Means of accessing IDS compatible data using Python

IMAS-Python
7 July 2026

ydejong@ignitioncomputing.com

IGNITION COMPUTING

EUROfusion

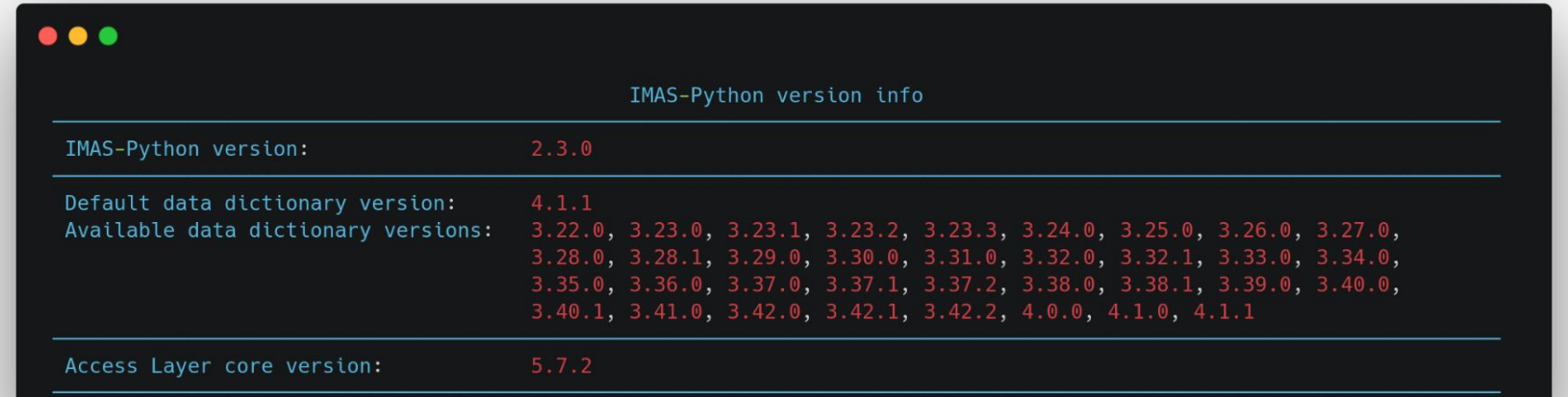
Outline

- What is IMAS-Python
- Installing IMAS-Python
- Using IMAS-Python
- Extras

What is IMAS-Python

- Pure Python codebase
  - Easy to extend, test and validate

- Works with any version of the Data Dictionary
  - Data dictionary versions are parsed at runtime
  - ‘Batteries included’: recent DD versions included in installation



Installing IMAS-Python

- Installation is easy!
- On SDCC just load the module
  - Check module list on other clusters
- Local install through pip
  - NetCDF and Xarray extras recommended!

```
$ module load IMAS-Python
```

```
$ pip install imas-python[netcdf,xarray]
```

Creating an IDS

- Import imas
- Create the IDS

```
>>> import imas
>>> ids_factory = imas.IDSFactory()
>>> core_profiles = ids_factory.core_profiles()
07:28:12 INFO     Parsing data dictionary version 4.1.1 @dd_zip.py:89
```

Inspecting the IDS

- Explore the child nodes
- IMAS-Python has Tab-autocomplete!
- (Be careful, some IDSs can have a large number of children)

```
>>> imas.util.inspect(core_profiles)
┌──────────────────────────── IDS: core_profiles (DD version 4.1.1) ────────────────────────────┐
│ Core plasma profiles                                                                          │
│ ┌──────────────────────────────────── Attributes ─────────────────────────────────────┐       │
│ │ has_value = False                                                                  │       │
│ │  metadata = <IDSMetadata for 'core_profiles'>                                      │       │
│ └────────────────────────────────────────────────────────────────────────────────────┘       │
│ ┌──────────────────────────────────── Child nodes ────────────────────────────────────┐       │
│ │                  code = <IDSStructure (IDS:core_profiles, code)>                   │       │
│ │            covariance = <IDSStructure (IDS:core_profiles, covariance)>             │       │
│ │     global_quantities = <IDSStructure (IDS:core_profiles, global_quantities)>      │       │
│ │        ids_properties = <IDSStructure (IDS:core_profiles, ids_properties)>         │       │
│ │           profiles_1d = <IDSStructArray (IDS:core_profiles, profiles_1d with 0 items)>     │
│ │           profiles_2d = <IDSStructArray (IDS:core_profiles, profiles_2d with 0 items)>     │
│ │            statistics = <IDSStructArray (IDS:core_profiles, statistics with 0 items)>     │
│ │                  time = <IDSNumericArray (IDS:core_profiles, time, empty FLT_1D)>  │       │
│ │ vacuum_toroidal_field = <IDSStructure (IDS:core_profiles, vacuum_toroidal_field)>   │       │
│ └────────────────────────────────────────────────────────────────────────────────────┘       │
└───────────────────────────────────────────────────────────────────────────────────────────────┘
```

