Interfacing OMFIT with ITER IMAS via OMAS

O. Meneghini
B.C. Lyons, J. McClenaghan
S.P. Smith, [2] J. Ferreira
[3] J. Hollocombe, [3] M. Romanelli

[1] General Atomics, San Diego
[2] Instituto Superior Tcnico Lisboa
[3] UKAEA, Culham Science Centre

3rd IAEA Technical Meeting on
Fusion Data Processing, Validation
and Analysis

2831 May 2019

GENERAL ATOMICS

Interfacing OMFIT with ITER IMAS via OMAS

1 OMFIT framework and IMAS data dictionary

2 Manipulating IMAS data with OMAS library

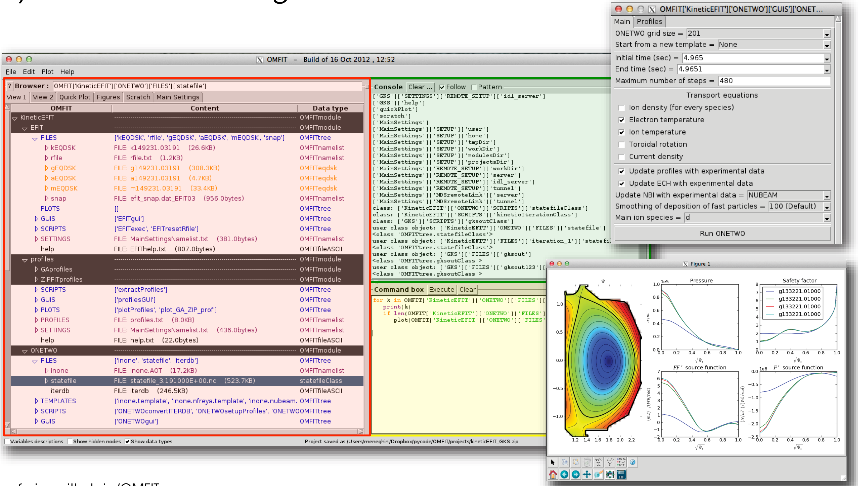
3 Integrated modeling with OMFIT and IMAS

4 Scaling IMAS performance for HPC and ML

5 Conclusions

OMFIT – One Modeling Framework for Integrated Tasks

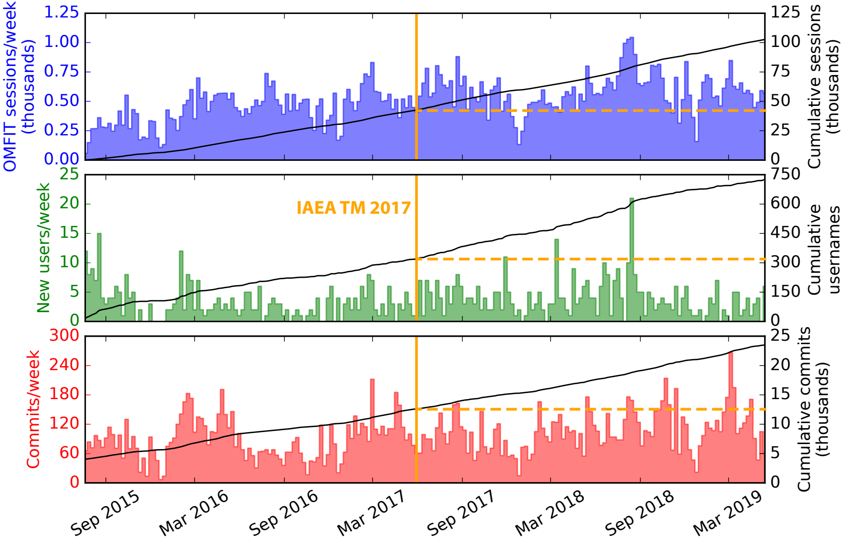
"A versatile framework designed to facilitate experimental data analysis and enable integrated simulations"



http://gafusion.github.io/OMFIT-source

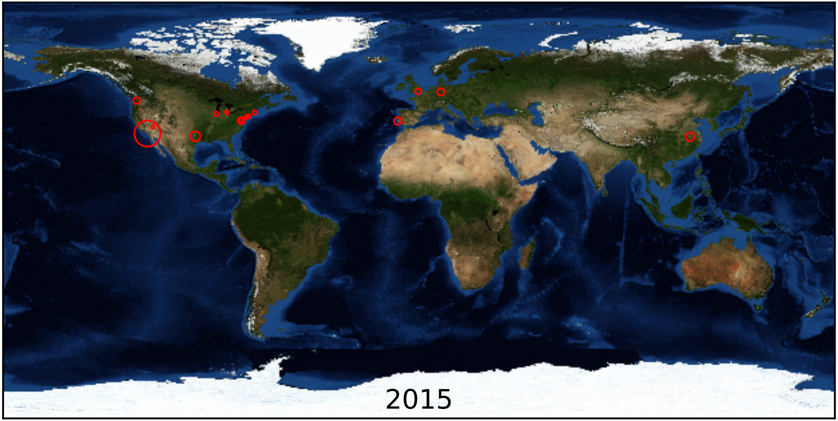
O. Meneghini, S. Smith, et al. Nuclear Fusion, **55** 083008 (2015)

