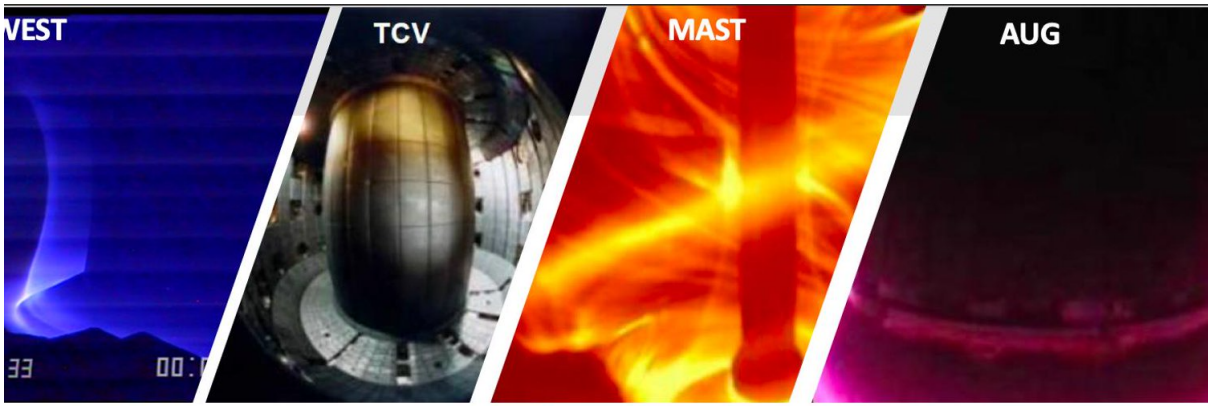


EUROfusion Integrated Modelling workflows Documentation

Release 3.0

WPCD

Jan 23, 2020

CONTENTS

CHAPTER

ONE

INTRODUCTION TO THE EUROFUSION PROJECT CODE DEVELOPMENT FOR INTEGRATED MODELLING

The EUROfusion Project on Code Development for Integrated Modelling (WP-CD) supports the achievement of the European Fusion Roadmap at Horizon 2020 goals, via the development of existing modelling codes with a particular focus on integrated modelling. The primary objectives of WPCD are:

1. Provide a suite of codes that can be validated on existing Tokamaks and used for JT-60SA, ITER and DEMO predictions:
    - build on the existing modelling codes developed by the EUROfusion Consortium members including the Integrated Modelling (EU-IM) infrastructure, toolset and codes developed under the former EFDA ITM Task Force,
    - add new physics to the existing models
    - couple codes into integrated workflows
    - optimize codes.
2. Specific ITER simulation work in support of ITER IO and F4E with specified deliverables.

WPCD operates under a work plan aiming to provide in the long term a full suite of integrated simulation workflows, incorporating core-edge-SOL/PFC coupling, first-principles models and control elements. A central task is the development of the modular European Transport Simulator, ETS, which is being deployed to JET and MST modelling infrastructures for validation and application to experimental analysis.

In addition to code and workflow development, rigorous code verification is also performed under WPCD, within the EU-IM framework; whereas validation of the released integrated modelling workflows against the experiments is performed under WPJET1 and WPMST1.

The Work Package is run as a project and managed by a project leader (M. Romanelli, UKAEA, michele.romanelli@ukaea.uk)

In 2019 the structure of the project has been reviewed and changed.

The project is now organised into three coordinated areas or subprojects reflecting the present priorities: Enabling Workflow Exploitation, Workflow Development, Workflow adaptation to IMAS. The Enabling Workflow Exploitation Area (EWE) will coordinate the development of pre-processing tools for the routine use of ETS and the equilibrium-MHD-stability workflow in EUROfusion facilities, the development of visualization tools, synthetic diagnostics and the provision of training to users. The Workflow Development Area or subproject will coordinate the continuous development of existing and new workflows in IMAS addressing specific modelling needs of the other EUROfusion work packages. The Workflow

adaptation to IMAS Area will operate in strong collaboration with ITER IO and will ensure the complete adaptation of existing workflows to IMAS using the most updated Data Dictionary.

1.1 The European Integrated Modelling (EU-IM) approach

The choice of Integrated Modelling made by the former EFDA ITM and pursued now under EUROfusion WPCD is unique and original: it entails the development of a comprehensive and completely generic tokamak simulator including both the physics and the machine, which can be applied for any fusion device. The simulation platform was designed to be fully modular, flexible, and independent of a programming language.The choice of modularity implies that each module contains a single physical model and that the communication between the modules is standardised: a set of common common rules (ontology) clearly specify the format of the data to be consistently exchanged between modules (data-structure). The complexity of coupling the codes together is therefore transferred to the definition of a generic data-structure (allowing to describe and exchange information concerning both physical quantities and technical objects, not assuming the origin of those), extensible to allow the integration of new physics, as well as more elaborate machine geometries and experimental data in the future. A central project is the development of the so-called **European Transport Simulator (ETS)** aimed to meet all the EU-IM requirements, namely modularity, flexibility and standardized interfaces. In terms of the physics, the ETS is designed to solve the standard set of one-dimensional time dependent equations which describe the evolution of the core plasma. The solver itself is designed with a modular approach enabling the separation of the physics from the numerics, thereby facilitating the testing/usage of the numerical schemes that best suit a particular physical simulation.

1.2 Mission

The European Integrated Tokamak Modelling Task Force (ITM-TF) operated under EFDA from 2004 until 2013. The main mission of the ITM TF was to provide a software infrastructure framework for EU integrated modelling activities (EU-IM) as well as a validated suite of simulation codes for the modelling of present experiments, ITER and DEMO plasmas. The ITM TF operated until 2013 under a work programme formulated to support this goal, structuring the EU modelling effort around existing experiments and ITER scenario prediction while maintaining a long term strategic aim to provide a validated set of European modelling tools for ITER exploitation. The EU-IM effort was then pursued under the EUROfusion project WPCD, progressing towards the achievement of a main milestone, “Extended Core Transport simulator used for analysis of JET1 and MST1 campaigns and developing facilities”.

1.3 Achievements

During the first phase of the EU-IM effort, surveys and cross-verification of the available European models and numerical codes were performed within the individual integrated modelling projects and the data-structure was extensively discussed and defined. Equilibrium, linear MHD stability, core transport and RF wave propagation, as well as the poloidal field systems and a few diagnostics were the first topics addressed. Data structures were finalised for these and then expanded to address, among others, non-linear MHD, edge physics, turbulence and neutral beam propagation. In parallel to the development of the physics concepts, the EU-IM effort developed the tools to manipulate the data structure and use it in fully flexible and modular simulation workflows. The EU-IM database contains machine descriptions from JET, Tore Supra, MAST, (as well as FTU, FAST), AUG, ITER, JT-60SA as well as some experimental data from Tore Supra (WEST) and JET. The EU-IM futher achieved the development of the first

release version of a fully modular and versatile transport simulator, the ETS, with all the essential functionalities. The validation of the ETS simulator first started in 2010 against the state-of-the-art transport codes and ETS recently started to be used for the first physics applications. Next steps are the validation of the simulator for a complete discharge on existing experimental data with the available modules, the integration of more quantitative physics models ("ab-initio") and the integration of the whole modelling of the device.

The main WPCD achievements are listed below: 1. An high-resolution equilibrium and linear MHD stability chain, for core and pedestal, applicable to peeling-ballooning type instabilities has been released for the analysis of equilibria from any tokamak integrated in the EU-IM platform, including ITER and DEMO. A predictive J-alpha MHD pedestal stability analysis workflow has also been developed and is in test release stage. 2. The fixed boundary core European Transport Simulator ETS, with various equilibrium modules and a full hierarchy of transport models, impurities, pellets, neutrals, sawteeth, Neoclassical Tearing Modes (NTM) modules, and full integration of Heating and Current Drive sources (Electron Cyclotron, Neutral Beam Injection, Ion Cyclotron, alpha), including synergies has been released.The released ETS workflow has been implemented in JET modelling infrastructure and went through validation. 3. A feedback controlled free boundary transport simulator prototype is operational and under verification. 4. A Scrape-Off-Layer (SOL) turbulence workflow including a synthetic probe, directly reading from experimental database has been developed and applied to analyse ASDEX-Upgrade divertor power deposition. 5. Benchmarks of EC, IC and NBI codes within the EU-IM infrastructure were carried out on identified test cases and presented in conference (Topical Conference on Radiofrequency Power in Plasmas, EPS, IAEA Technical Meeting on Energetic Particles (EP)). 7. Prototypes of self-consistent coupling between core and edge transport codes were demonstrated, in particular automated direct coupling of the ETS core transport code to the 2D edge transport code SOLPS. 8. SOLPS technical optimization studies (parareal algorithm, speed-up techniques, reduced physics models) provided an assessment of speed-up techniques to be possibly integrated in SOLPS-ITER. 9. A prototype acyclic workflow for modelling the SOL and interaction with Plasma Facing Components (PFC) was demonstrated by coupling the 2D transport code SOPLS to the 3D Monte Carlo PWI and impurity transport code ERO.