Inspecting the IDS

- Find all paths ending with temperature
- `util.find_paths`
  - Supports regex

```
>>> print('\n'.join(imas.util.find_paths(core_profiles, "temperature$")))
profiles_1d/electrons/temperature
profiles_1d/ion/temperature
profiles_1d/ion/state/temperature
profiles_1d/neutral/temperature
profiles_1d/neutral/state/temperature
profiles_2d/ion/temperature
profiles_2d/ion/state/temperature
```

Reading data

- Open IMAS URI
  - `imas:hdf5?path=...`
  - `imas:mdsplus?path=...`
- Also supports netcdf!
  - `my_netcdf_file.nc`
- `get` will load all data in memory
  Use `get(ids, lazy=True)`!

```
>>> import imas
>>> entry = imas.DBEntry("imas:hdf5?path=test_entry", "r")
>>> eq = entry.get("equilibrium")
>>> imas.util.inspect(eq)
                    IDS: equilibrium (DD version 4.1.1)
Description of a 2D, axi-symmetric, tokamak equilibrium; result of an equilibrium code.
                              Attributes
  has_value = True
   metadata = <IDSMetadata for 'equilibrium'>

                              Child nodes
                code = <IDSStructure (IDS:equilibrium, code)>
           grids_ggd = <IDSStructArray (IDS:equilibrium, grids_ggd with 0 items)>
      ids_properties = <IDSStructure (IDS:equilibrium, ids_properties)>
                time = <IDSNumericArray (IDS:equilibrium, time, FLT_1D)>
                       numpy.ndarray([  1.202     , 432.93759781, 798.92426448])
          time_slice = <IDSStructArray (IDS:equilibrium, time_slice with 3 items)>
vacuum_toroidal_field = <IDSStructure (IDS:equilibrium, vacuum_toroidal_field)>
```

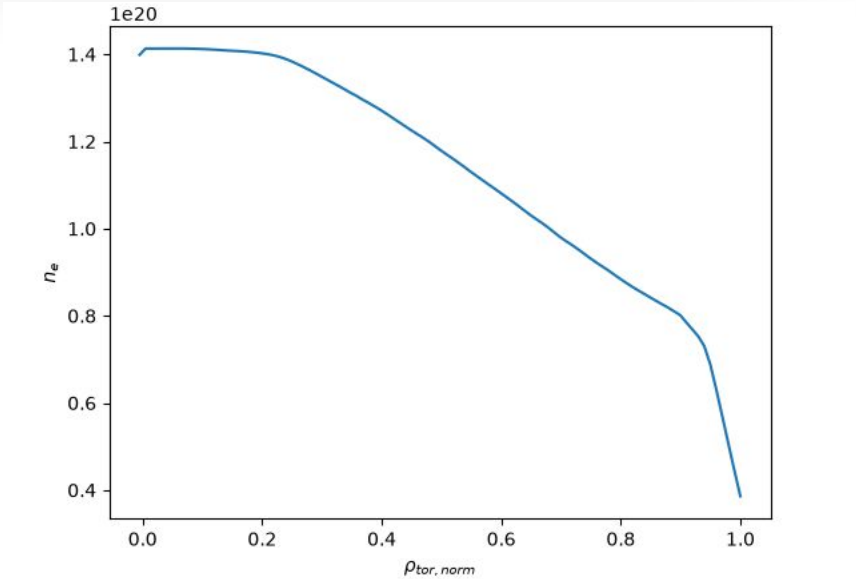
IGNITION COMPUTING

EUROfusion

Plotting data

```
>>> n_e = core_profiles.profiles_1d[1].electrons.density
>>> plt.plot(n_e.coordinates[0], n_e)
>>> plt.xlabel(r"$\rho_{tor, norm}$")
>>> plt.ylabel(r"$n_e$")
>>> plt.ticklabel_format(axis="y", scilimits=(-1, 1))
>>> plt.show()
```

- Find the quantity you want to plot
  - `electron.density`
- `.coordinates` contain the coordinates of the value!