OMFIT usage is steadily growing both in number of users as well as geographically

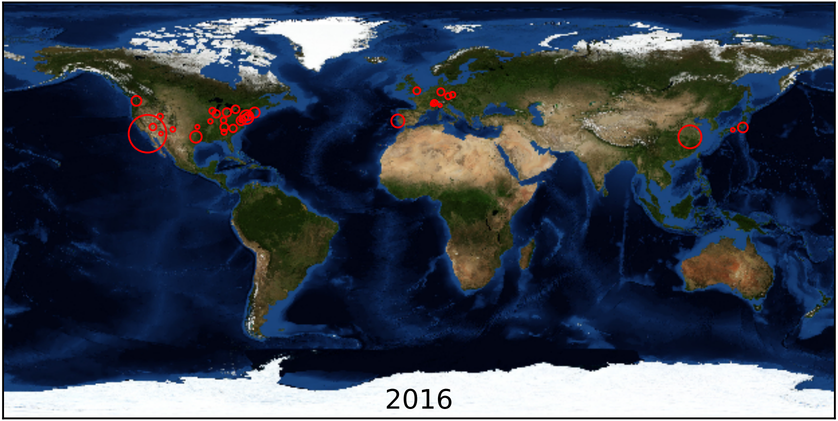


GENERAL ATOMICS

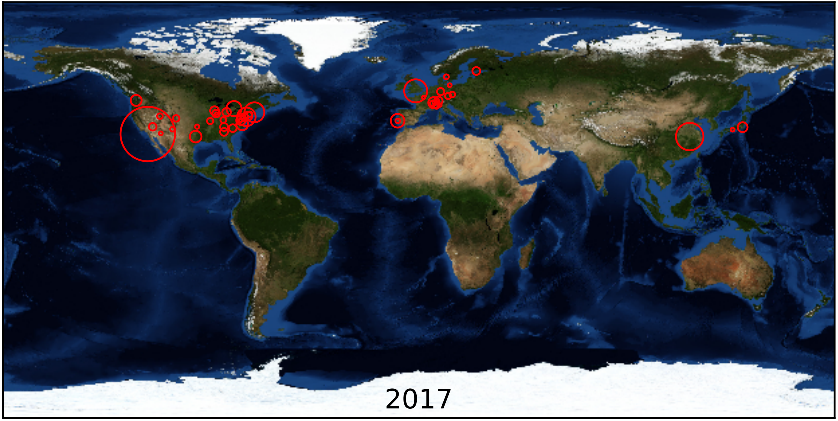
OMFIT usage is steadily growing both in number of users as well as geographically



OMFIT usage is steadily growing both in number of users as well as geographically

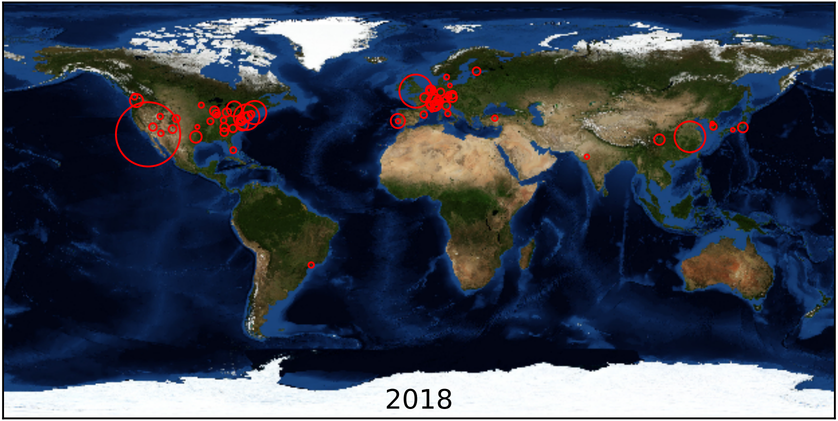


OMFIT usage is steadily growing both in number of users as well as geographically



GENERAL ATOMICS

OMFIT usage is steadily growing both in number of users as well as geographically



GENERAL ATOMICS

OMFIT – One Modeling Framework for Integrated Tasks

1. **Integrated Modeling Framework**
   - Enables data exchange among different components and coordinates their execution in complex workflows

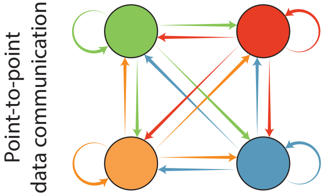
2. **Lightweight & pure Python**
   - Remote execution of interactive/batch jobs
   - Installs & runs anywhere: public/private, cluster/laptop

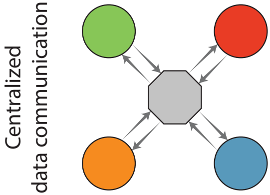
3. **Free-form hierarchical data structure**
   - No a-priori decision of what is stored and how
   - Support for most fusion-relevant data formats
   - Does not exclude use of data structures from other frameworks

