IMAS Data Dictionary: an Introduction

F. Imbeaux (CEA)

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

What is IMAS ?

- IMAS is the ITER Integrated Modelling and Analysis Suite
- Infrastructure :
  - Data Dictionary : a machine generic ontology for magnetic fusion :
    - What data exist ?
    - What are they called ?
    - How are they structured ?
  - Data Access : functions to read/write objects defined in the Data Dictionary (available in Fortran, C++, Python, Matlab, Java)
  - Workflow component generator : encapsulate physics codes (using IDSs as I/O) to turn them into components that can be coupled in a workflow
- Physics applications : components and workflows (e.g. ETS, HFPS, …)

The IMAS Data Dictionary : a fusion standard

- The IMAS backbone is a machine-generic ontology : the physics Data Dictionary
  - Capable of covering all experiment subsystems and plasma physics, and is extensible
  - It represents simulation and experimental data with the same data structures, enabling direct comparisons
  - The Interface Data Structures (IDSs) are specific entry points of the Data Dictionary. They typically describe a tokamak subsystem (diagnostic, heating system, …) or an abstract physical concept (equilibrium, set of plasma profiles, wave propagation, MHD, …)
  - They define standard interfaces between physics components in an IMAS workflow
- The IMAS Data Dictionary is being promoted as the standard to enable Interoperability in the FAIR and open science requirements for FP9 (Fair4Fusion project, EUROfusion Data Management Plan)
- The IMAS Data Dictionary and Access Layer are now available Open Source

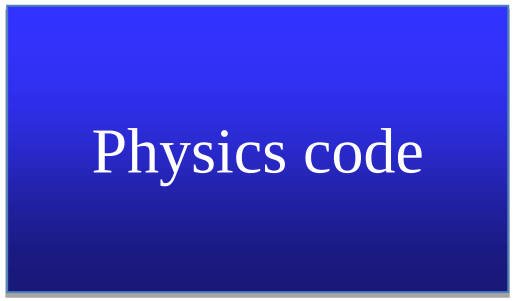
What TSVVs and DTE can do with the IMAS infrastructure ?

- **Share/store/publish data using a fusion-standard ontology**
  - Store simulations results and compare to an experiment
  - Exchange simulation results with other codes (benchmarking, reuse of input datasets)
  - Create a catalogue of simulations that can be searched/browsed (various catalogue prototypes are under development : simDB, DMP, …)
  - Make data FAIR and Open

- **Assemble a workflow of physics components** (Integrated Modelling, Simulation post-processing, Plasma Reconstruction Chains, …)
  - E.g. process synthetic diagnostic output from a simulation and compare to an experiment

