module implementing various utility functions for the AMNS interface.

Functions/Subroutines

- character ∗24 function amns_utility::int_to_string (int)

  convert an integer to a string

16.104 amns_utility.f90

```
00001
00007
00008 module amns_utility
00009
00010   use amns_types
00011
00012   interface string
00013
00014     module procedure int_to_string
00015
00016   end interface string
00017
00018 contains
00019
00023   character*24 function int_to_string(int)
00024     integer, intent(in) :: int
00025     write(int_to_string,'(I24)') int
00026     int_to_string=adjustl(int_to_string)
00027   end function
00028
00029 end module amns_utility
```

16.105 src/libamns/call_utils.f90 File Reference

Modules

- module call_utils

  utils module from Silvio Gori's grid package

Functions/Subroutines

- subroutine, public call_utils::assert (lcond, message)
- subroutine, public call_utils::warning (lcond, message)
- subroutine, public call_utils::exiting (lcond, message)
- subroutine, public call_utils::exitall ()
- subroutine, public call_utils::sub_init (subin)
- subroutine, public call_utils::sub_end ()

16.106 call_utils.f90

```
00001
00006 module call_utils
00007
00008
00009   use  f90_kind
00010
00011   implicit none
00012
00013   private
00014
00015   integer(ikind), parameter :: max_call_deep=100
00016   character(skind) :: sub_called(max_call_deep)
00017   integer(ikind)   :: sub_called_pos=0
00018
00019
00020   public  :: assert, warning, exiting, exitall, sub_init, sub_end
00021
00022 contains
00023
00024   !-----------------------------------------------
00025   !------- assert ----------------------------------
00026   !-----------------------------------------------
00027   subroutine assert (lcond, message)
00028
00029     logical, intent(in)          :: lcond
00030     character(len=*), intent(in) :: message
00031
00032     if(.not.lcond)then
00033        print*, message
00034        call exitall()
00035     endif
00036   end subroutine assert
00037
00038
00039   !-----------------------------------------------
00040   !------- warning ---------------------------------
00041   !-----------------------------------------------
00042   subroutine warning (lcond, message)
00043
00044     logical, intent(in)          :: lcond
00045     character(len=*), intent(in) :: message
00046     if(.not.lcond)then
00047        print*,"warning!: "// message
00048     endif
00049   end subroutine warning
00050
00051   !-----------------------------------------------
00052   !------- exiting ---------------------------------
00053   !-----------------------------------------------
00054   subroutine exiting (lcond, message)
00055
00056     logical, intent(in)          :: lcond
00057     character(len=*), intent(in) :: message
00058
00059     if(.not.lcond)then
00060        print*,"Panik !: "// message
00061        call exitall()
00062     endif
00063   end subroutine exiting
00064
00065
00066   !-----------------------------------------------
00067   !------- exitall  --------------------------------
00068   !-----------------------------------------------
00069
00070   subroutine exitall()
00071
00072     integer :: i
00073
00074     do i=sub_called_pos,1,-1
00075        print*, trim(sub_called(sub_called_pos))
00076     enddo
00077     stop 'Error - exitall@utils: at eof'
00078
00079   end subroutine exitall
00080
00081   !-----------------------------------------------
00082   !--------- sub_init ------------------------------
00083   !-----------------------------------------------
00084   subroutine sub_init(subin)
00085     character(len=*), intent(in) :: subin
00086     ! print *,' +++> ',subin    , sub_called_pos, max_call_deep
00087     call assert(sub_called_pos<max_call_deep,'sub_init error sub_called_pos>=max_call_deep')
00088     sub_called_pos=sub_called_pos+1
00089     sub_called(sub_called_pos)=trim(subin)
```

```
00090   end subroutine sub_init
00091
00092   !-----------------------------------------------
00093   !--------- sub_end -----------------------------
00094   !-----------------------------------------------
00095   subroutine sub_end()
00096
00097     ! print *,' ---> ',sub_called(sub_called_pos)
00098     sub_called_pos=sub_called_pos-1
00099
00100
00101   end subroutine sub_end
00102
00103 end module call_utils
00104
```

16.107 src/libamns/data_suport.f90 File Reference

Modules

- module data_suport

Functions/Subroutines

- subroutine, public data_suport::delete (grid)

  deallocate a grid
- subroutine, public data_suport::set_option (grid, warning)

  set the "with_warning" flag in grid to the value of "warning"
- subroutine, public data_suport::interpol (w, x, y, z, fd1, fd2, fd3, fd4, grid, data_error)

  interpolate in the grid
- logical function data_suport::sorted (x)