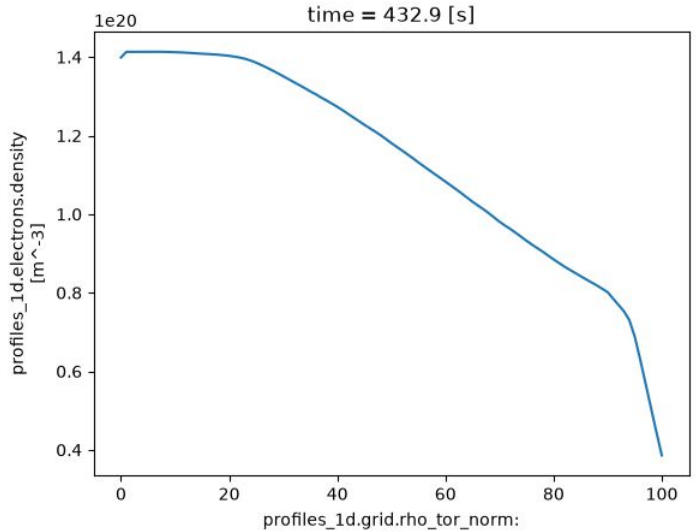


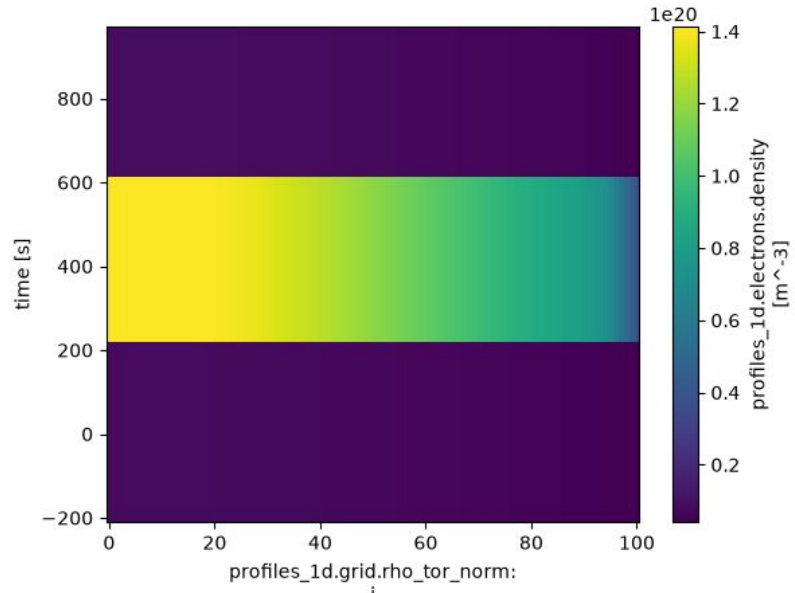
IGNITION COMPUTING EUROfusion

Plotting data (Xarray)

```
>>> ds = imas.util.to_xarray(core_profiles)
>>> n_e = ds["profiles_1d.electrons.density"]
>>> n_e.sel(time=400, method='nearest').plot()
>>> plt.show()
>>> n_e.plot()
```

- Entries can be converted to Xarray
  - Gives you units for free!
  - Easy plotting!





Creating an IDS

- Start with an empty IDS
  - `imas.IDSFactory()`
- Fill it with data

```
>>> factory = imas.IDSFactory()
>>> cp = factory.core_profiles()
>>> cp.ids_properties.homogeneous_time = imas.ids_defs.IDS_TIME_MODE_HOMOGENEOUS
>>> cp.ids_properties.comment = "Synthetic IDS created for demo"
>>> cp.ids_properties.creation_date = datetime.date.today().isoformat()
>>> cp.time = [1.0, 2.5, 4.0] # Timesteps
>>> rho_tor_norm = np.linspace(0, 1, num=64) # Main coordinate
>>> cp.profiles_1d.resize(len(cp.time))
>>> # Generate a profile
>>> for index, t in enumerate(cp.time):
...     t_e = np.exp(-16 * rho_tor_norm**2) + (1 - np.tanh(4 * rho_tor_norm - 3)) * t / 8
...     t_e *= t * 500
...     # Store the generated t_e as electron temperature
...     cp.profiles_1d[index].electrons.temperature = t_e
...
```

Creating an IDS

- Start with an empty IDS
  - `imas.IDSFactory()`
- Fill it with data
- Was that correct?
  - No, we forgot to set the coordinates!

```
>>> cp.validate()
Traceback (most recent call last):
  File "<python-input-108>", line 1, in <module>
    cp.validate()
    ~~~~~~~~~~~^^
  File "/home/yannick/projects/imaspython-training/.venv/lib64/python3.14/site-packages/imas/ids_toplevel.py", line 283, in validate
    raise exc.with_traceback(None) from None
imas.exception.CoordinateError: Element `profiles_1d[0]/electrons/temperature` has incorrect shape (64,): its coordinate in dimension 1 (`profiles_1d[0]/grid/rho_tor_norm`) has size 0.
```