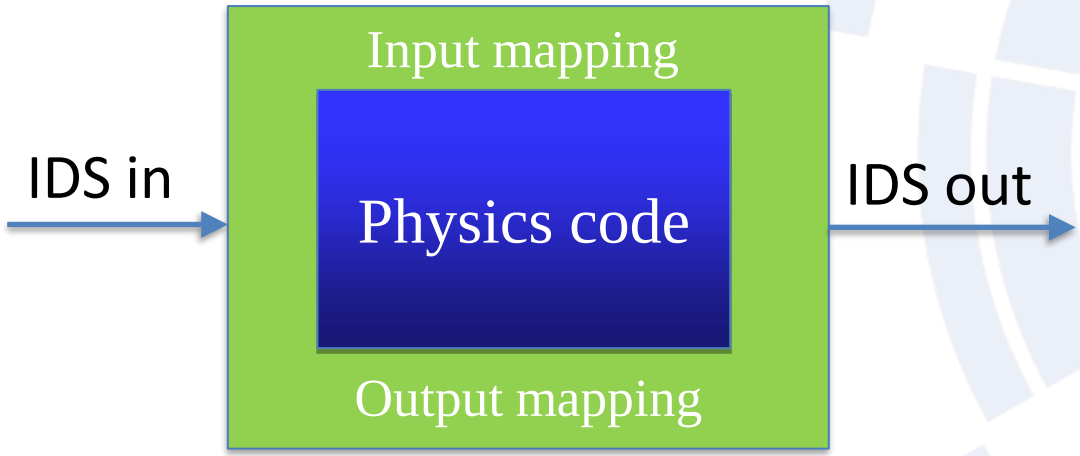
IMAS integration steps

- Step 0 : the physics code



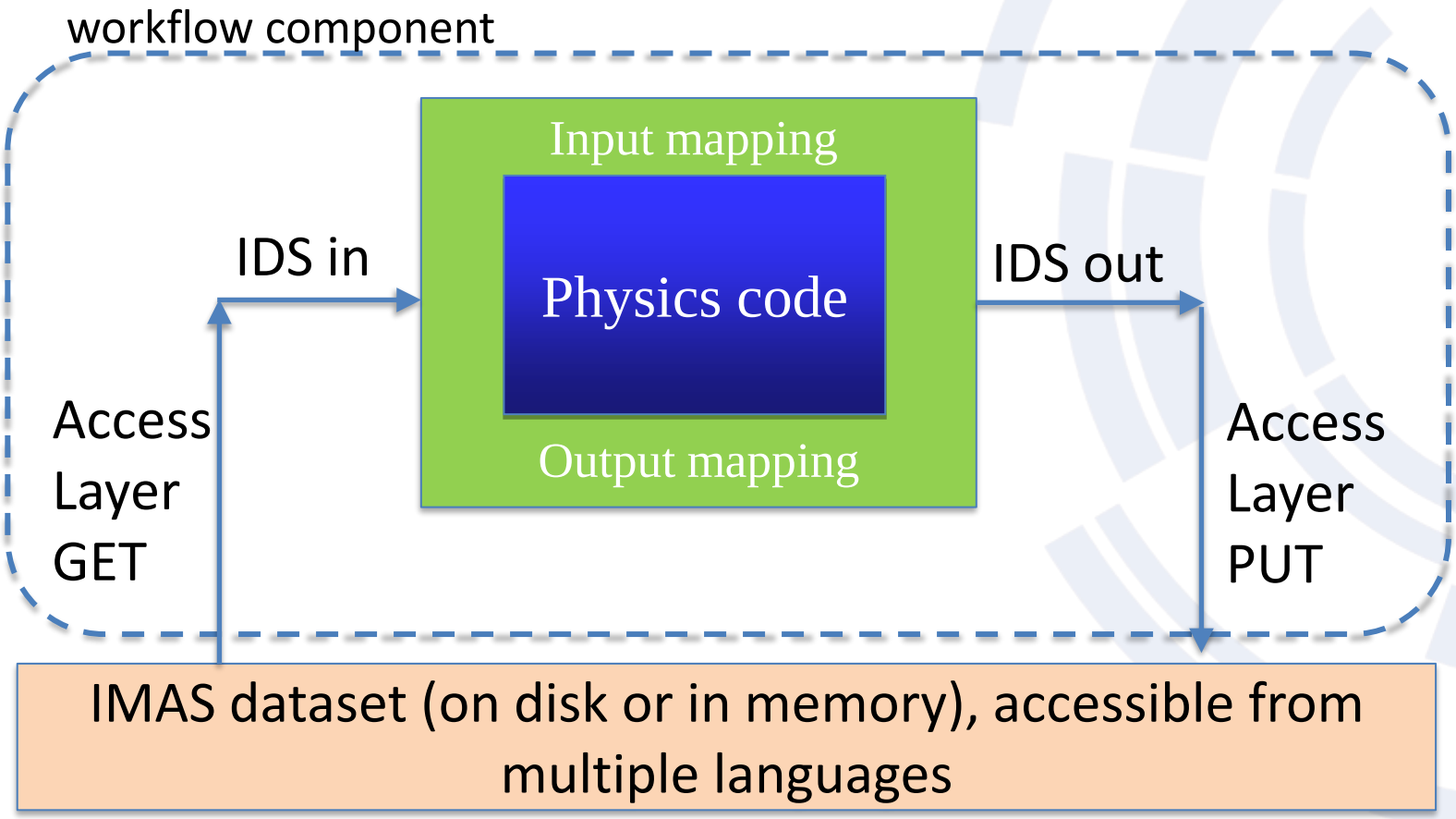
IMAS integration steps

- Step 1 : the TSVV physics code with I/O mapped to IMAS Data Dictionary



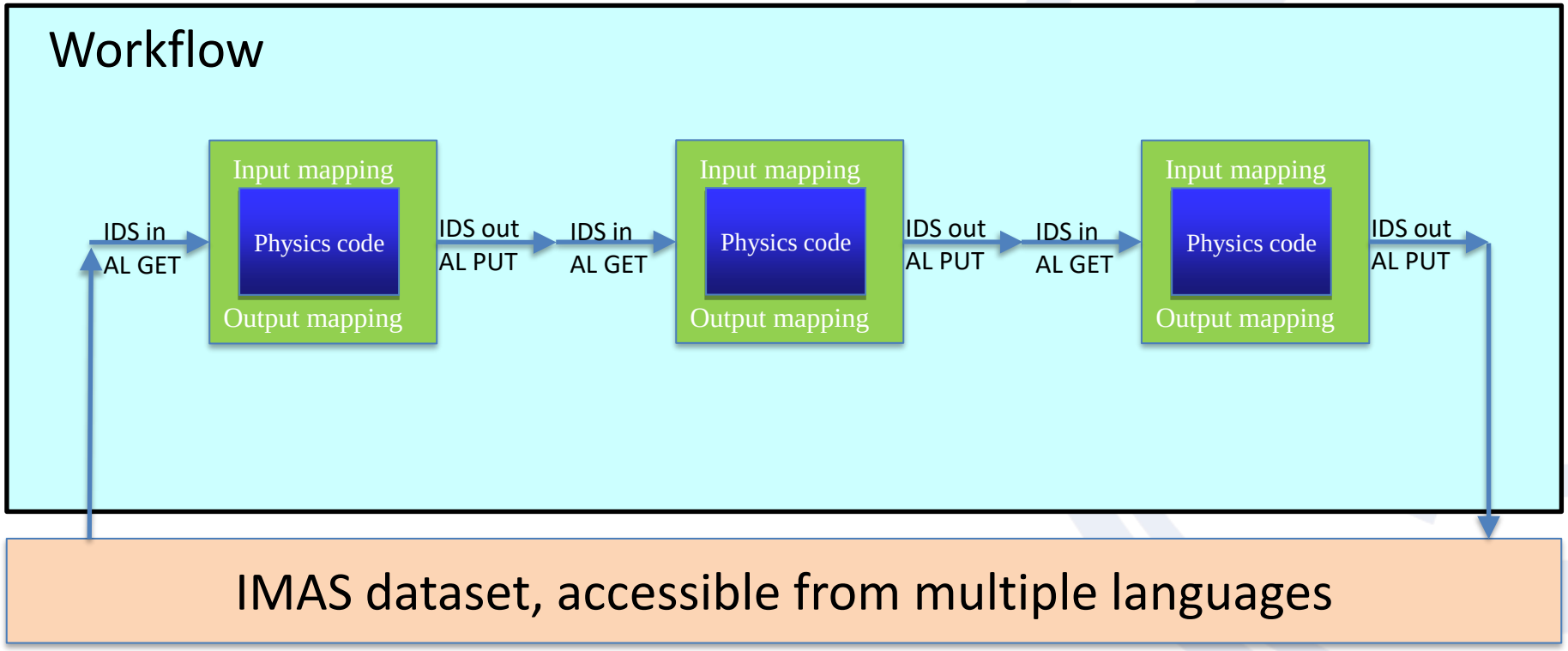
IMAS integration steps

- Step 2 : the physics code with I/O mapped to IMAS Data Dictionary uses the Access Layer to read/write data
- NB : this step, encapsulated, results in a potential workflow component



IMAS integration steps

- Step 3: multiple physics code with I/O mapped to IMAS Data Dictionary use the Access Layer to communicate together in a workflow



What means « adapting a code to IMAS » ?