4. **Interactive and graphical or scripted**
   - Accelerate time consuming IM tasks: develop, setup, visualize, share

5. **Version control and community**
   - Grow at scale cheaply and remain focused

Standardized data format enables centralized data communication but requires coordination among all parties





Free-form data format:

1. Does not require third-parties agreement
2. Workflow dependent integration
3. Does not scale theoretically

Standardized data format:

1. Requires coordination among ALL parties
2. Interfaces are workflow independent (plug new models & play)
3. Scales well theoretically

imas – ITER Integrated Modeling and Analysis Suite

IMAS is the data schema and storage infrastructure that support ITER plasma operations and research

⇒ our community best attempt to build a standard fusion format

- **IMAS data schema: Interface Data Structure (IDS)**
  - Data organized 48 IDSs for different physics
  - For both experimental and simulated data
  - Each IDS is structured as a hierarchical tree
    - Data are leaf nodes (scalars / arrays)
    - Structures (arrays of) are branches
- **IMAS storage infrastructure:** Access Layer (AL)
  - Layer that passes data between components and to/from storage
  - C/C++, Fortran (F95), Java, Matlab, Python
- **Significant effort is going into making IMAS a standard**
  - All ITER data will only be available through IMAS
  - European tokamaks making notable progress adopting IMAS

Need to interface frameworks with IMAS but it is hard to build on top of an infrastructure that evolves at its foundations

Back-end replacement is under way: from UAL to AL (based on UDA)

- Addresses some performance issues, provides client-server capabilities, enables dynamic mapping of existing data to IMAS

Currently **AL is tightly linked to the data-schema**, which requires re-compile IMAS and physics codes for each data-schema release

- Proposed new HDC API to be independent of data-schema

Major upgrades are welcome, but they are a problem when building a functional integrated modeling environment

- eg. European effort: CPO/UAL → IDS/UAL → IDS/AL → IDS/AL(HDC)

Some long-standing limitations remain:

- IMAS infrastructures is heavy, and hard to install and manage
- Independently of the programming language, the IMAS API does not provide any useful functionality besides data storage

Solution: Store data according to IMAS schema, but do not rely on the IMAS infrastructure itself

Shortcomings and rapidly evolving IMAS infrastructure demand an **approach that decouples our integrated modeling environments from IMAS, while ensuring their compatibility**

⇓ IDEA ⇓

If one organizes data in compliance with IMAS schema, then there must be a way to automatically save/load data from/to IMAS

⇓ IMPLEMENTATION ⇓

OMAS

Interfacing OMFIT with ITER IMAS via OMAS

1 OMFIT framework and IMAS data dictionary

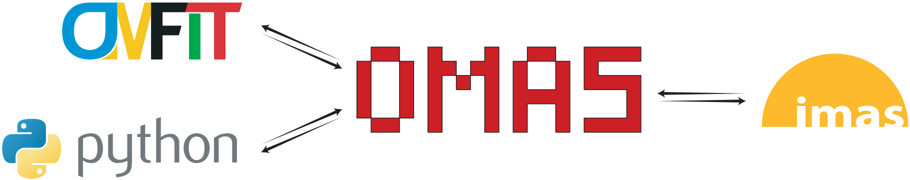
2 Manipulating IMAS data with OMAS library

3 Integrated modeling with OMFIT and IMAS

4 Scaling IMAS performance for HPC and ML

5 Conclusions

OMAS – Ordered Multidimensional Array Structure



Python library designed to simplify the interface of third-party codes with the ITER Integrated Modeling and Analysis Suite (**IMAS**).

- It provides a convenient Python API
- capable of storing data with different file/database formats
- in a form that is always compatible with the IMAS data model
- avoiding speed, stability, portability, usability issues associated with IMAS infrastructure

model predictive control faster than realtime –¿ projected horizion shot validator