1.4 Publications

-G.L. Falchetto, et al., and the EUROfusion-IM Team, MULTI-MACHINE ANALYSIS OF EU EXPERIMENTS USING THE EUROFUSION INTEGRATED MODELLING (EU-IM) FRAMEWORK, P1.1081, 46th EPS conference, Milan, 2019.

-G.I. Pokol, et al., "Runaway electron modelling in the ETS self-consistent core transport simulator", Nuclear Fusion 59, 076024 (2019). https://doi.org/10.1088/1741-4326/ab13da

-Y.-S. Na et al.,"On Benchmarking of Simulations of Particle Transport in ITER", Nuclear Fusion 59 (7), 076026, 2019.

-A.H. Nielsen, et al. "Synthetic edge and SOL diagnostics - a bridge between experiments and theory", THD/P7-4 IAEA CN-258 2018. https://conferences.iaea.org/indico/event/151/papers/5806/files/4686-Nielsen-THD-P7-4.pdf, Nuclear Fusion accepted

-R. Coelho, et al., "Plasma equilibrium reconstruction of JET discharges using the IMAS modelling infrastructure", TH/P5-27, IAEA CN-258, 2018. https://conferences.iaea.org/indico/event/151/papers/6136/files/4812-Paper_IAEA_Coelho_v7.pdf

-P. Strand, et al., "Towards a predictive modelling capacity for DT plasmas: European Transport Simulator (ETS) verification and validation", TH/P6-14, IAEA CN-258, 2018. https://conferences.iaea.org/indico/event/151/papers/5943/files/4801-IAEA_FEC18_THP6_14_Strand.pdf

-S. Nowak, et al., "Analysis and modelling of NTMs dynamics in JET discharges using the European Transport Simulator (ETS) and integrated modelling tools", TH/P6-26, IAEA CN-258 2018. https://conferences.iaea.org/indico/event/151/papers/6039/files/4878-64930_snowak_iaea2018_paper_final_rev.pdf

-G.I. Pokol, et al., "Runaway electron modelling in the ETS self-consistent core transport simulator", TH/P8-15, IAEA CN-258, 2018. https://conferences.iaea.org/indico/event/151/papers/6162/files/4621-Pokol_IAEA-FEC_2018-paper.pdf

-V. Basiuk, P. Huynh, A. Merle, S. Nowak, O. Sauter, "Towards self-consistent plasma modelisation in presence of neoclassical tearing mode and sawteeth: effects on transport coefficients", Plasma Phys. Control. Fusion 59 (12), 125012 (2017) https://doi.org/10.1088/1361-6587/aa8c8c

-D. Samaddar, D.P. Coster, X. Bonnin, C. Bergmeister, E. Havlickova, L.A. Berry, W.R. Elwasif, D.B. Batchelor, "Temporal parallelization of edge plasma simulations using the parareal algorithm and the SOLPS code", Computer Physics Communications 221, 19-27 (2017). https://doi.org/10.1016/j.cpc.2017.07.012

-M. Baelmans, P. Borner, K. Ghoos, G. Samaey, "Efficient code simulation strategies for B2-EIRENE", Nuclear Materials and Energy 12, 858-863 (2017) https://doi.org/10.1016/j.nme.2016.10.028

-D.P. Coster, "Exploring the edge operating space of fusion reactors using reduced physics models", Nuclear Materials and Energy 12, 1055-1060 (2017) https://doi.org/10.1016/j.nme.2016.12.033

-G.L. Falchetto, et al., and the EUROfusion-IM Team, "EUROfusion Integrated Modelling (EU-IM) capabilities and selected physics applications", Proc. 26th IAEA Fusion Energy Conference, Kyoto, Japan, IAEA CN-234, TH/ P2-13, 2016. https://nucleus.iaea.org/sites/fusionportal/Shared%20Documents/FEC%202016/fec2016-preprints/preprint0362.pdf

-Y.-S. Na et al., "On Benchmarking of Simulations of Particle Transport in ITER", Proc. 26th IAEA Fusion Energy Conference, Kyoto, Japan, TH/P2-24, IAEA CN-234, 2016.

-G.L. Falchetto, et al.,and ITM-TF contributors, "The European Integrated Tokamak Modelling (ITM) effort: achievements and first physics results", Nuclear Fusion 54, 043018, 2014.

-D. Kalupin et al, "Numerical analysis of JET discharges with the European Transport Simulator", Nucl. Fusion 53, 123007, 2013.

-D. Penko "3D tokamak Wall description within ITER Integrated Modelling and Analysis (IMAS) framework", Proc. 45th European Physical Society Conference on Plasma Physics (EPS), 2018.

-O. Asunta, R. Coelho, D. Kalupin, T. Johnson, T. Franke and EU-IM Team, "Predictions of neutral beam current drive in DEMO using BBNBI and ASCOT within the European Transport Simulator", 42nd EPS 2015, P2.156 ECA Vol. 39E ISBN 2-914771-98-3.

-R. Bilato, N. Bertelli, M. Brambilla, R. Dumont, E.F. Jaeger, T. Johnson, E. Lerche, O. Sauter, D. Van Eester and L. Villard, "Status of the benchmark activity of ICRF full-wave codes within EUROfusion WPCD and beyond", 21st Topical Conference on Radiofrequency Power in Plasmas, 2015.

Some posters describing the ITM-TF achievements were presented in an "ITM EXPO" at the 2011 EPS fusion conference in Strasbourg.

1.5 Contributors

The EUROfusion-IM Team members are defined in the link: http://euro-fusionscipub.org/eu-im

ITM-TF contributors were defined in the Appendix of G.L. Falchetto et al., Nuclear Fusion 54,043018, 2014. This list reproduces the status of of members in 2012 and is not exhaustive. A grateful thank you to all those who contributed and promoted EU-IM since its beginnigs.

1.6 Glossary

Collaborative Development Environment (CDE) A **collaborative development environment (CDE)** is an online meeting space where a software development project's stakeholders can work together, no matter what timezone or region they are in, to discuss, document , and produce project deliverables. The name was coined by Grady Booch.

Consistent Physical Object (CPO) A Consistent Physical Object (CPO) is a physics based, hierarchical data structure employed by the EU-IM for a complete description of a physics area, e.g. equilibrium. All EU-IM code modules interact through the exchange of CPOs. The CPOs also form the basic block of data written to the EU-IM database.

Content Management System (CMS) A **content management system (CMS)** is the collection of procedures used to manage work flow in a collaborative environment. These procedures can be manual or computer-based. The procedures are designed to:

- Allow for a large number of people to contribute to and share stored data
- Control access to data, based on user roles. User roles define what information each user can view or edit
- Aid in easy storage and retrieval of data
- Reduce repetitive duplicate input
- Improve the ease of report writing
- Improve communication between usersq

In a CMS, data can be defined as nearly anything - documents, movies, pictures, phone numbers, scientific data, etc. CMSs are frequently used for storing, controlling, revising, semantically enriching, and publishing documentation.

FC2K FC2K is a tool for wrapping a Fortran or C++ source code into a Kepler actor. Before using it, your physics code should be EU-IM-compliant (i.e. use CPOs as input/output).

Gforge Gforge hosts all projects (software and infrastructure) under the EU-IM.

EUROfusion Gateway The EUROfusion Gateway is a computer cluster presently hosted at CINECA, Bologna, Italy. It is used as central repository of the EU-IM software, as well as platfrom for developments and fusion simulations.

EU-IM Portal The EU-IM Portal is the web portal for the EU-IM, i.e. it hosts the EU-IM web pages and projects under Gforge.