  return true if the passed array is sorted

16.108 data_suport.f90

```
00001
00008 module data_suport
00009
00010   use f90_kind
00011   use call_utils
00012   use unit_h
00013   use strings
00014   use m_mrgrnk
00015
00016   implicit none
00017
00018   private
00019
00020   !data types
00021   !
00022   !--- axes_t ----
00023   type axes_t
00024      real(rkind)   , allocatable  :: x(:)
00025      integer(ikind)               :: dim_x
00026      integer(ikind)               :: lbound
00027      integer(ikind), dimension(2) :: extrapolation_type = (/ 0, 0 /)
00028      logical                      :: is_lin=.false.
00029      logical                      :: is_log=.false.
00030      logical                      :: is_equi=.false.
00031      logical                      :: allocated
00032      integer                      :: last_access
00033      character(len=132)           :: label='', units=''
00034      character(len=132), allocatable :: value_labels(:)
00035   end type axes_t
00036
00037   !
00038   !--- grid_t ----
00039   ! update this and all nesessary routines below for 5D
00040   type grid_t
00041      real(rkind)   , allocatable  :: f1d(:)        ! Values 1D
00042      real(rkind)   , allocatable  :: f2d(:,:)      ! Values 2D
00043      real(rkind)   , allocatable  :: f3d(:,:,:)    ! Values 3D
00044      real(rkind)   , allocatable  :: f4d(:,:,:,:)  ! Values 4D
00045      type (axes_t) , allocatable  :: axe(:)
```

```
00046      logical                      :: exist_f1d=.false.
00047      logical                      :: exist_f2d=.false.
00048      logical                      :: exist_f3d=.false.
00049      logical                      :: exist_f4d=.false.
00050      logical                      :: is_lin=.false.
00051      logical                      :: is_log=.false.
00052      logical                      :: with_warning=.false.
00053      character(len=skind)         :: name=""
00054      integer                      :: interpol_function = 0
00055      character(len=132)           :: result_label=""
00056      character(len=132)           :: result_unit=""
00057      character(len=132)           :: state_label=""
00058      integer                      :: ndim=0
00059    end type grid_t
00060
00061    !
00062    !--- data_error ----
00063    type data_error_t
00064       integer(ikind)              :: ierr
00065       character (len=128)         :: cerr
00066    end type data_error_t
00067
00068
00069    ! update this and all nesessary routines below for 5D
00070
00071    !! \todo Update this interface for 5D and up
00072    interface new_grid
00073       module procedure new_grid_1d,  &
00074            new_grid_2d,  &
00075            new_grid_3d,  &
00076            new_grid_4d
00077    end interface new_grid
00078
00079    public :: grid_t, axes_t
00080    public :: interpol, new_grid, delete
00081    public :: set_option
00082    public :: data_error_t
00083
00084  contains
00085
00086
00088    !-----------------------------------------
00089    !------- new_grid_1 --------------------------
00090    !-----------------------------------------
00091
00092    subroutine new_grid_1d(grid,f,x)
00093      implicit none
00094      real(rkind), intent(in) :: f(:)
00095      real(rkind), intent(in), optional :: x(:)
00096      type(grid_t),intent(out):: grid
00097      integer :: optargs
00098
00099      ! error check
00100      optargs=0
00101      call assert(.not.grid%exist_f1d,"Error: grid%f1d is still allocated")
00102      call assert(.not.grid%exist_f2d,"Error: grid%f2d is still allocated")
00103      call assert(.not.grid%exist_f3d,"Error: grid%f3d is still allocated")
00104      call assert(.not.grid%exist_f4d,"Error: grid%f4d is still allocated")
00105      if(present(x)) optargs=optargs+1
00106      if(optargs.eq.1) then
00107         call assert(size(f)==size(x), "Error: size(f)" //size(f)// " /= =size(x) "//size(x))
00108         call assert(sorted(x(:)),"not sorted x")
00109         call assert(size(x(:))>=2,"not sorted x")
00110         allocate(grid%axe(1))
00111         allocate(grid%axe(1)%x(size(x)))
00112         grid%axe(1)%x(:) = x(:)
00113         grid%axe(1)%dim_x = size(x)
00114         grid%axe(1)%allocated = .true.
00115      endif
00116      allocate(grid%f1d(size(f,1)))
00117      grid%f1d(:)=f(:)
00118      grid%exist_f1d=.true.
00119    end subroutine new_grid_1d
00120
00122    !-----------------------------------------
00123    !------- new_grid_2d --------------------------
00124    !-----------------------------------------
00125
00126    subroutine new_grid_2d(grid,f,x,y)
00127      implicit none
00128      real(rkind), intent(in) :: f(:,:)
00129      real(rkind), intent(in), optional :: x(:)
00130      real(rkind), intent(in), optional :: y(:)
00131      type(grid_t),intent(out):: grid
00132      integer :: optargs
00133
00134      ! error check
```

```
00135     optargs=0
00136     call assert(.not.grid%exist_f1d, "Error: grid%f1d is still allocated")
00137     call assert(.not.grid%exist_f2d, "Error: grid%f2d is still allocated")
00138     call assert(.not.grid%exist_f3d, "Error: grid%f3d is still allocated")
00139     call assert(.not.grid%exist_f4d, "Error: grid%f4d is still allocated")
00140     if(present(x)) optargs=optargs+1
00141     if(present(y)) optargs=optargs+1
00142     if(optargs.eq.2) then
00143        call assert(size(f,1)==size(x), "Error: size(f,1)" //size(f,1)// ' /= =size(x) '//size(x))
00144        call assert(size(f,2)==size(y), "Error: size(f,2)" //size(f,2)// ' /= =size(y) '//size(y))
00145        call assert(sorted(x(:)),"not sorted x")
00146        call assert(sorted(y(:)),"not sorted y")
00147        call assert(size(x(:))>=2,"not sorted x")
00148        call assert(size(y(:))>=2,"not sorted y")
00149        allocate(grid%axe(2))
00150        allocate(grid%axe(1)%x(size(x)))
00151        allocate(grid%axe(2)%x(size(y)))
00152        grid%axe(1)%x(:) = x(:)
00153        grid%axe(2)%x(:) = y(:)
00154        grid%axe(1)%dim_x = size(x)
00155        grid%axe(2)%dim_x = size(y)
00156        grid%axe(1)%allocated = .true.
00157        grid%axe(2)%allocated = .true.
00158     else
00159        call assert(optargs==0, "Error: wrong number of optionl arguments for new_grid_2d")
00160     endif
00161     allocate(grid%f2d(size(f,1),size(f,2)))
00162     grid%f2d(:,:)=f(:,:)
00163     grid%exist_f2d=.true.
00164   end subroutine new_grid_2d
00165
00167   !-----------------------------------------
00168   !------- new_grid_3d -------------------------
00169   !-----------------------------------------
00170
00171   subroutine new_grid_3d(grid,f,x,y,z)
00172     implicit none
00173     real(rkind), intent(in) :: f(:,:,:)
00174     real(rkind), intent(in), optional :: x(:)
00175     real(rkind), intent(in), optional :: y(:)
00176     real(rkind), intent(in), optional :: z(:)
00177     type(grid_t),intent(out):: grid
00178     integer :: optargs
00179
00180     ! error check
00181     optargs=0
00182     call assert(.not.grid%exist_f1d, "Error: grid%f1d is still allocated")
00183     call assert(.not.grid%exist_f2d, "Error: grid%f2d is still allocated")
00184     call assert(.not.grid%exist_f3d, "Error: grid%f3d is still allocated")
00185     call assert(.not.grid%exist_f4d, "Error: grid%f4d is still allocated")
00186     if(present(x)) optargs=optargs+1
00187     if(present(y)) optargs=optargs+1
00188     if(present(z)) optargs=optargs+1
00189     if(optargs.eq.3) then
00190        call assert(size(f,1)==size(x), "Error: size(fx,1)" //size(f,1)// ' /= =size(x) '//size(x))
00191        call assert(size(f,2)==size(y), "Error: size(fx,2)" //size(f,2)// ' /= =size(y) '//size(y))
00192        call assert(size(f,3)==size(z), "Error: size(fx,3)" //size(f,3)// ' /= =size(z) '//size(z))
00193        call assert(sorted(x(:)),"not sorted x")
00194        call assert(sorted(y(:)),"not sorted y")
00195        call assert(sorted(y(:)),"not sorted z")
00196        call assert(size(x(:))>=2,"not sorted x")
00197        call assert(size(y(:))>=2,"not sorted y")
00198        call assert(size(z(:))>=2,"not sorted z")
00199        allocate(grid%axe(3))
00200        allocate(grid%axe(1)%x(size(x)))
00201        allocate(grid%axe(2)%x(size(y)))
00202        allocate(grid%axe(3)%x(size(z)))
00203        grid%axe(1)%x(:) = x(:)
00204        grid%axe(2)%x(:) = y(:)
00205        grid%axe(3)%x(:) = z(:)
00206        grid%axe(1)%dim_x = size(x)
00207        grid%axe(2)%dim_x = size(y)
00208        grid%axe(3)%dim_x = size(z)
00209        grid%axe(1)%allocated = .true.
00210        grid%axe(2)%allocated = .true.
00211        grid%axe(3)%allocated = .true.
00212     else
00213        call assert(optargs==0, "Error: wrong number of optionl arguments for new_grid_3d")
00214     endif
00215     allocate(grid%f3d(size(f,1),size(f,2),size(f,3)))
00216     grid%f3d(:,:,:)=f(:,:,:)
00217     grid%exist_f3d=.true.
00218   end subroutine new_grid_3d
00219
00221   !-----------------------------------------
00222   !------- new_grid_4d -------------------------
00223   !-----------------------------------------
```

```
00224
00225   subroutine new_grid_4d(grid,f,w,x,y,z)
00226     implicit none
00227     real(rkind), intent(in) :: f(:,:,:,:)
00228     real(rkind), intent(in), optional :: w(:)
00229     real(rkind), intent(in), optional :: x(:)
00230     real(rkind), intent(in), optional :: y(:)
00231     real(rkind), intent(in), optional :: z(:)
00232     type(grid_t),intent(out):: grid
00233     integer :: optargs
00234
00235     ! error check
00236     optargs=0
00237     call assert(.not.grid%exist_f1d, "Error: grid%f1d is still allocated")
00238     call assert(.not.grid%exist_f2d, "Error: grid%f2d is still allocated")
00239     call assert(.not.grid%exist_f3d, "Error: grid%f3d is still allocated")
00240     call assert(.not.grid%exist_f4d, "Error: grid%f4d is still allocated")
00241     if(present(w)) optargs=optargs+1
00242     if(present(x)) optargs=optargs+1
00243     if(present(y)) optargs=optargs+1
00244     if(present(z)) optargs=optargs+1
00245     if(optargs.eq.4) then
00246        call assert(size(f,1)==size(w), "Error: size(fx,1)" //size(f,1)// ' /= =size(w) '//size(w))
00247        call assert(size(f,2)==size(x), "Error: size(fx,2)" //size(f,2)// ' /= =size(x) '//size(x))
00248        call assert(size(f,3)==size(y), "Error: size(fx,3)" //size(f,3)// ' /= =size(y) '//size(y))
00249        call assert(size(f,4)==size(z), "Error: size(fx,4)" //size(f,4)// ' /= =size(z) '//size(z))
00250        call assert(sorted(w(:)),"not sorted w")
00251        call assert(sorted(x(:)),"not sorted x")
00252        call assert(sorted(y(:)),"not sorted y")
00253        call assert(sorted(z(:)),"not sorted z")
00254        call assert(size(w(:))>=2,"not sorted w")
00255        call assert(size(x(:))>=2,"not sorted x")
00256        call assert(size(y(:))>=2,"not sorted y")
00257        call assert(size(z(:))>=2,"not sorted z")
00258        allocate(grid%axe(4))
00259        allocate(grid%axe(1)%x(size(w)))
00260        allocate(grid%axe(2)%x(size(x)))
00261        allocate(grid%axe(3)%x(size(y)))
00262        allocate(grid%axe(4)%x(size(z)))
00263        grid%axe(1)%x(:) = w(:)
00264        grid%axe(2)%x(:) = x(:)
00265        grid%axe(3)%x(:) = y(:)
00266        grid%axe(4)%x(:) = z(:)
00267        grid%axe(1)%dim_x = size(w)
00268        grid%axe(2)%dim_x = size(x)
00269        grid%axe(3)%dim_x = size(y)
00270        grid%axe(4)%dim_x = size(z)
00271        grid%axe(1)%allocated = .true.
00272        grid%axe(2)%allocated = .true.
00273        grid%axe(3)%allocated = .true.
00274        grid%axe(4)%allocated = .true.
00275     else
00276        call assert(optargs==0, "Error: wrong number of optionl arguments for new_grid_4d")
00277     endif
00278     allocate(grid%f4d(size(f,1),size(f,2),size(f,3),size(f,4)))
00279     grid%f4d(:,:,:,:)=f(:,:,:,:)
00280     grid%exist_f4d=.true.
00281   end subroutine new_grid_4d
00282
00283
00285   !-----------------------------------------
00286   !------- delete ----------------------------
00287   !-----------------------------------------
00288   !this routine can also been used when the grid is not set.
00289   subroutine delete(grid)
00290     implicit none
00291     type(grid_t),intent(inout):: grid
00292
00293     if(allocated(grid%f1d))then
00294        deallocate(grid%f1d)
00295        grid%exist_f1d=.false.
00296     endif
00297     if(allocated(grid%f2d))then
00298        deallocate(grid%f2d)
00299        grid%exist_f2d=.false.
00300     endif
00301     if(allocated(grid%f3d))then
00302        deallocate(grid%f3d)
00303        grid%exist_f3d=.false.
00304     endif
00305     if(allocated(grid%f4d))then
00306        deallocate(grid%f4d)
00307        grid%exist_f4d=.false.
00308     endif
00309     if(allocated(grid%axe)) then
00310        deallocate(grid%axe)
00311     endif
```

```
00312   end subroutine delete
00313
00315   !---------------------------------
00316   !----- set_option ----------------
00317   !---------------------------------
00318   subroutine set_option(grid,warning)
00319     implicit none
00320     type(grid_t)         :: grid
00321     logical,optional     :: warning
00322
00323     if(present(warning)) grid%with_warning =warning
00324
00325   end subroutine set_option
00326
00340
00341   subroutine interpol(w,x,y,z,fd1,fd2,fd3,fd4,grid,data_error)
00342     use amns_external_functions
00343     implicit none
00344     real (rkind), target           ,intent(in) :: w(:)
00345     real (rkind), target, optional ,intent(in) :: x(:)
00346     real (rkind), target, optional ,intent(in) :: y(:)
00347     real (rkind), target, optional ,intent(in) :: z(:)
00348     real (rkind),         optional ,intent(out):: fd1(:)
00349     real (rkind),         optional ,intent(out):: fd2(:,:)
00350     real (rkind),         optional ,intent(out):: fd3(:,:,:)
00351     real (rkind),         optional ,intent(out):: fd4(:,:,:,:)
00352     type (grid_t)                  ,intent(inout):: grid
00353     type (data_error_t),             intent(out):: data_error
00354
00355     ! interpolation index
00356     !
00357     real (rkind) :: dw(size(w))
00358     integer      :: iw(size(w))
00359     integer      :: iw_sort(size(w))
00360     real (rkind), target :: w_s(size(w))
00361     real (rkind), pointer :: w_p(:)
00362
00363     real (rkind), allocatable         :: dx(:)  !size(x))
00364     integer , allocatable             :: ix(:)  !size(x))
00365     integer , allocatable             :: ix_sort(:)  !size(x))
00366     real (rkind), allocatable, target :: x_s(:)  !size(x))
00367     real (rkind), pointer             :: x_p(:)
00368
00369     real (rkind), allocatable         :: dy(:)  !size(yvec))
00370     integer , allocatable             :: iy(:)  !size(yvec))
00371     integer, allocatable              :: iy_sort(:)  !size(yvec))
00372     real (rkind), allocatable, target :: y_s(:)  !size(yvec))
00373     real (rkind), pointer             :: y_p(:)
00374
00375     real (rkind), allocatable         :: dz(:)  !size(zvec))
00376     integer  , allocatable            :: iz(:)  !size(zvec))
00377     integer , allocatable             :: iz_sort(:)  !size(zvec))
00378     real (rkind), allocatable, target :: z_s(:)  !size(zvec))
00379     real (rkind), pointer             :: z_p(:)
00380
00381     !local
00382     logical  :: with_x      ! =present(x)
00383     logical  :: with_y      ! =present(y)
00384     logical  :: with_z      ! =present(z)
00385     logical  :: one_d       ! =present(fd1)
00386     logical  :: two_d       ! =present(fd2)
00387     logical  :: three_d     ! =present(fd3)
00388     logical  :: four_d      ! =present(fd4)
00389     logical  :: is_sorted_w = .false. ! =present(fgrid3)
00390     logical  :: is_sorted_x = .false. ! =present(fgrid3)
00391     logical  :: is_sorted_y = .false. ! =present(fgrid3)
00392     logical  :: is_sorted_z = .false. ! =present(fgrid3)
00393
00394     ! help swap
00395     integer :: iswap
00396     !loop index
00397     integer :: i, idpc
00398
00399     type (fun_err_t) :: fun_err
00400
00401     data_error%ierr = 0
00402     data_error%cerr = ''
00403     fun_err%ierr = 0
00404     fun_err%cerr = ''
00405
00406     with_x   =present(x)
00407     with_y   =present(y)
00408     with_z   =present(z)
00409     one_d    =present(fd1)
00410     two_d    =present(fd2)
00411     three_d  =present(fd3)
00412     four_d   =present(fd4)
```

```
00413
00414     !-irregular calls---
00415     call assert(grid%exist_f1d .or. grid%exist_f2d .or.grid%exist_f3d.or.grid%exist_f4d, &
00416          "Data not allocated for interpolation")
00417
00418     if(one_d) then
00419        call assert(one_d.and. .not.(two_d.or.three_d.or.four_d), &
00420             "Type mismatch, only one f vector type possible!!")
00421        if(with_x) call assert(size(x)==size(w), "Size w:"//size(w)// &
00422             "/= Size x:"//size(x))
00423        if(with_y) call assert(size(y)==size(w), "Size w:"//size(w)// &
00424             "/= Size y:"//size(y))
00425        if(with_z) call assert(size(z)==size(w), "Size w:"//size(x)// &
00426             "/= Size z:"//size(z))
00427     endif
00428
00429     if(two_d)then
00430        call assert(with_x.and..not.(with_y.or.with_z),"expect x present, y and z not present")
00431     endif
00432     if(three_d)then
00433        call assert(with_y .and. .not. with_z,"expect x and y present and z not present")
00434     endif
00435     if(four_d)then
00436        call assert(with_y .and. with_z,"expect x y and zvec present")
00437     endif
00438
00439     select case(grid%interpol_function)
00440
00441     case (0)
00443        if(with_x)then
00444           allocate(dx(size(x)),ix(size(x)),ix_sort(size(x)),x_s(size(x)))
00445        endif
00446        if(with_y)then
00447           allocate(dy(size(y)),iy(size(y)),iy_sort(size(y)),y_s(size(y)))
00448        endif
00449        if(with_z)then
00450           allocate(dz(size(z)),iz(size(z)),iz_sort(size(z)),z_s(size(z)))
00451        endif
00452
00453        is_sorted_w=sorted(w(:))
00454        if(with_x)is_sorted_x=sorted(x(:))
00455        if(with_y)is_sorted_y=sorted(y(:))
00456        if(with_z)is_sorted_z=sorted(z(:))
00457
00458     ! if the vectors are not sorted, we create a sorted vector, and save index map
00459     ! pointer x_p will show a sorted vector
00460
00461        if(.not.is_sorted_w) then
00462           iw_sort(:)= (/ (i,i=1,size(iw_sort(:))   ) /)
00463           call mrgrnk(w(:),iw_sort(:))
00464           w_s(:)=w(iw_sort(:))
00465           w_p => w_s
00466           if(grid%axe(1)%is_log) w_p = log10(w_p)
00467        else
00468           if(grid%axe(1)%is_log) then
00469              w_s = log10(w)
00470              w_p => w_s
00471           else
00472              w_p => w
00473           endif
00474        endif
00475
00476        if(with_x) then
00477           if(.not.is_sorted_x) then
00478              ix_sort(:)= (/ (i,i=1,size(ix_sort(:))   ) /)
00479              call mrgrnk(x(:),ix_sort(:))
00480              x_s(:)=x(ix_sort(:))
00481              x_p => x_s
00482              if(grid%axe(2)%is_log) x_p = log10(x_p)
00483           else
00484              if(grid%axe(2)%is_log) then
00485                 x_s = log10(x)
00486                 x_p => x_s
00487              else
00488                 x_p => x
00489              endif
00490           endif
00491        endif
00492
00493        if(with_y) then
00494           if(.not.is_sorted_y)then
00495              iy_sort(:)= (/ (i,i=1,size(iy_sort(:))   ) /)
00496              call mrgrnk(y(:),iy_sort(:))
00497              y_s(:)=y(iy_sort(:))
00498              y_p => y_s
00499              if(grid%axe(3)%is_log) y_p = log10(y_p)
00500           else
```

```
00501             if(grid%axe(3)%is_log) then
00502                y_s = log10(y)
00503                y_p => y_s
00504             else
00505                y_p => y
00506             endif
00507          endif
00508       endif
00509
00510       if(with_z) then
00511          if(.not.is_sorted_z)then
00512             iz_sort(:)= (/ (i,i=1,size(iz_sort(:))   ) /)
00513             call mrgrnk(z(:),iz_sort(:))
00514             z_s(:)=z(ix_sort(:))
00515             z_p => z_s
00516             if(grid%axe(4)%is_log) z_p = log10(z_p)
00517          else
00518             z_p => z
00519             if(grid%axe(4)%is_log) then
00520                z_s = log10(z)
00521                z_p => z_s
00522             else
00523                z_p => z
00524             endif
00525          endif
00526       endif
00527
00528
00529    !--get index of w in the grid
00530       call interpol_index(w_p(:),grid%axe(1)%x(:),iw(:),dw(:),grid)
00531    ! reset ix to original xvec positions
00532    ! next comand work, because internaly the f90 compiler works with copy comands
00533
00534       if(.not. is_sorted_w)then
00535          iw(iw_sort(:))=iw(:)
00536          dw(iw_sort(:))=dw(:)
00537       endif
00538
00539
00540       if(with_x) then
00541          idpc=size(x)/2
00542          call interpol_index(x_p(:),grid%axe(2)%x(:),ix(:),dx(:),grid)
00543          if(.not. is_sorted_x)then
00544             ix(ix_sort(:))=ix(:)
00545             dx(ix_sort(:))=dx(:)
00546          endif
00547       endif
00548
00549       if(with_y) then
00550          call interpol_index(y_p(:),grid%axe(3)%x(:),iy(:),dy(:),grid)
00551          if(.not. is_sorted_y)then
00552             iy(iy_sort(:))=iy(:)
00553             dy(iy_sort(:))=dy(:)
00554          endif
00555       endif
00556
00557       if(with_z) then
00558          call interpol_index(z_p(:),grid%axe(4)%x(:),iz(:),dz(:),grid)
00559          if(.not. is_sorted_z)then
00560             iz(iz_sort(:))=iz(:)
00561             dz(iz_sort(:))=dz(:)
00562          endif
00563       endif
00564    !
00565    !-- now interpolate
00566
00567       if(with_z)then
00568          if(one_d)then
00569             call interpo_4_1(iw(:),ix(:),iy(:),iz(:),dw(:),dx(:),dy(:),dz(:),fd1(:),grid)
00570          else
00571             call interpo_4_4(iw(:),ix(:),iy(:),iz(:),dw(:),dx(:),dy(:),dz(:),fd4(:,:,:,:),grid)
00572          endif
00573       elseif(with_y)then
00574          if(one_d)then
00575             call interpo_3_1(iw(:),ix(:),iy(:),dw(:),dx(:),dy(:),fd1(:),grid)
00576          else
00577             call interpo_3_3(iw(:),ix(:),iy(:),dw(:),dx(:),dy(:),fd3(:,:,:),grid)
00578          endif
00579       elseif(with_x)then
00580          if(one_d)then
00581             call interpo_2_1(iw(:),ix(:),dw(:),dx(:),fd1(:),grid)
00582          else
00583             call interpo_2_2(iw(:),ix(:),dw(:),dx(:),fd2(:,:),grid)
00584          endif
00585       else
00586          call interpo_1_1(iw(:),dw(:),fd1(:),grid)
00587       endif
```

```
00588
00589         if(with_x)then
00590            deallocate(dx,ix,ix_sort,x_s)
00591         endif
00592         if(with_y)then
00593            deallocate(dy,iy,iy_sort,y_s)
00594         endif
00595         if(with_z)then
00596            deallocate(dz,iz,iz_sort,z_s)
00597         endif
00598
00599         if(grid%is_log) then
00600            if(one_d)   fd1 = 10.0_rkind ** fd1
00601            if(two_d)   fd2 = 10.0_rkind ** fd2
00602            if(three_d) fd3 = 10.0_rkind ** fd3
00603            if(four_d)  fd4 = 10.0_rkind ** fd4
00604         endif
00605
00606      case (1001)    ! grid%interpol_function
00608         call assert(.not.(with_x.or.with_y.or.with_z), "nuclear data '1001' must be a function of 1
       parameter")
00609         call assert(grid%exist_f2d, "nuclear data '1001' requires 2d table of data")
00610         call assert(one_d, "nuclear data '1001' expected to be passed an effective 1d array")
00611         call nuclear_data_1001(grid%f2d, w, fd1, grid%with_warning, fun_err)
00612
00613      case (1002)    ! grid%interpol_function
00615         call assert(.not.(with_x.or.with_y.or.with_z), "nuclear data '1002' must be a function of 1
       parameter")
00616         call assert(grid%exist_f2d, "nuclear data '1002' requires 2d table of data")
00617         call assert(one_d, "nuclear data '1002' expected to be passed an effective 1d array")
00618         call nuclear_data_1002(grid%f2d, w, fd1, grid%with_warning, fun_err)
00619
00620      case (1003)    ! grid%interpol_function
00622         call assert(.not.(with_x.or.with_y.or.with_z), "resonant charge data '1003' must be a function
       of 1 parameter")
00623         call assert(grid%exist_f1d, "resonant charge transfer data '1003' requires 1d table of data")
00624         call assert(one_d, "resonant charge transfer '1003' expected to be passed an effective 1d
       array")
00625         call rct_data_1003(grid%f1d, w, fd1, grid%with_warning, fun_err)
00626
00627      case (1004)    ! grid%interpol_function
00629         call assert(.not.(with_y.or.with_z), "sputter data '1004' must be a function of 2 parameters")
00630         call assert(grid%exist_f2d, "sputter yield '1004' requires 2d table of data")
00631         call assert(one_d, "sputter yield '1004' expected to be passed an 2d array")
00632         call sputter_data_1004(grid%f2d, w, x, fd1, grid%with_warning, fun_err)
00633
00634      case (1005)    ! grid%interpol_function
00636         call assert(.not.(with_y.or.with_z), "reflection yield '1005' must be a function of 2
       parameters")
00637         call assert(grid%exist_f2d, "reflection yield '1005' requires 2d table of data")
00638         call assert(one_d, "reflection yield '1005' expected to be passed an 2d array")
00639         call reflect_data_1005(grid%f2d, w, x, fd1, grid%with_warning, fun_err)
00640
00641      case (1006)    ! grid%interpol_function
00643         call assert(.not.(with_x.or.with_y.or.with_z), "nuclear data '1006' must be a function of 1
       parameter")
00644         call assert(grid%exist_f2d, "nuclear data '1006' requires 2d table of data")
00645         call assert(one_d, "nuclear data '1006' expected to be passed an effective 1d array")
00646         call nuclear_data_1006(grid%f2d, w, fd1, grid%with_warning, fun_err)
00647
00648      case default
00649         write(*,*) 'Case for grid%interpol_function = ', grid%interpol_function, ' not yet coded'
00650
00651      end select
00652
00653      if(fun_err%ierr.ne.0) then
00654         data_error%ierr = fun_err%ierr
00655         data_error%cerr = fun_err%cerr
00656      endif
00657
00658    end subroutine interpol
00659
00661    !---------------------------------------
00662    !--- sorted ----------------------------
00663    !---------------------------------------
00664    ! assending order
00665    function sorted(x) result(result)
00666      real(rkind), intent (in) :: x(:)
00667      logical ::result
00668
00669      !  logical:: order
00670      integer:: i
00671
00672      result=.true.
00673      do i=1,size(x)-1
00674         if(x(i)>x(i+1))then
00675            result=.false.
```

```
00676           exit
00677        endif
00678     enddo
00679   end function sorted
00680
00682   !----------------------------------------
00683   !------ interpol_index --------------------
00684   !----------------------------------------
00685   subroutine interpol_index(x, axe, ix, dx, grid)
00686     implicit none
00687     real(rkind)   , intent(in) :: x(:)
00688     real(rkind)   , intent(out):: axe(:)
00689     real(rkind)   , intent(out):: dx(:)
00690     integer(ikind), intent(out):: ix(:)
00691     type(grid_t)  , intent(in) :: grid
00692
00693
00694     real(rkind)   :: dx_help(size(dx))
00695     integer(ikind):: ix_lb,ix_ub
00696
00697     ! after this call
00698     !  ix_lb points to the 1st point >= xaxe(1)
00699     !  ux_ub points to the last point <= xaxe(size(axe))
00700     !  for these outside points ix is set to 0 or size(axe)+1 and
00701     !    dx has been set properly
00702     call bound_check(axe(:), x(:),ix(:),dx(:),ix_lb,ix_ub, grid)
00703
00704     if(ix_lb>ix_ub) return
00705     ! find other location of x in respect of bounds
00706     call locate(axe, x(ix_lb),ix(ix_lb),dx(ix_lb))
00707     call hunt(axe, x(ix_lb+1:ix_ub),ix(ix_lb+1:ix_ub),dx(ix_lb+1:ix_ub))
00708     ! else
00709     !   call locate(axe, xvec(:),ix(:),dx(:))
00710     ! endif
00711
00712   end subroutine interpol_index
00713
00715   !----------------------------------------
00716   !--- bound_check ------------------------
00717   !----------------------------------------
00718   subroutine bound_check(axe, x, ix, dx, lb, ub, grid)
00719     implicit none
00720     real(rkind)   , intent(in) :: axe(:)
00721     real(rkind)   , intent(in) :: x(:)
00722     real(rkind)   , intent(out):: dx(:)
00723     integer(ikind), intent(out):: ix(:)
00724     integer(ikind), intent(out):: lb, ub
00725     type(grid_t)  , intent(in) :: grid
00726
00727
00728     integer:: i
00729
00730     !axe and x are sorted in ascending order
00731     !left bound
00732     lb=1
00733     do i=1, size(x)
00734        if(x(i)>=axe(1)) then
00735           exit
00736        else
00737           ix(i)=0
00738           lb=lb+1
00739           dx(i)=(x(i)-axe(1))/(axe(2)-axe(1))
00740           if(grid%with_warning) call warning(.false.,"Warning "// x(i) &
00741                // " < "// axe(1)// 'in grid: ?')
00742        endif
00743     enddo
00744
00745     ! right bound
00746     ub=size(x)
00747     do i=size(x),1,-1
00748        if(x(i)<=axe(size(axe))) then
00749           exit
00750        else
00751           ix(i)=size(axe)+1
00752           ub=ub-1
00753           dx(i)=(x(i)-axe(size(axe)-1))/(axe(size(axe))-axe(size(axe)-1))
00754           if(grid%with_warning) call warning(.false.," Point "// x(i) &
00755                // " out of bound in grid ")
00756        endif
00757     enddo
00758
00759   end subroutine bound_check
00760
00762   !----------------------------------------
00763   !------- interpo_1_1 ----------------------
00764   !----------------------------------------
00765   subroutine interpo_1_1(ix,dx,fgrid1,grid)
```

```
00766     implicit none
00767     integer(ikind), intent(in) :: ix(:)
00768     real(rkind), intent(out):: fgrid1(:)
00769     real(rkind), intent(in) :: dx(:)
00770     type(grid_t),intent(in) :: grid
00771
00772     integer :: i
00773     integer :: ii1, ii2
00774     !lin
00775     do i=1,size(ix(:))
00776        ii1=ix(i)
00777        if(ix(i)<lbound(grid%f1d,1)) ii1=lbound(grid%f1d,1)
00778        if(ix(i)>ubound(grid%f1d,1)) ii1=ubound(grid%f1d,1)-1
00779        ii2=ii1+1
00780
00781        fgrid1(i)=grid%f1d(ii1) &
00782             +(grid%f1d(ii2) &
00783             -grid%f1d(ii1))*dx(i)
00784     enddo
00785   end subroutine interpo_1_1
00786
00787
00789   !-----------------------------------------
00790   !------- interpo_2_1 ----------------------
00791   !-----------------------------------------
00792   subroutine interpo_2_1(ix,iy,dx,dy,fgrid1,grid)
00793     implicit none
00794     integer(ikind), intent(in) :: ix(:)
00795     integer(ikind), intent(in) :: iy(:)
00796     real(rkind), intent(out):: fgrid1(:)
00797     real(rkind), intent(in) :: dx(:)
00798     real(rkind), intent(in) :: dy(:)
00799     type(grid_t),intent(in) :: grid
00800
00801     real(rkind)::f1, f2, f3
00802     integer :: i
00803     integer :: ii1, ii2, jj1, jj2
00804     !lin
00805     do i=1,size(ix(:))
00806        ii1=ix(i)
00807        if(ii1<lbound(grid%f2d,1)) ii1=lbound(grid%f2d,1)
00808        if(ii1>=ubound(grid%f2d,1)) ii1=ubound(grid%f2d,1)-1
00809        ii2=ii1+1
00810        jj1=iy(i)
00811        if(jj1<lbound(grid%f2d,2)) jj1=lbound(grid%f2d,2)
00812        if(jj1>=ubound(grid%f2d,2)) jj1=ubound(grid%f2d,2)-1
00813        jj2=jj1+1
00814 !!!
00815        if(ii1.lt.1.or.ii2.gt.ubound(grid%f2d,1)) then
00816           write(*,*) 'Error: ii1, ii2 = ', ii1, ii2
00817        endif
00818        if(jj1.lt.1.or.jj2.gt.ubound(grid%f2d,2)) then
00819           write(*,*) 'Error: jj1, jj2 = ', jj1, jj2
00820        endif
00821        f1=grid%f2d(ii1,jj1) &
00822             +(grid%f2d(ii2,jj1) &
00823             -grid%f2d(ii1,jj1))*dx(i)
00824        f2=grid%f2d(ii1,jj2) &
00825             +(grid%f2d(ii2,jj2) &
00826             -grid%f2d(ii1,jj2))*dx(i)
00827        fgrid1(i)=f1+(f2-f1)*dy(i)
00828     enddo
00829   end subroutine interpo_2_1
00830
00832   !-----------------------------------------
00833   !------- interpo_2_2 ----------------------
00834   !-----------------------------------------
00835   subroutine interpo_2_2(ix,iy,dx,dy,fgrid2,grid)
00836     implicit none
00837     integer(ikind), intent(in) :: ix(:)
00838     integer(ikind), intent(in) :: iy(:)
00839     real(rkind), intent(out):: fgrid2(:,:)
00840     real(rkind), intent(in) :: dx(:)
00841     real(rkind), intent(in) :: dy(:)
00842     type(grid_t),intent(in) :: grid
00843
00844     real(rkind)::f1, f2
00845     integer :: i, j
00846     integer :: ii1, ii2, jj1, jj2
00847     !lin
00848     do j=1,size(iy(:))
00849        jj1=iy(j)
00850        if(jj1<lbound(grid%f2d,2)) jj1=lbound(grid%f2d,2)
00851        if(jj1>ubound(grid%f2d,2)) jj1=ubound(grid%f2d,2)-1
00852        jj2=jj1+1
00853        do i=1,size(ix(:))
00854           ii1=ix(i)
```

```
00855           if(ii1<lbound(grid%f2d,1)) ii1=lbound(grid%f2d,1)
00856           if(ii1>ubound(grid%f2d,1)) ii1=ubound(grid%f2d,1)-1
00857           ii2=ii1+1
00858           f1=grid%f2d(ii1,jj1) &
00859                +(grid%f2d(ii2,jj1) &
00860                -grid%f2d(ii1,jj1))*dx(i)
00861           f2=grid%f2d(ii1,jj2) &
00862                +(grid%f2d(ii2,jj2) &
00863                -grid%f2d(ii1,jj2))*dx(i)
00864           fgrid2(i,j)=f1+(f2-f1)*dy(j)
00865        enddo
00866     enddo
00867   end subroutine interpo_2_2
00868
00870   !-----------------------------------------
00871   !------- interpo_3_1 ----------------------
00872   !-----------------------------------------
00873   subroutine interpo_3_1(ix,iy,iz,dx,dy,dz,fgrid1,grid)
00874     implicit none
00875     integer(ikind), intent(in) :: ix(:)
00876     integer(ikind), intent(in) :: iy(:)
00877     integer(ikind), intent(in) :: iz(:)
00878     real(rkind), intent(out):: fgrid1(:)
00879     real(rkind), intent(in) :: dx(:)
00880     real(rkind), intent(in) :: dy(:)
00881     real(rkind), intent(in) :: dz(:)
00882     type(grid_t),intent(in) :: grid
00883
00884     real(rkind)::f1, f2, f3, f4
00885     integer :: i
00886     integer :: ii1, ii2, jj1, jj2, kk1, kk2
00887     !lin
00888     do i=1,size(ix(:))
00889        ii1=ix(i)
00890        if(ii1<lbound(grid%f3d,1)) ii1=lbound(grid%f3d,1)
00891        if(ii1>ubound(grid%f3d,1)) ii1=ubound(grid%f3d,1)-1
00892        ii2=ii1+1
00893        jj1=iy(i)
00894        if(jj1<lbound(grid%f3d,2)) jj1=lbound(grid%f3d,2)
00895        if(jj1>ubound(grid%f3d,2)) jj1=ubound(grid%f3d,2)-1
00896        jj2=jj1+1
00897        kk1=iz(i)
00898        if(kk1<lbound(grid%f3d,3)) kk1=lbound(grid%f3d,3)
00899        if(kk1>ubound(grid%f3d,3)) kk1=ubound(grid%f3d,3)-1
00900        kk2=kk1+1
00901        f1=grid%f3d(ii1,jj1,kk1) &
00902             +(grid%f3d(ii2,jj1,kk1) &
00903             -grid%f3d(ii1,jj1,kk1))*dx(i)
00904        f2=grid%f3d(ii1,jj2,kk1) &
00905             +(grid%f3d(ii2,jj2,kk1) &
00906             -grid%f3d(ii1,jj2,kk1))*dx(i)
00907        f3=f1+(f2-f1)*dy(i)
00908        f1=grid%f3d(ii1,jj1,kk2) &
00909             +(grid%f3d(ii2,jj1,kk2) &
00910             -grid%f3d(ii1,jj1,kk2))*dx(i)
00911        f2=grid%f3d(ii1,jj2,kk2) &
00912             +(grid%f3d(ii2,jj2,kk2) &
00913             -grid%f3d(ii1,jj2,kk2))*dx(i)
00914        f4=f1+(f2-f1)*dy(i)
00915        fgrid1(i)=f3+(f4-f3)*dz(i)
00916     enddo
00917   end subroutine interpo_3_1
00918
00920   !-----------------------------------------
00921   !------- interpo_3_3 ----------------------
00922   !-----------------------------------------
00923   subroutine interpo_3_3(ix,iy,iz,dx,dy,dz,fgrid3,grid)
00924     implicit none
00925     integer(ikind), intent(in) :: ix(:)
00926     integer(ikind), intent(in) :: iy(:)
00927     integer(ikind), intent(in) :: iz(:)
00928     real(rkind), intent(out):: fgrid3(:,:,:)
00929     real(rkind), intent(in) :: dx(:)
00930     real(rkind), intent(in) :: dy(:)
00931     real(rkind), intent(in) :: dz(:)
00932     type(grid_t),intent(in) :: grid
00933
00934     real(rkind)::f1, f2, f3, f4
00935     integer :: i,j,k
00936     integer :: ii1, ii2, jj1, jj2, kk1, kk2
00937     !lin
00938     do k=1,size(iz(:))
00939        kk1=iz(k)
00940        if(kk1<lbound(grid%f3d,3)) kk1=lbound(grid%f3d,3)
00941        if(kk1>ubound(grid%f3d,3)) kk1=ubound(grid%f3d,3)-1
00942        kk2=kk1+1
00943        do j=1,size(iy(:))
```

```
00944           jj1=iy(j)
00945           if(jj1<lbound(grid%f3d,2)) jj1=lbound(grid%f3d,2)
00946           if(jj1>ubound(grid%f3d,2)) jj1=ubound(grid%f3d,2)-1
00947           jj2=jj1+1
00948           do i=1,size(ix(:))
00949              ii1=ix(i)
00950              if(ii1<lbound(grid%f3d,1)) ii1=lbound(grid%f3d,1)
00951              if(ii1>ubound(grid%f3d,1)) ii1=ubound(grid%f3d,1)-1
00952              ii2=ii1+1
00953              f1=grid%f3d(ii1,jj1,kk1)&
00954                   +(grid%f3d(ii2,jj1,kk1) &
00955                   -grid%f3d(ii1,jj1,kk1))*dx(i)
00956              f2=grid%f3d(ii1,jj2,kk1) &
00957                   +(grid%f3d(ii2,jj2,kk1) &
00958                   -grid%f3d(ii1,jj2,kk1))*dx(i)
00959              f3=f1+(f2-f1)*dy(j)
00960              f1=grid%f3d(ii1,jj1,kk2) &
00961                   +(grid%f3d(ii2,jj1,kk2) &
00962                   -grid%f3d(ii1,jj1,kk2))*dx(i)
00963              f2=grid%f3d(ii1,jj2,kk2) &
00964                   +(grid%f3d(ii2,jj2,kk2) &
00965                   -grid%f3d(ii1,jj2,kk2))*dx(i)
00966              f4=f1+(f2-f1)*dy(j)
00967              fgrid3(i,j,k)=f3+(f4-f3)*dz(k)
00968           enddo
00969        enddo
00970     enddo
00971   end subroutine interpo_3_3
00972
00974   !----------------------------------------
00975   !------- interpo_4_1 ----------------------
00976   !----------------------------------------
00977   subroutine interpo_4_1(iw,ix,iy,iz,dw,dx,dy,dz,fgrid1,grid)
00978     implicit none
00979     integer(ikind), intent(in) :: iw(:)
00980     integer(ikind), intent(in) :: ix(:)
00981     integer(ikind), intent(in) :: iy(:)
00982     integer(ikind), intent(in) :: iz(:)
00983     real(rkind), intent(out):: fgrid1(:)
00984     real(rkind), intent(in) :: dw(:)
00985     real(rkind), intent(in) :: dx(:)
00986     real(rkind), intent(in) :: dy(:)
00987     real(rkind), intent(in) :: dz(:)
00988     type(grid_t),intent(in) :: grid
00989
00990     real(rkind)::f1, f2, f3, f4, f5, f6
00991     integer :: i
00992     integer :: ll1,ll2,ii1,ii2,jj1,jj2,kk1,kk2
00993     !lin
00994     do i=1,size(iw(:))
00995        ll1=iw(i)
00996        if(ll1<lbound(grid%f4d,1)) ll1=lbound(grid%f4d,1)
00997        if(ll1>ubound(grid%f4d,1)) ll1=ubound(grid%f4d,1)-1
00998        ll2=ll1+1
00999        ii1=ix(i)
01000        if(ii1<lbound(grid%f4d,2)) ii1=lbound(grid%f4d,2)
01001        if(ii1>ubound(grid%f4d,2)) ii1=ubound(grid%f4d,2)-1
01002        ii2=ii1+1
01003        jj1=iy(i)
01004        if(jj1<lbound(grid%f4d,3)) jj1=lbound(grid%f4d,3)
01005        if(jj1>ubound(grid%f4d,3)) jj1=ubound(grid%f4d,3)-1
01006        jj2=jj1+1
01007        kk1=iz(i)
01008        if(kk1<lbound(grid%f4d,4)) kk1=lbound(grid%f4d,4)
01009        if(kk1>ubound(grid%f4d,4)) kk1=ubound(grid%f4d,4)-1
01010        kk2=kk1+1
01011        ! building interpolation for first cube
01012        f1=grid%f4d(ll1,ii1,jj1,kk1) &
01013             +(grid%f4d(ll2,ii1,jj1,kk1) &
01014             -grid%f4d(ll1,ii1,jj1,kk1))*dw(i)
01015        f2=grid%f4d(ll1,ii2,jj1,kk1) &
01016             +(grid%f4d(ll2,ii2,jj1,kk1) &
01017             -grid%f4d(ll1,ii2,jj1,kk1))*dw(i)
01018        f3=f1+(f2-f1)*dx(i)
01019        f1=grid%f4d(ll1,ii1,jj2,kk1) &
01020             +(grid%f4d(ll2,ii1,jj2,kk1) &
01021             -grid%f4d(ll1,ii1,jj2,kk1))*dw(i)
01022        f2=grid%f4d(ll1,ii2,jj2,kk1) &
01023             +(grid%f4d(ll2,ii2,jj2,kk1) &
01024             -grid%f4d(ll1,ii2,jj2,kk1))*dw(i)
01025        f4=f1+(f2-f1)*dx(i)
01026
01027        f5=f3+(f4-f3)*dy(i)
01028        ! done
01029        ! build second cube
01030        f1=grid%f4d(ll1,ii1,jj1,kk2) &
01031             +(grid%f4d(ll2,ii1,jj1,kk2) &
```

```
01032            -grid%f4d(ll1,ii1,jj1,kk2))*dw(i)
01033       f2=grid%f4d(ll1,ii2,jj1,kk2) &
01034            +(grid%f4d(ll2,ii2,jj1,kk2) &
01035            -grid%f4d(ll1,ii2,jj1,kk2))*dw(i)
01036       f3=f1+(f2-f1)*dx(i)
01037       f1=grid%f4d(ll1,ii1,jj2,kk2) &
01038            +(grid%f4d(ll2,ii1,jj2,kk2) &
01039            -grid%f4d(ll1,ii1,jj2,kk2))*dw(i)
01040       f2=grid%f4d(ll1,ii2,jj2,kk2) &
01041            +(grid%f4d(ll2,ii2,jj2,kk2) &
01042            -grid%f4d(ll1,ii2,jj2,kk2))*dw(i)
01043       f4=f1+(f2-f1)*dx(i)
01044
01045       f6=f3+(f4-f3)*dy(i)
01046       ! done
01047       fgrid1(i)=f5+(f6-f5)*dz(i)
01048
01049     enddo
01050
01051   end subroutine interpo_4_1
01052
01053
01055   !-----------------------------------------
01056   !------- interpo_4_4 ----------------------
01057   !-----------------------------------------
01058   subroutine interpo_4_4(iw,ix,iy,iz,dw,dx,dy,dz,fgrid1,grid)
01059     implicit none
01060     integer(ikind), intent(in) :: iw(:)
01061     integer(ikind), intent(in) :: ix(:)
01062     integer(ikind), intent(in) :: iy(:)
01063     integer(ikind), intent(in) :: iz(:)
01064     real(rkind), intent(out):: fgrid1(:,:,:,:)
01065     real(rkind), intent(in) :: dw(:)
01066     real(rkind), intent(in) :: dx(:)
01067     real(rkind), intent(in) :: dy(:)
01068     real(rkind), intent(in) :: dz(:)
01069     type(grid_t),intent(in) :: grid
01070
01071     real(rkind)::f1, f2, f3, f4, f5, f6
01072     integer :: l,i,j,k
01073     integer :: ll1,ll2,ii1,ii2,jj1,jj2,kk1,kk2
01074     !lin
01075     do k=1,size(iz(:))
01076        kk1=iz(k)
01077        if(kk1<lbound(grid%f4d,4)) kk1=lbound(grid%f4d,4)
01078        if(kk1>ubound(grid%f4d,4)) kk1=ubound(grid%f4d,4)-1
01079        kk2=kk1+1
01080        do j=1,size(iy(:))
01081           jj1=iy(j)
01082           if(jj1<lbound(grid%f4d,3)) jj1=lbound(grid%f4d,3)
01083           if(jj1>ubound(grid%f4d,3)) jj1=ubound(grid%f4d,3)-1
01084           jj2=jj1+1
01085           do i=1,size(ix(:))
01086              ii1=ix(i)
01087              if(ii1<lbound(grid%f4d,2)) ii1=lbound(grid%f4d,2)
01088              if(ii1>ubound(grid%f4d,2)) ii1=ubound(grid%f4d,2)-1
01089              ii2=ii1+1
01090              do l=1,size(iw(:))
01091                 ll1=iw(l)
01092                 if(ll1<lbound(grid%f4d,1)) ll1=lbound(grid%f4d,1)
01093                 if(ll1>ubound(grid%f4d,1)) ll1=ubound(grid%f4d,1)-1
01094                 ll2=ll1+1
01095   ! building interpolation for first cube
01096                 f1=grid%f4d(ll1,ii1,jj1,kk1) &
01097                      +(grid%f4d(ll2,ii1,jj1,kk1) &
01098                      -grid%f4d(ll1,ii1,jj1,kk1))*dw(l)
01099                 f2=grid%f4d(ll1,ii2,jj1,kk1) &
01100                      +(grid%f4d(ll2,ii2,jj1,kk1) &
01101                      -grid%f4d(ll1,ii2,jj1,kk1))*dw(l)
01102                 f3=f1+(f2-f1)*dx(i)
01103                 f1=grid%f4d(ll1,ii1,jj2,kk1) &
01104                      +(grid%f4d(ll2,ii1,jj2,kk1) &
01105                      -grid%f4d(ll1,ii1,jj2,kk1))*dw(l)
01106                 f2=grid%f4d(ll1,ii2,jj2,kk1) &
01107                      +(grid%f4d(ll2,ii2,jj2,kk1) &
01108                      -grid%f4d(ll1,ii2,jj2,kk1))*dw(l)
01109                 f4=f1+(f2-f1)*dx(i)
01110
01111                 f5=f3+(f4-f3)*dy(i)
01112   ! done
01113   ! build second cube
01114                 f1=grid%f4d(ll1,ii1,jj1,kk2) &
01115                      +(grid%f4d(ll2,ii1,jj1,kk2) &
01116                      -grid%f4d(ll1,ii1,jj1,kk2))*dw(l)
01117                 f2=grid%f4d(ll1,ii2,jj1,kk2) &
01118                      +(grid%f4d(ll2,ii2,jj1,kk2) &
01119                      -grid%f4d(ll1,ii2,jj1,kk2))*dw(l)
```

```
01120                 f3=f1+(f2-f1)*dx(i)
01121                 f1=grid%f4d(ll1,ii1,jj2,kk2) &
01122                      +(grid%f4d(ll2,ii1,jj2,kk2) &
01123                      -grid%f4d(ll1,ii1,jj2,kk2))*dw(l)
01124                 f2=grid%f4d(ll1,ii2,jj2,kk2) &
01125                      +(grid%f4d(ll2,ii2,jj2,kk2) &
01126                      -grid%f4d(ll1,ii2,jj2,kk2))*dw(l)
01127                 f4=f1+(f2-f1)*dx(i)
01128
01129                 f6=f3+(f4-f3)*dy(i)
01130   ! done
01131                 fgrid1(l,i,j,k)=f5+(f6-f5)*dz(i)
01132
01133               enddo
01134             enddo
01135           enddo
01136         enddo
01137
01138   end subroutine interpo_4_4
01139
01141   !-----------------------------------------------
01142   !-------- hunt -----------------------------------
01143   !-----------------------------------------------
01144
01145   subroutine hunt(axe,x,ix,dx)
01146     implicit none
01147     ! be careful, the axe is not necessarily starting with low bound=1
01148     ! to use the hunt search from num.rec. to get index of x the vector
01149     ! of axe must be adjusted to start with idx 1. This is done by calling
01150     ! hunt with the internal x vec. of axe. The calling mechanisme will
01151     ! addmit then that the pass vector starts with one.
01152     !
01153     ! The return indexes are then reset to the correct index after calling hunt_nrv
01154     !
01155     ! Additionaly dx is computed for interpolation
01156
01157     real(rkind)   ,  intent(inout) :: axe(:)
01158     real(rkind),     intent(in)    :: x(:)
01159     real(rkind),     intent(out)   :: dx(:)
01160     integer(ikind), intent(out)   :: ix(:)
01161     integer(ikind), save :: last_access=1
01162
01163     integer :: lbnd, ubnd
01164     integer :: i
01165
01166     lbnd=lbound(axe(:),1)
01167     ubnd=ubound(axe(:),1)
01168
01169
01170     ! call hunt_nrv with internal vector
01171     call hunt_nrv(axe(:), x(:), ix(:), last_access)
01172     ! addjust to bound of axe
01173     ix(:)=ix(:)+lbnd-1
01174
01175     do i=1,size(x)
01176        if(ix(i)<ubnd)then
01177           dx(i)=(x(i)-axe(ix(i)))/(axe(ix(i)+1)-axe(ix(i)))
01178        else
01179           dx(i)=0
01180        endif
01181     enddo
01182   end subroutine hunt
01183
01185   !-----------------------------------------------
01186   !-------- hunt_nrv -------------------------------
01187   !-----------------------------------------------
01188   ! it is guaranteed by calling that xx and x are indexed from 1 to size(..)
01189   !
01190   ! just like hunt in num recip
01191   ! but x is a vextor and a colecting index vector is pass
01192   !
01193   subroutine hunt_nrv(xx,x,ix,jlo)
01194     implicit none
01195     !..given an array xx of length n and a value x, this routine returns a value
01196     !..jlo such that x is between xx(jlo) and xx(jlo+1). the array xx must be
01197     !..monotonic. j=0 or j=n indicates that x is out of range. on input jlo is a
01198     !..guess for the table entry, and should be used as input for the next hunt.
01199     !..one assumes the next table entry is relatively close to the old one.
01200
01201     real(rkind),     intent(in)    :: xx(:)
01202     real(rkind),     intent(in)    :: x(:)
01203     integer(ikind), intent(out)   :: ix(:)
01204     integer(ikind), intent(inout) :: jlo
01205
01206     integer :: n,inc,jhi,jm
01207     logical :: ascnd
01208
```

```
01209     integer :: i
01210
01211     n=size(xx)
01212
01213     !..logical function; true if ascending table, false if descending
01214     ascnd = (xx(n) >= xx(1))
01215
01216     do i=1,size(x)
01217        !..initial guess is not useful; go to bisection immediatly
01218        if (jlo <= 0 .or. jlo > n) then
01219           jlo=0
01220           jhi=n+1
01221        else
01222           inc=1                        ! set the initial hunting increment
01223
01224    !..this is the hunt up section
01225           if (x(i) >= xx(jlo) .eqv. ascnd) then
01226              do
01227                 jhi=jlo+inc
01228                 if (jhi > n) then             ! done hunting since end of table
01229                    jhi=n+1
01230                    exit
01231                 else
01232                    if (x(i) < xx(jhi) .eqv. ascnd) exit
01233                    jlo=jhi                      !not done hunting
01234                    inc=inc+inc                  !so double increment
01235                 end if
01236              end do                           ! and try again
01237           else                               ! hunt done
01238              jhi=jlo
01239              do
01240                 jlo=jhi-inc
01241                 if (jlo < 1) then             ! hunt done, since end of table
01242                    jlo=0
01243                    exit
01244                 else
01245                    if (x(i) >= xx(jlo) .eqv. ascnd) exit
01246                    jhi=jlo                      !not done hunting
01247                    inc=inc+inc                  !so double increment
01248                 end if
01249              end do                           ! and try again
01250           end if
01251        end if                                ! done hunting, value bracketed
01252        do                                    ! Hunting is done, begin final bisection phase
01253           if (jhi-jlo <= 1) then
01254              if (x(i) == xx(n)) jlo=n-1
01255              if (x(i) == xx(1)) jlo=1
01256              exit
01257           else
01258              jm=(jhi+jlo)/2
01259              if (x(i) >= xx(jm) .eqv. ascnd) then
01260                 jlo=jm
01261              else
01262                 jhi=jm
01263              end if
01264           end if
01265        end do
01266        ix(i)=jlo
01267     enddo!(i)
01268   end subroutine hunt_nrv
01269
01271   !-------------------------------------------------------------
01272   !------------ locate ---------------------------------
01273   !-------------------------------------------------------------
01274   !
01275   ! same as hunt search, using midpoint method
01276   !
01277
01278   subroutine locate(axe, x, ix, dx)
01279     implicit none
01280     !..given an array xx of length n, and a value of x, this routine returns
01281     !..a value j such that x is between xx(j) and xx(j+1). the array xx must be
01282     !..monotonic. j=0 or j=n indicates that x is out of range. bisection is used
01283     !..to find the entry
01284
01285     real(rkind),    intent(inout) :: axe(:)
01286     real(rkind),    intent(in)    :: x
01287     real(rkind),    intent(out)   :: dx
01288     integer(ikind), intent(out)   :: ix
01289
01290     integer  :: lbnd, ubnd
01291     integer  :: i
01292
01293     lbnd=lbound(axe(:),1)
01294     ubnd=ubound(axe(:),1)
01295
01296     ! get index
```

```
01297     ix=locate_nrv(axe(:),x)
01298     ! addjust to bound of axe
01299     ix=ix+lbnd-1
01300
01301     if(ix<ubnd)then
01302        dx=(x-axe(ix))/(axe(ix+1)-axe(ix))
01303     else
01304        dx=0
01305     endif
01306
01307     ! last search for hunt search
01308     ! axe%last_access=ix(size(x))+lbnd-1
01309
01310   end subroutine locate
01311
01312
01314   !-------------------------------------------------------
01315   !------------ locate_nrv ---------------------------------
01316   !-------------------------------------------------------
01317
01318   function locate_nrv(xx,x) result (result)
01319     !..given an array xx of length n, and a value of x, this routine returns
01320     !..a value j such that x is between xx(j) and xx(j+1). the array xx must be
01321     !..monotonic. j=0 or j=n indicates that x is out of range. bisection is used
01322     !..to find the entry
01323
01324     real(rkind),  intent(in) :: xx(:)
01325     real(rkind),  intent(in) :: x
01326
01327     integer      :: result
01328     integer      :: n,jl,jm,ju
01329     logical      :: ascnd
01330
01331     integer  :: i
01332
01333     n=size(xx)
01334     ascnd = (xx(n) >= xx(1))
01335     jl=0
01336     ju=n+1
01337     do
01338        if (ju-jl <= 1) exit
01339    !..compute a midpoint, and replace either the upper or lower limit
01340        jm=(ju+jl)/2
01341        if (ascnd .eqv. (x >= xx(jm))) then
01342           jl=jm
01343        else
01344           ju=jm
01345        end if
01346     end do
01347     if (x == xx(1)) then
01348        result=1
01349     else if (x == xx(n)) then
01350        result=n-1
01351     else
01352        result=jl
01353     end if
01354
01355   end function locate_nrv
01356
01357 end module data_suport
```