Creating an IDS

- Start with an empty IDS
  - `imas.IDSFactory()`
- Fill it with data
- Was that correct?
  - No, we forgot to set the coordinates!
  - Let’s fix it :)

```
>>> for index in range(3):
...     cp.profiles_1d[index].grid.rho_tor_norm = rho_tor_norm
>>> cp.validate()
```

Storing an IDS to disk

- Let’s save the IDS using the HDF5 backend

```
>>> with imas.DBEntry("imas:hdf5?path=my_dataset", "w") as entry:
...     entry.put(cp)
...
```

```
~/projects/imaspython-training ❯ l -lah my_dataset
total 36K
drwxr-xr-x. 1 yannick yannick   50 Jul  9 08:48 .
drwxr-xr-x. 1 yannick yannick   86 Jul  9 08:48 ..
-rw-r--r--. 1 yannick yannick  29K Jul  9 08:48 core_profiles.h5
-rw-r--r--. 1 yannick yannick 2.0K Jul  9 08:48 master.h5
```

Storing an IDS to disk

- What about other backends?
  - NnetCDF

  - MDSPlus
    - Requires `saxonche` package (`pip install saxonche`)

Using NetCDF

- Easy to share
  - Don’t need IMAS installation to open data

```
>>> ds = xr.open_dataset("my_dataset.nc", group="core_profiles/0")
>>> ds
<xarray.Dataset> Size: 3kB
Dimensions:                                          (time: 3,
                                                      profiles_1d.grid.rho_tor_norm:i: 64)
Coordinates:
  * time                                             (time) float64 24B 1.0 ...
    profiles_1d.grid.rho_tor_norm                    (time, profiles_1d.grid.rho_tor_norm:i) float64 2kB ...
Dimensions without coordinates: profiles_1d.grid.rho_tor_norm:i
Data variables:
    ids_properties                                   |S1 1B ...
    ids_properties.comment                           object 8B ...
    ids_properties.homogeneous_time                  float64 8B ...
    ids_properties.creation_date                     object 8B ...
    ids_properties.version_put                       |S1 1B ...
    ids_properties.version_put.data_dictionary       object 8B ...
    ids_properties.version_put.access_layer          object 8B ...
    ids_properties.version_put.access_layer_language object 8B ...
    profiles_1d                                      |S1 1B ...
    profiles_1d.grid                                 |S1 1B ...
    profiles_1d.electrons                            |S1 1B ...
    profiles_1d.electrons.temperature                (time, profiles_1d.grid.rho_tor_norm:i) float64 2kB ...
```

IGNITION COMPUTING

Lazy loading

- `.get` loads entire IDS in memory
  - Can be too big!
  - Slow to load large datasets
- Lazy loading will fetch data on demand!

```
>>> entry = imas.DBEntry("imas:hdf5?path=my_dataset", "r")
>>> core_profiles = entry.get("core_profiles", lazy=True)
>>> core_profiles.profiles_1d[1].electrons.temperature
<IDSNumericArray (IDS:core_profiles, profiles_1d[1]/electrons/temperature, FLT_1D)>
numpy.ndarray([2029.31826316, 2024.02853251, 2008.76694596, 1983.88716603,
       1949.96526243, 1907.7767444 , 1858.26621592, 1802.51128823,
       1741.68264694, 1677.00230707, 1609.70209148, 1540.98424447,
       1471.98585725, 1403.74845919, 1337.19374658, 1273.10601006,
       1212.12141376, 1154.72390077, 1101.24717434, 1051.88194835,
       1006.68748408,  965.60633514,  928.48120541,  895.07287666,
        865.07827013,  838.1478549 ,  813.90178903,  791.94436245,
        771.87648892,  753.3061575 ,  735.85689299,  719.1743835 ,
        702.9315099 ,  686.83205417,  670.61337395,  654.04831163,
        636.94656202,  619.15565948,  600.56166917,  581.08958716,
        560.70337876,  539.40552343,  517.23589789,  494.26982365,
        470.61513532,  446.40819193,  421.80884775,  396.99450965,
        372.15351865,  347.47818593,  323.15787093,  299.37250306,
        276.28691401,  254.04627189,  232.77280375,  212.56387564,
        193.49138639,  175.60233724,  158.92037186,  143.4480454 ,
        129.16957344,  116.05382774,  104.05737846,   93.1274235 ])
```