Universal Access Layer (UAL) The UAL (Universal Access Layer) is a multi-language library that allows exchanging Consistent Physical Objects (CPOs) between various modules, and to write to an EU-IM database.

actor Modular element within a Kepler scientific workflow. Actors take execution instructions from a director. In other words, actors specify what processing occurs while the director specifies when it occurs. In the EU-IM Kepler workflows, most actors are modules which contain physics codes.

calibration The process of adjusting numerical or physical modelling parameters in the computational model for the purpose of improving agreement with experimental data.

data mapping An XML file containing all the mapping essentials for mapping from a local experimental database for a specific tokamak device to the EU-IM database. The mapping essentials include for instance the download method, local signal names, gains and offsets, time base, and eventual interpolation option to ensure that only one time base is set for each CPO that is built from multiple local signals. A java code (exp2ITM developed under ISIP), with the MD and DM files as inputs, is then run to connect to the local device database, retrieve the required experimental data and populate the EU-IM database instance for that shot/device and dataversion.

director A director controls (or directs) the execution of a workflow, just as a film director oversees a cast and crew.

error A recognisable deficiency in any phase or activity of modelling and simulation that is not due to lack of knowledge.

kepler Kepler is a software application for the analysis and modeling of scientific data. Kepler simplifies the effort required to create executable models by using a visual representation of these processes. These representations, or "scientific workflows", display the flow of data among discrete analysis and modeling components.

machine description The machine description (MD) of a device builds on the set of engineering and diagnostic settings characterising a tokamak device. This includes, for instance, the vessel/limiter description, the PF coils and circuiting and lines of sight of diagnostics. In practice, all MD information is encapsulated in an XML file that emanates from the MD tagged datastructure schemas. An MD instance of a given device is then stored into the EU-IM database as shot 0 for that device database.

model A representation of a physical system or process intended to enhance our ability to understand, predict, or control its behaviour.

- A **conceptual model** consists of the observations, mathematical modelling data, and mathematical (e.g., partial differential) equations that describe the physical system. It will also include initial and boundary conditions.
- The **computational model** is the computer program or code that implements the conceptual model. It includes the algorithms and iterative strategies. Parameters for the computational model include the number of grid points, algorithm inputs, and similar parameters, etc.

modelling The process of construction or modification of a model

prediction Use of a model to foretell the state of a physical system under conditions for which the model has not been validated.

simulation The exercise or use of a model.

uncertainty A potential deficiency in any phase or activity of the modelling process that is due to the lack of knowledge.

validation The process of determining the degree to which a model is an accurate representation of the real world form the perspective of the intended uses of the model.

verification The process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model.

CHAPTER

TWO

INFRASTRUCTURE

The infrastructure documentation is hosted at the link below

https://confluence.man.poznan.pl/community/display/WFMS

In a workflow, physics modules exchange physics data in the form of standardised blocks of information: the Consistent Physical Objects (CPOs). The list of CPOs as well as their inner structure defines the EU-IM Data Structure. All physics modules should use these standardised interfaces for I/O.

The most recent datastructure for EU-IM workflows can be browsed at the link below

> Data structure 4.10b.10

2.1 Kepler

2.1.1 Introduction to Kepler - basics

Kepler is a workflow engine and design platform for analyzing and modeling scientific data. Kepler provides a graphical interface and a library of pre-defined components to enable users to construct scientific workflows which can undertake a wide range of functionality. It is primarily designed to access, analyse, and visualise scientific data but can be used to construct whole programs or run pre-existing simulation codes.

Kepler builds upon the mature Ptolemy II framework, developed at the University of California, Berkeley. Kepler itself is developed and maintained by the cross-project Kepler collaboration.

The main components in a Kepler workflow are actors, which are used in a design (inherited from Ptolemy II) that separates workflow components ("actors") from workflow orchestration ("directors"), making components more easily reusable. Workflows can work at very levels of granularity, from low-level workflows (that explicitly move data around or start and monitor remote jobs, for example) to high-level workflows that interlink complex steps/actors. Actors can be reused to construct more complex actors enabling complex functionality to be encapsulated in easy to use packages. A wide range of actors are available for use and reuse.

2.1.1.1 Installing Kepler and tutorial workflows

You can download Kepler from the following page https://kepler-project.org/users/downloads

In order to install Kepler and tutorial related workflows you have to follow the instruction at

https://confluence.man.poznan.pl/community/display/WFMS/5.3.+Kepler+Basics#id-5.3.KeplerBasics-1InstallingKepler

Now you can start Kepler application and proceed to tutorial examples

2.1.2 Kepler IMAS actors

Imas actor	Deacription
UALSliceCollector	Store one slice from input IDS into different run. This way, it is possible to collect intermediate results during workflow execution.
UALPython	Allows to run external Python process and pass input/output data between workflow and process itself. This actor is, most commonly, used for data visualization. User can pass Python script directly to the actor.
UALMuxParam	Provides similar behavior to ualmux/UALMux, however, this actors has two additional ports: - fieldDescription - contains name of the filed that will be modified - fieldValue - contains new value of the field Main difference between ualmux/ ualmuxparam actors lays in it's ability to be used in a loop that modify different field inside IDS. You can simply provide different field name for different loop's step.
UALMux	Provides a method for putting data inside IDS inputs - inputIds/inputCpo - cpo we are going to modify - inTime - time index at which data are supposed to be updated - name of the field is specified as port name - new value of the field is passed as value sent to the port outputs: - outputIds/outputCpo - modified IDS - outTime - actual time index (depend on approximation mode)
UALInit	Initializes input pulse file, creates run work and provides ID S description for other actors inputs: - user - name of the user for input data file

2.1.3 IMAS Kepler based configuration

2.1.3.1 Running Kepler using IMAS environment

2.1.3.1.1 Setting up environment

Please do not forget to set JAVA memory settings:

```
export _JAVA_OPTIONS="-Xss20m -Xms8g -Xmx8g"
```

2.1.3.1.1.1 Backing up old files

Before first configuration of Kepler, make sure to backup your old data files

```
cd ~
mv .kepler .kepler~
mv KeplerData KeplerData~
mv .ptolemyII .ptolemyII~
```

2.1.3.1.2 Creating place to store your personal installations of Kepler

IMAS based installations are stored inside **$HOME/kepler** directory.

Before proceeding further, make sure to create **kepler** directory

```
# create directory inside $HOME
cd ~
mkdir kepler
```

2.1.3.1.3 Running Kepler (default release)

In order to start Kepler you have use helper scripts that will install and configure your personal copy of Kepler

- load IMAS module

```
module load imas
module load kepler
# NOTE! It might be that you don't have Kepler copy inside your $HOME
# in that case you need to install it kepler_install_light
```

- Start Kepler

```
# run alias that will execute Kepler
kepler
```

2.1.4 FC2K - Embedding user codes into Kepler

This tutorial is designed to introduce the concept of using FC2K tool in order to build Kepler compatible actors.

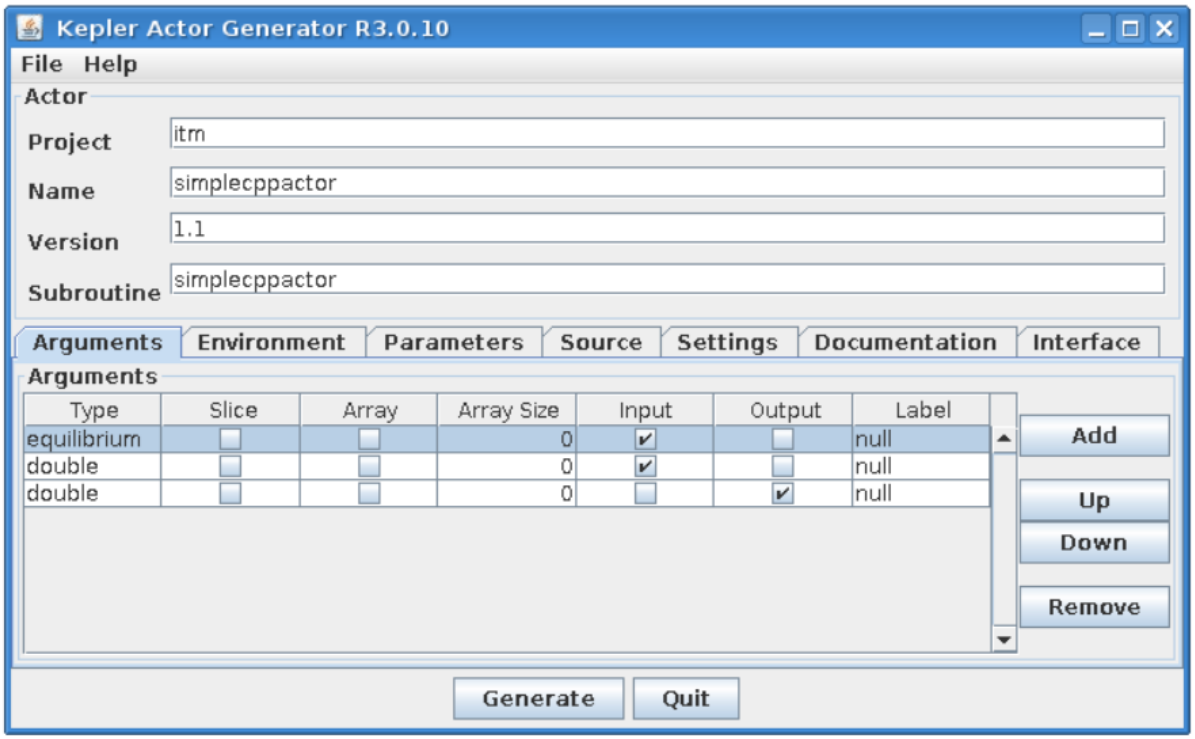
```
This tutorial explains
how to set up codes for FC2K
how to build actor using FC2K
how to incorporate actor within Kepler workflow
```