16.109 src/libamns/eckstein_yields.f90 File Reference

Modules

- module eckstein_yields

Functions/Subroutines

- real(ids_real) function eckstein_yields::etf (M1, M2, Z1, Z2)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::omegal (epsl)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::sn (epsl)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::seyield (E0, M1, M2, Z1, Z2, q, lambda, u, ETh)

AMNS External utility function ...

- real(ids_real) function eckstein_yields::sayield (E0, theta, f, b, c, Esp)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::reyieldlight (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::reyieldself (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::reyield (E0, M1, M2, Z1, Z2, A1, A2, A3, A4)

  AMNS External utility function ...
- real(ids_real) function eckstein_yields::rayield (angledeg, C1, C2, C3, C4)

  AMNS External utility function ...

Variables

- real(ids_real), parameter eckstein_yields::mathpi = 3.14159265358979323846264338327 95
- real(ids_real), parameter eckstein_yields::degtorad = 0.01745329251994329576923690768489

16.110 eckstein_yields.f90

```
00001
00007 module eckstein_yields
00008
00009   use ids_types  ! IGNORE
00010   implicit none
00011
00012   real(ids_real), parameter :: mathpi = 3.14159265358979323846264338327 95
00013   real(ids_real), parameter :: degtorad = 0.01745329251994329576923690768489
00014
00015 contains
00016
00022   function etf(M1, M2, Z1, Z2)
00023     implicit none
00024     real(ids_real) :: etf, m1, m2, z1, z2
00025     etf = (30.74*(m1 + m2)*z1*sqrt(z1**0.6666666666666666 + z2**0.6666666666666666)*z2)/m2
00026   end function etf
00027
00033   function omegal(epsl)
00034     implicit none
00035     real(ids_real) :: omegal, epsl
00036     omegal = 0.008*(epsl**0.1504) + 0.1728*sqrt(epsl) + epsl
00037   end function omegal
00038
00044   function sn(epsl)
00045     implicit none
00046     real(ids_real) :: sn, epsl
00047     sn = (0.5*log(1 + 1.2288*epsl))/omegal(epsl)
00048   end function sn
00049
00055   function seyield(E0, M1, M2, Z1, Z2, q, lambda, u, ETh)
00056     implicit none
00057     real(ids_real) :: seyield, e0, m1, m2, z1, z2, q, lambda, u, eth, epsl, snucl, omega, etoeth
00058     epsl = e0/etf(m1, m2, z1, z2)
00059     snucl = sn(epsl)
00060     omega = omegal(epsl)
00061     etoeth =  ((e0/eth) - 1.0)**u
00062     seyield = (etoeth*q*snucl)/(etoeth + (lambda/omega))
00063   end function seyield
00064
00070   function sayield(E0, theta, f, b, c, Esp)
00071     implicit none
00072     real(ids_real) :: sayield, e0, theta, thetarad, f, b, c, esp
00073     real(ids_real) :: thetastar, cosfact
00074     thetarad = theta * degtorad
00075     thetastar = mathpi - acos(sqrt(1/(1 + e0/esp)))
00076     cosfact = cos(((mathpi/2.)**c)*((thetarad/thetastar)**c))
00077     sayield = exp(b*(1 - 1/cosfact))/(cosfact**f)
00078   end function sayield
00079
00085   function reyieldlight(E0, M1, M2, Z1, Z2, A1, A2, A3, A4)
00086     implicit none
00087     real(ids_real) :: reyieldlight, e0, m1, m2, z1, z2, a1, a2, a3, a4, eps
00088
00089     eps = e0/etf(m1, m2, z1, z2)
00090
00091     reyieldlight = (a1 * eps**a2) / (1.0 + a3 * eps**a4)
```

```
00092
00093   end function reyieldlight
00094
00100   function reyieldself(E0, M1, M2, Z1, Z2, A1, A2, A3, A4)
00101     implicit none
00102     real(ids_real) :: reyieldself, e0, m1, m2, z1, z2, a1, a2, a3, a4, eps
00103
00104     eps = e0/etf(m1, m2, z1, z2)
00105
00106     if(a3 * eps**a4 .lt. 700) then  ! prevent overflow DPC
00107        reyieldself = exp(a1*eps**a2)/(1.0 + exp(a3 * eps**a4))
00108     else
00109        reyieldself = exp(a1*eps**a2 - a3 * eps**a4)
00110     endif
00111
00112   end function reyieldself
00113
00119   function reyield(E0, M1, M2, Z1, Z2, A1, A2, A3, A4)
00120     implicit none
00121     real(ids_real) :: reyield, e0, m1, m2, z1, z2, a1, a2, a3, a4, eps
00122     if (m1 .ge. m2) then
00123 !       write(*,*) 'M1 = ', M1, ' is >= M2 = ',M2,' Using  ryieldself'
00124        reyield = reyieldself(e0, m1, m2, z1, z2, a1, a2, a3, a4)
00125     else
00126 !       write(*,*) 'M1 = ', M1, ' is <  M2 = ',M2,' Using  ryieldlight'
00127        reyield = reyieldlight(e0, m1, m2, z1, z2, a1, a2, a3, a4)
00128     end if
00129   end function reyield
00130
00136   function rayield(angledeg, C1, C2, C3, C4)
00137     implicit none
00138     real(ids_real) :: rayield, angledeg, c1, c2, c3, c4
00139
00140     rayield = c1 + c2 * tanh(c3 * angledeg * degtorad + c4)
00141
00142   end function rayield
00143
00144 end module eckstein_yields
```

16.111 src/libamns/f90_kind.f90 File Reference

Modules

- module f90_kind

  f90_kind module from Silvio Gori's grid package

Variables

- integer, parameter f90_kind::rkind = kind(1.0d0)
- integer, parameter f90_kind::ikind = kind(1)
- integer, parameter f90_kind::skind = 256

16.112 f90_kind.f90

```
00001
00006
00007 module f90_kind
00008
00009   implicit none
00010
00011   public
00012
00013   integer, parameter :: rkind  = kind(1.0d0)
00014   integer, parameter :: ikind  = kind(1)
00015   integer, parameter :: skind  = 256
00016
00017 end module f90_kind
00018
```

16.113 src/libamns/git_version_AMNS.h File Reference

16.114 git_version_AMNS.h

```
00001       character*132, parameter :: git_version_AMNS = '1.5.0'
```

16.115 src/libamns/interface_to_amns.f90 File Reference

Modules

- module interface_to_amns

Functions/Subroutines

- subroutine interface_to_amns::get_amns_data (reaction_type, reactants, grid, properties_comment, properties_source, properties_provider, properties_creation_date, code_name, code_commit, code_version, code_repository, source, provider, citation, shot, run, backend, user, ds_version, ierr, error_description, debug)

  transfer data from the IDS to the internal data structure
- character(len=answer_length) function interface_to_amns::assign_if_associated (ids_str, def_str)

  utility to assign if associated
- subroutine interface_to_amns::end_amns_data

  deallocate idss

Variables

- type(amns_ids_list), pointer interface_to_amns::first => null()