- Basic level (minimal requirement for EUROfusion standard software)
  - Agree on a minimal set of input/output data to be mapped to IMAS
  - Create mapping script that will do the mapping and read/write data to an IMAS database
- Full IMAS interface
  - Map full I/O to IMAS, including code-specific parameters, and optionally restart files as well
- IMAS component
  - Full IMAS interface + generate component directly usable in workflows (Python, Muscle 3, …)

You may have needs that are not covered yet by the IMAS Data Dictionary

- You may contribute to the Data Dictionary !
- The IMAS Data Dictionary is extensible
- It has precise lifecycle procedure to be able to evolve and be jointly developed by multiple teams
- It has precise design rules to ensure global homogeneity
- Question/feature request ? : go to https://github.com/iterorganization/imas-data-dictionary and create an “issue” or a “discussion” → the repository is open !

Now we pause for questions before …

Going deeper inside the IMAS DD structure

Deeper insight : Interface Data Structure

- The Data Dictionary has a tree structure, for the sake of clarity
- At the top level, a collection of modular structures (IDSs) representing
  - Abstract physical quantities (e.g. distribution functions)
  - Tokamak subsystems (e.g. PF systems)
- By default, data access is made at the level of these structures (Write and Read)
  - NB : Partial Get is now available to read only a subset of an IDS structure
- These modular structures have the appropriate granularity for exchange in an IM workflow → they also represent standardised interfaces for communication between codes, named Interface Data Structure (IDS)
  - Each has an “ids_properties” substructure (metadata + comments + timebase usage)
  - Each has a “code” substructure (trace the code-specific parameters and the name/version of the code that has generated this IDS)

Data Dictionary documentation

- Openly available here : https://imas-data-dictionary.readthedocs.io/en/latest/reference_ids.html
- The IDS reference first shows the list of all IDSs. For each of them, a detailed documentation:
  - Full path name: name of all variables of the IDSs, with their path in the structure. **Replace "/" by the structure operator in a programming language, e.g. "%" in Fortran (core_profiles%global_quantities%ip), "." in C++, Matlab, Java, Python (core_profiles.global_quantities.ip)**
  - Definition
  - Units
  - In {}, whether it is STATIC (constant over a range of pulses, e.g. machine configuration), CONSTANT (constant over the pulse or the simulation), or DYNAMIC (time-dependent within the pulse or the simulation)
  - Data_Type in Blue: indicates whether it is a string, an integer or a real, and its dimension (0D, 1D, 2D, …)
  - Coordinates: for each dimension, the full path name to the related coordinate. If the dimension simply refers to a quantity not present in the Data Model, it is indicated as "1…N"
  - DD lifecycle information
- Also on this site: search DD tools, IDS migration guide, GGD guide, DD developer documentation

Timebases in IDSs

- “time” is a reserved node name for any timebase in the DD. Such nodes are recognized and used by the Access Layer when getting or putting time slices (GET_SLICE / PUT_SLICE functions).
- An IDS may contain quantities with different timebases in order to have the ability to describe experimental data as it is acquired in the experiment.
- However, an IDS can also be filled in a synchronous way (i.e. all dynamic quantities are stored on a unique timebase, located at the root of the IDS)
- There are therefore two possible usages of the IDS, with two possible locations for the “time” coordinate related to a given node. **Homogeneous_time is set by the data provider**.

Value of ids_properties/homogeneous_time	Location of the time coordinate for dynamic nodes
0	Dynamic nodes may be asynchronous, their timebase is located as indicated in the “Coordinates” column of the documentation.
1	All dynamic nodes are synchronous, their common timebase is the “time” node located at the IDS root (e.g. equilibrium/time).
2	Means that no dynamic node is filled in the IDS (dynamic nodes will skipped by the Access Layer)

You don’t need to fill everything when writing an IDS

- An IDS can contain a fairly large number of physical quantities and covers a wide range of use cases. Therefore there will be many cases in which the IDSs are only partially filled.
- The only requirement regarding empty fields are:
  - The ids_properties/homogeneous_time field must be filled (unless the IDS is static)
    - If homogeneous_time = 1, the root time must be filled
  - When a quantity is filled, the coordinates of this quantity must be filled as well
- Not meeting these requirements will cause PUT methods to return an error.