2.1.4.1 FC2K basics

2.1.4.1.1 What FC2K actually does?

- Generates a Fortran/CPP wrapper, which intermediates between Kepler actor and user code in terms of:
  - reading/writing of in/out physical data (IDS)
  - passing arguments of standard types to/from the actor
- Creates a Kepler actor that:
  - calls a user code
  - provides error handling
  - calls debugger (if run in "debug" mode)
  - calls batch submission commands for MPI based actors

2.1.4.1.2 FC2K main window

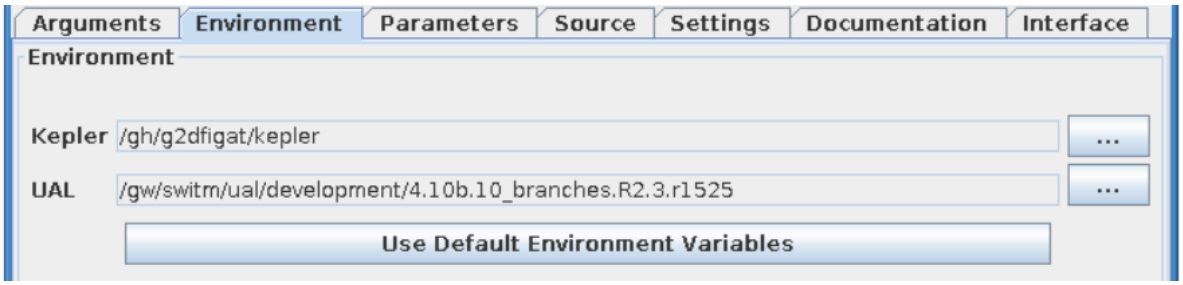


2.1.4.1.3 Actor description

This group of graphical controls allows to set the description of the actor and its "place" in hierarchy of Kepler elements in Kepler "Component" browser

- **Project** - defines a branch in Kepler "Component" browser where an actor will be placed
- **Name** - a user defined name of the actor
- **Version** - a user defined version of user codes
- **Subroutine** - A name of user subroutine (Fortran) or function (C++)

2.1.4.1.4 Environment

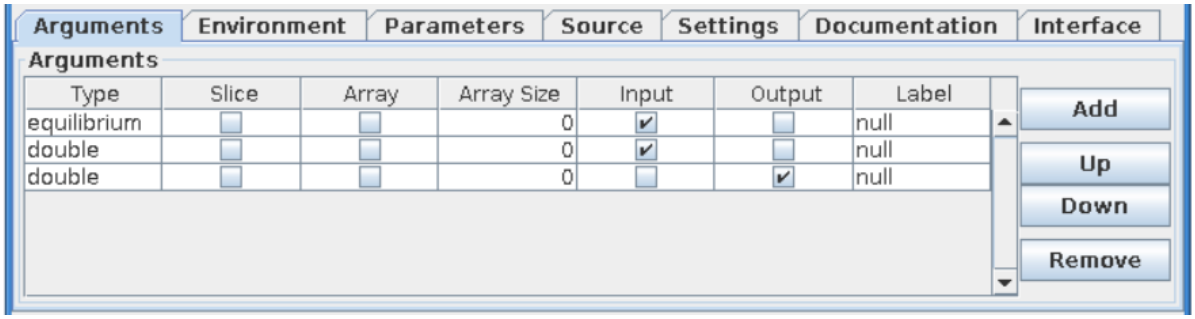


The Environment text fields shows UAL and Kepler locations.

- **Kepler** - Kepler location (usually the same as $KEPLER)
- **UAL** - IMAS UAL location (usually the same as $IMAS_PREFIX)

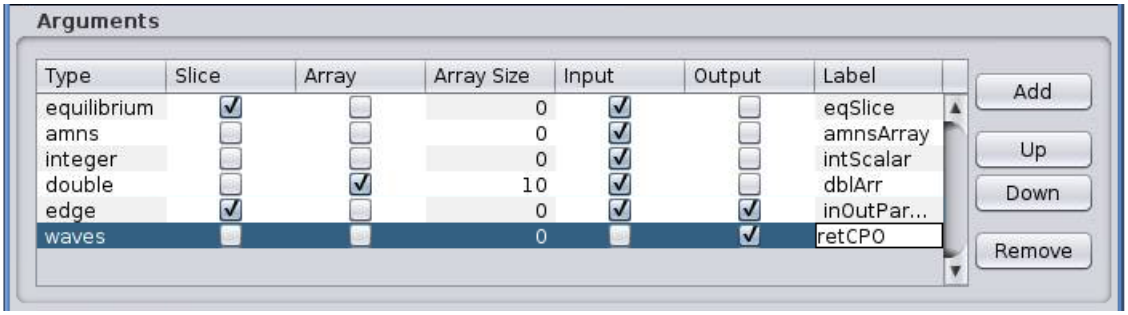
2.1.4.1.5 "Arguments" tab explained

Below you can find explanation of FC2K arguments tab.



- **Type** - Defines a type of an argument. It is possible to choose either IDS based type (e.g. equilibrium, topinfo, etc.) or primitive type (e.g. int, long, double, char)

- **Single slice** - Determines if IDS is passed as single slice or an array. (This setting is valid for IDS types only)
  - if turned **ON** - Only one slice is passed. An actor will get an additional port to define a time.
  - if turned **OFF** - All IDSes for given shot run is passed.
- **Is array** - Determines if a primitive type is passed as a scalar or an array
  - if turned **ON** - An argument is passed as an array. It requires definition of array size (dynamic array are not supported)
  - if turned **OFF** - An argument is passed as a scalar.
- **Array size** - Defines the size of an array of primitive types
- **Input** - Defines argument as an input
- **Output** - Defines argument as an output
- **Label** - User defined name of an argument (and actor port)

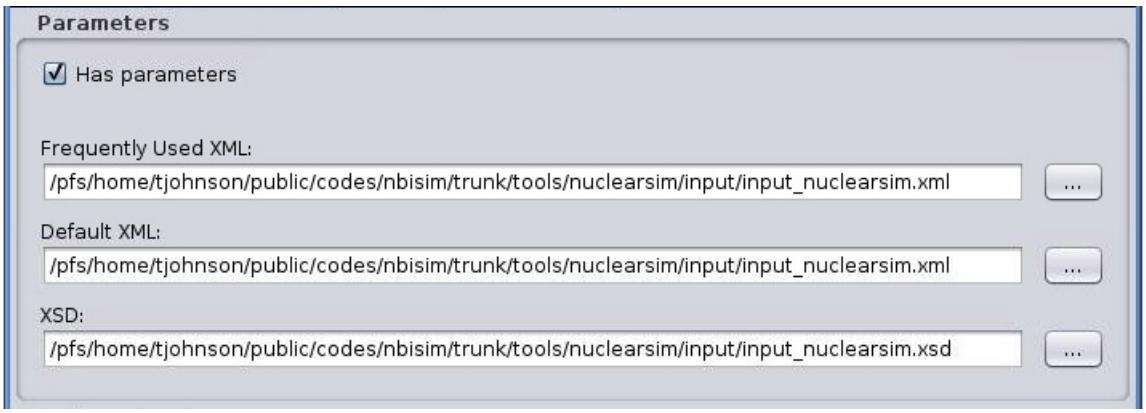


Please take a look on a screenshot above:

- *equilibrium* - an input parameter - one IDS (slice)
- *amns* - an input parameter - all amns IDS slices stored in given shot/run
- *integer* - an input parameter - a scalar
- *double* - an input parameter - an array of size 10
- *edge* - an in/out parameter - single slice of “edge” IDS
- *waves* - an output parameter - all slices of “waves” IDS