Eg. Sample OMAS usage for mapping some equilibrium data to IMAS compatible schema

```python
# Load gEQDSK in OMFIT
eq=OMFITgeqdsk(OMFITsrc+'/../samples/g133221.01000')

# Instantiate new OMAS data structure (ODS)
ods=omas()

# ==== EQUILIBRIUM ====
# 0D data
ods['equilibrium.time_slice.0.global_quantities.ip']=eq['CURRENT']
ods['equilibrium.time_slice.0.global_quantities.magnetic_axis.r']=eq['RMAXIS']
ods['equilibrium.time_slice.0.global_quantities.magnetic_axis.z']=eq['ZMAXIS']
ods['equilibrium.time_slice.0.global_quantities.magnetic_axis.b_field_tor']=eq['BCENTR']*eq['RCENTR']/eq['RMAXIS']

# 1D data
ods['equilibrium.time_slice.0.profiles_1d.psi']=linspace(eq['SIMAG'],eq['SIBRY'],len(eq['PRES']))
ods['equilibrium.time_slice.0.profiles_1d.phi']=eq['AuxQuantities']['PHI']

# 2D data
ods['equilibrium.time_slice.0.profiles_2d.0.grid.dim1']=eq['AuxQuantities']['R']
ods['equilibrium.time_slice.0.profiles_2d.0.grid.dim2']=eq['AuxQuantities']['Z']
ods['equilibrium.time_slice.0.profiles_2d.0.b_field_tor']=eq['AuxQuantities']['Bt']
ods['equilibrium.time_slice.0.profiles_2d.0.psi']=eq['PSIRZ']
ods['equilibrium.time_slice.0.profiles_2d.0.phi']=eq['AuxQuantities']['PHIRZ']

# ==== WALL ====
ods['wall.description_2d.0.limiter.type.name']='DIII-D'
ods['wall.description_2d.0.limiter.type.index']=0
ods['wall.description_2d.0.limiter.type.description']='DIII-D first wall'
ods['wall.description_2d.0.limiter.unit.0.outline.r']=eq['RLIM']
ods['wall.description_2d.0.limiter.unit.0.outline.z']=eq['ZLIM']

# ==== SAVE/LOAD from/to pickle FILE ====
save_omas_pkl(ods,'test.omas')
ods1=load_omas_pkl('test.omas')
if not different_ods(ods, ods1):
  print('OMAS data got saved to and loaded from file correctly')

# ==== SAVE/LOAD from/to IMAS ====
paths=save_omas_imas(ods, user='meneghini', tokamak='D3D', version='3.10.1', shot=133221, run=0, new=True)
ods1=load_omas_imas(user='meneghini', tokamak='D3D', version='3.10.1', shot=133221, run=0, paths=paths)
if not different_ods(ods, ods1):
   print('OMAS data got saved to and loaded from IMAS correctly')
```

- ods
  - equilibrium
    - time_slice
      - 0
        - global_quantities
          - ip
          - magnetic_axis
            - b_field_tor
            - r
            - z
        - profiles_1d
          - phi
          - psi
        - profiles_2d
          - 0
            - b_field_tor
            - grid
              - dim1
              - dim2
            - phi
            - psi
        - time
  - wall
    - description_2d
      - 0
        - limiter
          - type
            - description
            - index
            - name
          - unit
            - 0
              - outline
                - r
                - z

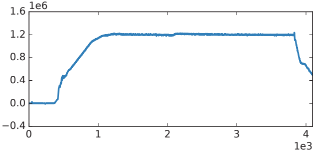
OMAS enhances familiar Python dictionaries and lists with functionalities useful for manipulating IMAS data

Different syntax to access data:

```
ods['equilibrium']['time_slice'][0]['profiles_2d'][0]['psi']   # standard Python dictionary format
ods['equilibrium.time_slice[0].profiles_2d[0].psi']            # IMAS hierarchical tree format
ods['equilibrium.time_slice.0.profiles_2d.0.psi']              # dot separated string format
ods[['equilibrium','time_slice',0,'profiles_2d',0,'psi']]      # list of nodes format
```

Slice array of structures:

```
plot(ods['equilibrium.time_slice.:.global_quantities.ip'])
```



Data validation and graceful error handling:

```
ods['equilibrium.time_slice.0.bad_location.ip']

Exception: `equilibrium.time_slice.0.bad_location` is not a valid IMAS location
                                     ^
                                     Did you mean: 'boundary'
                                                   'coordinate_system'
                                                   'ggd'
                                                   'global_quantities'
                                                   'profiles_1d'
                                                   'profiles_2d'
                                                   'time'
```

OMAS does the tedious heavy lifting so that it does not have to be done in the physics codes

Seamless handling of uncertain data:

```
ods['thomson_scattering.channel[0].t_e.data'] = unumpy.uarray(te,te_err)
```

from IMAS ↑ ↓ to IMAS

```
thomson_scattering%channel[0]%t_e%data
thomson_scattering%channel[0]%t_e%data%error_upper
```

Automatic grids interpolation:

```
# Define working coordinates
coordinates['equilibrium.time_slice.0.profiles_1d.psi'] = new_psi
with omas_environment(ods, coordsio=coordinates):
    plot('equilibrium.time_slice.0.profiles_1d.pressure')  # get data on working coordinates
```

Automatic Coordinate Conventions (COCOS) transformations:

```
# Automatic COCOS transformations
with omas_environment(ods, cocosio=2):
    ods['equilibrium.time_slice.0.profiles_1d.psi'] = psi_in_COCOS2 # set psi in COCOS2
    print(ods['equilibrium.time_slice.0.profiles_1d.psi'])  # get psi in COCOS2
print(ods['equilibrium.time_slice.0.profiles_1d.psi'])  # get psi in COCOS11
```

⇓

16 possible COCOS:

- Direction of φ
- Direction of θ
- Sign of $\nabla\varphi \times \nabla\psi$
- 2π normalization $\nabla\varphi \times \nabla\psi$

Calculation of derived quantities:

```
# calculate derived quantities
ods.physics_core_profiles_pressures()
ods['core_profiles.profiles_1d[0].ion[0]pressure']
ods['core_profiles.profiles_1d[0].ion[0]pressure_thermal']
ods['core_profiles.profiles_1d[0].ion[1]pressure']
ods['core_profiles.profiles_1d[0].ion[1]pressure_thermal']
ods['core_profiles.profiles_1d[0].pressure_thermal']
ods['core_profiles.profiles_1d[0].pressure_ion_total']
ods['core_profiles.profiles_1d[0].pressure_perpendicular']
ods['core_profiles.profiles_1d[0].pressure_parallel']
ods['core_profiles.profiles_1d[0].pressure']
ods['core_profiles.profiles_1d[0].pressure_electron_total']
ods['core_profiles.profiles_1d[0].pressure_fast']
```