Arrays of structure

- Arrays of structures are frequently used in IDSs, to describe a list of elements that may have nodes of different sizes, in order to avoid creating large sparse arrays
- The two typical cases are :
- Case 1: lists of objects that may contain asynchronous nodes, e.g. PF coils may be acquired with different timebases :
  - pf_active%coil(i)%current%data(itime)
  - pf_active%coil(i)%current%time(itime)
  - These Case 1 AoS are used essentially in IDSs representing tokamak subsystems
- Case 2: list of time slices. The structure contains only dynamic and synchronous nodes, e.g. equilibrium/time_slice(:). This time slice representation allows the size of the children to vary as a function of time (e.g. variable grid size). These Case 2 AoS are used essentially in IDSs representing abstract physical quantities

Lifecycle information

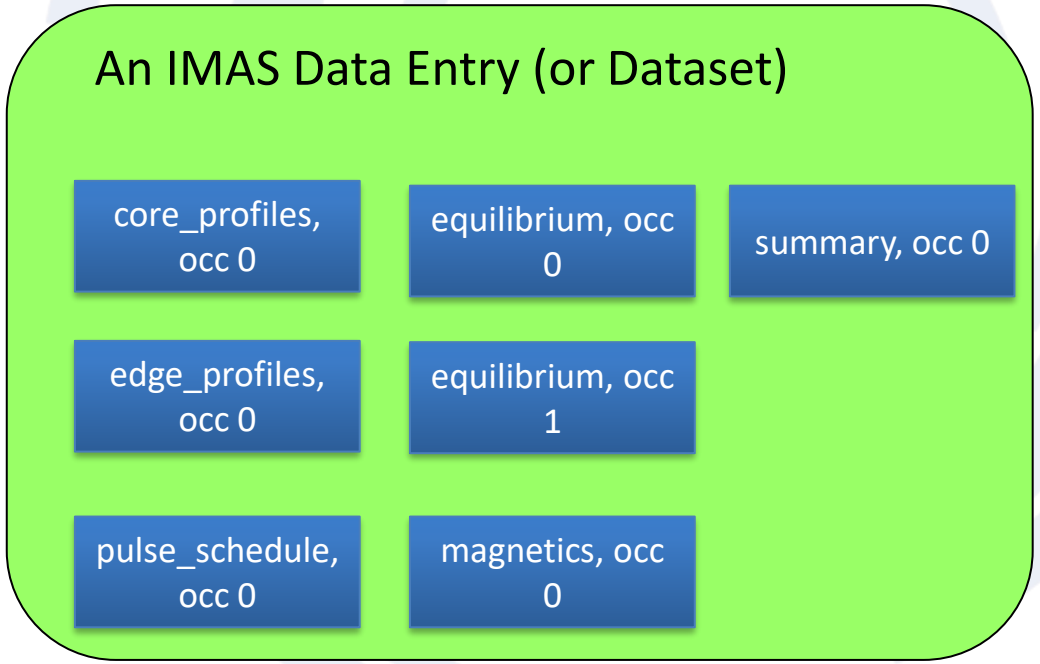
- The DD is a living object that evolves and expands with the needs. Therefore the lifecycle status of each node is documented
  - Some parts of the DD are recent and may evolve rapidly “lifecycle_status = alpha”
  - Some other parts are used for a longer time and are more stable “lifecycle_status = active”. If they need to be changed, they will become “obsolescent” but will not suddenly disappear (until a Major Release)
  - Some other parts are deprecated and shouldn’t be used “lifecycle_status = obsolescent”
- The lifecycle status of IDS nodes is described in the documentation. It applies to all descendants of a node, unless a descendant carries a different lifecycle status
- Two major versions are currently used
  - DD v3 (latest 3.42.2): mostly used outside of IO – frozen since 2024 (only technical patches allowed)
  - DD v4 (latest 4.1.1): used at IO – some non-backward compatibles changes w.r.t. v3 – allowed to evolve