16.116 interface_to_amns.f90

```
00001
00008 module interface_to_amns
00009
00010   use amns_types
00011   use ids_routines  ! IGNORE
00012   type (amns_ids_list), pointer :: first => null()
00013
00014 contains
00015
00016 ! zn:                   nuclear charge
00017 ! za:                   ionization state
00018 ! code                  reaction type
00019 ! grid                  output data (has a structure of the type "grid")
00020 ! inpdata               input data: amns ids
00021 ! nproc:                number of resulting states
00022 ! nz:                   number of initial states (if bundle: number of initial states)
00023 ! tdim(2,nproc):        dimesionality (1,nproc) and start positions (2,nproc) of tables (e.g. 1D: only
       T dependence, 2D: T and n dependence, ...)
00024
00030   subroutine get_amns_data(reaction_type, reactants, grid, &
00031        properties_comment, properties_source, &
00032        properties_provider, properties_creation_date, &
00033        code_name, code_commit, &
00034        code_version, code_repository, &
00035        source, provider, citation, &
00036        shot, run, backend, user, ds_version, &
00037        ierr, error_description, debug)
00038     use ids_types  ! IGNORE
00039     use amns_utility
00040     use f90_kind
00041 ! do not remove the comment!!!
00042     use ids_schemas   ! IGNORE
00043     use ids_routines  ! IGNORE
00044 !    use read_structures ! IGNORE
00045     implicit none
00046
00047     type (grid_t), intent(out) :: grid
00048     character*(answer_length), intent(out) :: &
00049          properties_comment, properties_source, &
00050          properties_provider, properties_creation_date, &
00051          code_name, code_commit, &
00052          code_version, code_repository, &
00053          source, provider, citation
```

```
00054     type(amns_reaction_type), intent(in) :: reaction_type
00055     type(amns_reactants_type), intent(in) :: reactants
00056     real (kind=ids_real) :: zn, za
00057     character (len=*) :: backend, user, ds_version
00058     integer :: ierr
00059     character*128 :: error_description
00060     logical :: debug
00061     integer :: iza, nz, nproc, i, k, j, isearch, ndim, ncoord
00062     type (amns_ids_list), pointer :: last, work => null()
00063 ! for calling AMNS IDS
00064     character(len=3)::      treename
00065     integer ::              idx, shot, run
00066     logical :: found, done, match
00067     character (len=256) :: file
00068     integer :: nreac, nprod, nlhs, nrhs, ireac
00069     real (kind=ids_real) :: zn_ids, za_ids
00070     integer :: ids_status
00071     character(STRMAXLEN):: uri
00072
00073     ierr=0
00074     error_description=''
00075     iza = -1
00076
00077     work => first
00078     last => first
00079     found = .false.
00080 !!    write(*,*) associated(first), associated(last), associated(work)
00081     do while(associated(work))
00082 !!      write(*,*) work%shot, shot
00083        if(work%shot.eq.shot.and.work%run.eq.run) then
00084           found=.true.
00085           exit
00086        endif
00087        last => work
00088        work => work%next
00089 !!      write(*,*) associated(first), associated(last), associated(work)
00090     enddo
00091 !!    write(*,*) associated(first), associated(last), associated(work)
00092     if(.not.found) then
00093        if(.not.associated(first)) then
00094           allocate(first)
00095 !!       write(*,*) first%shot, first%run, associated(first%prev), associated(first%next)
00096           work => first
00097 !!       write(*,*) associated(first), associated(work), work%shot
00098        else if(.not.associated(last%next)) then
00099           allocate(last%next)
00100           work => last%next
00101           work%prev => last
00102        endif
00103        if(work%shot.eq.0) then
00104           treename = 'ids'
00105           if(backend.eq.'ascii') then
00106 !            write(file,'(a,"_",i6.6,"_",i6.6,".IDS")') 'amns',shot,run
00107 !            if(debug) write(*,*) 'Reading data from ', trim(file), ' for ', shot, run
00108 !            call open_read_file(1, trim(file))
00109 !            call read_ids(work%amns_ids, 'amns')
00110 !            call close_read_file
00111           else
00112 #ifdef AL
00113              if(backend.eq.'mdsplus') then
00114                 if(debug) write(*,*) 'Using mdsplus backend'
00115                 !                  call
     imas_open_env(treename,shot,run,idx,trim(USER),'amns',ds_version)
00116                 !call al_begin_pulse_action(MDSPLUS_BACKEND, shot, run, &
00117                 !     trim(USER), 'amns', trim(ds_version), idx)
00118                 !call al_open_pulse(idx, OPEN_PULSE, '', ids_status)
00119                 call al_build_uri_from_legacy_parameters(mdsplus_backend, shot, run, &
00120                      trim(user), 'amns', trim(ds_version), '', uri, ids_status)
00121                 call al_begin_dataentry_action(uri, open_pulse, idx, ids_status)
00122
00123              elseif(backend.eq.'hdf5') then
00124                 if(debug) write(*,*) 'Using hdf5 backend'
00125 !                call imas_open_hdf5(treename,shot,run,idx)
00126                 stop 'HFD5 nor currently supported'
00127              else
00128                 error_description = 'Backend not yet supported: ' // trim(backend)
00129                 write(*,*) error_description
00130                 ierr=1
00131                 return
00132              endif
00133              call ids_get(idx,'amns_data',work%amns_ids)
00134              call imas_close(idx)
00135 #else
00136              error_description = 'The AMNS routines were compiled without AL support. ' // &
00137                   'Use the ascii backend'
00138              write(*,*) error_description
00139              ierr=2
```

```
00140             return
00141 #endif
00142          endif
00143          work%shot=shot
00144          work%run=run
00145       endif
00146    endif
00147
00148    nproc= SIZE(work%amns_ids%process)
00149
00150    k=0
00151    do isearch = 1, nproc
00152       if (debug) write(*,'(a,i4)') 'Considering IDS process ', isearch
00153 ! first look for the right process
00154       if(reaction_type%string .ne. work%amns_ids%process(isearch)%label(1)) cycle
00155 ! found the right process, now search for the right reaction pattern
00156 ! first calculate the number of reactants and products in the database
00157       if(associated(work%amns_ids%process(isearch)%reactants)) then
00158          nreac = size(work%amns_ids%process(isearch)%reactants)
00159       else
00160          nreac = 0
00161       endif
00162       if(associated(work%amns_ids%process(isearch)%products)) then
00163          nprod = size(work%amns_ids%process(isearch)%products)
00164       else
00165          nprod = 0
00166       endif
00167 ! then calculate the number of reactants and products for the requested reaction
00168       nlhs=0
00169       nrhs=0
00170       if(allocated(reactants%components)) then
00171          do ireac = 1, size(reactants%components)
00172             if(reactants%components(ireac)%lr .eq. 0) then
00173                if (nrhs .ne. 0) then
00174                   error_description = 'Reactants should precede products'
00175                   write (*,*) error_description
00176                   ierr=3
00177                   return
00178                endif
00179                nlhs = nlhs + 1
00180             else if(reactants%components(ireac)%lr .eq. 1) then
00181                nrhs = nrhs + 1
00182             else
00183                write(error_description,*) 'Unknown lr option ', reactants%components(ireac)%lr, &
00184                     ' for component ', ireac
00185                write(*,*) error_description
00186                ierr=4
00187                return
00188             endif
00189          enddo
00190       endif
00191 ! the numbers should match
00192       if (debug) write(*,'(a,2i4,a,2i4)') &
00193            'nreac, nprod  IDS: ', nreac, nprod, &
00194            '  REQUEST: ', nlhs, nrhs
00195       if(nreac .ne. nlhs) cycle
00196       if(nprod .ne. nrhs) cycle
00197       match = .true.
00198 ! check the reactants
00199       do ireac = 1, nreac
00200          if (debug) write(*,'(a,i4)') 'Considering reactant ', ireac
00201 ! now check that the zn's match
00202          zn_ids = 0
00203          if(associated(work%amns_ids%process(isearch)%reactants(ireac)%element)) then
00204             if(size(work%amns_ids%process(isearch)%reactants(ireac)%element) .eq. 1) then
00205 !               if(work%amns_ids%process(isearch)%reactants(ireac)%element(1)%multiplicity .eq. 1)
     then
00206                   zn_ids = work%amns_ids%process(isearch)%reactants(ireac)%element(1)%z_n
00207 !                else
00208 !                   write(*,*) 'Case with multiplicity != 1 not yet coded'
00209 !                   write(*,*) ireac,
     work%amns_ids%process(isearch)%reactants(ireac)%element(1)%multiplicity
00210 !                   stop  'ERROR' ! for the moment
00211 !                endif
00212             else
00213                error_description = 'Case with number of elements != 1 not yet coded'
00214                write(*,*) error_description
00215                ierr=5
00216                return
00217             endif
00218          else
00219             error_description = 'Must have at least one constituent'
00220             write(*,*) error_description
00221             ierr=6
00222             return
00223          endif
00224          if (debug) write(*,'(a,f7.3,a,f7.3)') &
```

```
00225                ’zn  IDS: ’, zn_ids, &
00226                ’  REQUEST: ’, reactants%components(ireac)%zn
00227           match = match .and. (nint(reactants%components(ireac)%zn) .eq. nint(zn_ids))
00228           if (.not. match) cycle
00229 ! now check that the za’s match
00230           za_ids = 0
00231           if(work%amns_ids%process(isearch)%reactants(ireac)%relative_charge .eq. -1) then
00232 ! charge is not relevant
00233              if (debug) write(*,’(a)’) &
00234                   ’za not relevant’
00235           else if(work%amns_ids%process(isearch)%reactants(ireac)%relative_charge .eq. 0) then
00236 ! charge is absolute
00237              if (debug) write(*,’(a,f7.3,a,f7.3)’) &
00238                   ’za (absolute charge)  IDS: ’,
       work%amns_ids%process(isearch)%reactants(ireac)%charge, &
00239                   ’  REQUEST: ’, reactants%components(ireac)%za
00240              match = match .and. (work%amns_ids%process(isearch)%reactants(ireac)%charge .eq.
       reactants%components(ireac)%za)
00241           else if(work%amns_ids%process(isearch)%reactants(ireac)%relative_charge .eq. 1 ) then
00242 ! charge is relative
00243 ! ???
00244              iza = reactants%components(ireac)%za + 1
00245              if (debug) write(*,’(a,f7.3,a,f7.3)’) &
00246                   ’za (relative charge)  IDS: ’,
       work%amns_ids%process(isearch)%reactants(ireac)%charge, &
00247                   ’  REQUEST: ’, reactants%components(ireac)%za
00248           else
00249              write(error_description,*) ’Unrecognized value for relative charge ’,
       work%amns_ids%process(isearch)%reactants(ireac)%relative_charge
00250              write(*,*) error_description
00251              ierr=7
00252              return
00253           endif
00254           if (.not. match) cycle
00255 ! now check that the am’s match
00256           if(reaction_type%isotope_resolved .eq. 1) then
00257              if(debug) write(*,’(a,f7.3,a,f7.3)’) &
00258                   ’am  IDS: ’, work%amns_ids%process(isearch)%reactants(ireac)%mass, &
00259                   ’  REQUEST: ’, reactants%components(ireac)%mi
00260              if(nint(work%amns_ids%process(isearch)%reactants(ireac)%mass) .ne. 0 .and.
       nint(reactants%components(ireac)%mi) .ne. 0) then
00261                 match = match .and. (nint(work%amns_ids%process(isearch)%reactants(ireac)%mass) .eq.
       nint(reactants%components(ireac)%mi))
00262              endif
00263           endif
00264           if (.not. match) cycle
00265        enddo
00266        if (.not. match) cycle
00267 ! check the products
00268        do ireac = 1, nprod
00269           if (debug) write(*,’(a,i4)’) ’Considering product ’, ireac
00270 ! now check that the zn’s match
00271           zn_ids = 0
00272           if(associated(work%amns_ids%process(isearch)%products(ireac)%element)) then
00273              if(size(work%amns_ids%process(isearch)%products(ireac)%element) .eq. 1) then
00274 !                if(work%amns_ids%process(isearch)%products(ireac)%element(1)%multiplicity .eq. 1)
       then
00275                   zn_ids = work%amns_ids%process(isearch)%products(ireac)%element(1)%z_n
00276 !                else
00277 !                   write(*,*) ’Case with multiplicity != 1 not yet coded’
00278 !                   stop  ’ERROR’ ! for the moment
00279 !                endif
00280              else
00281                 error_description = ’Case with number of elements != 1 not yet coded’
00282                 write(*,*) error_description
00283                 ierr=8
00284                 return
00285              endif
00286           else
00287              error_description = ’Must have at least one constituent’
00288              write(*,*) error_description
00289              ierr=9
00290              return
00291           endif
00292           if (debug) write(*,’(a,f7.3,a,f7.3)’) &
00293                ’zn  IDS: ’, zn_ids, &
00294                ’  REQUEST: ’, reactants%components(ireac+nlhs)%zn
00295           match = match .and. (nint(reactants%components(ireac+nlhs)%zn) .eq. nint(zn_ids))
00296 ! now check that the za’s match
00297           za_ids = 0
00298           if(work%amns_ids%process(isearch)%products(ireac)%relative_charge .eq. -1) then
00299 ! charge is not relevant
00300              if (debug) write(*,’(a)’) &
00301                   ’za not relevant’
00302           else if(work%amns_ids%process(isearch)%products(ireac)%relative_charge .eq. 0) then
00303 ! charge is absolute
00304              if (debug) write(*,’(a,f7.3,a,f7.3)’) &
```

```
00305                   'za (absolute charge)  IDS: ',
      work%amns_ids%process(isearch)%products(ireac)%charge, &
00306                   '  REQUEST: ', reactants%components(ireac+nlhs)%za
00307              match = match .and. (work%amns_ids%process(isearch)%products(ireac)%charge .eq.  &
00308                   reactants%components(ireac+nlhs)%za)
00309           else if(work%amns_ids%process(isearch)%products(ireac)%relative_charge .eq. 1 ) then
00310 ! charge is relative
00311 ! ???
00312              if (debug) write(*,'(a,f7.3,a,f7.3)') &
00313                   'za (relative charge)  IDS: ',
      work%amns_ids%process(isearch)%products(ireac)%charge, &
00314                   '  REQUEST: ', reactants%components(ireac+nlhs)%za
00315           else
00316              write(error_description,*) 'Unrecognized value for relative charge ',
      work%amns_ids%process(isearch)%reactants(ireac)%relative_charge
00317              write(*,*) error_description
00318              ierr=10
00319              return
00320           endif
00321           if (.not. match) cycle
00322 ! now check that the am's match
00323           if(reaction_type%isotope_resolved .eq. 1) then
00324              if(debug) write(*,'(a,f7.3,a,f7.3)') &
00325                   'am  IDS: ', work%amns_ids%process(isearch)%products(ireac)%mass, &
00326                   '  REQUEST: ', reactants%components(ireac+nlhs)%mi
00327              if(nint(work%amns_ids%process(isearch)%products(ireac)%mass) .ne. 0 .and.
      nint(reactants%components(ireac+nlhs)%mi) .ne. 0) then
00328                 match = match .and. (nint(work%amns_ids%process(isearch)%products(ireac)%mass) .eq.
      nint(reactants%components(ireac+nlhs)%mi))
00329              endif
00330           endif
00331           if (.not. match) cycle
00332        enddo
00333        if (.not. match) cycle
00334        match = .true.
00335        k = isearch
00336        exit
00337     enddo
00338     if(k .eq. 0) then
00339        write(error_description,*) 'GET_AMNS_DATA: Case ',trim(reaction_type%string),' not yet
      implemented'
00340        if(debug) then
00341           write(*,'(a)') trim(error_description)
00342           do isearch = 1, nproc
00343              write(*,'(a,i4,a,a)') 'Process ', isearch, ' = ' ,
      trim(work%amns_ids%process(isearch)%label(1))
00344           enddo
00345        endif
00346        ierr=11
00347        return
00348     endif
00349
00350     ndim = work%amns_ids%process(k)%table_dimension         ! dimensionality of the requested matrix
00351     ncoord=work%amns_ids%process(k)%coordinate_index
00352     if(iza .eq. -1) iza=1
00353     if(size(work%amns_ids%process(k)%charge_state) .eq. 1) iza=1
00354     if(iza .gt. size(work%amns_ids%process(k)%charge_state)) then
00355        error_description = 'iza out of bounds'
00356        write(*,*) error_description
00357        ierr=12
00358        return
00359     endif
00360
00361 ! some of the 1d functions require 2d tables
00362     if(ndim.eq.1) then
00363        if(.not.associated(work%amns_ids%process(k)%charge_state(iza)%table_1d)) then
00364           if(debug) write(*,*) 'Special treatment for 1d functions requiring a 2d table'
00365           if(associated(work%amns_ids%process(k)%charge_state(iza)%table_2d)) then
00366              ndim=2
00367           endif
00368        endif
00369     endif
00370
00371     done=.true.
00372     select case (ndim)
00373     case(1)
00374        if(ncoord.gt.0) then
00375           call new_grid(grid, &
00376                work%amns_ids%process(k)%charge_state(iza)%table_1d(:), &
00377                work%amns_ids%coordinate_system(ncoord)%coordinate(1)%values(:))
00378        else
00379           call new_grid(grid, &
00380                work%amns_ids%process(k)%charge_state(iza)%table_1d(:))
00381        endif
00382     case(2)
00383        if(ncoord.gt.0) then
00384           call new_grid(grid, &
```

```
00385              work%amns_ids%process(k)%charge_state(iza)%table_2d(:,:), &
00386              work%amns_ids%coordinate_system(ncoord)%coordinate(1)%values(:), &
00387              work%amns_ids%coordinate_system(ncoord)%coordinate(2)%values(:))
00388         else
00389            call new_grid(grid, &
00390              work%amns_ids%process(k)%charge_state(iza)%table_2d(:,:))
00391         endif
00392      case(3)
00393         if(ncoord.gt.0) then
00394            call new_grid(grid, &
00395              work%amns_ids%process(k)%charge_state(iza)%table_3d(:,:,:), &
00396              work%amns_ids%coordinate_system(ncoord)%coordinate(1)%values(:), &
00397              work%amns_ids%coordinate_system(ncoord)%coordinate(2)%values(:), &
00398              work%amns_ids%coordinate_system(ncoord)%coordinate(3)%values(:))
00399         else
00400            call new_grid(grid, &
00401              work%amns_ids%process(k)%charge_state(iza)%table_3d(:,:,:))
00402         endif
00403      case(4)
00404         if(ncoord.gt.0) then
00405            call new_grid(grid, &
00406              work%amns_ids%process(k)%charge_state(iza)%table_4d(:,:,:,:), &
00407              work%amns_ids%coordinate_system(ncoord)%coordinate(1)%values(:), &
00408              work%amns_ids%coordinate_system(ncoord)%coordinate(2)%values(:), &
00409              work%amns_ids%coordinate_system(ncoord)%coordinate(3)%values(:), &
00410              work%amns_ids%coordinate_system(ncoord)%coordinate(4)%values(:))
00411         else
00412            call new_grid(grid, &
00413              work%amns_ids%process(k)%charge_state(iza)%table_4d(:,:,:,:))
00414         endif
00415      case(5)
00416         done=.false.
00417         write(*,*) '0D and 5D can be implemented later'
00418      end select
00419
00420      if(done) then
00421         grid%ndim=work%amns_ids%process(k)%table_dimension
00422         if(work%amns_ids%process(k)%result_transformation == 0) then
00423            grid%is_lin= .true.
00424         elseif(work%amns_ids%process(k)%result_transformation == 1) then
00425            grid%is_log= .true.
00426         else
00427            grid%interpol_function = work%amns_ids%process(k)%result_transformation
00428         endif
00429
00430 ! work%amns_ids%process(k)%label is an array of elements divided into 132 bytes
00431 ! long strings. But state_label inside grid is just 132 character long string
00432 ! we will take first element from label to fit it
00433
00434         if(associated(work%amns_ids%process(k)%charge_state)) then
00435            if(size(work%amns_ids%process(k)%charge_state).ge.iza) then
00436               grid%state_label=assign_if_associated(work%amns_ids%process(k)%charge_state(iza)%label,
     '')
00437            endif
00438         else
00439            grid%state_label=''
00440         endif
00441         grid%result_label=assign_if_associated(work%amns_ids%process(k)%result_label, '')
00442         grid%result_unit=assign_if_associated(work%amns_ids%process(k)%result_units, '')
00443         if(ncoord.gt.0) then
00444            do j=1,ndim
00445               if(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%transformation == 0) then
00446                  grid%axe(j)%is_lin = .true.
00447               elseif(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%transformation == 1) then
00448                  grid%axe(j)%is_log = .true.
00449               else
00450                  write(*,*) 'The coordinate transformation is unknown, for process',
     trim(reaction_type%string)
00451               endif
00452               if(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%spacing == 0) then
00453                  grid%axe(j)%is_equi = .false.
00454               elseif(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%spacing == 1) then
00455                  grid%axe(j)%is_equi = .true.
00456               else
00457                  write(*,*) 'The coordinate spacing is unknown, for process',
     trim(reaction_type%string)
00458               endif
00459               grid%axe(j)%lbound= lbound(grid%axe(j)%x, 1)
00460               if(associated(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%extrapolation_type))
     then
00461                  if(size(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%extrapolation_type) .eq.
     2) then
00462                     grid%axe(j)%extrapolation_type(:) =
     work%amns_ids%coordinate_system(ncoord)%coordinate(j)%extrapolation_type(:)
00463                  else
00464                     write(*,*) 'ERROR: coordinate extrapolation_type not of length 2 in amns_data, 0
     assumed!'
```

```
00465                   grid%axe(j)%extrapolation_type = 0
00466                endif
00467             else
00468                write(*,*) 'ERROR: coordinate extrapolation_type not associated in amns_data, 0
      assumed!'
00469                grid%axe(j)%extrapolation_type = 0
00470             endif
00471             if(associated(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%label)) then
00472                grid%axe(j)%label = work%amns_ids%coordinate_system(ncoord)%coordinate(j)%label(1)
00473             endif
00474             if(associated(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%units)) then
00475                grid%axe(j)%units = work%amns_ids%coordinate_system(ncoord)%coordinate(j)%units(1)
00476             endif
00477             if(associated(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%value_labels)) then
00478
      allocate(grid%axe(j)%value_labels(size(work%amns_ids%coordinate_system(ncoord)%coordinate(j)%value_labels)))
00479                grid%axe(j)%value_labels(:) =
      work%amns_ids%coordinate_system(ncoord)%coordinate(j)%value_labels(:)
00480             endif
00481
00482          enddo
00483       endif
00484    endif
00485
00486    properties_comment = assign_if_associated(work%amns_ids%ids_properties%comment, "Not specified in
      the AMNS IDS")
00487    properties_source = assign_if_associated(work%amns_ids%ids_properties%source, "Not specified in
      the AMNS IDS")
00488    properties_provider = assign_if_associated(work%amns_ids%ids_properties%provider, "Not specified
      in the AMNS IDS")
00489    properties_creation_date = assign_if_associated(work%amns_ids%ids_properties%creation_date, "Not
      specified in the AMNS IDS")
00490    code_name = assign_if_associated(work%amns_ids%code%name, "Not specified in the AMNS IDS")
00491    code_commit = assign_if_associated(work%amns_ids%code%commit, "Not specified in the AMNS IDS")
00492    code_version = assign_if_associated(work%amns_ids%code%version, "Not specified in the AMNS IDS")
00493    code_repository = assign_if_associated(work%amns_ids%code%repository, "Not specified in the AMNS
      IDS")
00494
00495    source = assign_if_associated(work%amns_ids%process(k)%source, "Not specified in the AMNS IDS")
00496    provider = assign_if_associated(work%amns_ids%process(k)%provider, "Not specified in the AMNS
      IDS")
00497    citation = assign_if_associated(work%amns_ids%process(k)%citation, "Not specified in the AMNS
      IDS")
00498
00499  end subroutine get_amns_data
00500
00504  function assign_if_associated(ids_str, def_str)
00505    implicit none
00506    character(len=*), dimension(:), pointer, intent(in) :: ids_str
00507    character(len=*), intent(in) :: def_str
00508    character(len=answer_length) :: assign_if_associated
00509    if(associated(ids_str)) then
00510       if(size(ids_str).ge.1) then
00511          assign_if_associated = ids_str(1)
00512       else
00513          assign_if_associated = def_str
00514       endif
00515    else
00516       assign_if_associated= def_str
00517    endif
00518  end function assign_if_associated
00519
00523  subroutine end_amns_data
00524
00525 !    use deallocate_structures ! IGNORE
00526    implicit none
00527
00528    type (amns_ids_list), pointer :: work, next => null()
00529
00530    work => first
00531
00532    do while(associated(work))
00533       next => work%next
00534       call ids_deallocate(work%amns_ids)
00535       deallocate(work)
00536       work => next
00537    enddo
00538
00539    first => null()
00540
00541  end subroutine end_amns_data
00542
00543 end module interface_to_amns
```