OMAS can load/store IMAS data in a variety of formats

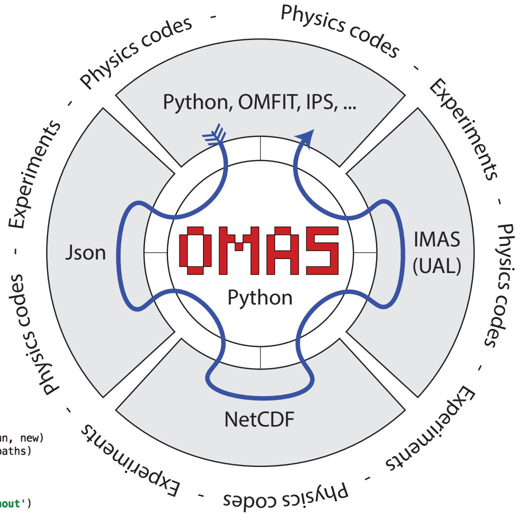
Already support for multiple storage systems:

Format	Storage type	Remote	Libraries required
omas	Python memory	-	-
pickle	Python binary file	-	-
Json	ASCII files	-	-
NetCDF	Binary files	-	netCDF4
HDF5	Binary files	-	h5py
S3	Object store	yes	boto
IMAS	Database	yes	imas
UDA	Database	yes	pyuda

- Users can choose in what format to save their data
- IMAS is just one of the supported formats
- Plugin approach makes it trivial to support new storage formats (eg. MDS+, ...)

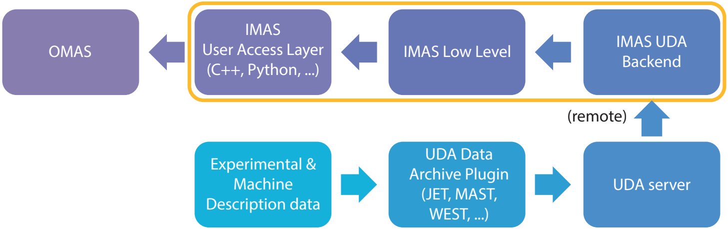
Eg. Save/Load OMAS data through different storage formats

```python
# load some sample data
ods_start = ods_sample()

# save/load Python pickle
filename = 'test.pkl'
save_omas_pkl(ods_start, filename)
ods = load_omas_pkl(filename)

# save/load ASCII Json
filename = 'test.json'
save_omas_json(ods, filename)
ods = load_omas_json(filename)

# save/load NetCDF
filename = 'test.nc'
save_omas_nc(ods, filename)
ods = load_omas_nc(filename)

# remote save/load S3
filename = 'test.s3'
save_omas_s3(ods, filename)
ods = load_omas_s3(filename)

# save/load IMAS
user = os.environ['USER']
tokamak = 'D3D'
version = os.environ.get('IMAS_VERSION','3.10.1')
shot = 1
run = 0
new = True
paths = save_omas_imas(ods,  user, tokamak, version, shot, run, new)
ods_end = load_omas_imas(user, tokamak, version, shot, run, paths)

# check data
if not different_ods(ods_start, ods_end):
   print('OMAS data got saved to and loaded correctly throughout')
```

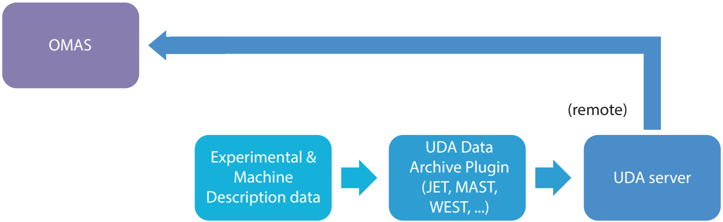


Direct OMAS interface to UDA greatly simplifies software stack to access IMAS data

Data has to traverse across many software layers



OMAS with UDA removes dependency on IMAS API altogether:



Open source: `pip install omas`
Documented: `http://gafusion.github.io/omas`

Installation — OMAS

https://gafusion.github.io/omas/install.html

OMAS | Concept | Examples | Installation | ITER | OMFIT | Data schema | API | In this page

Google: omas imas

Installation

OMAS runs both with *Python2* and *Python3*.

Pypi: To install OMAS with pip (for users):

```
pip install --upgrade omas
```

where *upgrade* is used to update the omas installation to the latest version.