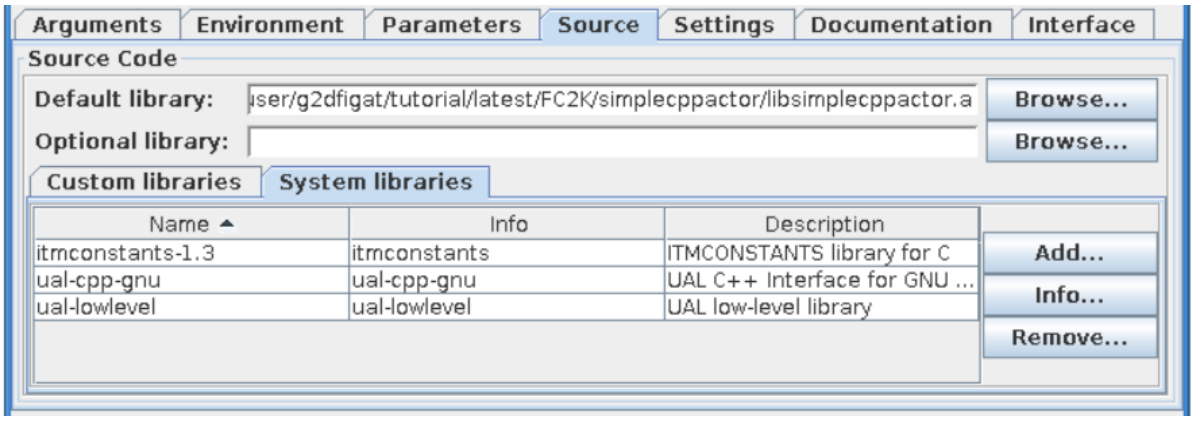
2.1.4.1.6 “Parameters” tab explained

Code specific parameters are all parameters which are specific to the code (like switches, scaling parameters, and parameters for built-in analytical models) as well as parameters to explicitly overrule fields in the ITM data structures.



- **Frequently Used XML** - Actual value of the code parameters
- **Default XML** - Default values of the code parameters
- **Schema** - A (XSD) XML schema

2.1.4.1.7 "Source" tab explained



The purpose of this tab is to define all code related issues:

- a programming language
- utilized compiler,
- type of code execution (sequential of parallel)
- libraries being used

2.1.4.1.7.1 Libraries

"Main library"

A "Main library" field allows to define a path to library containing user subroutine/function.

"Optional library"

A “Optional library” field allows to define a path to optional library containing user subroutine/function.

“Custom libraries”

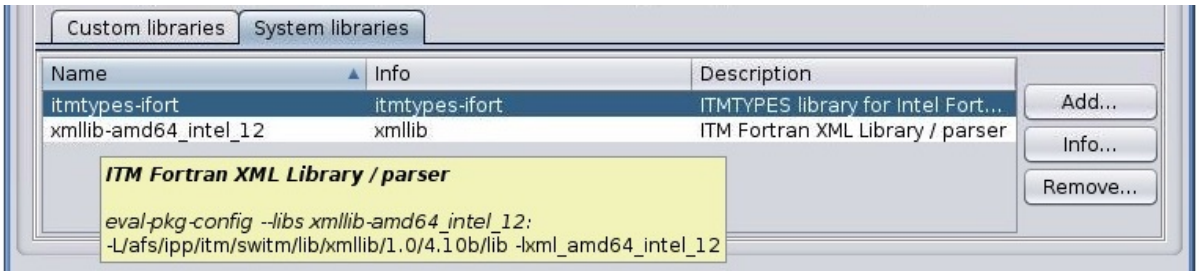
“Custom libraries” are non-standard static libraries required for building the user code.

Available operations on libraries list:

- *“Add...”* - Adds a new library to the list
- *“Edit...”* - Edits library path
- *“Remove”* - Removes a new library from the list

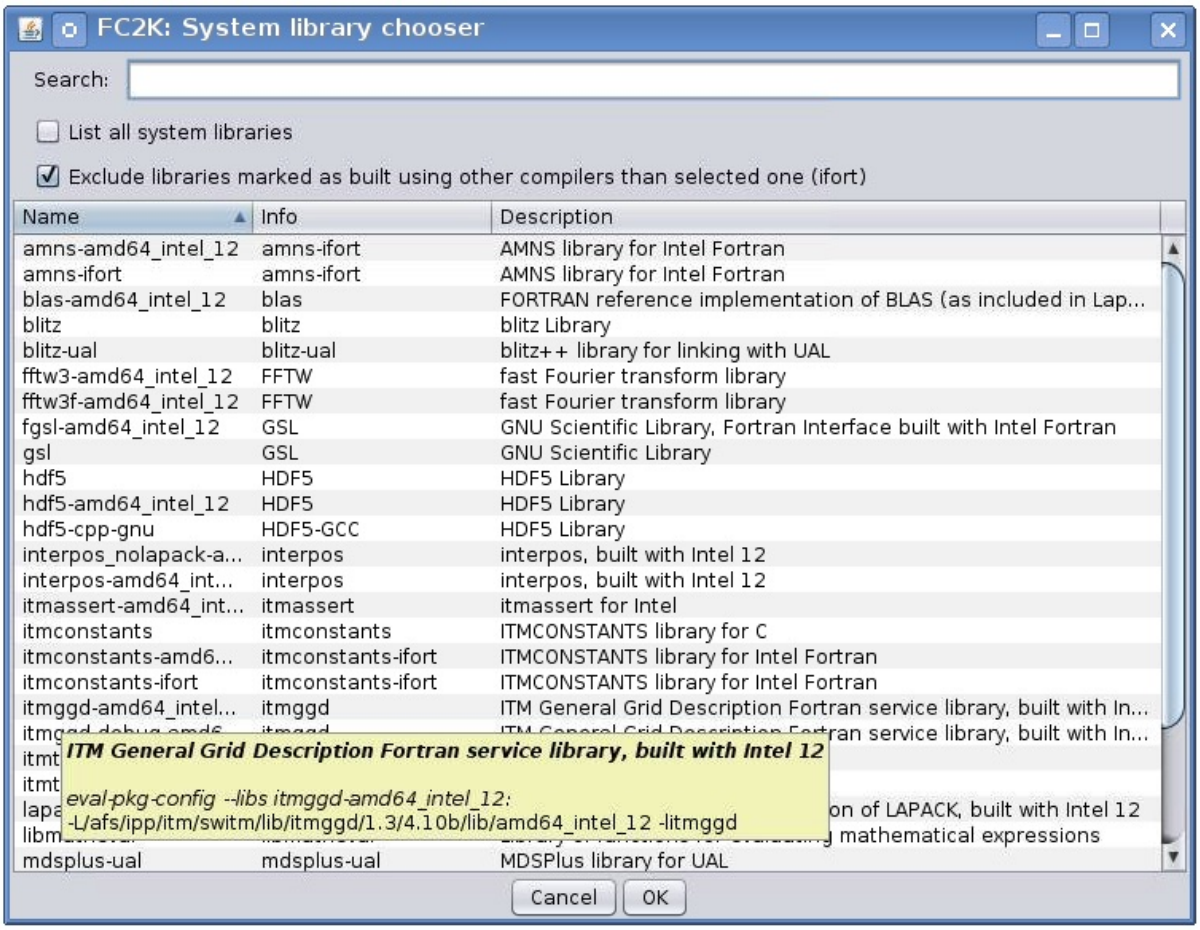
“System libraries”

“System libraries” are system libraries handled by pkg-config mechanism and required for building the user code.



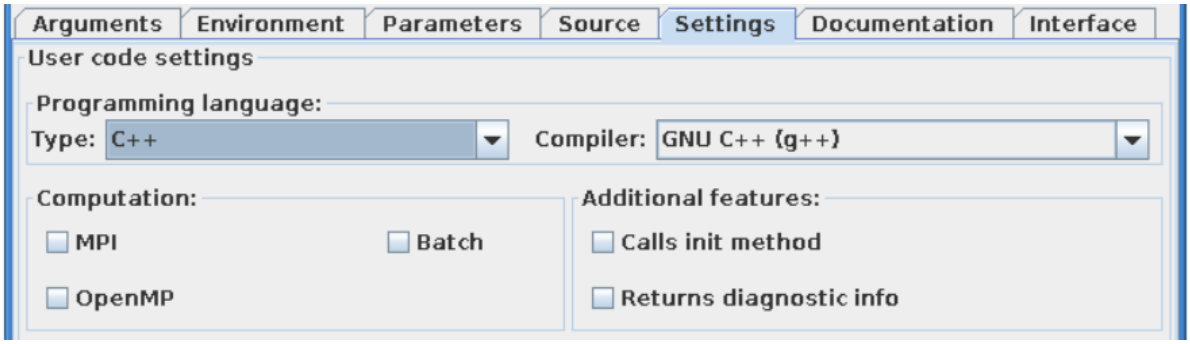
A user can:

- add library from the list,
- remove library
- display detailed info (library definition returned by pkg-config mechanism)



2.1.4.1.8 "Settings" tab explained

A user, using this tab, selects programming language of codes provided, compiler used to built library and type of code execution (sequential or parallel)