IGNITION COMPUTING

EUROfusion

Converting data dictionary versions

- Automatic conversion within major versions

```
>>> entry = imas.DBEntry("imas:hdf5?path=my_dataset", "r", dd_version="4.0.0")
09:15:01 INFO     Parsing data dictionary version 4.0.0 @dd_zip.py:89
```

- Explicit conversion between major versions

```
>>> ids = entry.get("core_profiles")
>>> ids = imas.convert_ids(ids, "3.32.0")
09:15:21 INFO     Starting conversion of IDS core_profiles from version 4.0.0 to version 3.32.0. @ids_convert.py:598
09:15:21 INFO     Conversion of IDS core_profiles finished. @ids_convert.py:636
```

Converting data dictionary versions

- By default, IMAS-Python uses the latest version of data dictionary!
- IMAS Python will raise an error if automatic conversion is not possible
- Read more: https://imas-python.readthedocs.io/en/latest/multi-dd.html#loading-idss-from-a-different-major-version

Resampling

- Convenience functions to resample data
  - NBI energy at timesteps 0.5 and 1.5

```python
import imas
nbi = imas.IDSFactory().new("nbi")
nbi.ids_properties.homogeneous_time = imas.ids_defs.IDS_TIME_MODE_HOMOGENEOUS
nbi.time = [1, 2, 3]
nbi.unit.resize(1)
nbi.unit[0].energy.data = 2 * nbi.time

imas.util.resample(
    nbi.unit[0].energy.data,
    nbi.time,
    [0.5, 1.5],
    nbi.ids_properties.homogeneous_time,
    inplace=True,
    fill_value="extrapolate",
)
```

https://imas-python.readthedocs.io/en/stable/resampling.html

CLI

- imas version: show version information
- imas convert: convert an ids to a different dd version
- imas print: print the contents of an IDS
- imas analyze-db: analyzes Data Entries
- More info: https://imas-python.readthedocs.io/en/stable/cli.html

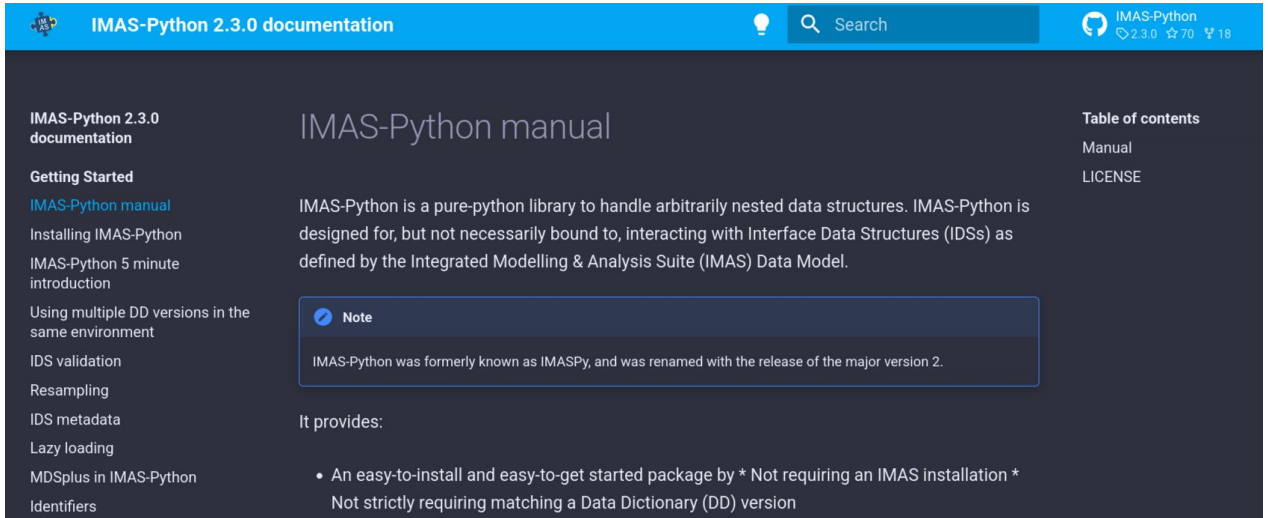
Projects using IMAS-Python

- Tokamunch
- Imas-validator
- Imas-idstools
- Imas-paraview
- Imas-simdb
- Imas-muscle3
- Fusio
- Torax
- Waveform-editor
- Cherab-imas
- Data Dictionary documentation

- Let us know if you are using IMAS-Python!

Checkout the documentation!

- Extensive documentation: https://imas-python.readthedocs.io/en/stable/
- IMAS-Python 101 course: https://imas-python.readthedocs.io/en/stable/courses/basic_user_training.html



IGNITION COMPUTING EUROfusion