Now we move to …

IMAS Data entries

IMAS Data Entries for storing simulation input/output

- A Data Entry is a collection of IDSs, gathered as a logical dataset (e.g. all IDSs corresponding to a given simulation output)
- A specific IDS, “summary” is the placeholder for description of the content of the dataset
- Multiple occurrences of a given IDS can co-exist, e.g. multiple equilibria calculated by different codes / assumptions
- Historically a Data Entry is defined (similarly as a pulse dataset in an experimental database) by:
  - IMAS “major version” (=“3” or “4”)
  - User name
  - Machine name
  - Pulse number
  - Run number (multiple simulations related to the same pulse)
- With recent versions of the Access Layer, a data entry is defined via a URI in a more generic way:
  - imas:[//host/]backend?query[#fragment]
  - imas:mdsplus?pulse=123;run=2;user=public;database=ITER;version=3
  - imas:hdf5?path=/home/username/runfolder
  - imas:ascii?path=./debug
- Choose a File Backend to write to disk (MDS+, ASCII, HDF5)
  - Create or open a Data Entry
  - Then GET or PUT individual IDSs from/in it

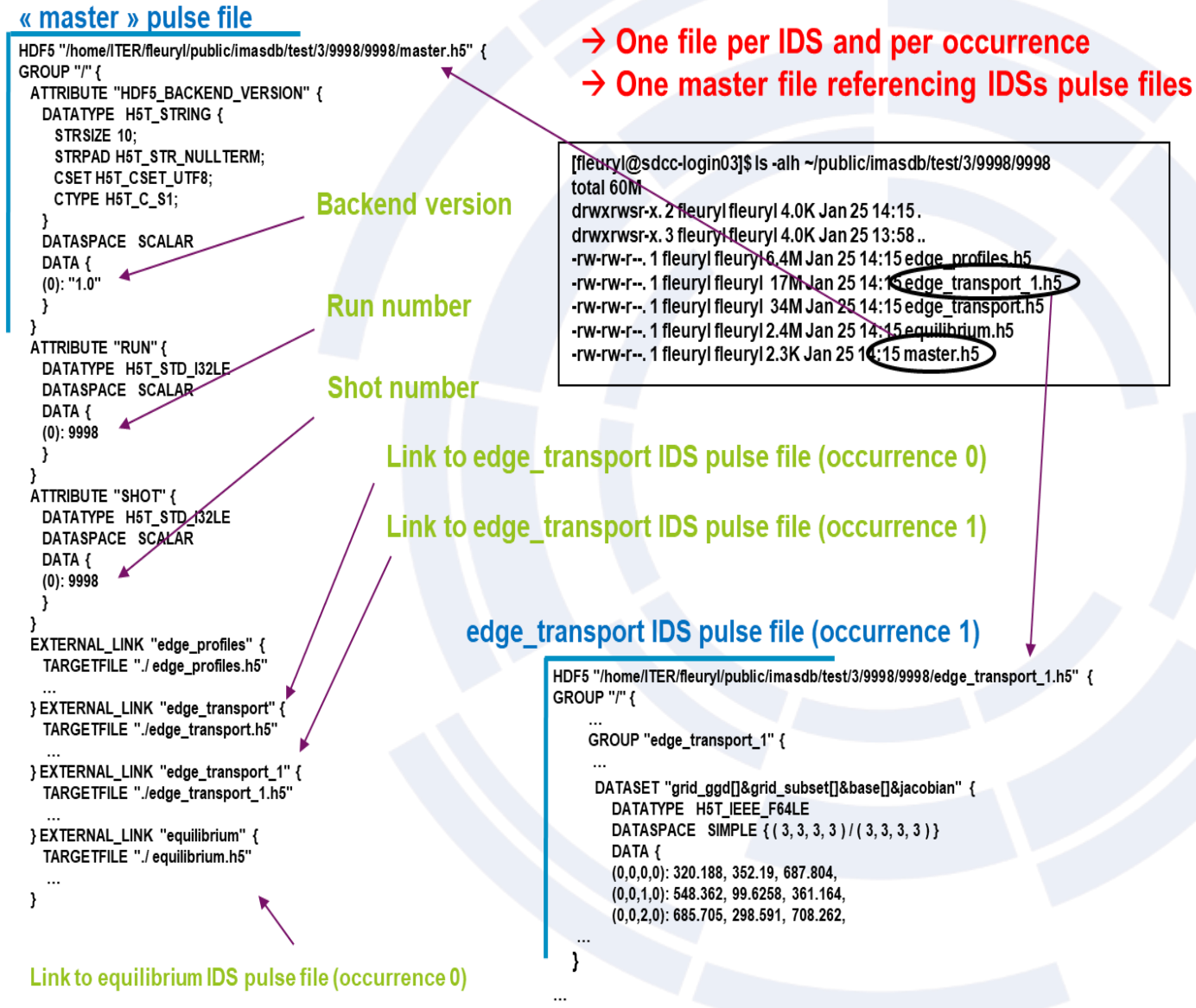


Which IMAS Backend to use ?

- Three Backends are available to write IMAS data to files
  - HDF5 Backend is the most recent and most optimized Backend for IMAS → we recommend using this one to produce new datasets
  - MDS+ Backend it the historical one, many datasets are available with this backend. Drawback : creates huge data files even for small amounts of data
  - ASCII Backend is not recommended for large data size, but may be interesting for rapid testing purposes. Reduced functionalities (no time slice operation)
- There are also other Backends
  - Memory Backend allows faster exchange of IDSs in memory (e.g. between components written in different languages)
  - UDA Backend allows reading (not writing, so far) data remotely, and includes an optional data mapping step (e.g. for reading experimental data not natively in IMAS format)