16.117 src/libamns/m_mrgrnk.f90 File Reference

Data Types

- interface m_mrgrnk::mrgrnk

Modules

- module m_mrgrnk

16.118 m_mrgrnk.f90

```
00001
00005 Module m_mrgrnk
00006 ! http://www.fortran-2000.com/rank/mrgrnk.f90
00007 Integer, Parameter :: kdp = selected_real_kind(15)
00008 public :: mrgrnk
00009 private :: kdp
00010 private :: r_mrgrnk, i_mrgrnk, d_mrgrnk
00011 interface mrgrnk
00012   module procedure d_mrgrnk, r_mrgrnk, i_mrgrnk
00013 end interface mrgrnk
00014 contains
00015
00016 Subroutine d_mrgrnk (XDONT, IRNGT)
00017 ! ____________________________________________________
00018 !   MRGRNK = Merge-sort ranking of an array
00019 !   For performance reasons, the first 2 passes are taken
00020 !   out of the standard loop, and use dedicated coding.
00021 ! ____________________________________________________
00022 ! ____________________________________________________
00023       Real (kind=kdp), Dimension (:), Intent (In) :: xdont
00024       Integer, Dimension (:), Intent (Out) :: IRNGT
00025 ! ____________________________________________________
00026       Real (kind=kdp) :: xvala, xvalb
00027 !
00028       Integer, Dimension (SIZE(IRNGT)) :: JWRKT
00029       Integer :: LMTNA, LMTNC, IRNG1, IRNG2
00030       Integer :: NVAL, IIND, IWRKD, IWRK, IWRKF, JINDA, IINDA, IINDB
00031 !
00032       nval = min(SIZE(xdont), SIZE(irngt))
00033       Select Case (nval)
00034       Case (:0)
00035          Return
00036       Case (1)
00037          irngt(1) = 1
00038          Return
00039       Case Default
00040          Continue
00041       End Select
00042 !
00043 !  Fill-in the index array, creating ordered couples
00044 !
00045       Do iind = 2, nval, 2
00046          If (xdont(iind-1) <= xdont(iind)) Then
00047             irngt(iind-1) = iind - 1
00048             irngt(iind) = iind
00049          Else
00050             irngt(iind-1) = iind
00051             irngt(iind) = iind - 1
00052          End If
00053       End Do
00054       If (modulo(nval, 2) /= 0) Then
00055          irngt(nval) = nval
00056       End If
00057 !
00058 !  We will now have ordered subsets A - B - A - B - ...
00059 !  and merge A and B couples into     C   -   C   - ...
00060 !
00061       lmtna = 2
00062       lmtnc = 4
00063 !
00064 !  First iteration. The length of the ordered subsets goes from 2 to 4
00065 !
00066       Do
00067          If (nval <= 2) Exit
00068 !
00069 !   Loop on merges of A and B into C
00070 !
00071          Do iwrkd = 0, nval - 1, 4
00072             If ((iwrkd+4) > nval) Then
```

```
00073                If ((iwrkd+2) >= nval) Exit
00074 !
00075 !   1 2 3
00076 !
00077                If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) Exit
00078 !
00079 !   1 3 2
00080 !
00081                If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00082                   irng2 = irngt(iwrkd+2)
00083                   irngt(iwrkd+2) = irngt(iwrkd+3)
00084                   irngt(iwrkd+3) = irng2
00085 !
00086 !   3 1 2
00087 !
00088                Else
00089                   irng1 = irngt(iwrkd+1)
00090                   irngt(iwrkd+1) = irngt(iwrkd+3)
00091                   irngt(iwrkd+3) = irngt(iwrkd+2)
00092                   irngt(iwrkd+2) = irng1
00093                End If
00094                Exit
00095             End If
00096 !
00097 !   1 2 3 4
00098 !
00099             If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) cycle
00100 !
00101 !   1 3 x x
00102 !
00103             If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00104                irng2 = irngt(iwrkd+2)
00105                irngt(iwrkd+2) = irngt(iwrkd+3)
00106                If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00107 !   1 3 2 4
00108                   irngt(iwrkd+3) = irng2
00109                Else
00110 !   1 3 4 2
00111                   irngt(iwrkd+3) = irngt(iwrkd+4)
00112                   irngt(iwrkd+4) = irng2
00113                End If
00114 !
00115 !   3 x x x
00116 !
00117             Else
00118                irng1 = irngt(iwrkd+1)
00119                irng2 = irngt(iwrkd+2)
00120                irngt(iwrkd+1) = irngt(iwrkd+3)
00121                If (xdont(irng1) <= xdont(irngt(iwrkd+4))) Then
00122                   irngt(iwrkd+2) = irng1
00123                   If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00124 !   3 1 2 4
00125                      irngt(iwrkd+3) = irng2
00126                   Else
00127 !   3 1 4 2
00128                      irngt(iwrkd+3) = irngt(iwrkd+4)
00129                      irngt(iwrkd+4) = irng2
00130                   End If
00131                Else
00132 !   3 4 1 2
00133                   irngt(iwrkd+2) = irngt(iwrkd+4)
00134                   irngt(iwrkd+3) = irng1
00135                   irngt(iwrkd+4) = irng2
00136                End If
00137             End If
00138          End Do
00139 !
00140 !  The Cs become As and Bs
00141 !
00142          lmtna = 4
00143          Exit
00144       End Do
00145 !
00146 !  Iteration loop. Each time, the length of the ordered subsets
00147 !  is doubled.
00148 !
00149       Do
00150          If (lmtna >= nval) Exit
00151          iwrkf = 0
00152          lmtnc = 2 * lmtnc
00153 !
00154 !   Loop on merges of A and B into C
00155 !
00156          Do
00157             iwrk = iwrkf
00158             iwrkd = iwrkf + 1
00159             jinda = iwrkf + lmtna
```

```
00160             iwrkf = iwrkf + lmtnc
00161             If (iwrkf >= nval) Then
00162                If (jinda >= nval) Exit
00163                iwrkf = nval
00164             End If
00165             iinda = 1
00166             iindb = jinda + 1
00167 !
00168 !   Shortcut for the case when the max of A is smaller
00169 !   than the min of B. This line may be activated when the
00170 !   initial set is already close to sorted.
00171 !
00172 !          IF (XDONT(IRNGT(JINDA)) <= XDONT(IRNGT(IINDB))) CYCLE
00173 !
00174 !  One steps in the C subset, that we build in the final rank array
00175 !
00176 !  Make a copy of the rank array for the merge iteration
00177 !
00178             jwrkt(1:lmtna) = irngt(iwrkd:jinda)
00179 !
00180             xvala = xdont(jwrkt(iinda))
00181             xvalb = xdont(irngt(iindb))
00182 !
00183             Do
00184                iwrk = iwrk + 1
00185 !
00186 !  We still have unprocessed values in both A and B
00187 !
00188                If (xvala > xvalb) Then
00189                   irngt(iwrk) = irngt(iindb)
00190                   iindb = iindb + 1
00191                   If (iindb > iwrkf) Then
00192 !  Only A still with unprocessed values
00193                      irngt(iwrk+1:iwrkf) = jwrkt(iinda:lmtna)
00194                      Exit
00195                   End If
00196                   xvalb = xdont(irngt(iindb))
00197                Else
00198                   irngt(iwrk) = jwrkt(iinda)
00199                   iinda = iinda + 1
00200                   If (iinda > lmtna) exit! Only B still with unprocessed values
00201                   xvala = xdont(jwrkt(iinda))
00202                End If
00203 !
00204             End Do
00205          End Do
00206 !
00207 !  The Cs become As and Bs
00208 !
00209          lmtna = 2 * lmtna
00210       End Do
00211 !
00212       Return
00213 !
00214 End Subroutine d_mrgrnk
00215
00216 Subroutine r_mrgrnk (XDONT, IRNGT)
00217 ! ______________________________________________________________
00218 !   MRGRNK = Merge-sort ranking of an array
00219 !   For performance reasons, the first 2 passes are taken
00220 !   out of the standard loop, and use dedicated coding.
00221 ! ______________________________________________________________
00222 ! ______________________________________________________________
00223       Real, Dimension (:), Intent (In) :: XDONT
00224       Integer, Dimension (:), Intent (Out) :: IRNGT
00225 ! ______________________________________________________________
00226       Real :: XVALA, XVALB
00227 !
00228       Integer, Dimension (SIZE(IRNGT)) :: JWRKT
00229       Integer :: LMTNA, LMTNC, IRNG1, IRNG2
00230       Integer :: NVAL, IIND, IWRKD, IWRK, IWRKF, JINDA, IINDA, IINDB
00231 !
00232       nval = min(SIZE(xdont), SIZE(irngt))
00233       Select Case (nval)
00234       Case (:0)
00235          Return
00236       Case (1)
00237          irngt(1) = 1
00238          Return
00239       Case Default
00240          Continue
00241       End Select
00242 !
00243 !  Fill-in the index array, creating ordered couples
00244 !
00245       Do iind = 2, nval, 2
00246          If (xdont(iind-1) <= xdont(iind)) Then
```

```
00247             irngt(iind-1) = iind - 1
00248             irngt(iind) = iind
00249          Else
00250             irngt(iind-1) = iind
00251             irngt(iind) = iind - 1
00252          End If
00253       End Do
00254       If (modulo(nval, 2) /= 0) Then
00255          irngt(nval) = nval
00256       End If
00257 !
00258 !  We will now have ordered subsets A - B - A - B - ...
00259 !  and merge A and B couples into     C   -   C   - ...
00260 !
00261       lmtna = 2
00262       lmtnc = 4
00263 !
00264 !  First iteration. The length of the ordered subsets goes from 2 to 4
00265 !
00266       Do
00267          If (nval <= 2) Exit
00268 !
00269 !   Loop on merges of A and B into C
00270 !
00271          Do iwrkd = 0, nval - 1, 4
00272             If ((iwrkd+4) > nval) Then
00273                If ((iwrkd+2) >= nval) Exit
00274 !
00275 !   1 2 3
00276 !
00277                If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) Exit
00278 !
00279 !   1 3 2
00280 !
00281                If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00282                   irng2 = irngt(iwrkd+2)
00283                   irngt(iwrkd+2) = irngt(iwrkd+3)
00284                   irngt(iwrkd+3) = irng2
00285 !
00286 !   3 1 2
00287 !
00288                Else
00289                   irng1 = irngt(iwrkd+1)
00290                   irngt(iwrkd+1) = irngt(iwrkd+3)
00291                   irngt(iwrkd+3) = irngt(iwrkd+2)
00292                   irngt(iwrkd+2) = irng1
00293                End If
00294                Exit
00295             End If
00296 !
00297 !   1 2 3 4
00298 !
00299             If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) cycle
00300 !
00301 !   1 3 x x
00302 !
00303             If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00304                irng2 = irngt(iwrkd+2)
00305                irngt(iwrkd+2) = irngt(iwrkd+3)
00306                If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00307 !   1 3 2 4
00308                   irngt(iwrkd+3) = irng2
00309                Else
00310 !   1 3 4 2
00311                   irngt(iwrkd+3) = irngt(iwrkd+4)
00312                   irngt(iwrkd+4) = irng2
00313                End If
00314 !
00315 !   3 x x x
00316 !
00317             Else
00318                irng1 = irngt(iwrkd+1)
00319                irng2 = irngt(iwrkd+2)
00320                irngt(iwrkd+1) = irngt(iwrkd+3)
00321                If (xdont(irng1) <= xdont(irngt(iwrkd+4))) Then
00322                   irngt(iwrkd+2) = irng1
00323                   If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00324 !   3 1 2 4
00325                      irngt(iwrkd+3) = irng2
00326                   Else
00327 !   3 1 4 2
00328                      irngt(iwrkd+3) = irngt(iwrkd+4)
00329                      irngt(iwrkd+4) = irng2
00330                   End If
00331                Else
00332 !   3 4 1 2
00333                   irngt(iwrkd+2) = irngt(iwrkd+4)
```

```
00334                   irngt(iwrkd+3) = irng1
00335                   irngt(iwrkd+4) = irng2
00336                End If
00337             End If
00338          End Do
00339 !
00340 !  The Cs become As and Bs
00341 !
00342          lmtna = 4
00343          Exit
00344       End Do
00345 !
00346 !  Iteration loop. Each time, the length of the ordered subsets
00347 !  is doubled.
00348 !
00349       Do
00350          If (lmtna >= nval) Exit
00351          iwrkf = 0
00352          lmtnc = 2 * lmtnc
00353 !
00354 !   Loop on merges of A and B into C
00355 !
00356          Do
00357             iwrk = iwrkf
00358             iwrkd = iwrkf + 1
00359             jinda = iwrkf + lmtna
00360             iwrkf = iwrkf + lmtnc
00361             If (iwrkf >= nval) Then
00362                If (jinda >= nval) Exit
00363                iwrkf = nval
00364             End If
00365             iinda = 1
00366             iindb = jinda + 1
00367 !
00368 !   Shortcut for the case when the max of A is smaller
00369 !   than the min of B. This line may be activated when the
00370 !   initial set is already close to sorted.
00371 !
00372 !          IF (XDONT(IRNGT(JINDA)) <= XDONT(IRNGT(IINDB))) CYCLE
00373 !
00374 !  One steps in the C subset, that we build in the final rank array
00375 !
00376 !  Make a copy of the rank array for the merge iteration
00377 !
00378             jwrkt(1:lmtna) = irngt(iwrkd:jinda)
00379 !
00380             xvala = xdont(jwrkt(iinda))
00381             xvalb = xdont(irngt(iindb))
00382 !
00383             Do
00384                iwrk = iwrk + 1
00385 !
00386 !  We still have unprocessed values in both A and B
00387 !
00388                If (xvala > xvalb) Then
00389                   irngt(iwrk) = irngt(iindb)
00390                   iindb = iindb + 1
00391                   If (iindb > iwrkf) Then
00392 !  Only A still with unprocessed values
00393                      irngt(iwrk+1:iwrkf) = jwrkt(iinda:lmtna)
00394                      Exit
00395                   End If
00396                   xvalb = xdont(irngt(iindb))
00397                Else
00398                   irngt(iwrk) = jwrkt(iinda)
00399                   iinda = iinda + 1
00400                   If (iinda > lmtna) exit! Only B still with unprocessed values
00401                   xvala = xdont(jwrkt(iinda))
00402                End If
00403 !
00404             End Do
00405          End Do
00406 !
00407 !  The Cs become As and Bs
00408 !
00409          lmtna = 2 * lmtna
00410       End Do
00411 !
00412       Return
00413 !
00414 End Subroutine r_mrgrnk
00415 Subroutine i_mrgrnk (XDONT, IRNGT)
00416 ! ____________________________________________________________
00417 !   MRGRNK = Merge-sort ranking of an array
00418 !   For performance reasons, the first 2 passes are taken
00419 !   out of the standard loop, and use dedicated coding.
00420 ! ____________________________________________________________
```

```
00421 ! ____________________________________________________________
00422       Integer, Dimension (:), Intent (In)  :: XDONT
00423       Integer, Dimension (:), Intent (Out) :: IRNGT
00424 ! ____________________________________________________________
00425       Integer :: XVALA, XVALB
00426 !
00427       Integer, Dimension (SIZE(IRNGT)) :: JWRKT
00428       Integer :: LMTNA, LMTNC, IRNG1, IRNG2
00429       Integer :: NVAL, IIND, IWRKD, IWRK, IWRKF, JINDA, IINDA, IINDB
00430 !
00431       nval = min(SIZE(xdont), SIZE(irngt))
00432       Select Case (nval)
00433       Case (:0)
00434          Return
00435       Case (1)
00436          irngt(1) = 1
00437          Return
00438       Case Default
00439          Continue
00440       End Select
00441 !
00442 !  Fill-in the index array, creating ordered couples
00443 !
00444       Do iind = 2, nval, 2
00445          If (xdont(iind-1) <= xdont(iind)) Then
00446             irngt(iind-1) = iind - 1
00447             irngt(iind) = iind
00448          Else
00449             irngt(iind-1) = iind
00450             irngt(iind) = iind - 1
00451          End If
00452       End Do
00453       If (modulo(nval, 2) /= 0) Then
00454          irngt(nval) = nval
00455       End If
00456 !
00457 !  We will now have ordered subsets A - B - A - B - ...
00458 !  and merge A and B couples into     C   -   C   - ...
00459 !
00460       lmtna = 2
00461       lmtnc = 4
00462 !
00463 !  First iteration. The length of the ordered subsets goes from 2 to 4
00464 !
00465       Do
00466          If (nval <= 2) Exit
00467 !
00468 !   Loop on merges of A and B into C
00469 !
00470          Do iwrkd = 0, nval - 1, 4
00471             If ((iwrkd+4) > nval) Then
00472                If ((iwrkd+2) >= nval) Exit
00473 !
00474 !   1 2 3
00475 !
00476                If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) Exit
00477 !
00478 !   1 3 2
00479 !
00480                If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00481                   irng2 = irngt(iwrkd+2)
00482                   irngt(iwrkd+2) = irngt(iwrkd+3)
00483                   irngt(iwrkd+3) = irng2
00484 !
00485 !   3 1 2
00486 !
00487                Else
00488                   irng1 = irngt(iwrkd+1)
00489                   irngt(iwrkd+1) = irngt(iwrkd+3)
00490                   irngt(iwrkd+3) = irngt(iwrkd+2)
00491                   irngt(iwrkd+2) = irng1
00492                End If
00493                Exit
00494             End If
00495 !
00496 !   1 2 3 4
00497 !
00498             If (xdont(irngt(iwrkd+2)) <= xdont(irngt(iwrkd+3))) cycle
00499 !
00500 !   1 3 x x
00501 !
00502             If (xdont(irngt(iwrkd+1)) <= xdont(irngt(iwrkd+3))) Then
00503                irng2 = irngt(iwrkd+2)
00504                irngt(iwrkd+2) = irngt(iwrkd+3)
00505                If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00506 !   1 3 2 4
00507                   irngt(iwrkd+3) = irng2
```

```
00508                Else
00509 !   1 3 4 2
00510                   irngt(iwrkd+3) = irngt(iwrkd+4)
00511                   irngt(iwrkd+4) = irng2
00512                End If
00513 !
00514 !   3 x x x
00515 !
00516             Else
00517                irng1 = irngt(iwrkd+1)
00518                irng2 = irngt(iwrkd+2)
00519                irngt(iwrkd+1) = irngt(iwrkd+3)
00520                If (xdont(irng1) <= xdont(irngt(iwrkd+4))) Then
00521                   irngt(iwrkd+2) = irng1
00522                   If (xdont(irng2) <= xdont(irngt(iwrkd+4))) Then
00523 !   3 1 2 4
00524                      irngt(iwrkd+3) = irng2
00525                   Else
00526 !   3 1 4 2
00527                      irngt(iwrkd+3) = irngt(iwrkd+4)
00528                      irngt(iwrkd+4) = irng2
00529                   End If
00530                Else
00531 !   3 4 1 2
00532                   irngt(iwrkd+2) = irngt(iwrkd+4)
00533                   irngt(iwrkd+3) = irng1
00534                   irngt(iwrkd+4) = irng2
00535                End If
00536             End If
00537          End Do
00538 !
00539 !  The Cs become As and Bs
00540 !
00541          lmtna = 4
00542          Exit
00543       End Do
00544 !
00545 !  Iteration loop. Each time, the length of the ordered subsets
00546 !  is doubled.
00547 !
00548       Do
00549          If (lmtna >= nval) Exit
00550          iwrkf = 0
00551          lmtnc = 2 * lmtnc
00552 !
00553 !   Loop on merges of A and B into C
00554 !
00555          Do
00556             iwrk = iwrkf
00557             iwrkd = iwrkf + 1
00558             jinda = iwrkf + lmtna
00559             iwrkf = iwrkf + lmtnc
00560             If (iwrkf >= nval) Then
00561                If (jinda >= nval) Exit
00562                iwrkf = nval
00563             End If
00564             iinda = 1
00565             iindb = jinda + 1
00566 !
00567 !   Shortcut for the case when the max of A is smaller
00568 !   than the min of B. This line may be activated when the
00569 !   initial set is already close to sorted.
00570 !
00571 !          IF (XDONT(IRNGT(JINDA)) <= XDONT(IRNGT(IINDB))) CYCLE
00572 !
00573 !  One steps in the C subset, that we build in the final rank array
00574 !
00575 !  Make a copy of the rank array for the merge iteration
00576 !
00577             jwrkt(1:lmtna) = irngt(iwrkd:jinda)
00578 !
00579             xvala = xdont(jwrkt(iinda))
00580             xvalb = xdont(irngt(iindb))
00581 !
00582             Do
00583                iwrk = iwrk + 1
00584 !
00585 !  We still have unprocessed values in both A and B
00586 !
00587                If (xvala > xvalb) Then
00588                   irngt(iwrk) = irngt(iindb)
00589                   iindb = iindb + 1
00590                   If (iindb > iwrkf) Then
00591 !  Only A still with unprocessed values
00592                      irngt(iwrk+1:iwrkf) = jwrkt(iinda:lmtna)
00593                      Exit
00594                   End If
```

```
00595                   xvalb = xdont(irngt(iindb))
00596                Else
00597                   irngt(iwrk) = jwrkt(iinda)
00598                   iinda = iinda + 1
00599                   If (iinda > lmtna) exit! Only B still with unprocessed values
00600                   xvala = xdont(jwrkt(iinda))
00601                End If
00602 !
00603             End Do
00604          End Do
00605 !
00606 !  The Cs become As and Bs
00607 !
00608          lmtna = 2 * lmtna
00609       End Do
00610 !
00611       Return
00612 !
00613 End Subroutine i_mrgrnk
00614 end module m_mrgrnk
```

16.119 src/libamns/strings.f90 File Reference

Data Types

- interface strings::operator(//)
- interface strings::operator(//)
- interface strings::operator(//)

Modules

- module strings

  strings module from Silvio Gori's grid package

Functions/Subroutines

- function strings::print_int_r (char, zahl)
- function strings::print_int_l (zahl, char)
- character(len=len(char)+22) function strings::print_dble_r (char, zahl)
- character(len=len(char)+22) function strings::print_dble_l (zahl, char)
- character(len=len(char)+12) function strings::print_real_r (char, zahl)
- character(len=len(char)+12) function strings::print_real_l (zahl, char)

16.120 strings.f90

```
00001
00013
00014 !########################################tall#
00015 !---------- strings --------------------
00016 !########################################
00017 module strings
00018
00019   use f90_kind
00020
00021   implicit none
00022
00023   public
00024
00025
00026   interface operator(//)
00027      module procedure print_int_r
00028      module procedure print_int_l
00029   end interface
00030
00031   interface operator(//)
00032      module procedure print_dble_r
00033      module procedure print_dble_l
00034   end interface
00035
00036   interface operator(//)
00037      module procedure print_real_r
00038      module procedure print_real_l
```

```
00039   end interface
00040
00041 contains
00042
00043   ! int right
00044   !----------
00045   function print_int_r(char, zahl ) result(string)
00046
00047     character(len=*), intent(in) :: char
00048     integer(ikind) , intent(in) :: zahl
00049     character(len=len(char)+int(log10(dble(max(1,zahl))))+ &
00050          1-floor(sign(0.1d0,dble(zahl))))::string
00051
00052     write(string,'(i0)')zahl
00053     string=char//trim(adjustl(string))
00054
00055   end function print_int_r
00056
00057   ! int left
00058   !---------
00059   function print_int_l(zahl, char ) result(string)
00060     character(len=*), intent(in) :: char
00061     integer(ikind) , intent(in) :: zahl
00062     character(len=len(char)+int(log10(dble(max(1,zahl))))+ &
00063          1-floor(sign(0.1d0,dble(zahl)))) :: string
00064
00065     write(string,'(i0)')zahl
00066     string=trim(adjustl(string))//char
00067   end function print_int_l
00068
00069   !dble right
00070   !----------
00071   function print_dble_r(char, zahl ) result(string)
00072     character(len=*), intent(in) :: char
00073     real(rkind) , intent(in) :: zahl
00074     character(len=len(char)+22)::string
00075
00076     write(string,'(1p,e22.15)')zahl
00077     string=char//trim(adjustl(string))
00078
00079   end function print_dble_r
00080
00081   !dble left
00082   !----------
00083   function print_dble_l(zahl, char ) result(string)
00084     character(len=*), intent(in) :: char
00085     real(rkind) , intent(in) :: zahl
00086     character(len=len(char)+22)::string
00087
00088     write(string,'(1p,e22.15)')zahl
00089     string=char//trim(adjustl(string))
00090
00091   end function print_dble_l
00092
00093   !real right
00094   !----------
00095   function print_real_r(char, zahl ) result(string)
00096     character(len=*), intent(in) :: char
00097     real , intent(in) :: zahl
00098     character(len=len(char)+12)::string
00099
00100     write(string,'(1p,e12.6)')dble(zahl)
00101     string=char//trim(adjustl(string))
00102
00103   end function print_real_r
00104
00105   !real left
00106   !----------
00107   function print_real_l(zahl, char ) result(string)
00108     character(len=*), intent(in) :: char
00109     real , intent(in) :: zahl
00110     character(len=len(char)+12)::string
00111
00112     write(string,'(1p,e12.6)')dble(zahl)
00113     string=char//trim(adjustl(string))
00114
00115   end function print_real_l
00116 end module strings
```

16.121 src/libamns/unit_h.f90 File Reference

Modules

- module unit_h

  unit_h module from Silvio Gori's grid package

Functions/Subroutines

- integer(ikind) function, public unit_h::next_unit ()
- subroutine, public unit_h::skip_comment_line (iunit)
- subroutine, public unit_h::read_error (iunit)

16.122 unit_h.f90

```
00001
00006
00007 module unit_h
00008
00009   use f90_kind
00010   use call_utils
00011
00012   implicit none
00013
00014   private
00015
00016
00017   integer, save      :: counter=2000
00018   integer, parameter :: maxcount=9999     !! try 9999 maximum
00019   !  public subs
00020   !
00021   public :: next_unit, skip_comment_line, read_error
00022
00023 contains
00024   !-----------------------------------------------
00025   !--------- next_unit ---------------------------
00026   !-----------------------------------------------
00027   !
00028   function next_unit() result(unit_num)     !
00029
00030     integer(ikind)     :: unit_num          !! return the next free unit number
00031
00032     logical            :: op                !! false if unit not in use
00033
00034     call sub_init("next_unit")
00035     do
00036        counter = counter +1
00037        inquire (counter, opened = op)       ! check if unit is open ! bug in f95n
00038        if (.not.op) exit                    ! found next free unit
00039        if (counter > maxcount) &
00040             call assert(.false.,' fatal : next_unit@unit_h could not find free unit number')
00041     enddo
00042
00043     unit_num = counter
00044
00045     call sub_end()
00046
00047   end function next_unit
00048
00049
00050   !---------------------------------------
00051   !------- skip_comment_line --------------
00052   !---------------------------------------
00053   ! skips lines that starts with "!"
00054   subroutine skip_comment_line(iunit)
00055
00056     integer(ikind), intent(in) :: iunit
00057
00058     character(skind) :: char
00059
00060     integer :: iostat
00061
00062     do
00063        read(iunit,'(a)',iostat=iostat) char
00064        !print *, '#', trim(char),'#'
00065        if(iostat /= 0) return
00066        if(char(1:1)/='!'.and.len(trim(char))>0) then
00067           backspace iunit
00068           return
```

```
00069         endif
00070       enddo
00071
00072     end subroutine skip_comment_line
00073
00074     !-----------------------------------------------
00075     !-------- read_error ---------------------------
00076     !-----------------------------------------------
00077     subroutine read_error(iunit)
00078       integer(ikind)  :: iunit
00079
00080       character(len=256) :: last_record
00081
00082       backspace(iunit)
00083
00084       read(iunit,'(a256)') last_record
00085
00086       print *, 'read_error@unit_h: last string read'
00087       print *, trim(last_record)
00088
00089       call exiting(.false., 'read_error@unit_h: last string read:"'//last_record//'"')
00090
00091     end subroutine read_error
00092
00093     !
00094 end module unit_h
00095
00096
```

16.123 src/py/amns/amns.pyx File Reference

Classes

- class amns.Amns
- class amns.Table
- class amns.Reactants
- class amns.AmnsException