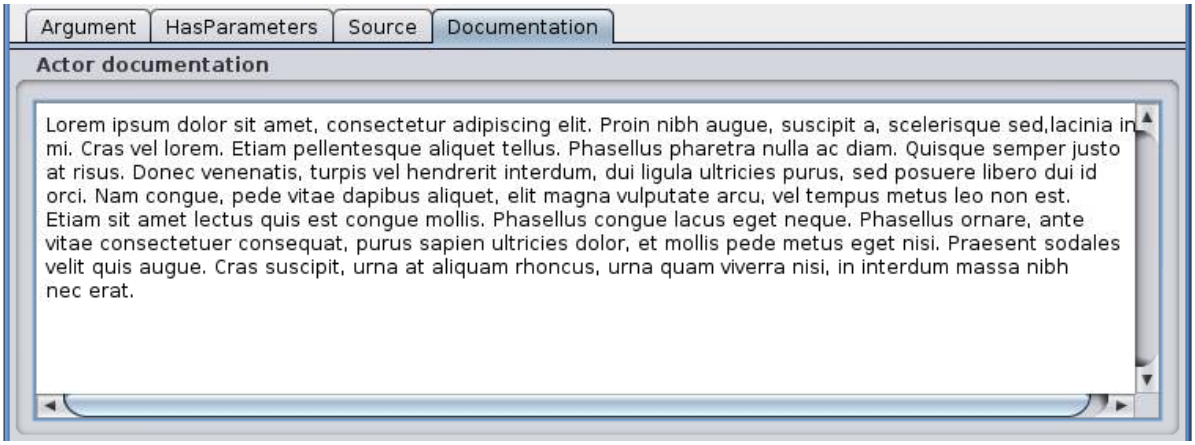
- **Programming languange:**
  - **Type** - Defines programming language of user codes. It could be set to:
    - Fortran
    - _C/C++
- **Compiler** - Defines compiler being used. Possible values:
  - ifort, gfortran

  - gcc, g++

- **Computation:**
  - **Parallel MPI** - If turned **ON** uses MPI compilers (mpiifort for ifort, mpif90 for gfortran, mpigxx for C)
  - **OPENMP** - Defined if usage of OpenMP directives is turned ON/OFF
  - **Batch** - If turned **ON**, submits a user code to jobs queue (combined with Parallel MPI or OPENMP switch runs user code as parallel job)
- **Additional features:**
  - **Calls init method** - If user function needs any pre-initialization, an additional function will be called.
  - **Returns diagnostic info** - adds output diagnostic information

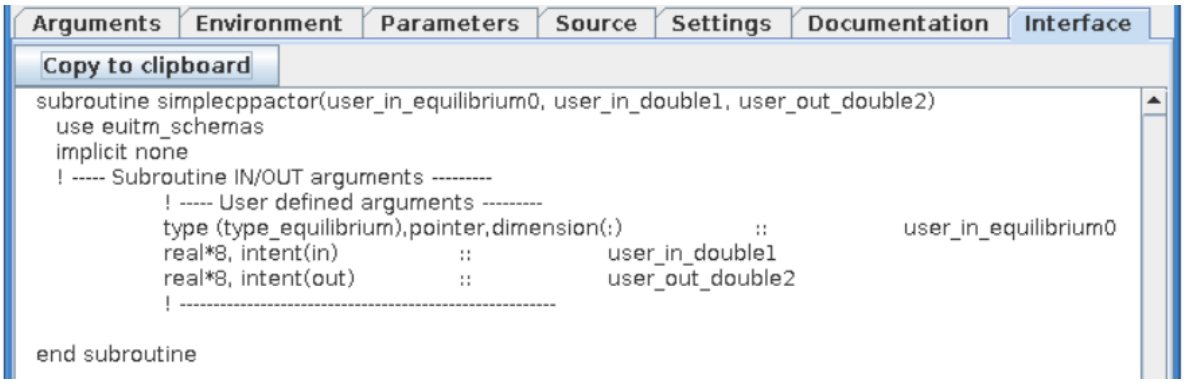
2.1.4.1.9 "Documentation" tab explained

The "Documentation" tab specifies an user-defined Kepler actor description. It could be displayed from actor pop-up menu.



2.1.4.1.10 "Interface" tab explained

The "Interface" tab specifies interface for Kepler actor.



2.1.4.2 Incorporating user codes into Kepler using FC2K - exercises

In this part of the tutorial you will learn how to incorporate Fortran and C++ codes into Kepler.

Hands-on exercises show:

- how to prepare C++ codes for FC2K
- how to prepare C++ library
- how set up Makefile
- how start and configure FC2K tool

2.1.4.2.1 Embedding Fortran codes into Kepler

Simple Fortran code

In this exercise you will execute simple Fortran code (multiplying input value by two) within Kepler.

Exercise1

Fortran UAL example (CPO handling)

In this exercise you will create Kepler actor that uses UAL.

Exercise2

2.1.4.2.2 Embedding C++ codes

Simple C++ code

Simple C++ code that will be incorporated into Kepler via FC2K tool - addition of one to the value passed into input port of the actor

Exercise3

C++ code within Kepler (CPO)

In this exercise you will create Kepler actor that uses UAL.

Exercise4

2.1.5 FC2K - developer guidelines

2.1.5.1 What code wrapper actually does?

The code wrapper intermediates between Kepler actor and user code:

- Passes variables of language built-in types (int, char, etc) from actor to the code
- Reads CPO(s) from UAL and passes data to user code
- Passes input code parameters (XML/XSD files) to user code
- Calls user subroutine/function
- Saves output CPO(s)

2.1.5.2 Development of Fortran codes

2.1.5.2.1 Subroutine syntax

subroutine name (<in/out arguments list> [,code_parameters] [,diagnostic_info])

- name - subroutine name
- in/out arguments list - a list of input and output subroutine arguments
- diagnostic_info - arbitrary output diagnostic information

2.1.5.2.2 Arguments list

- A mandatory position
- A list of input and output subroutine arguments including:
  - Fortran intrisic data types, eg:
    - integer :: input
    - character(50) :: charstring
    - integer,dimension(4) :: tabint
  - CPOs, eg:
    - type (type_equilibrium),pointer :: equilibriumin(:)
    - type (type_distsource),pointer :: distsourceout(:)

2.1.5.2.3 Code parameters

- user defined input parameters
- input / optional
- Argument of type: type_param

```
type type_param  !
    character(len=132), dimension(:), pointer ::parameters
    character(len=132), dimension(:), pointer ::default_param
    character(len=132), dimension(:), pointer ::schema
endtype
```

- Derived type type_param describes:
  - parameters - Actual value of the code parameters (instance of coparam/parameters in XML format).
  - default_param - Default value of the code parameters (instance of coparam/parameters in XML format).
  - schema - Code parameters schema.
- An example:
  - (type_param) :: codeparam{{

2.1.5.2.4 Diagnostic info

- arbitrary output diagnostic information
  - output / optional

```
!----  Diagnostic info  ----
integer, intent(out)     ::     user_out_outputFlag
character(len=:), pointer, intent(out)    ::    user_out_diagnosticInfo
```

- outputFlag - indicates if user subroutine was successfully executed
  - outpuflag = 0 - SUCCESS, no action is taken
  - outputFlag > 0 - WARNING, a warning message is displayed, workflow continuue execution
  - outputFlag < 0 - ERROR, actor throws an exception, workflow stops
- diagnosticInfo - an arbitrary string

2.1.5.2.5 Examples

```
**Example 1 Simple in/out argument types**
subroutine nocpo(input, output)
   integer, intent(in):: input
   integer, intent(out):: output
```

```
**Example 2 A CPO array as a subroutine argument**
subroutine equil2dist(equilibriumin, distsourceout)
    use euITM_schemas
    implicit none

    !input
    type (type_equilibrium), pointer :: equilibriumin(:)
    !output
    type (type_distsource), pointer :: distsourceout(:)
```

```
**Example 3 Usage of code input parameters**
subroutine teststring(coreprof,equi,tabint,tabchar,codeparam)
    use euITM_schemas

    implicit none

!input
    type(type_coreprof),pointer,dimension(:) :: coreprof
    integer, dimension(4), intent(in) :: tabint

!output
    type(type_equilibrium),pointer,dimension(:) :: equi
    character(50), intent(out) :: tabchar

!code parameters
    type(type_param), intent(in) :: codeparam
```

2.1.5.3 Development of C++ codes

2.1.5.3.1 Function syntax

void name (<in/out arguments list> [,code_parameters] [,diagnostic_info])

- name - function name

- code_parameters - optional - user defined input parameters
- diagnostic_info - arbitrary output diagnostic information