The development version of omas can also be installed with pip:

```
pip install --upgrade -e git+git@github.com:gafusion/omas#egg=omas
```

Conda: To install OMAS with conda (for users):

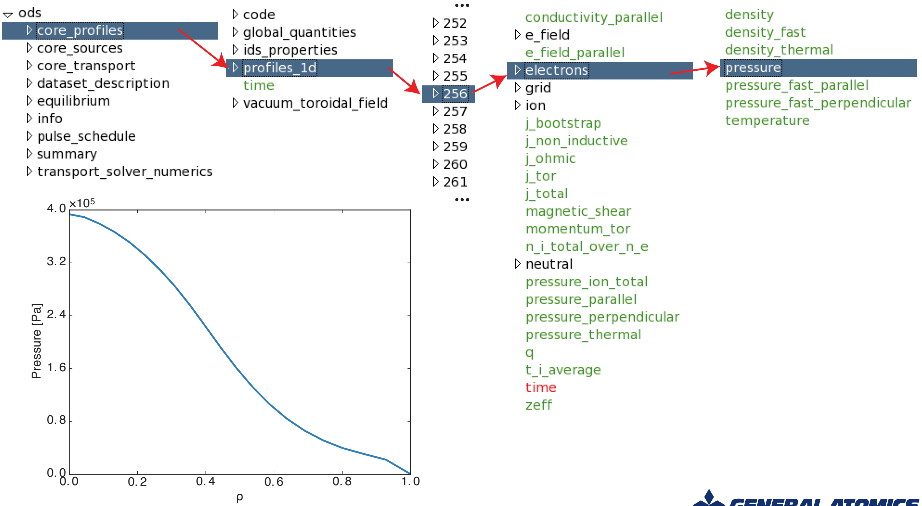
```
conda install -c conda-forge omas
conda update  -c conda-forge omas
```

GitHub To clone OMAS from GitHub (for developers):

```
git clone git@github.com:gafusion/omas.git
cd omas
sudo pip install --upgrade -e .[build_structures, build_documentation]      # Add this `omas` directory to your $PYTHONPATH
                                                                           # The [build_structures,build_documentation] options
                                                                           # install packages required for extra development purposes
```

Easy to start by accessing, exploring and working with data in the ITER IMAS scenario database (requires ITER account)

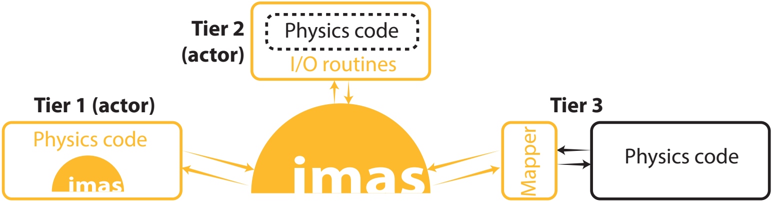
```python
import omas
ods = omas.load_omas_iter_scenario(shot=130010, run=1)
plot( ods['core_profiles']['profiles_1d'][256]['electrons']['pressure'] )
```



GENERAL ATOMICS

Interfacing OMFIT with ITER IMAS via OMAS

Three possible tiers of physics codes integration with IMAS



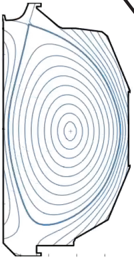
1. IMAS actors use IDS for internal data structures and I/O
   - Viable solution only for brand new codes
   - Codes depend on IMAS to run
2. IMAS actors use IDS for I/O (actors)
   - Requires modifying existing physics codes
   - Maintain two I/O systems to be able to run independently of IMAS
3. Translate legacy file formats to and from IDSs
   - Requires writing wrappers around legacy file formats
   - No changes to existing codes, which run independently of IMAS

OMFIT supports IMAS integration with all these tiers

OMFIT classes `.to_omas()` and `.from_omas()` provide an effective way to simplify tier 3 codes integration

OMFIT + OMAS

Mapper



```python
from omfit.classes.omfit_eqdsk import OMFITgeqdsk, OMFITsrc
from omas import *

# read gEQDSK file in OMFIT
eq = OMFITgeqdsk(OMFITsrc + '/../samples/g133221.01000')

# convert gEQDSK to OMAS data structure
ods = eq.to_omas()

# save OMAS data structure to IMAS
paths = save_omas_imas(ods, tokamak='DIII-D', new=True)

# load OMAS data structure from IMAS
ods1 = load_omas_imas(user, tokamak='DIII-D', paths=paths)

# generate gEQDSK file from OMAS data structure
eq1 = OMFITgeqdsk('g133221.02000').from_omas(ods1)

# save gEQDSK file
eq1.deploy()
```

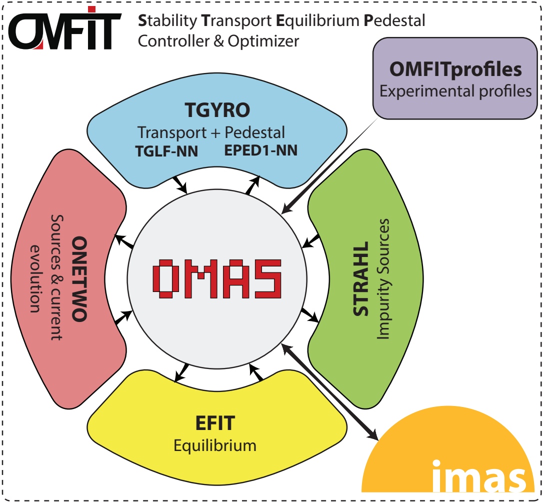
imas

- Many legacy codes share the same file formats!