Namespaces

- amns

16.124 amns.pyx

```
00001 # cython: language_level=3
00002 cimport camns_interface
00003 from camns_interface cimport *
00004 import numpy as np
00005 cimport numpy as np
00006
00007 # some SO for cython & numpy
00008 # http://stackoverflow.com/questions/3046305/simple-wrapping-of-c-code-with-cython
00009 # http://stackoverflow.com/questions/4495420/passing-numpy-arrays-to-c-code-wrapped-with-cython
00010 #
      http://stackoverflow.com/questions/8366095/have-pointer-to-data-need-numpy-array-in-fortran-order-want-to-use-cython
00011
00012 cdef class Amns:
00013 #     """Class for interfacing to the AMNS library"""
00014
00015     cdef void* _handle
00016 #         """AMNS handle (opaque)"""
00017
00018     def __init__(self, version_string=None, version_number=None, version_backend=None,
      version_user=None):
00019         """Use to initialize the Amns class
00020         in:
00021                 version_string: requested version (string, default None)
00022                 version_number: requested version (integer, default None)
00023                 version_backend: requested backend (string, default None)
00024                 version_user: requated user (string, default None)
00025         out:
00026                 instance of the Amns class
00027         """
00028         self._handle = NULL
00029         self._setup(version_string=version_string, version_number=version_number,
      version_backend=version_backend, version_user=version_user)
00030
```

```
00031     cdef void* handle(self):
00032         return self._handle
00033 
00034     def _setup(self, version_string=None, version_number=None, version_backend=None,
      version_user=None):
00035         """Perform the actual call to the C interface to the original primary Fortran SETUP routine"""
00036         cdef camns_interface.amns_c_error_type error_status
00037         cdef camns_interface.amns_c_version_type version
00038         version = camns_interface.get_default_amns_c_version_type()
00039         if (version_string == None) & ( version_number == None) & (version_backend == None) &
      (version_user == None):
00040             camns_interface.IMAS_AMNS_CC_SETUP(&self._handle, &error_status)
00041         else:
00042 #            print(version)
00043             if version_string  != None:
00044                 if (type(version_string)) == bytes:
00045                     version.string = version_string
00046                 else:
00047                     tmp_c_str = version_string.encode('UTF-8')
00048                     version.string = tmp_c_str
00049             if version_number  != None: version.number = version_number
00050             if version_backend != None:
00051                 if (type(version_backend)) == bytes:
00052                     version.backend = version_backend
00053                 else:
00054                     tmp_c_str = version_backend.encode('UTF-8')
00055                     version.backend = tmp_c_str
00056             if version_user    != None:
00057                 if (type(version_user)) == bytes:
00058                     version.user = version_user
00059                 else:
00060                     tmp_c_str = version_user.encode('UTF-8')
00061                     version.user = tmp_c_str
00062 #            print(version)
00063             camns_interface.IMAS_AMNS_CC_SETUP_VERSION(&self._handle, &version, &error_status)
00064         if error_status.flag:
00065             raise AmnsException(error_status.string.decode('UTF-8'))
00066 
00067     cdef camns_interface.amns_c_answer_type lquery(self, queryString):
00068         cdef camns_interface.amns_c_query_type query
00069         cdef camns_interface.amns_c_answer_type answer
00070         cdef camns_interface.amns_c_error_type error_status
00071         if type(queryString) == bytes:
00072             query.string = queryString
00073         else:
00074             tmp_c_str = queryString.encode('UTF-8')
00075             query.string = tmp_c_str
00076         camns_interface.IMAS_AMNS_CC_QUERY(self._handle, &query, &answer, &error_status);
00077         if error_status.flag:
00078             raise AmnsException(error_status.string.decode('UTF-8'))
00079         return answer
00080 
00081     def query(self, queryString):
00082         """Provide the interface to IMAS_AMNS_QUERY
00083         in:
00084                 queryString: string containing query (string, required)
00085         out:
00086                 answer_string: string containing reply
00087                 answer_number: number containing any numerical answer
00088 
00089         """
00090         cdef camns_interface.amns_c_query_type query
00091         cdef camns_interface.amns_c_answer_type answer
00092         cdef camns_interface.amns_c_error_type error_status
00093         if type(queryString) == bytes:
00094             query.string = queryString
00095         else:
00096             tmp_c_str = queryString.encode('UTF-8')
00097             query.string = tmp_c_str
00098         camns_interface.IMAS_AMNS_CC_QUERY(self._handle, &query, &answer, &error_status);
00099         if error_status.flag:
00100             raise AmnsException(error_status.string.decode('UTF-8'))
00101         return answer.string.decode('UTF-8'), answer.number
00102 
00103     @property
00104     def version(self):
00105         """Query for version of the AMNS data"""
00106         cdef camns_interface.amns_c_answer_type answer
00107         answer = self.lquery(b"version")
00108         return answer.number, answer.string.decode('UTF-8')
00109 
00110     @property
00111     def prop_comment(self):
00112         """Query for prop_comment associated with the AMNS data"""
00113         cdef camns_interface.amns_c_answer_type answer
00114         answer = self.lquery(b"prop_comment")
00115         return answer.string.decode('UTF-8')
```

```
00116
00117     @property
00118     def prop_source(self):
00119         """Query for prop_source associated with the AMNS data"""
00120         cdef camns_interface.amns_c_answer_type answer
00121         answer = self.lquery(b"prop_source")
00122         return answer.string.decode('UTF-8')
00123
00124     @property
00125     def prop_provider(self):
00126         """Query for prop_provider associated with the AMNS data"""
00127         cdef camns_interface.amns_c_answer_type answer
00128         answer = self.lquery(b"prop_provider")
00129         return answer.string.decode('UTF-8')
00130
00131     @property
00132     def prop_creation(self):
00133         """Query for prop_creation associated with the AMNS data"""
00134         cdef camns_interface.amns_c_answer_type answer
00135         answer = self.lquery(b"prop_creation")
00136         return answer.string.decode('UTF-8')
00137
00138     @property
00139     def code_name(self):
00140         """Query for code_name associated with the AMNS data"""
00141         cdef camns_interface.amns_c_answer_type answer
00142         answer = self.lquery(b"code_name")
00143         return answer.string.decode('UTF-8')
00144
00145     @property
00146     def code_commit(self):
00147         """Query for code_commit associated with the AMNS data"""
00148         cdef camns_interface.amns_c_answer_type answer
00149         answer = self.lquery(b"code_commit")
00150         return answer.string.decode('UTF-8')
00151
00152     @property
00153     def code_version(self):
00154         """Query for code_version associated with the AMNS data"""
00155         cdef camns_interface.amns_c_answer_type answer
00156         answer = self.lquery(b"code_version")
00157         return answer.string.decode('UTF-8')
00158
00159     @property
00160     def code_repository(self):
00161         """Query for code_repository associated with the AMNS data"""
00162         cdef camns_interface.amns_c_answer_type answer
00163         answer = self.lquery(b"code_repository")
00164         return answer.string.decode('UTF-8')
00165
00166     def set(self, setString):
00167         """Provide the interface to IMAS_AMNS_SET
00168         in:
00169                 setString: string containing setting (string, required)
00170         out:
00171                 None
00172
00173         """
00174         cdef camns_interface.amns_c_set_type set
00175         cdef camns_interface.amns_c_error_type error_status
00176         if type(setString) == bytes:
00177             set.string = setString
00178         else:
00179             tmp_c_str = setString.encode('UTF-8')
00180             set.string = tmp_c_str
00181         camns_interface.IMAS_AMNS_CC_SET(self._handle, &set, &error_status);
00182         if error_status.flag:
00183             raise AmnsException(error_status.string.decode('UTF-8'))
00184
00185     def finalize(self):
00186         """Provide the interface to IMAS_AMNS_FINISH
00187         in:
00188                 None
00189         out:
00190                 None
00191         """
00192         cdef camns_interface.amns_c_error_type error_status
00193         camns_interface.IMAS_AMNS_CC_FINISH(&self._handle, &error_status)
00194         if error_status.flag:
00195             raise AmnsException(error_status.string.decode('UTF-8'))
00196
00197     def get_table(self, reactionString, Reactants reactants, isotope_resolved=None):
00198         """Provide the interface to the Table class used for tables
00199         in:
00200                 reactionString: name of reaction (string, required)
00201                 reactants: reactants object (of type amns.Reactants, required)
00202                 isotope_resolved: 0 if not isotope resolved, 1 if isotope resolved (integer, default
```

```
    None)
00203         out:
00204                 amns table (of type amns.Table)
00205         """
00206         return Table(reactionString, reactants, self, isotope_resolved)
00207
00208
00209 cdef class Table:
00210 #     """Class for interfacing to Tables in the the AMNS library"""
00211
00212     cdef void* _handle
00213     cdef Amns _parentAmns
00214
00215     def __init__(self, reactionString, Reactants reactants, Amns parentAmns, isotope_resolved=None):
00216         """Create a new table
00217
00218         called by AMNS.get_table
00219
00220         in:
00221                 reactionString: name of reaction (string, required)
00222                 reactants: reactants object (of type amns.Reactants, required)
00223                 Amns: object of class Amns (of type Amns, required)
00224                 isotope_resolved: 0 if not isotope resolved, 1 if isotope resolved (integer, default
    None)
00225         out:
00226                 instance of Table class
00227         """
00228         cdef camns_interface.amns_c_error_type error_status
00229         cdef camns_interface.amns_c_reaction_type reaction
00230         reaction = camns_interface.get_default_amns_c_reaction_type()
00231         self._parentAmns = parentAmns
00232         if type(reactionString) == bytes:
00233             reaction.string = reactionString
00234         else:
00235             tmp_c_str = reactionString.encode('UTF-8')
00236             reaction.string = tmp_c_str
00237         if isotope_resolved : reaction.isotope_resolved = isotope_resolved
00238         camns_interface.IMAS_AMNS_CC_SETUP_TABLE(self._parentAmns.handle(),
00239                                                  &reaction,
00240                                                  reactants.handle(), &self._handle,
00241                                                  &error_status)
00242         if error_status.flag:
00243             raise AmnsException(error_status.string.decode('UTF-8'))
00244
00245         self.set(b"nowarn")
00246
00247
00248     cdef camns_interface.amns_c_answer_type lquery(self, queryString):
00249         cdef camns_interface.amns_c_query_type query
00250         cdef camns_interface.amns_c_answer_type answer
00251         cdef camns_interface.amns_c_error_type error_status
00252         if type(queryString) == bytes:
00253             query.string = queryString
00254         else:
00255             tmp_c_str = queryString.encode('UTF-8')
00256             query.string = tmp_c_str
00257         camns_interface.IMAS_AMNS_CC_QUERY_TABLE(self._handle, &query, &answer, &error_status);
00258         if error_status.flag:
00259             raise AmnsException(error_status.string.decode('UTF-8'))
00260         return answer
00261
00262     def query(self, queryString):
00263         """Provide the interface to IMAS_AMNS_QUERY_TABLE
00264         in:
00265                 queryString: string containing query (string, required)
00266         out:
00267                 answer_string: string containing reply
00268                 answer_number: number containing any numerical answer
00269
00270         """
00271         cdef camns_interface.amns_c_query_type query
00272         cdef camns_interface.amns_c_answer_type answer
00273         cdef camns_interface.amns_c_error_type error_status
00274         if type(queryString) == bytes:
00275             query.string = queryString
00276         else:
00277             tmp_c_str = queryString.encode('UTF-8')
00278             query.string = tmp_c_str
00279         camns_interface.IMAS_AMNS_CC_QUERY_TABLE(self._handle, &query, &answer, &error_status);
00280         if error_status.flag:
00281             raise AmnsException(error_status.string.decode('UTF-8'))
00282         return answer.string.decode('UTF-8'), answer.number
00283
00284     def set(self, setString):
00285         """Provide the interface to IMAS_AMNS_SET_TABLE
00286         in:
00287                 setString: string containing setting (string, required)
```

```
00288         out:
00289                 None
00290 
00291         """
00292         cdef camns_interface.amns_c_set_type set
00293         cdef camns_interface.amns_c_error_type error_status
00294         if type(setString) == bytes:
00295             set.string = setString
00296         else:
00297             tmp_c_str = setString.encode('UTF-8')
00298             set.string = tmp_c_str
00299         camns_interface.IMAS_AMNS_CC_SET_TABLE(self._handle, &set, &error_status);
00300         if error_status.flag:
00301             raise AmnsException(error_status.string.decode('UTF-8'))
00302 
00303     def finalize(self):
00304         """Provide the interface to IMAS_AMNS_FINISH_TABLE
00305         in:
00306                 None
00307         out:
00308                 None
00309         """
00310         cdef camns_interface.amns_c_error_type error_status
00311         camns_interface.IMAS_AMNS_CC_FINISH_TABLE(&self._handle, &error_status)
00312         if error_status.flag:
00313             raise AmnsException(error_status.string.decode('UTF-8'))
00314 
00315     @property
00316     def no_of_reactants(self):
00317         """Query for no_of_reactants associated with the AMNS table"""
00318         cdef camns_interface.amns_c_answer_type answer
00319         answer = self.lquery(b"no_of_reactants")
00320         return answer.number
00321 
00322     @property
00323     def ndim(self):
00324         """Query for ndim associated with the AMNS table"""
00325         cdef camns_interface.amns_c_answer_type answer
00326         answer = self.lquery(b"ndim")
00327         return answer.number
00328 
00329     @property
00330     def source(self):
00331         """Query for source associated with the AMNS table"""
00332         cdef camns_interface.amns_c_answer_type answer
00333         answer = self.lquery(b"source")
00334         return answer.string.decode('UTF-8')
00335 
00336     @property
00337     def provider(self):
00338         """Query for provider associated with the AMNS table"""
00339         cdef camns_interface.amns_c_answer_type answer
00340         answer = self.lquery(b"provider")
00341         return answer.string.decode('UTF-8')
00342 
00343     @property
00344     def citation(self):
00345         """Query for citation associated with the AMNS table"""
00346         cdef camns_interface.amns_c_answer_type answer
00347         answer = self.lquery(b"citation")
00348         return answer.string.decode('UTF-8')
00349 
00350     @property
00351     def filled(self):
00352         """Query for filled status associated with the AMNS table"""
00353         cdef camns_interface.amns_c_answer_type answer
00354         answer = self.lquery(b"filled")
00355         return answer.string.decode('UTF-8')
00356 
00357     @property
00358     def reaction_type(self):
00359         """Query for reaction_type associated with the AMNS table"""
00360         cdef camns_interface.amns_c_answer_type answer
00361         answer = self.lquery(b"reaction_type")
00362         return answer.string.decode('UTF-8')
00363 
00364     @property
00365     def reactants(self):
00366         """Query for reactants associated with the AMNS table"""
00367         cdef camns_interface.amns_c_answer_type answer
00368         answer = self.lquery(b"reactants")
00369         return answer.string.decode('UTF-8')
00370 
00371     @property
00372     def coordinates(self):
00373         """Query for coordinates associated with the AMNS table"""
00374         cdef camns_interface.amns_c_answer_type answer
```

```
00375         answer = self.lquery(b"coordinates")
00376         return answer.string.decode('UTF-8')
00377
00378     @property
00379     def version(self):
00380         """Query for version associated with the AMNS table"""
00381         cdef camns_interface.amns_c_answer_type answer
00382         answer = self.lquery(b"version")
00383         return answer.string.decode('UTF-8'), answer.number
00384
00385     @property
00386     def state_label(self):
00387         """Query for state_label associated with the AMNS table"""
00388         cdef camns_interface.amns_c_answer_type answer
00389         answer = self.lquery(b"state_label")
00390         return answer.string.decode('UTF-8')
00391
00392     @property
00393     def result_unit(self):
00394         """Query for result_unit associated with the AMNS table"""
00395         cdef camns_interface.amns_c_answer_type answer
00396         answer = self.lquery(b"result_unit")
00397         return answer.string.decode('UTF-8')
00398
00399     @property
00400     def result_label(self):
00401         """Query for result_label associated with the AMNS table"""
00402         cdef camns_interface.amns_c_answer_type answer
00403         answer = self.lquery(b"result_label")
00404         return answer.string.decode('UTF-8')
00405
00406     @property
00407     def interp_fun(self):
00408         """Query for interp_fun associated with the AMNS table"""
00409         cdef camns_interface.amns_c_answer_type answer
00410         answer = self.lquery(b"interp_fun")
00411         return answer.number
00412
00413     @property
00414     def prop_comment(self):
00415         """Query for prop_comment associated with the AMNS table"""
00416         cdef camns_interface.amns_c_answer_type answer
00417         answer = self.lquery(b"prop_comment")
00418         return answer.string.decode('UTF-8')
00419
00420     @property
00421     def prop_source(self):
00422         """Query for prop_source associated with the AMNS table"""
00423         cdef camns_interface.amns_c_answer_type answer
00424         answer = self.lquery(b"prop_source")
00425         return answer.string.decode('UTF-8')
00426
00427     @property
00428     def prop_provider(self):
00429         """Query for prop_provider associated with the AMNS table"""
00430         cdef camns_interface.amns_c_answer_type answer
00431         answer = self.lquery(b"prop_provider")
00432         return answer.string.decode('UTF-8')
00433
00434     @property
00435     def prop_creation(self):
00436         """Query for prop_creation associated with the AMNS table"""
00437         cdef camns_interface.amns_c_answer_type answer
00438         answer = self.lquery(b"prop_creation")
00439         return answer.string.decode('UTF-8')
00440
00441     @property
00442     def code_name(self):
00443         """Query for code_name associated with the AMNS table"""
00444         cdef camns_interface.amns_c_answer_type answer
00445         answer = self.lquery(b"code_name")
00446         return answer.string.decode('UTF-8')
00447
00448     @property
00449     def code_commit(self):
00450         """Query for code_commit associated with the AMNS table"""
00451         cdef camns_interface.amns_c_answer_type answer
00452         answer = self.lquery(b"code_commit")
00453         return answer.string.decode('UTF-8')
00454
00455     @property
00456     def code_version(self):
00457         """Query for code_version associated with the AMNS table"""
00458         cdef camns_interface.amns_c_answer_type answer
00459         answer = self.lquery(b"code_version")
00460         return answer.string.decode('UTF-8')
00461
```

```
00462     @property
00463     def code_repository(self):
00464         """Query for code_repository associated with the AMNS table"""
00465         cdef camns_interface.amns_c_answer_type answer
00466         answer = self.lquery(b"code_repository")
00467         return answer.string.decode('UTF-8')
00468 
00469     def data(self, p1, p2 = None, p3 = None, p4 = None):
00470         """Return the AMNS data based on the arguments
00471         in:
00472                 p1: array of input values for argument 1 (float array, required)
00473                 p2: array of input values for argument 2 (optional, float array of the same shape as
      p1)
00474                 p3: array of input values for argument 3 (optional, float array of the same shape as
      p1)
00475                 p4: array of input values for argument 4 (optional, float array of the same shape as
      p1)
00476         out:
00477                 data evaluated (float array of the same shape as p1)
00478         """
00479         nargs = 1
00480         if p2 is not None: nargs += 1
00481         if p3 is not None: nargs += 1
00482         if p4 is not None: nargs += 1
00483         if nargs != self.ndim:
00484             raise AmnsException("Number of parameters does not match table dimensions (ndim="
00485                                 + str(self.ndim) + ")")
00486 
00487         refShape = p1.shape
00488         if p2 is not None and p2.shape != refShape:
00489             raise AmnsException("Shape of parameter 2 inconsistent with shape of parameter 1")
00490         if p3 is not None and p3.shape != refShape:
00491             raise AmnsException("Shape of parameter 3 inconsistent with shape of parameter 1")
00492         if p4 is not None and p4.shape != refShape:
00493             raise AmnsException("Shape of parameter 4 inconsistent with shape of parameter 1")
00494 
00495         if len(refShape) == 0:
00496             res = self._data_0d(nargs,p1,p2,p3,p4)
00497         elif len(refShape) == 1:
00498             res = self._data_1d(nargs,p1,p2,p3,p4)
00499         elif len(refShape) == 2:
00500             res = self._data_2d(nargs,p1,p2,p3,p4)
00501         elif len(refShape) == 3:
00502             res = self._data_3d(nargs,p1,p2,p3,p4)
00503         else:
00504             raise AmnsException("Unsupported rank of input arguments: %s" % (len(refShape),))
00505 
00506         return res
00507 
00508     def _data_0d(self, int nargs,
00509                        np.double_t p1,
00510                        np.double_t p2 = 0.0,
00511                        np.double_t p3 = 0.0,
00512                        np.double_t p4 = 0.0):
00513         cdef camns_interface.amns_c_error_type error_status
00514         cdef np.ndarray[np.double_t, ndim=1] res
00515 
00516         res = np.empty(1, order='F')
00517 
00518         if nargs == 1:
00519             camns_interface.IMAS_AMNS_CC_RX_0_A(self._handle,
00520                                                 <double*> res.data,
00521                                                 <double> p1, &error_status);
00522         elif nargs == 2:
00523             camns_interface.IMAS_AMNS_CC_RX_0_B(self._handle,
00524                                                 <double*> res.data,
00525                                                 <double> p1,
00526                                                 <double> p2, &error_status);
00527         elif nargs == 3:
00528             camns_interface.IMAS_AMNS_CC_RX_0_C(self._handle,
00529                                                 <double*> res.data,
00530                                                 <double> p1,
00531                                                 <double> p2,
00532                                                 <double> p3, &error_status);
00533         elif nargs == 4:
00534             camns_interface.IMAS_AMNS_CC_RX_0_D(self._handle,
00535                                                 <double*> res.data,
00536                                                 <double> p1,
00537                                                 <double> p2,
00538                                                 <double> p3,
00539                                                 <double> p4, &error_status);
00540 
00541         if error_status.flag:
00542             raise AmnsException(error_status.string.decode('UTF-8'))
00543         return res
00544 
00545     def _data_1d(self, int nargs,
```

```
00546                       np.ndarray[np.double_t, ndim=1] p1,
00547                       np.ndarray[np.double_t, ndim=1] p2 = None,
00548                       np.ndarray[np.double_t, ndim=1] p3 = None,
00549                       np.ndarray[np.double_t, ndim=1] p4 = None):
00550         cdef camns_interface.amns_c_error_type error_status
00551         cdef np.ndarray[np.double_t, ndim=1] res
00552 
00553         res = np.empty_like(p1, order='F')
00554 
00555         if nargs == 1:
00556             camns_interface.IMAS_AMNS_CC_RX_1_A(self._handle, p1.shape[0],
00557                                                 <double*> res.data,
00558                                                 <double*> p1.data, &error_status);
00559         elif nargs == 2:
00560             camns_interface.IMAS_AMNS_CC_RX_1_B(self._handle, p1.shape[0],
00561                                                 <double*> res.data,
00562                                                 <double*> p1.data,
00563                                                 <double*> p2.data, &error_status);
00564         elif nargs == 3:
00565             camns_interface.IMAS_AMNS_CC_RX_1_C(self._handle, p1.shape[0],
00566                                                 <double*> res.data,
00567                                                 <double*> p1.data,
00568                                                 <double*> p2.data,
00569                                                 <double*> p3.data, &error_status);
00570         elif nargs == 4:
00571             camns_interface.IMAS_AMNS_CC_RX_1_D(self._handle, p1.shape[0],
00572                                                 <double*> res.data,
00573                                                 <double*> p1.data,
00574                                                 <double*> p2.data,
00575                                                 <double*> p3.data,
00576                                                 <double*> p4.data, &error_status);
00577 
00578         if error_status.flag:
00579             raise AmnsException(error_status.string.decode('UTF-8'))
00580         return res
00581 
00582     def _data_2d(self, int nargs,
00583                       np.ndarray[np.double_t, ndim=2] p1,
00584                       np.ndarray[np.double_t, ndim=2] p2 = None,
00585                       np.ndarray[np.double_t, ndim=2] p3 = None,
00586                       np.ndarray[np.double_t, ndim=2] p4 = None):
00587 
00588         cdef camns_interface.amns_c_error_type error_status
00589         cdef np.ndarray[np.double_t, ndim=2] res
00590 
00591         res = np.empty_like(p1, order='F')
00592 
00593         if nargs == 1:
00594             camns_interface.IMAS_AMNS_CC_RX_2_A(self._handle, p1.shape[0], p1.shape[1],
00595                                                 <double*> res.data,
00596                                                 <double*> p1.data, &error_status);
00597         elif nargs == 2:
00598             camns_interface.IMAS_AMNS_CC_RX_2_B(self._handle, p1.shape[0], p1.shape[1],
00599                                                 <double*> res.data,
00600                                                 <double*> p1.data,
00601                                                 <double*> p2.data, &error_status);
00602         elif nargs == 3:
00603             camns_interface.IMAS_AMNS_CC_RX_2_C(self._handle, p1.shape[0], p1.shape[1],
00604                                                 <double*> res.data,
00605                                                 <double*> p1.data,
00606                                                 <double*> p2.data,
00607                                                 <double*> p3.data, &error_status);
00608         elif nargs == 4:
00609             camns_interface.IMAS_AMNS_CC_RX_2_D(self._handle, p1.shape[0], p1.shape[1],
00610                                                 <double*> res.data,
00611                                                 <double*> p1.data,
00612                                                 <double*> p2.data,
00613                                                 <double*> p3.data,
00614                                                 <double*> p4.data, &error_status);
00615 
00616         if error_status.flag:
00617             raise AmnsException(error_status.string.decode('UTF-8'))
00618         return res
00619 
00620     def _data_3d(self, int nargs,
00621                       np.ndarray[np.double_t, ndim=3] p1,
00622                       np.ndarray[np.double_t, ndim=3] p2 = None,
00623                       np.ndarray[np.double_t, ndim=3] p3 = None,
00624                       np.ndarray[np.double_t, ndim=3] p4 = None):
00625         cdef camns_interface.amns_c_error_type error_status
00626         cdef np.ndarray[np.double_t, ndim=3] res
00627 
00628         res = np.empty_like(p1, order='F')
00629 
00630         if nargs == 1:
00631             camns_interface.IMAS_AMNS_CC_RX_3_A(self._handle, p1.shape[0], p1.shape[1], p1.shape[2],
00632                                                 <double*> res.data,
```

```
00633                                                 <double*> p1.data, &error_status);
00634         elif nargs == 2:
00635             camns_interface.IMAS_AMNS_CC_RX_3_B(self._handle, p1.shape[0], p1.shape[1], p1.shape[2],
00636                                                 <double*> res.data,
00637                                                 <double*> p1.data,
00638                                                 <double*> p2.data, &error_status);
00639         elif nargs == 3:
00640             camns_interface.IMAS_AMNS_CC_RX_3_C(self._handle, p1.shape[0], p1.shape[1], p1.shape[2],
00641                                                 <double*> res.data,
00642                                                 <double*> p1.data,
00643                                                 <double*> p2.data,
00644                                                 <double*> p3.data, &error_status);
00645         elif nargs == 4:
00646             camns_interface.IMAS_AMNS_CC_RX_3_D(self._handle, p1.shape[0], p1.shape[1], p1.shape[2],
00647                                                 <double*> res.data,
00648                                                 <double*> p1.data,
00649                                                 <double*> p2.data,
00650                                                 <double*> p3.data,
00651                                                 <double*> p4.data, &error_status);
00652
00653         if error_status.flag:
00654             raise AmnsException(error_status.string.decode('UTF-8'))
00655         return res
00656
00657
00658 cdef class Reactants:
00659 #     """Class providing the machanism for indicating reactants and products"""
00660     cdef void* _handle
00661     cdef _reactants
00662
00663     def __cinit__(self):
00664         pass
00665
00666     def __init__(self):
00667         """Use to initialize the Reactants class
00668         in:
00669                 None
00670         out:
00671                 instance of the Reactants class
00672         """
00673
00674         self._handle = NULL
00675         self._reactants= []
00676         pass
00677
00678     cdef void* handle(self):
00679         cdef camns_interface.amns_c_reactant_type reactant
00680         if self._handle == NULL:
00681             camns_interface.IMAS_AMNS_CC_SETUP_REACTANTS(&self._handle, "", 0, len(self._reactants));
00682             for i, reactant in enumerate(self._reactants):
00683                 camns_interface.IMAS_AMNS_CC_SET_REACTANT(self._handle, i+1, &reactant);
00684         return self._handle
00685
00686     def _invalidate_handle(self):
00687         if self._handle != NULL:
00688             camns_interface.IMAS_AMNS_CC_FINISH_REACTANTS(&self._handle)
00689             self._handle = NULL
00690
00691     def add(self, zn, za, mi, lr=None, real_specifier=None, int_specifier=None):
00692         """Add to the list of reactants/products
00693         in:
00694                 zn: nuclear charge (float, required)
00695                 za: ion charge (float, required)
00696                 mi: nuclear mass (float, required)
00697                 lr: if 0, reactant; if 1, product (integer, default None)
00698                 real_specifier: optional additional real (float, default None)
00699                 int_specifier: optional additional integer (integer, default None)
00700         out:
00701                 None
00702         """
00703         cdef camns_interface.amns_c_reactant_type r
00704         r = camns_interface.get_default_amns_c_reactant_type()
00705         self._invalidate_handle()
00706         r.ZN = zn
00707         r.ZA = za
00708         r.MI = mi
00709         if lr: r.LR = lr
00710         if real_specifier: r.real_specifier = real_specifier
00711         if int_specifier: r.int_specifier = int_specifier
00712         self._reactants.append(r)
00713
00714     def __len__(self):
00715         """Return the number of reactants/products"""
00716         return len(self._reactants)
00717
00718     def __str__(self):
00719         """Return a string version of all the reactants/products"""
```

```
00720         return str(self._reactants)
00721
00722     def test(self):
00723         cdef void* handle
00724         handle = self.handle()
00725
00726     def value(self):
00727         """Return all of the reactants/products
00728         in:
00729                 None
00730         out:
00731                 current list of reactants
00732         """
00733         return self._reactants
00734
00735
00736 class AmnsException(Exception):
00737     pass
```