- There is also a NetCDF export function in IMAS-Python but it changes the data format (tensorization), to be used to export data to non-IMAS users. No time slice operation

IMAS file structure when using the HDF5 Backend

- If using the “legacy” Data Entry definitions: all pulse files are located in the user’s account under the folder: ~/public/imasdb/DatabaseName/IMASMajorVersion/SHOT/RUN
- Otherwise they are located under the folder you specified
- Modular organization:
- One file per IDS and occurrence
- One master file with the references



Conclusion : you are ready to start playing with IMAS

- The IMAS infrastructure features a standard, machine-generic Data Dictionary for experiment and simulation data, together with an Access Layer in multiple programming languages
- ITER Organization succeeded to make it a true fusion standard among ITER members
- IMAS facilitates:
  - Building FAIR fusion databases
  - Sharing data accross experiments and simulation tools
  - Interfacing codes

- Data Dictionary documentation: https://imas-data-dictionary.readthedocs.io/en/latest/reference_ids.html
- Question/feature request ? : go to https://github.com/iterorganization/imas-data-dictionary and create an “issue” or a “discussion”
- All Open Source
- See next courses for an Introduction to Access Layer in Python and Matlab
- Our ACH can support you during your usage of IMAS

BACKUP SLIDES

- BACKUP SLIDES

Minimal workflow for reading from / writing to IMAS
In green : what the TSVVs have to do

- The IMAS infrastructure has an API (Access Layer, AL) to read (GET) and write (PUT) IDS structures from a variable in your favorite language (Fortran, C++, Matlab, Python, Java) to a file (MDS+, ASCII, HDF5).
- OPEN the data entry of interest (for your input)
- GET the IDSs of interest for the input to your code
- Map the IMAS input to your code's data model
- Run your code
- Map your code output to IMAS
- OPEN/CREATE your output data entry
- PUT the output IDSs to the output data entry
- CLOSE data entries
- You are done !