Example OMFIT-IMAS tier 3 integration for self-consistent 1.5D core-pedestal scenario modeling

→ Friday talk on use of STEP for ITER modeling

- OMFIT STEP module combines codes (*"steps"*) to support arbitrary workflows
  - open loop prediction
  - control
  - optimization
- Data exchanged between steps always as IDSs via OMAS
- Can be initialized from different OMFIT modules and IMAS
- Results can be written to IMAS at any stage



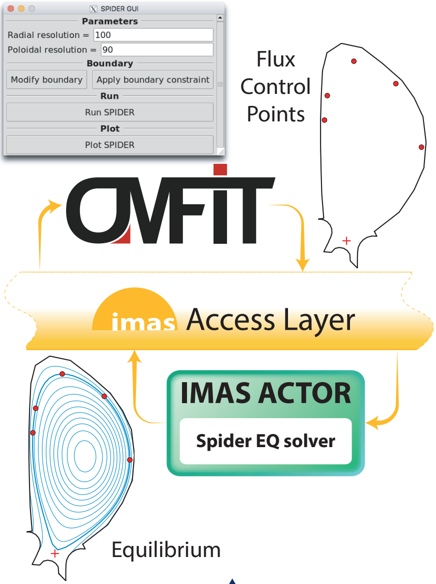
Example OMFIT-IMAS tier 1&2 integration for existing IMAS Python actors originally developed for the Kepler framework

EUROFUSION devoted significant effort to adapt EU codes to work with IMAS (Tier 1&2)

- Large library of IMAS actors is available
- Typically run via Kepler framework
- But can be run directly from Python too 👍

OMAS enables seamlessly data transfer from OMFIT to IMAS and actors execution

Game-changer: Convenience and readiness of OMFIT to run workflows of IMAS Python actors



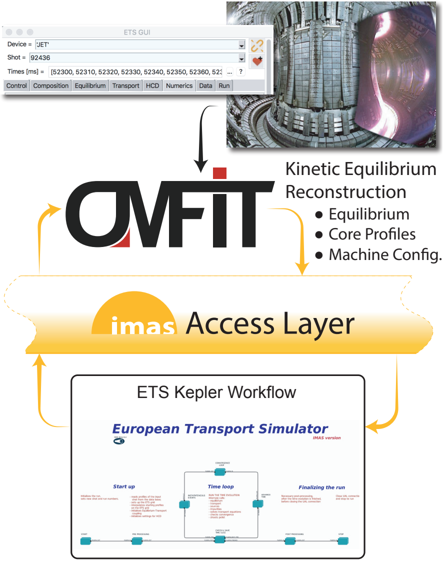
Example OMFIT-IMAS tier 1&2 integration for European Transport Solver (ETS) Kepler workflow

ETS is a new modular transport solver completely developed within the the Kepler framework

ETS module in OMFIT:

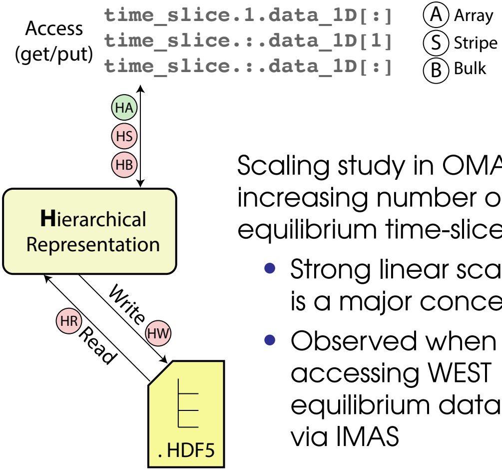
(J. Ferreira - IPFN & M. Romanelli - CCFE)

1. Prepares input data for ETS
   - Use data from OMFIT kinetic equilibrium reconstruction module (DIII-D, NSTX, **JET**, **MAST**, C-Mod, KSTAR, AUG, COMPASS, ...)
2. Provides user-friendly GUI
3. Executes Kepler workflow
4. Visualizes simulation results



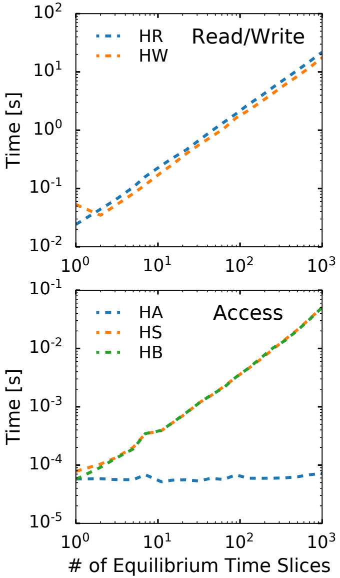
Interfacing OMFIT with ITER IMAS via OMAS

Hierarchical organization hinders IMAS's ability to efficiently manipulate large data sets



Scaling study in OMAS for increasing number of equilibrium time-slices

- Strong linear scaling is a major concern
- Observed when accessing WEST equilibrium data via IMAS