16.125 src/py/amns/camns_interface.pxd File Reference

Namespaces

- camns_interface

16.126 camns_interface.pxd

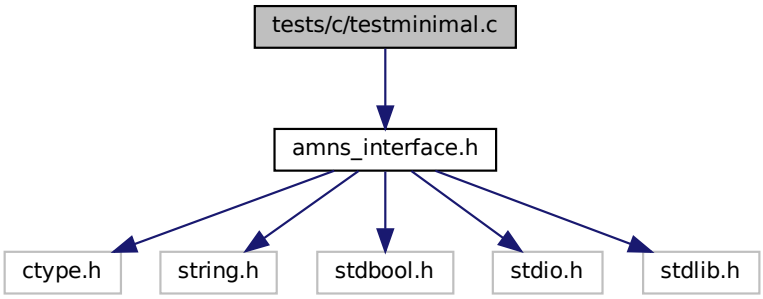
```
00001 # cython: language_level=3
00002 from libcpp cimport bool
00003
00004 cdef extern from "amns_interface.h":
00005
00006     ctypedef struct amns_c_version_type:
00007         char *string
00008         int number
00009         char *backend
00010         char *user
00011
00012     amns_c_version_type get_default_amns_c_version_type()
00013
00014     ctypedef struct amns_c_reactant_type:
00015         double ZN, ZA, MI
00016         int LR
00017         double real_specifier
00018         int int_specifier
00019
00020     amns_c_reactant_type get_default_amns_c_reactant_type()
00021
00022     ctypedef struct amns_c_error_type:
00023         bool flag
00024         char *string
00025
00026     amns_c_error_type get_default_amns_c_error_type()
00027
00028     ctypedef struct  amns_c_reaction_type:
00029         char *string
00030         int isotope_resolved
00031
00032     amns_c_reaction_type get_default_amns_c_reaction_type()
00033
00034     ctypedef struct amns_c_set_type:
00035         char *string
00036
00037     ctypedef struct amns_c_query_type:
00038         char *string
00039
00040     ctypedef struct  amns_c_answer_type:
00041         char *string
00042         int number
00043
00044     void IMAS_AMNS_CC_SETUP(void **handle_out, amns_c_error_type *error_status)
00045     void IMAS_AMNS_CC_SETUP_VERSION(void **handle_in, amns_c_version_type *version, amns_c_error_type
    *error_status)
00046     void IMAS_AMNS_CC_FINISH(void **handle_inout, amns_c_error_type *error_status)
00047     void IMAS_AMNS_CC_FINISH_TABLE(void **handle_rx_inout, amns_c_error_type *error_status)
00048     void IMAS_AMNS_CC_SET(void *handle_in, amns_c_set_type *set, amns_c_error_type *error_status)
00049     void IMAS_AMNS_CC_QUERY(void *handle_in, amns_c_query_type *query, amns_c_answer_type *answer,
    amns_c_error_type *error_status)
00050     void IMAS_AMNS_CC_SETUP_TABLE(void *handle_in, amns_c_reaction_type *reaction_type, void
    *reactant_handle_in, void **handle_rx_out, amns_c_error_type *error_status)
```

```
00051     void IMAS_AMNS_CC_QUERY_TABLE(void *handle_rx_in, amns_c_query_type *query, amns_c_answer_type
      *answer, amns_c_error_type *error_status)
00052     void IMAS_AMNS_CC_SET_TABLE(void *handle_rx_in, amns_c_set_type *set, amns_c_error_type
      *error_status)
00053
00054     # ...
00055
00056     void IMAS_AMNS_CC_RX_0_A(void *handle_rx_in, double *out, double arg1, amns_c_error_type
      *error_status)
00057     void IMAS_AMNS_CC_RX_0_B(void *handle_rx_in, double *out, double arg1, double arg2,
      amns_c_error_type *error_status)
00058     void IMAS_AMNS_CC_RX_0_C(void *handle_rx_in, double *out, double arg1, double arg2, double arg3,
      amns_c_error_type *error_status)
00059     void IMAS_AMNS_CC_RX_0_D(void *handle_rx_in, double *out, double arg1, double arg2, double arg3,
      double arg4, amns_c_error_type *error_status)
00060     void IMAS_AMNS_CC_RX_1_A(void *handle_rx_in, int nx, double *out, double *arg1, amns_c_error_type
      *error_status)
00061     void IMAS_AMNS_CC_RX_1_B(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2,
      amns_c_error_type *error_status)
00062     void IMAS_AMNS_CC_RX_1_C(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2,
      double *arg3, amns_c_error_type *error_status)
00063     void IMAS_AMNS_CC_RX_1_D(void *handle_rx_in, int nx, double *out, double *arg1, double *arg2,
      double *arg3, double *arg4, amns_c_error_type *error_status)
00064     void IMAS_AMNS_CC_RX_2_A(void *handle_rx_in, int nx, int ny, double *out, double *arg1,
      amns_c_error_type *error_status)
00065     void IMAS_AMNS_CC_RX_2_B(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double
      *arg2, amns_c_error_type *error_status)
00066     void IMAS_AMNS_CC_RX_2_C(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double
      *arg2, double *arg3, amns_c_error_type *error_status)
00067     void IMAS_AMNS_CC_RX_2_D(void *handle_rx_in, int nx, int ny, double *out, double *arg1, double
      *arg2, double *arg3, double *arg4, amns_c_error_type *error_status)
00068     void IMAS_AMNS_CC_RX_3_A(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1,
      amns_c_error_type *error_status)
00069     void IMAS_AMNS_CC_RX_3_B(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1,
      double *arg2, amns_c_error_type *error_status)
00070     void IMAS_AMNS_CC_RX_3_C(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1,
      double *arg2, double *arg3, amns_c_error_type *error_status)
00071     void IMAS_AMNS_CC_RX_3_D(void *handle_rx_in, int nx, int ny, int nz, double *out, double *arg1,
      double *arg2, double *arg3, double *arg4, amns_c_error_type *error_status)
00072
00073     # ...
00074
00075     void IMAS_AMNS_CC_SETUP_REACTANTS(void **reactants_handle_out, char *string_in, int index_in, int
      n_reactants)
00076     void IMAS_AMNS_CC_SET_REACTANT(void *reactants_handle_in, int reactant_index, amns_c_reactant_type
      *reactant_in)
00077     void IMAS_AMNS_CC_GET_REACTANT(void *reactants_handle_in, int reactant_index, amns_c_reactant_type
      *reactant_out)
00078     void IMAS_AMNS_CC_FINISH_REACTANTS(void **reactants_handle_inout)
00079
```

16.127 tests/c/testminimal.c File Reference

`#include "amns_interface.h"`

Include dependency graph for testminimal.c:



Functions

- int main (int argc, char ∗argv[ ])

16.127.1 Function Documentation

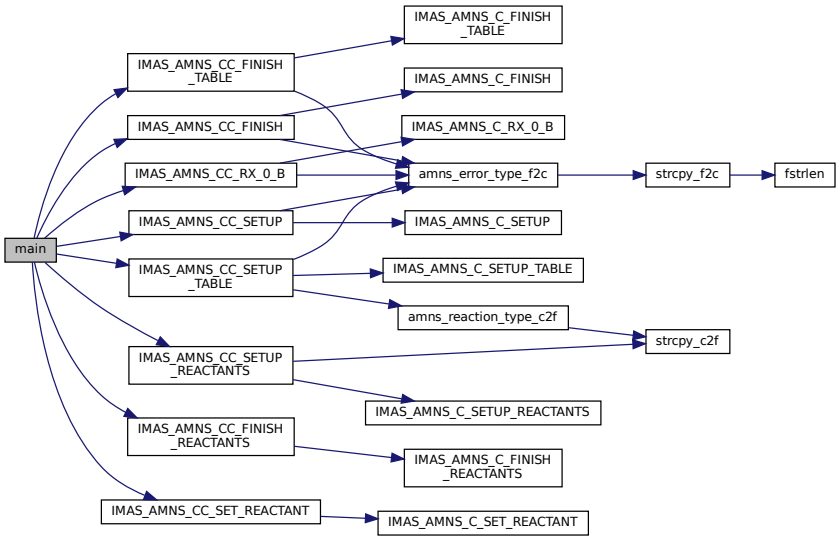
16.127.1.1 main()

```
int main (
            int argc,
            char * argv[ ] )
```

Definition at line 10 of file testminimal.c.

```
00011 {
00012   void* amns_handle = NULL;
00013   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00014   void* reactants_handle = NULL;
00015   amns_c_reactant_type species1  = {.ZN=6, .ZA=1, .MI=12, .LR=0};
00016   amns_c_reactant_type species2  = {.ZN=1, .ZA=0, .MI=2 , .LR=0};
00017   amns_c_reactant_type species3  = {.ZN=6, .ZA=0, .MI=12, .LR=1};
00018   amns_c_reactant_type species4  = {.ZN=1, .ZA=1, .MI=2 , .LR=1};
00019   amns_c_reaction_type xx_rx  = {.string = "CX"};
00020   void* amns_cx_handle;
00021   double rate;
00022
00023   IMAS_AMNS_CC_SETUP(&amns_handle, &error_stat);
00024   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00025   IMAS_AMNS_CC_SETUP_REACTANTS(&reactants_handle, "", 0, 4);
00026   IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 1, &species1);
00027   IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 2, &species2);
00028   IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 3, &species3);
00029   IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 4, &species4);
00030   IMAS_AMNS_CC_SETUP_TABLE(amns_handle, &xx_rx, reactants_handle, &amns_cx_handle, &error_stat);
00031   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00032   IMAS_AMNS_CC_RX_0_B(amns_cx_handle, &rate, 100.0, 1e20, &error_stat);
00033   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00034   printf("Rate = %25.15e\n", rate);
00035   IMAS_AMNS_CC_FINISH_TABLE(&amns_cx_handle, &error_stat);
00036   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00037   IMAS_AMNS_CC_FINISH_REACTANTS(&reactants_handle);
00038   IMAS_AMNS_CC_FINISH(&amns_handle, &error_stat);
00039   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00040   return 0;
00041 }
```

Here is the call graph for this function:



16.128 testminimal.c

```
00001
00008 #include "amns_interface.h"
00009
00010 int main(int argc, char *argv[])
00011 {
00012   void* amns_handle = NULL;
00013   amns_c_error_type error_stat = DEFAULT_AMNS_C_ERROR_TYPE;
00014   void* reactants_handle = NULL;
00015   amns_c_reactant_type species1  = {.ZN=6, .ZA=1, .MI=12, .LR=0};
00016   amns_c_reactant_type species2  = {.ZN=1, .ZA=0, .MI=2 , .LR=0};
00017   amns_c_reactant_type species3  = {.ZN=6, .ZA=0, .MI=12, .LR=1};
00018   amns_c_reactant_type species4  = {.ZN=1, .ZA=1, .MI=2 , .LR=1};
00019   amns_c_reaction_type xx_rx  = {.string = "CX"};
00020   void* amns_cx_handle;
00021   double rate;
00022
00023   IMAS_AMNS_CC_SETUP(&amns_handle, &error_stat);
00024   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00025   IMAS_AMNS_CC_SETUP_REACTANTS(&reactants_handle, "", 0, 4);
00026   IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 1, &species1);
00027   IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 2, &species2);
00028   IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 3, &species3);
00029   IMAS_AMNS_CC_SET_REACTANT(reactants_handle, 4, &species4);
00030   IMAS_AMNS_CC_SETUP_TABLE(amns_handle, &xx_rx, reactants_handle, &amns_cx_handle, &error_stat);
00031   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00032   IMAS_AMNS_CC_RX_0_B(amns_cx_handle, &rate, 100.0, 1e20, &error_stat);
00033   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00034   printf("Rate = %25.15e\n", rate);
00035   IMAS_AMNS_CC_FINISH_TABLE(&amns_cx_handle, &error_stat);
00036   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00037   IMAS_AMNS_CC_FINISH_REACTANTS(&reactants_handle);
00038   IMAS_AMNS_CC_FINISH(&amns_handle, &error_stat);
00039   printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
00040   return 0;
00041 }
```

16.129 tests/f90/testminimal.f90 File Reference

Functions/Subroutines

- program minimal

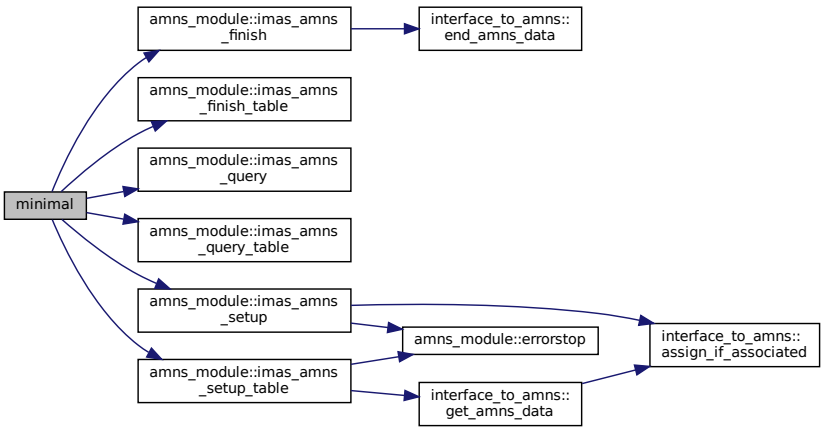
16.129.1 Function/Subroutine Documentation

16.129.1.1 minimal()

`program minimal`

Definition at line 7 of file testminimal.f90.

Here is the call graph for this function:



16.130 testminimal.f90

```
00001
00007 program minimal
00008   use ids_types  ! IGNORE
00009   use amns_types  ! IGNORE
00010   use amns_module  ! IGNORE
00011
00012   implicit none
00013
00014   type (amns_handle_type) :: amns                              ! AMNS global handle
00015   type (amns_handle_rx_type)  :: amns_rx                       ! AMNS table handle
00016   type (amns_reaction_type) :: xx_rx
00017   type (amns_reactants_type) :: species
00018   type (amns_answer_type) :: answer
00019   real (kind=ids_real) :: te=100.0_ids_real, ne=1e20_ids_real, rate
00020
00021 ! set up the AMNS system  -- here we force to version 1
00022   call imas_amns_setup(amns, version=amns_version_type(", 1, ", "))
00023 ! query to find when the index file was created
00024   call imas_amns_query(amns, amns_query_type("prop_creation"), answer)
00025   write(*,*) 'Database created ', trim(answer%string)
00026 ! set up reactants/products
00027   allocate(species%components(4))
00028   species%components = &
00029        (/ amns_reactant_type(6, 1, 12, 0), amns_reactant_type(1, 0, 2, 0), &
00030           amns_reactant_type(6, 0, 12, 1), amns_reactant_type(1, 1, 2, 1) /)
00031 ! set up reaction
00032   xx_rx%string='CX'
00033 ! set up table
00034   call imas_amns_setup_table(amns, xx_rx, species, amns_rx)
00035 ! query to find information about the table
00036   call imas_amns_query_table(amns_rx, amns_query_type("prop_creation"), answer)
00037   write(*,*) 'Database created ', trim(answer%string)
00038   call imas_amns_query_table(amns_rx, amns_query_type("citation"), answer)
00039   write(*,*) 'Citation ', trim(answer%string)
00040 ! get results
00041   call imas_amns_rx(amns_rx, rate, te, ne)
00042   write(*,*) 'Rate = ', rate
00043 ! finish with table
00044   call imas_amns_finish_table(amns_rx)
00045 ! finish with amns
00046   call imas_amns_finish(amns)
00047
00048 end program minimal
```

16.131 tests/java/AmnsMinimal.java File Reference

Classes

- class AmnsMinimal

16.132 AmnsMinimal.java

```
00001 import amns.*;
00002 import amns.type.*;
00003
00010 public class AmnsMinimal {
00011
00013     public static void main(String[] args) {
00014
00015         // this is a pointer from C
00016         long ptrAmnsHandle = 0;
00017         AmnsErrorType error_stat = new AmnsErrorType();
00018
00019         // this is a pointer from C
00020         long ptrReactantsHandle = 0;
00021
00022         // Definition of species
00023         AmnsReactantType species1 = new AmnsReactantType();
00024         species1.ZN = 6;
00025         species1.ZA = 1;
00026         species1.MI = 12;
00027         species1.LR = 0;
00028
00029         AmnsReactantType species2 = new AmnsReactantType();
00030         species2.ZN = 1;
00031         species2.ZA = 0;
00032         species2.MI = 2;
00033         species2.LR = 0;
00034
00035         AmnsReactantType species3 = new AmnsReactantType();
00036         species3.ZN = 6;
00037         species3.ZA = 0;
00038         species3.MI = 12;
00039         species3.LR = 1;
00040
00041         AmnsReactantType species4 = new AmnsReactantType();
00042         species4.ZN = 1;
00043         species4.ZA = 1;
00044         species4.MI = 2;
00045         species4.LR = 1;
00046
00047         long ptrCXHandle = 0;
00048         AmnsReactionType reactionType = new AmnsReactionType();
00049         reactionType.string = "CX";
00050         reactionType.isotopeResolved = 0;
00051
00052         double rate = 0;
00053
00054         // Class to manage Amns calls to low level
00055         Amns amns = new Amns();
00056
00057         // Setup
00058         String imas_amns_debug = System.getenv("IMAS_AMNS_JAVA_DEBUG");
00059         Boolean amns_debug = (imas_amns_debug != null && !imas_amns_debug.trim().isEmpty() &&
    !imas_amns_debug.equals("no") && !imas_amns_debug.equals("NO") );
00060
00061         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP");
00062         ptrAmnsHandle = amns.ImasAmnsCCSetup(error_stat);
00063         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " +
    ptrAmnsHandle);
00064         if (amns_debug) System.err.println("[JVM] Error.flag= " + error_stat.flag + " Error.string= "
    + error_stat.string);
00065
00066         int idx = 0;
00067         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP_REACTANTS");
00068         ptrReactantsHandle = amns.ImasAmnsCcSetupReactantsNumber(idx, 4);
00069         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
    ptrReactantsHandle);
00070
00071         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species1");
00072         amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 1, species1);
00073
00074         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species2");
00075         amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 2, species2);
00076
00077         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species3");
00078         amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 3, species3);
00079
00080         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SET_REACTANT - species4");
00081         amns.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 4, species4);
00082
00083         // we are calling now SETUP_TABLE using all the elements above
00084         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_SETUP_TABLE");
00085         ptrCXHandle = amns.ImasAmnsCCSetupTable(ptrAmnsHandle, reactionType
00086                                                 , ptrReactantsHandle
00087                                                 , error_stat);
00088         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
```

```
00089
00090         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_RX_0_B");
00091         rate = amns.ImasAmnsCCRX0B( ptrCXHandle, 100, 1.0e20, error_stat);
00092         System.out.println("Value of rate: " + rate);
00093
00094         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_TABLE");
00095         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
00096         ptrCXHandle =amns.ImasAmnsCCFinishTable( ptrCXHandle, error_stat);
00097         if (amns_debug) System.err.println("[JVM] After IMAS_AMNS_CC_FINISH_TABLE");
00098         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrCXHandle): " + ptrCXHandle);
00099
00100         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_REACTANTS");
00101         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      ptrReactantsHandle);
00102         ptrReactantsHandle = amns.ImasAmnsCCFinishReactants(ptrReactantsHandle);
00103         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH_REACTANTS");
00104         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrReactantsHandle): " +
      ptrReactantsHandle);
00105
00106         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH");
00107         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " +
      ptrAmnsHandle);
00108         ptrAmnsHandle = amns.ImasAmnsCCFinish(ptrAmnsHandle, error_stat);
00109         if (amns_debug) System.err.println("[JVM] Calling IMAS_AMNS_CC_FINISH");
00110         if (amns_debug) System.err.println("[JVM] Value of pointer (ptrAmnsHandle): " +
      ptrAmnsHandle);
00111     }
00112
00113 }
```

16.133 tests/matlab/javaclasspath.txt File Reference

16.134 tests/matlab/javalibrarypath.txt File Reference

16.135 tests/matlab/testminimal.m File Reference

Functions

- Make AMNS available in Matlab Import Java classes (interface to AMNS) import amns.type.AmnsAnswerType import amns.type.AmnsErrorType import amns.type.AmnsQueryType import amns.type.AmnsReactantType import amns.type.AmnsReactionType import amns.type.AmnsSetType import amns.Amns % minimal example in Matlab % class to manage Amns calls to low level amnsjava
- if (amns_error.flag ~=0) fprintf('Error = %s\n'
- amnsjava ImasAmnsCCQuery (ptrAmnsHandle, amns_query, amns_answer, amns_error) if(amns_error.flag ~=0) fprintf('Error
- amnsjava amns_error string end fprintf ('Version %i\n', amns_answer.number) % setup the reactants and products species
- D0 amnsjava ImasAmnsCCSetReactantIdx (ptrReactantsHandle, 2, species)
- C0 amnsjava ImasAmnsCCSetReactantIdx (ptrReactantsHandle, 3, species)
- amnsjava ImasAmnsCCQueryTable (ptrCXHandle, amns_query, amns_answer, amns_error) if(amns_↩ error.flag ~=0) fprintf('Error
- amnsjava amns_error string end fprintf ('Reactants/Products=%s\n', amns_answer.string) % calculate the rate at(100
- amns_error string end fprintf ('Rate=%.15e\n', rate) % close down ptrCXHandle

Variables

- variables for querying and setting amns_error = amns.type.AmnsErrorType
- amns_answer = amns.type.AmnsAnswerType
- amns_query = amns.type.AmnsQueryType
- amns_set = amns.type.AmnsSetType
- setup AMNS system ptrAmnsHandle = amnsjava.ImasAmnsCCSetup(amns_error)
- amns_error string end amns_query string = "version"
- idx = 0
- ptrReactantsHandle = amnsjava.ImasAmnsCcSetupReactantsNumber(idx, 4)

- species ZN =double(6)
- species ZA =double(1)
- species MI =double(12)
- species LR =0
- identify the reaction reactionType =amns.type.AmnsReactionType
- reactionType isotopeResolved =0
- Setup the table ptrCXHandle = amnsjava.ImasAmnsCCSetupTable(ptrAmnsHandle, reactionType, ptrReactantsHandle, amns_error)
- amnsjava amns_error string end rate = amnsjava.ImasAmnsCCRX0B(ptrCXHandle,100,1.0e20,amns_error)

16.135.1 Function Documentation

16.135.1.1 classes()

```
Make AMNS available in Matlab Import Java classes (
             interface to AMNS )
```

16.135.1.2 fprintf() [1/3]

```
amns_error string end fprintf (
             ' Rate = %.15e\n',
             rate  )
```

16.135.1.3 fprintf() [2/3]

```
amnsjava amns_error string end fprintf (
             'Reactants/ Products = %s\n',
             amns_answer.  string )
```

16.135.1.4 fprintf() [3/3]

```
amnsjava amns_error string end fprintf (
             'Version %i\n' ,
             amns_answer.  number )
```

16.135.1.5 if()

```
if (
             amns_error.flag ~  = 0 ) = %s\n'
```

16.135.1.6 ImasAmnsCCQuery()

```
amnsjava ImasAmnsCCQuery (
             ptrAmnsHandle ,
             amns_query ,
             amns_answer ,
             amns_error  )  [pure virtual]
```

16.135.1.7 ImasAmnsCCQueryTable()

```
amnsjava ImasAmnsCCQueryTable (
            ptrCXHandle ,
            amns_query ,
            amns_answer ,
            amns_error  )  [pure virtual]
```

16.135.1.8 ImasAmnsCCSetReactantIdx() [1/2]

```
D0 amnsjava ImasAmnsCCSetReactantIdx (
            ptrReactantsHandle ,
            2 ,
            species  )
```

16.135.1.9 ImasAmnsCCSetReactantIdx() [2/2]

```
C0 amnsjava ImasAmnsCCSetReactantIdx (
            ptrReactantsHandle ,
            3 ,
            species  )
```

16.135.2 Variable Documentation

16.135.2.1 amns_answer

```
amns_answer = amns.type.AmnsAnswerType
```
Definition at line 22 of file testminimal.m.

16.135.2.2 amns_error

```
variables for querying and setting amns_error = amns.type.AmnsErrorType
```
Definition at line 21 of file testminimal.m.

16.135.2.3 amns_query

```
amns_query = amns.type.AmnsQueryType
```
Definition at line 23 of file testminimal.m.

16.135.2.4 amns_set

```
amns_set = amns.type.AmnsSetType
```
Definition at line 24 of file testminimal.m.

16.135.2.5 idx

```
idx = 0
```
Definition at line 43 of file testminimal.m.

16.135.2.6 isotopeResolved

```
reactionType isotopeResolved =0
```
Definition at line 58 of file testminimal.m.

16.135.2.7 LR

```
species4 LR =0
```
Definition at line 45 of file testminimal.m.

16.135.2.8 MI

```
species MI =double(12)
```
Definition at line 45 of file testminimal.m.

16.135.2.9 ptrAmnsHandle

```
amns_error string end ptrAmnsHandle = amnsjava.ImasAmnsCCSetup(amns_error)
```
Definition at line 28 of file testminimal.m.

16.135.2.10 ptrCXHandle

```
Setup the table ptrCXHandle = amnsjava.ImasAmnsCCSetupTable(ptrAmnsHandle, reactionType, ptrReactantsHandle,
amns_error)
```
Definition at line 62 of file testminimal.m.

16.135.2.11 ptrReactantsHandle

```
amns_error string end ptrReactantsHandle = amnsjava.ImasAmnsCcSetupReactantsNumber(idx, 4)
```
Definition at line 44 of file testminimal.m.

16.135.2.12 rate

```
amnsjava amns_error string end rate = amnsjava.ImasAmnsCCRX0B(ptrCXHandle,100,1.0e20,amns_error)
```
Definition at line 76 of file testminimal.m.

16.135.2.13 reactionType

```
identify the reaction reactionType =amns.type.AmnsReactionType
```
Definition at line 56 of file testminimal.m.

16.135.2.14 string

```
amns_error string end amns_query string = "version"
```
Definition at line 33 of file testminimal.m.

16.135.2.15 ZA

```
species ZA =double(1)
```
Definition at line 45 of file testminimal.m.

16.135.2.16 ZN

```
species ZN =double(6)
```

Definition at line 45 of file testminimal.m.