2.1.5.3.2 Arguments list

- in/out arguments list
- mandatory
- a list of input and output function arguments including:
  - CPP intrisic data types, eg:
    - int &x
    - double &y
  - CPOs, eg:
    - ItmNs::Itm::antennas & ant
    - ItmNs::Itm::equilibriumArray & eq

2.1.5.3.3 Code parameters

- Optional
- User defined input parameters
- Argument of type: ItmNs:: codeparam_t &

```
typedef struct {
   char **parameters;
   char **default_param;
   char **schema;
} codeparam_t;
```

- A structure codeparam_t describes:
  - parameters - Actual value of the code parameters (instance of coparam/parameters in XML format).
  - default_param - Default value of the code parameters (instance of coparam/parameters in XML format).
  - schema - Code parameters schema.
- An example: ItmNs::codeparam_t & codeparam

2.1.5.3.4 Diagnostic info

- arbitrary output diagnostic information
- output / optional

```
void name(...., int* output_flag, char** diagnostic_info)
```

- output_flag - indicates if user subroutine was successfully executed

  - output_flag = 0 - SUCCESS, no action is taken
  - output_flag > 0 - WARNING, a warning message is displayed, workflow continue execution
  - output_flag < 0 - ERROR, actor throws an exception, workflow stops
- diagnostic_info - an arbitrary string

2.1.5.3.5 Examples

```
**Example 4. Simple in/out argument types**
void simplecppactornocpo(double &x, double &y)
```

```
**Example 5. A CPO array as a function argument**
void simplecppactor(ItmNs::Itm::equilibriumArray &eq, double &x, double &y)
```

```
**Example 6. Usage of init function and code input parameters**
void mycppfunctionbis_init();
void mycppfunction(ItmNs::Itm::summary& sum, ItmNs::Itm::equilibriumArray& eq, int& x,␣
↪ItmNs::Itm::coreimpur& cor,  double&  y, ItmNs::codeparam_t& codeparam)
```

2.1.5.4 Delivery of the user code

The user code should be delivered as a static library. Please find examples of the simple "makefiles" below:

```
**Example 6. Building of Fortran code**
F90 = $(ITM_INTEL_FC)
COPTS = -g -O0 -assume no2underscore -fPIC -shared-intel

INCLUDES = $(shell eval-pkg-config --cflags ual-$(ITM_INTEL_OBJECTCODE))

all: equilibrium2distsource.o libequilibrium2distsource

libequilibrium2distsource: equilibrium2distsource.o
       ar -rvs libequilibrium2distsource.a equilibrium2distsource.o

equilibrium2distsource.o: equilibrium2distsource.f90
       $(F90) $(COPTS) -c -o $@ $^ ${INCLUDES}

clean:
      rm -f *.o *.a
```

```
**Example 7. Building of C++ code**
CXX=g++
CXXFLAGS= -g -fPIC
CXXINCLUDES= ${shell eval-pkg-config --cflags ual-cpp-gnu}

all: libsimplecppactor.a

libsimplecppactor.a: simplecppactor.o
     ar -rvs $@ $^

simplecppactor.o: simplecppactor.cpp
     $(CXX) $(CXXFLAGS) $(CXXINCLUDES) -c -o $@ $^

clean:
     rm *.a *.o
```

2.1.6 FC2K - Example 1 - Embedding Fortran codes into Kepler (no CPOs)

The knowledge gained After this exercise you will:

- know how to prepare Fortran codes for FC2K
- know how to build Fortran library
- know how set up Makefile
- know how start and configure FC2K tool

In this exercise you will execute simple Fortran code within Kepler. In order to this follow the instructions:

2.1.6.1 Get familiar with codes that will be incorporated into Kepler

Go to Code Camp related materials within your home directory

```
shell> cd $TUTORIAL_DIR/FC2K/nocpo_example_1
```

You can find there various files. Pay particular attention to following ones:

- nocpo.f90 - Fortran source code that will be executed from Kepler
- Makefile - makefile that allows to build library file
- nocpo_fc2k.xml - parameters for FC2K application (NOTE! this file contains my own settings, we will modify them during tutorial)
- nocpo.xml - example workflow

2.1.6.2 Build the code by issuing

```
shell> make clean
shell> make
```

Codes are ready to be used within FC2K

2.1.6.3 Prepare environment for FC2K

Make sure that all required system settings are correctly set

```
shell> source $ITMSCRIPTDIR/ITMv1 kepler test 4.10b > /dev/null
```

2.1.6.4 Start FC2K application

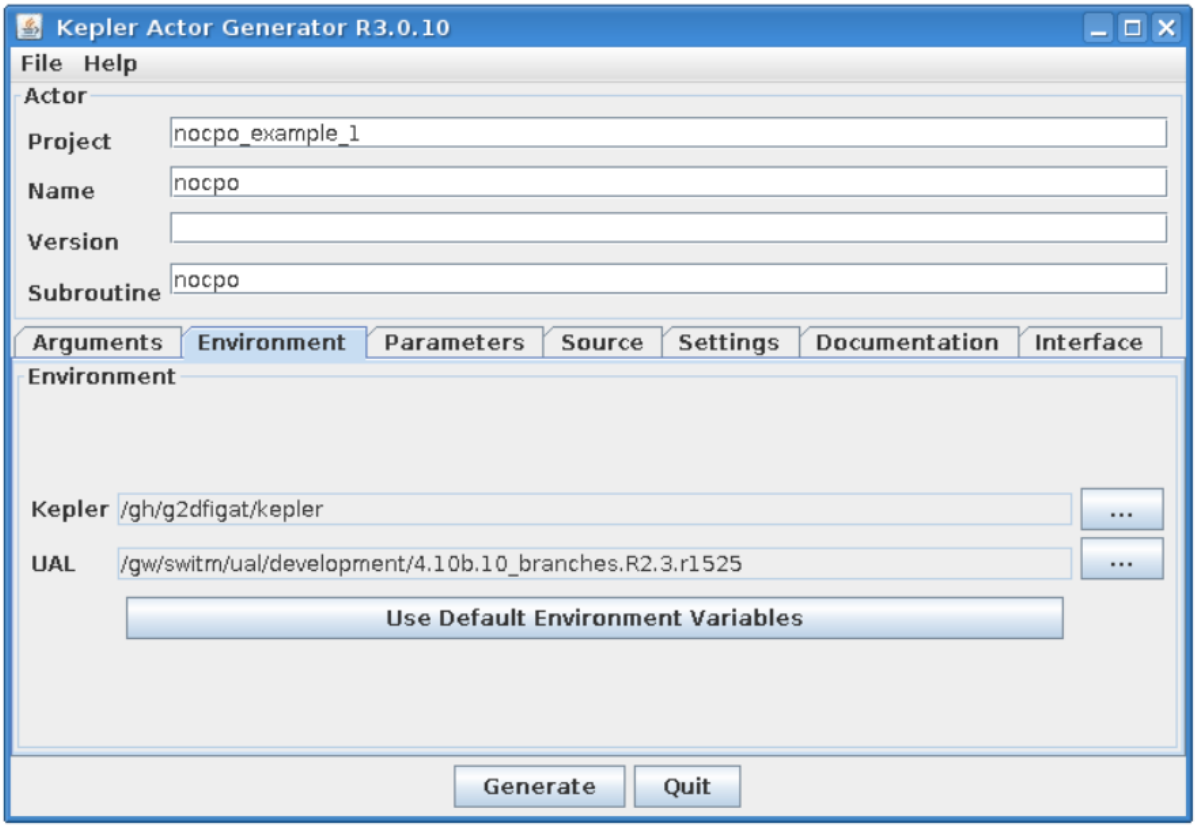
This is as simple as typing **fc2k** from terminal

```
shell> fc2k
```

After a while, you should see FC2K's main window.

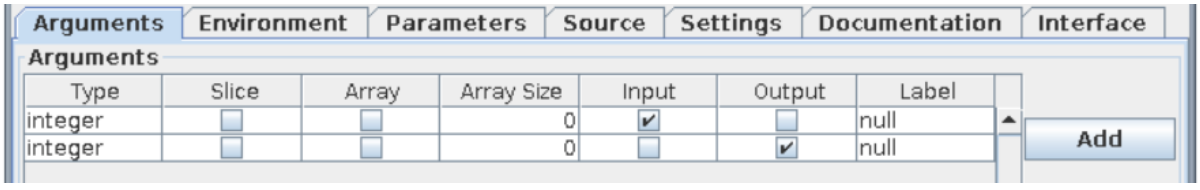
2.1.6.5 Open a nocpo_example_1 project

1. Choose **File -> Open** and navigate to **$TUTORIAL_DIR/FC2K/nocpo_example_1.**
2. Open file **nocpo_fc2k.xml.**
3. You should see new parameter settings loaded into FC2K.
4. After loading parameters you can notice that parameters point to locations within your home directory.