GENERAL ATOMICS

Proposed solution: Adopt tensors representation that is commonly used by HPC and ML applications

Prototyped in OMAS **mapping between hierarchical and tensors representations**

- Requires constant grids across arrays of structures
  - Across list of time slices, ions, sources, ...
  - Virtually always true! Adaptive grids are rarely used
- Extra tensor dimension could be used to efficiently store samples from distribution of uncertain quantities

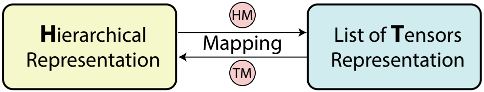
Hierarchical Repr.	Dimensions
[time_slice]	
[1]	
[data_0D]	()
[data_1D]	(x_dim)
[data_2D]	(r_dim,z_dim)
[2]	
[data_0D]	()
[data_1D]	(x_dim)
[data_2D]	(r_dim,z_dim)
. . .	
[n]	
[data_0D]	()
[data_1D]	(x_dim)
[data_2D]	(r_dim,z_dim)

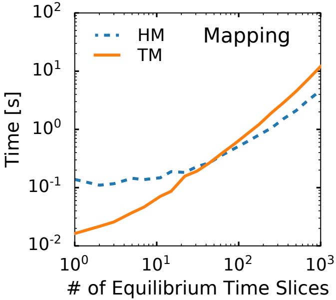
↑↓

List of Tensors Repr.	Dimensions
[time_slice.:.data_0D]	(t_dim)
[time_slice.:.data_1D]	(t_dim,x_dim)
[time_slice.:.data_2D]	(t_dim,r_dim,z_dim)

Mapping between hierarchical and tensor representations is also costly

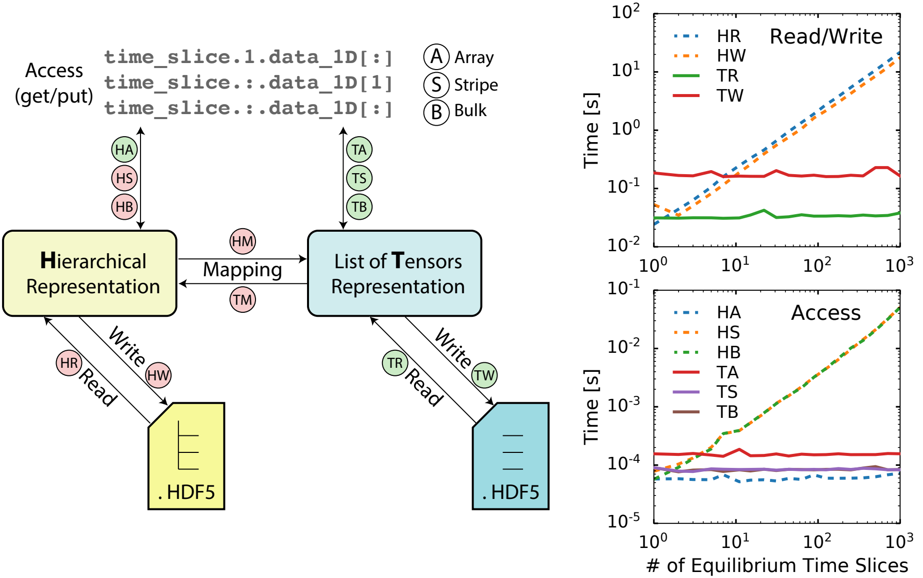
- Keep it to a minimum
  (write once read multiple times)
- Avoid it altogether
  (in memory tensor representation)





OMAS uses the same API independently of in-memory data representation (hierarchical or tensors) 👍

Tensors representation provides significantly better data storage and access performance for increasing dataset size



Interfacing OMFIT with ITER IMAS via OMAS

1 OMFIT framework and IMAS data dictionary

2 Manipulating IMAS data with OMAS library

3 Integrated modeling with OMFIT and IMAS

4 Scaling IMAS performance for HPC and ML

5 Conclusions

OMFIT framework is now fully compatible with the ITER Integrated Modeling and Analysis Suite

OMFIT free-form data structure supports different fusion formats

- IMAS is yet another data format

Powerful OMAS library simplifies interaction with IMAS in Python

- Open source, documented, and independent of OMFIT
- Leveraged by OMFIT to interface with IMAS
- Tensors representation could address IMAS scaling issues

OMFIT-IMAS integration actively used for leading edge fusion research

- Tier 3 example: STEP module
- Tiers 1&2 example: ETS Kepler workflow

Game-changing ability to combine benefits of OMFIT and IMAS actors

- Convenience and omnipresence of OMFIT
- Vast library of EUROFUSION IMAS actors
- Familiarity of Python

Following is a simplified summary slide for tensor representation

Tensors representation can be used to scale IMAS performance to handle large datasets

- Hierarchical representation does not allow bulk read/write of data
- Tensor representation commonly used for HPC and ML applications
- Mapping requires constant grids across arrays of structures
  - Across list of time slices, ions, sources, ...
  - Virtually always true, since adaptive grids are rarely used
- Prototyped and tested within OMAS library