16.136 testminimal.m

```
00001 % Make AMNS available in Matlab
00002
00003 % Import Java classes (interface to AMNS)
00004
00005 import amns.type.AmnsAnswerType
00006 import amns.type.AmnsErrorType
00007 import amns.type.AmnsQueryType
00008 import amns.type.AmnsReactantType
00009 import amns.type.AmnsReactionType
00010 import amns.type.AmnsSetType
00011 import amns.Amns
00012
00013 % minimal example in Matlab
00014
00015 % class to manage Amns calls to low level
00016
00017 amnsjava=amns.Amns;
00018
00019 % variables for querying and setting
00020
00021 amns_error   = amns.type.AmnsErrorType;
00022 amns_answer  = amns.type.AmnsAnswerType;
00023 amns_query   = amns.type.AmnsQueryType;
00024 amns_set     = amns.type.AmnsSetType;
00025
00026 % setup AMNS system
00027
00028 ptrAmnsHandle = amnsjava.ImasAmnsCCSetup(amns_error);
00029 if (amns_error.flag ~= 0)
00030   fprintf ('Error = %s\n' , amns_error.string)
00031 end
00032
00033 amns_query.string = "version";
00034 amnsjava.ImasAmnsCCQuery(ptrAmnsHandle, amns_query, amns_answer, amns_error)
00035 if (amns_error.flag ~= 0)
00036   fprintf ('Error = %s\n' , amns_error.string)
00037 end
00038 fprintf ('Version %i\n', amns_answer.number)
00039
00040 % setup the reactants and products
00041
00042 species = amns.type.AmnsReactantType;
00043 idx = 0;
00044 ptrReactantsHandle = amnsjava.ImasAmnsCcSetupReactantsNumber(idx, 4);
00045 species.ZN=double(6);  species.ZA=double(1);  species.MI=double(12);  species.LR=0;  % C1+
00046 amnsjava.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 1, species);
00047 species.ZN=double(1);  species.ZA=double(0);  species.MI=double(2);   species.LR=0;  % D0
00048 amnsjava.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 2, species);
00049 species.ZN=double(6);  species.ZA=double(0);  species.MI=double(12);  species.LR=1;  % C0
00050 amnsjava.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 3, species);
00051 species.ZN=double(1);  species.ZA=double(1);  species.MI=double(2);   species4.LR=1; % D1+
00052 amnsjava.ImasAmnsCCSetReactantIdx(ptrReactantsHandle, 4, species);
00053
00054 % identify the reaction
00055
00056 reactionType=amns.type.AmnsReactionType;
00057 reactionType.string='CX';
00058 reactionType.isotopeResolved=0;
00059
00060 % Setup the table
00061
00062 ptrCXHandle = amnsjava.ImasAmnsCCSetupTable(ptrAmnsHandle, reactionType, ptrReactantsHandle,
       amns_error);
00063 if (amns_error.flag ~= 0)
00064   fprintf ('Error = %s\n' , amns_error.string)
00065 end
00066
00067 amns_query.string = "reactants";
00068 amnsjava.ImasAmnsCCQueryTable(ptrCXHandle, amns_query, amns_answer, amns_error)
00069 if (amns_error.flag ~= 0)
00070   fprintf ('Error = %s\n' , amns_error.string)
00071 end
00072 fprintf ('Reactants/Products = %s\n', amns_answer.string)
00073
00074 % calculate the rate at (100, 1e20)
00075
00076 rate = amnsjava.ImasAmnsCCRX0B(ptrCXHandle,100,1.0e20,amns_error);
```

```
00077 if (amns_error.flag ~= 0)
00078   fprintf ('Error = %s\n' , amns_error.string)
00079 end
00080 fprintf ('Rate = %.15e\n' , rate)
00081
00082 % close down
00083
00084 ptrCXHandle =amnsjava.ImasAmnsCCFinishTable( ptrCXHandle, amns_error);
00085 if (amns_error.flag ~= 0)
00086   fprintf ('Error = %s\n' , amns_error.string)
00087 end
00088 ptrReactantsHandle = amnsjava.ImasAmnsCCFinishReactants(ptrReactantsHandle);
00089 if (amns_error.flag ~= 0)
00090   fprintf ('Error = %s\n' , amns_error.string)
00091 end
00092 ptrAmnsHandle = amnsjava.ImasAmnsCCFinish(ptrAmnsHandle, amns_error);
00093 if (amns_error.flag ~= 0)
00094   fprintf ('Error = %s\n' , amns_error.string)
00095 end
00096
```

16.137 tests/py/testminimal.py File Reference

Namespaces

- testminimal

Variables

- testminimal.amnsdb = amns.Amns()
- testminimal.r = amns.Reactants()
- testminimal.lr
- testminimal.table = amnsdb.get_table(b"CX", r)
- testminimal.dat = table.data(np.array([100.0]), np.array([1e20]))

16.138 testminimal.py

```
00001 #! /usr/bin/env python
00002 import amns
00003 import numpy as np
00004
00005 amnsdb = amns.Amns()
00006 r = amns.Reactants()
00007 r.add(6,1,12)
00008 r.add(1,0,2)
00009 r.add(6,0,12,lr=1)
00010 r.add(1,1,2,lr=1)
00011
00012 table = amnsdb.get_table(b"CX", r) # fix error: expected bytes, found str
00013 print("table.no_of_reactants", table.no_of_reactants)
00014 dat = table.data(np.array([100.0]), np.array([1e20]))
00015 print("Rate = \t%.15e" % (dat[0]))
00016 amnsdb.finalize()
```

16.139 verification/amns_verify.py File Reference

Namespaces

- amns_verify

Functions

- def amns_verify.plot (x1, y1, x2, y2, xlabel, ylabel, title, file=None, line1='bo-', line2='r+-')
- def amns_verify.adas (zn, za, mi, reac, ref, file=None, texfile=None)
- def amns_verify.nuclear_HB (r1, r2, p1, p2, reac, ref, file=None, texfile=None)
- def amns_verify.nuclear_HB_tt (r1, r2, p1, p2, reac, ref, file=None, texfile=None)
- def amns_verify.nuclear_HB_bt (r1, r2, p1, p2, reac, ref, file=None, texfile=None)
- def amns_verify.total_cross_section_EL (zn, za, mi, reac, ref, file=None)

- def amns_verify.differential_cross_section_EL (zn, za, mi, reac, ref, file=None)
- def amns_verify.rct (zn, za, mi, reac, ref, file=None)
- def amns_verify.sputter (zn_w, za_w, mi_w, zn_p, za_p, mi_p, reac, ref, file=None)
- def amns_verify.reflect (zn_w, za_w, mi_w, zn_p, za_p, mi_p, reac, ref, file=None)

Variables

- amns_verify.amnsdb = amns.Amns()
- amns_verify.texfile = open('amns_verify.tex', 'w')
- amns_verify.file

16.140 amns_verify.py

```
00001 #! /usr/bin/env python
00002 import os
00003 if not os.environ.get('DISPLAY'):
00004     # we can still plot, even without having a DISPLAY
00005     import matplotlib as mpl
00006     mpl.use('Agg')
00007
00008 import matplotlib.pyplot as plt
00009 from matplotlib import colors, ticker
00010 import amns
00011 import numpy as np
00012 import math as m
00013
00014 def plot(x1,y1,x2,y2,xlabel,ylabel,title,file=None,line1='bo-',line2='r+-'):
00015
00016     height=210/25.4
00017     width=297/25.4
00018     plt.figure(1, figsize=(width,height))
00019
00020     plt.clf()
00021     try:
00022         plt.loglog(x1, y1, line1, label='AMNS') # ValueError: Data has no positive values, and
       therefore can not be log-scaled.
00023         plt.loglog(x2, y2, line2, label='REF')
00024     except ValueError:
00025         print("Warning: Data has no positive values, not plotting log: %s, %s" % (title ,file))
00026         plt.plot(x1, y1, line1, label='AMNS')
00027         plt.plot(x2, y2, line2, label='REF')
00028     plt.xlabel(xlabel)
00029     plt.ylabel(ylabel)
00030     plt.title(title)
00031     y1_max=m.ceil(m.log10(np.nanmax(y1)))
00032     try:
00033         y1_min=m.floor(m.log10(np.nanmin(y1))) # ValueError: math domain error
00034     except ValueError:
00035         print("Warning: math domain error on (%e): %s, %s" % (np.nanmin(y1), title ,file))
00036         y1_min=m.floor(m.log10(np.nanmin(y1+1e-32)))
00037     y1_min=max(y1_max-20,y1_min)
00038     plt.ylim(10**y1_min,10**y1_max)
00039     plt.legend(loc=0)
00040     if (file is None):
00041         plt.show()
00042     else:
00043         try:
00044             plt.savefig(file, papertype='a4', orientation='landscape')
00045         except:
00046             plt.savefig(file)
00047
00048 def adas(zn, za, mi, reac, ref, file=None, texfile=None):
00049
00050     nx=[101,1]
00051
00052     x = np.loadtxt(ref)
00053     nxr=list(nx) ; nxr.reverse()
00054     te=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00055     ne=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00056
00057     r = amns.Reactants()
00058     if reac in ['RC']:
00059         r.add(zn,za,mi)
00060         r.add(0,-1,0)
00061         r.add(zn,za-1,mi,lr=1)
00062         r.add(0,-1,0,lr=1)
00063     elif reac in ['EI']:
00064         r.add(zn,za,mi)
00065         r.add(0,-1,0)
00066         r.add(zn,za+1,mi,lr=1)
```

```
00067         r.add(0,-1,0,lr=1)
00068     elif reac in ['CX']:
00069         r.add(zn,za,mi)
00070         r.add(1,0,2)
00071         r.add(zn,za-1,mi,lr=1)
00072         r.add(1,1,2,lr=1)
00073     elif reac in ['BR', 'LR', 'ZE', 'ZE2', 'EIP']:
00074         r.add(zn,za,mi)
00075         r.add(zn,za,mi,lr=1)
00076     else:
00077         raise ValueError('Invalid option %s' % (reac))
00078 
00079     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00080     res = table.data(te, ne)
00081 
00082     plot(te[:,0], res[:,0], x[:,0], x[:,2], 'Te', '%s [%s]' % (table.result_label, table.result_unit),
      '%s for ne=%s, ZN=%s, ZA=%s, MI=%s (%s)' % (reac, ne[0,0], zn, za, mi, table.state_label), file)
00083 
00084     max_rel_err = np.max(np.abs((table.data(x[:,0].copy(), x[:,1].copy()) - x[:,2]) / x[:,2]))
00085     print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
00086 
00087     if file:
00088         if texfile:
00089             print('', file=texfile)
00090             print('\\subsection{%s of %s}' %
      (table.result_label.replace('^','\\^').replace('_','\\_'),
      table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
00091             print('', file=texfile)
00092             print('Comparison of AMNS with %s' % (ref.replace('_','\\_'),) , file=texfile)
00093             print('', file=texfile)
00094             print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
      file=texfile)
00095             print('', file=texfile)
00096             print('Maximum relative error for %s = %10.03g' % (ref.replace('_','\\_'), max_rel_err),
      file=texfile)
00097             print('', file=texfile)
00098 
00099 
00100 def nuclear_HB(r1, r2, p1, p2, reac,ref, file=None, texfile=None):
00101 
00102     nx=[101,1]
00103 
00104     x = np.loadtxt(ref)
00105     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
00106 
00107     r = amns.Reactants()
00108     r.add(r1[0],r1[1],r1[2])
00109     r.add(r2[0],r2[1],r2[2])
00110     r.add(p1[0],p1[1],p1[2],lr=1)
00111     r.add(p2[0],p2[1],p2[2],lr=1)
00112     table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
00113     res = table.data(E)
00114 
00115     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s' % (reac,), file)
00116 
00117     max_rel_err = np.max(np.abs((table.data(x[:,0].copy()) - x[:,1]) / x[:,1]))
00118 
00119     print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
00120 
00121     if file:
00122         if texfile:
00123             print('', file=texfile)
00124             print('\\subsection{%s}' % (table.result_label.replace('^','\\^').replace('_','\\_'),),
      file=texfile)
00125             print('', file=texfile)
00126             print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
      file=texfile)
00127             print('', file=texfile)
00128             print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
      file=texfile)
00129             print('', file=texfile)
00130             print('Maximum relative error for %s = %10.03g' %
      (ref.replace('^','\\^').replace('_','\\_'), max_rel_err), file=texfile)
00131             print('', file=texfile)
00132 
00133 
00134 def nuclear_HB_tt(r1, r2, p1, p2, reac, ref, file=None, texfile=None):
00135 
00136     nx=[101,1]
00137 
00138     x = np.loadtxt(ref)
00139     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10).repeat(nx[1]).reshape(nx),order='F')
00140 
00141     height=210/25.4
00142     width=297/25.4
00143     plt.figure(1, figsize=(width,height))
```

```
00144
00145     r = amns.Reactants()
00146     r.add(r1[0],r1[1],r1[2])
00147     r.add(r2[0],r2[1],r2[2])
00148     r.add(p1[0],p1[1],p1[2],lr=1)
00149     r.add(p2[0],p2[1],p2[2],lr=1)
00150     table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
00151     res = table.data(E)
00152
00153     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Ti', '%s [%s]' % (table.result_label, table.result_unit),
      '%s' % (reac,), file)
00154
00155     max_rel_err = np.max(np.abs((table.data(x[:,0].copy()) - x[:,1]) / x[:,1]))
00156
00157     print('Maximum relative error for %s = %10.03g' % (ref, max_rel_err))
00158
00159     if file:
00160         if texfile:
00161             print('', file=texfile)
00162             print('\\subsection{%s of %s}' %
      (table.result_label.replace('^','\\^').replace('_','\\_'),
      table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
00163             print('', file=texfile)
00164             print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
      file=texfile)
00165             print('', file=texfile)
00166             print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
      file=texfile)
00167             print('', file=texfile)
00168             print('Maximum relative error for %s = %10.03g' %
      (ref.replace('^','\\^').replace('_','\\_'), max_rel_err), file=texfile)
00169             print('', file=texfile)
00170
00171
00172 def nuclear_HB_bt(r1, r2, p1, p2, reac, ref, file=None, texfile=None):
00173
00174     nx=[101,1]
00175
00176     x = np.loadtxt(ref)
00177     nxr=list(nx) ; nxr.reverse()
00178     Ti=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00179     E=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00180
00181     r = amns.Reactants()
00182     r.add(r1[0],r1[1],r1[2])
00183     r.add(r2[0],r2[1],r2[2])
00184     r.add(p1[0],p1[1],p1[2],lr=1)
00185     r.add(p2[0],p2[1],p2[2],lr=1)
00186     table = amnsdb.get_table(reac.encode('UTF-8'), r, isotope_resolved=1)
00187     res = table.data(Ti, E)
00188
00189     plot(Ti[:,0], res[:,0], x[:,0], x[:,2], 'Ti', '%s [%s]' % (table.result_label, table.result_unit),
      '%s for E=%s' % (reac, E[0,0]), file)
00190
00191     max_rel_err = np.max(np.abs((table.data(x[:,0].copy(), x[:,1].copy()) - x[:,2]) / x[:,2]))
00192
00193     if file:
00194         if texfile:
00195             print('', file=texfile)
00196             print('\\subsection{%s of %s}' %
      (table.result_label.replace('^','\\^').replace('_','\\_'),
      table.state_label.replace('^','\\^').replace('_','\\_')), file=texfile)
00197             print('', file=texfile)
00198             print('Comparison of AMNS with %s' % (ref.replace('^','\\^').replace('_','\\_'),) ,
      file=texfile)
00199             print('', file=texfile)
00200             print('\\centerline{\\includegraphics[origin=br,width=0.9\\columnwidth]{%s}}' % (file,),
      file=texfile)
00201             print('', file=texfile)
00202             print('Maximum relative error for %s = %10.03g' %
      (ref.replace('^','\\^').replace('_','\\_'), max_rel_err), file=texfile)
00203             print('', file=texfile)
00204
00205
00206 def total_cross_section_EL(zn,za,mi,reac,ref,file=None):
00207
00208     nx=[101,1]
00209
00210     x = np.loadtxt(ref)
00211     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00212
00213     r = amns.Reactants()
00214     r.add(zn,za,mi)
00215     r.add(zn,za,mi,lr=1)
00216     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00217     res = table.data(E)
00218
```

```
00219     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn, za, mi), file)
00220
00221
00222 def differential_cross_section_EL(zn,za,mi,reac,ref,file=None):
00223
00224     nx=[101,1]
00225
00226     x = np.loadtxt(ref)
00227     nxr=list(nx) ; nxr.reverse()
00228     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/10.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00229     angle=np.array((x[0,1]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00230
00231     r = amns.Reactants()
00232     r.add(zn,za,mi)
00233     r.add(zn,za,mi,lr=1)
00234     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00235     res = table.data(angle,E)
00236
00237     plot(E[:,0], res[:,0], x[:,0], x[:,2], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for angle=%s, ZN=%s, ZA=%s, MI=%s' % (reac, angle[0,0], zn, za, mi), file)
00238
00239
00240 def rct(zn,za,mi,reac,ref,file=None):
00241
00242     nx=[101,1]
00243
00244     x = np.loadtxt(ref)
00245     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/6.0))*0.1).repeat(nx[1]).reshape(nx),order='F')
00246
00247     r = amns.Reactants()
00248     r.add(zn,za,mi)
00249     r.add(zn,0.0,mi)
00250     r.add(zn,za-1,mi,lr=1)
00251     r.add(zn,1.0,mi,lr=1)
00252     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00253     res = table.data(E)
00254
00255     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn, za, mi), file, line2='r*')
00256
00257
00258 def sputter(zn_w,za_w,mi_w,zn_p,za_p,mi_p,reac,ref,file=None):
00259
00260     nx=[101,1]
00261
00262     x = np.loadtxt(ref, skiprows=6)
00263     nxr=list(nx) ; nxr.reverse()
00264     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
00265     A=np.array(np.array([0.0]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00266
00267     r = amns.Reactants()
00268     r.add(zn_w,za_w,mi_w)
00269     r.add(zn_p,za_p,mi_p)
00270     r.add(zn_w,za_w,mi_w, lr=1)
00271     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00272     res = table.data(E,A)
00273
00274     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn_p, za_p, mi_p), file, line2='r*')
00275
00276
00277 def reflect(zn_w,za_w,mi_w,zn_p,za_p,mi_p,reac,ref,file=None):
00278
00279     nx=[101,1]
00280
00281     x = np.loadtxt(ref, skiprows=6)
00282     nxr=list(nx) ; nxr.reverse()
00283     E=np.array((10**(np.arange(nx[0])/((nx[0]-1)/5.0))*10.0).repeat(nx[1]).reshape(nx),order='F')
00284     A=np.array(np.array([0.0]).repeat(nxr[1]).reshape(nxr).transpose(),order='F')
00285
00286     r = amns.Reactants()
00287     r.add(zn_w,za_w,mi_w)
00288     r.add(zn_p,za_p,mi_p)
00289     r.add(zn_p,za_p,mi_p, lr=1)
00290     table = amnsdb.get_table(reac.encode('UTF-8'), r)
00291     res = table.data(E,A)
00292
00293     plot(E[:,0], res[:,0], x[:,0], x[:,1], 'Energy', '%s [%s]' % (table.result_label,
      table.result_unit), '%s for ZN=%s, ZA=%s, MI=%s' % (reac, zn_p, za_p, mi_p), file, line2='r*')
00294
00295 amnsdb = amns.Amns()
00296
00297 if not os.path.exists('FIG'):
00298     os.makedirs('FIG')
00299
00300 texfile=open('amns_verify.tex', 'w')
```

```
00301
00302 print("'\\documentclass[a4wide,10pt]{article}
00303 \\usepackage[pdftex]{graphicx}
00304 \\usepackage[a4paper,includeheadfoot,margin=2.54cm]{geometry}
00305 \\usepackage{parskip,hyperref,multicol}  %% paralist,longtable,booktabs,spverbatim
00306 \\title{AMNS Verification Certificate}
00307 \\author{David.Coster@ipp.mpg.de on behalf of the AMNS Team}
00308 \\begin{document}
00309 \maketitle
00310 \\tableofcontents
00311 \\newpage
00312 \\begin{multicols}{2}"', file=texfile)
00313
00314 print('', file=texfile)
00315 print('\\section{Atomic rate coefficients using ADAS}', file=texfile)
00316 print('', file=texfile)
00317
00318 adas( 6,0,0, "EI",  'REF/scd96_c0.dat', 'FIG/EI.png',  texfile )
00319 adas( 6,1,0, "RC",  'REF/acd96_c1.dat', 'FIG/RC.png',  texfile )
00320 adas( 6,1,0, "CX",  'REF/ccd96_c1.dat', 'FIG/CX.png',  texfile )
00321 adas( 6,1,0, "BR",  'REF/prb96_c1.dat', 'FIG/BR.png',  texfile )
00322 adas( 6,0,0, "LR",  'REF/plt96_c0.dat', 'FIG/LR.png',  texfile )
00323 adas( 6,1,0, "EIP", 'REF/ecd96_c1.dat', 'FIG/EIP.png', texfile )
00324
00325 adas( 74,0,0, "LR",  'REF/plt89_w_01.dat', 'FIG/W_LR.png', texfile )
00326
00327
00328 print('', file=texfile)
00329 print('\\newpage', file=texfile)
00330 print('\\section{Nuclear cross sections}', file=texfile)
00331 print('', file=texfile)
00332
00333 nuclear_HB( (1,0,2), (1,0,2), (1,0,1), (1,0,3), "NUC_BB", 'REF/ref_D(D,p)T.dat',
       'FIG/D(D,p)T.png'      , texfile )
00334 nuclear_HB( (1,0,2), (1,0,2), (0,0,1), (2,0,3), "NUC_BB", 'REF/ref_D(D,n)^3He.dat',
       'FIG/D(D,n)^3He.png'   , texfile )
00335 nuclear_HB( (1,0,2), (1,0,3), (0,0,1), (2,0,4), "NUC_BB", 'REF/ref_D(T,n)^4He.dat',
       'FIG/D(T,n)^4He.png'   , texfile )
00336 nuclear_HB( (1,0,2), (2,0,3), (1,0,1), (2,0,4), "NUC_BB", 'REF/ref_D(^3He,p)^4He.dat',
       'FIG/D(^3He,p)^4He.png', texfile )
00337
00338 print('', file=texfile)
00339 print('\\newpage', file=texfile)
00340 print('\\section{Nuclear thermal-thermal rate coefficients}', file=texfile)
00341 print('', file=texfile)
00342
00343 nuclear_HB_tt( (1,0,2), (1,0,2), (1,0,1), (1,0,3), 'NUC_TT', 'REF/ref_tt_D(D,p)T_Ti.dat',
       'FIG/tt_D(D,p)T.png'      , texfile )
00344 nuclear_HB_tt( (1,0,2), (1,0,2), (0,0,1), (2,0,3), 'NUC_TT', 'REF/ref_tt_D(D,n)^3He_Ti.dat',
       'FIG/tt_D(D,n)^3He.png'   , texfile )
00345 nuclear_HB_tt( (1,0,2), (1,0,3), (0,0,1), (2,0,4), 'NUC_TT', 'REF/ref_tt_D(T,n)^4He_Ti.dat',
       'FIG/tt_D(T,n)^4He.png'   , texfile )
00346 nuclear_HB_tt( (1,0,2), (2,0,3), (1,0,1), (2,0,4), 'NUC_TT', 'REF/ref_tt_D(^3He,p)^4He_Ti.dat',
       'FIG/tt_D(^3He,p)^4He.png', texfile )
00347
00348 print('', file=texfile)
00349 print('\\newpage', file=texfile)
00350 print('\\section{Nuclear Beam-target rate-coefficients}', file=texfile)
00351 print('', file=texfile)
00352
00353 nuclear_HB_bt( (1,0,2), (1,0,2), (1,0,1), (1,0,3), 'NUC_BT', 'REF/ref_bt_3.dat', 'FIG/bt_D(D,p)T.png'
       , texfile )
00354 nuclear_HB_bt( (1,0,2), (1,0,2), (0,0,1), (2,0,3), 'NUC_BT', 'REF/ref_bt_4.dat',
       'FIG/bt_D(D,n)^3He.png'   , texfile )
00355 nuclear_HB_bt( (1,0,2), (1,0,3), (0,0,1), (2,0,4), 'NUC_BT', 'REF/ref_bt_1.dat',
       'FIG/bt_D(T,n)^4He.png'   , texfile )
00356 nuclear_HB_bt( (1,0,3), (1,0,2), (0,0,1), (2,0,4), 'NUC_BT', 'REF/ref_bt_2.dat',
       'FIG/bt_T(D,n)^4He.png'   , texfile )
00357 nuclear_HB_bt( (2,0,3), (1,0,2), (1,0,1), (2,0,4), 'NUC_BT', 'REF/ref_bt_6.dat',
       'FIG/bt_^3He(D,p)^4He.png', texfile )
00358 nuclear_HB_bt( (1,0,2), (2,0,3), (1,0,1), (2,0,4), 'NUC_BT', 'REF/ref_bt_5.dat',
       'FIG/bt_D(^3He,p)^4He.png', texfile )
00359
00360 #total_cross_section_EL       ( 74,0,184, 'EL',  'REF/W-total-elastic-cross-section.dat',
       'FIG/W_EL.png' )
00361 #differential_cross_section_EL( 74,0,184, 'dEL', 'REF/W-angular-diff-elastic-cross-section_90.dat',
       'FIG/W_dEL.png' )
00362
00363 #rct(  2,1,  4, 'RCT', 'REF/RCT_He.dat', 'FIG/RCT_He.png' )
00364 #rct( 10,1, 20, 'RCT', 'REF/RCT_Ne.dat', 'FIG/RCT_Ne.png' )
00365 #rct( 18,1, 40, 'RCT', 'REF/RCT_Ar.dat', 'FIG/RCT_Ar.png' )
00366 #rct( 36,1, 84, 'RCT', 'REF/RCT_Kr.dat', 'FIG/RCT_Kr.png' )
00367 #rct( 54,1,131, 'RCT', 'REF/RCT_Xe.dat', 'FIG/RCT_Xe.png' )
00368
00369 #sputter(74, 0, 184, 1, 0, 2, 'SPUT', 'REF/sputter_dw.y', 'FIG/sputter_W_by_D.png' )
00370 #reflect(74, 0, 184, 1, 0, 2, 'REFL', 'REF/reflect_dw.rn', 'FIG/reflect_D_on_W.png' )
00371
```

```
00372 amnsdb.finalize()
00373
00374 print("'\\end{multicols}
00375 \\end{document}"', file=texfile)
```

Index