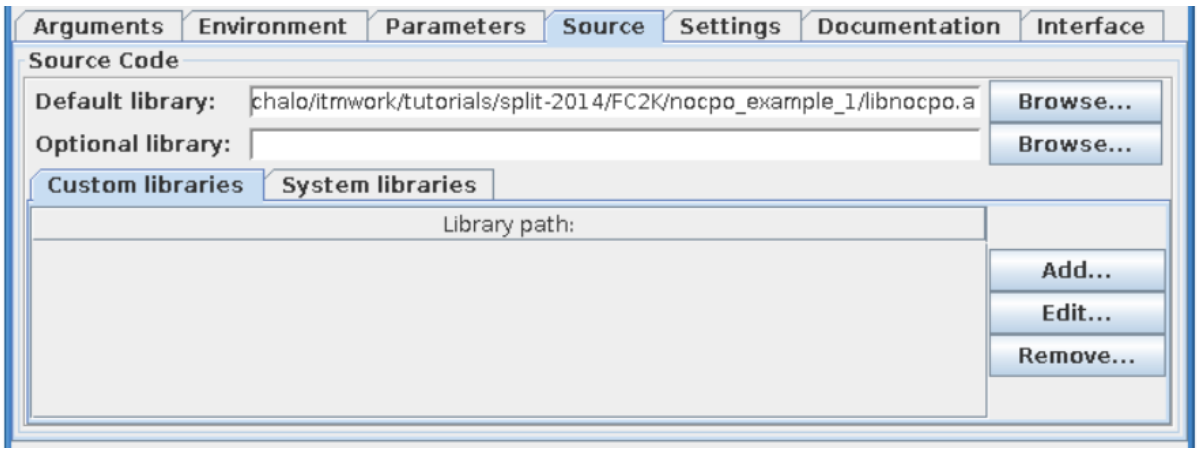
2.1.6.6 Project settings

Please take a look at the project settings.



Subroutine arguments:

- one input argument of type integer
- one output argument of type integer

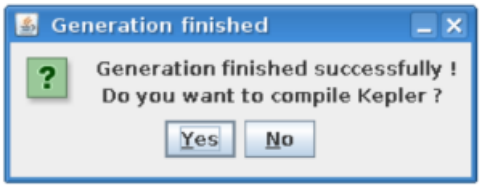


2.1.6.7 After all the settings are correct, you can generate actor

Simply press "Generate" button and wait till FC2K finishes the generation.

2.1.6.8 Confirm Kepler compilation

After actor is generated, FC2K offers to compile Kepler application. Make sure to compile it by pressing "Yes".



2.1.6.9 You can now start Kepler and use generated actor

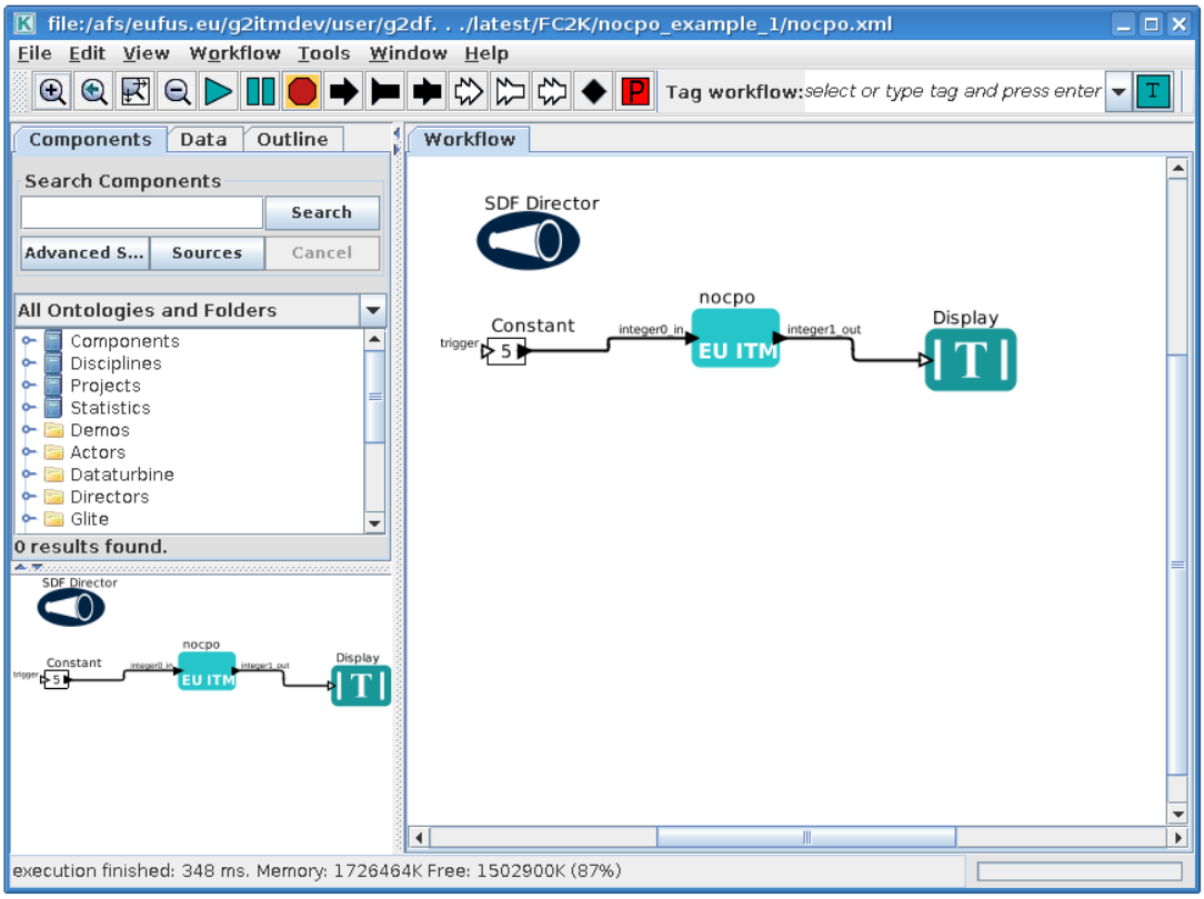
Open new terminal window and make sure that all environment settings are correctly set and execute Kepler.

```
shell> source $ITMSCRIPTDIR/ITMv1 kepler test 4.10b > /dev/null
shell> kepler.sh
```

After Kepler is started, open example workflow from the following location

```
$TUTORIAL_DIR/FC2K/nocpo_example_1/nocpo.xml
```

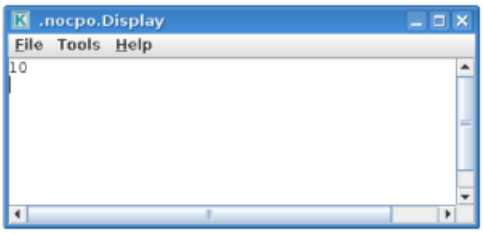
You should see similar workflow on screen.



2.1.6.10 Launch the workflow

You can start the workflow execution, by pressing “Play” button

After workflow finishes it’s execution, you should see result similar to one below:



Exercise no. 1 finishes here.

2.1.7 FC2K - Example 2 - Embedding Fortran code into Kepler (CPOs)

Exercise no. 2.

Fortran example (CPO handling)

(approx. 30 min)

The knowledge gained

After this exercise you will:

- know how to prepare Fortran codes that use UAL
- know how to prepare Fortran based library that uses UAL
- know how set up Makefile
- know how start and configure FC2K tool

In this exercise you will execute simple Fortran code that uses UAL. Code will be incorporated into Kepler. In order to do this follow the instructions:

2.1.7.1 Get familiar with codes that will be incorporated into Kepler

Go to Code Camp related materials within your home directory

```
shell> cd $TUTORIAL_DIR/FC2K/equilibrium2distsource/
```

You can find there various files. Pay particular attention to following ones:

- equilibrium2distsource.f90 - Fortran source code that will be executed fromKepler - this code uses UAL
- Makefile - makefile that allows to build library file
- cposlice2cposlicef_fc2k.xml - parameters for FC2K application (NOTE! this file contains my own settings, we will modify them during tutorial)
- cposlice2cposlicef_kepler.xml - example workflow

2.1.7.2 Build the code

A Fortran example could be built by issuing

```
shell> make clean -f make_ifort
shell> make -f make_ifort
```

Codes are ready to be used within FC2K

2.1.7.3 Prepare environment for FC2K

Make sure that all required system settings are correctly set

```
shell> source $ITMSCRIPTDIR/ITMv1 kepler test 4.10b > /dev/null
```

2.1.7.4 Start FC2K application

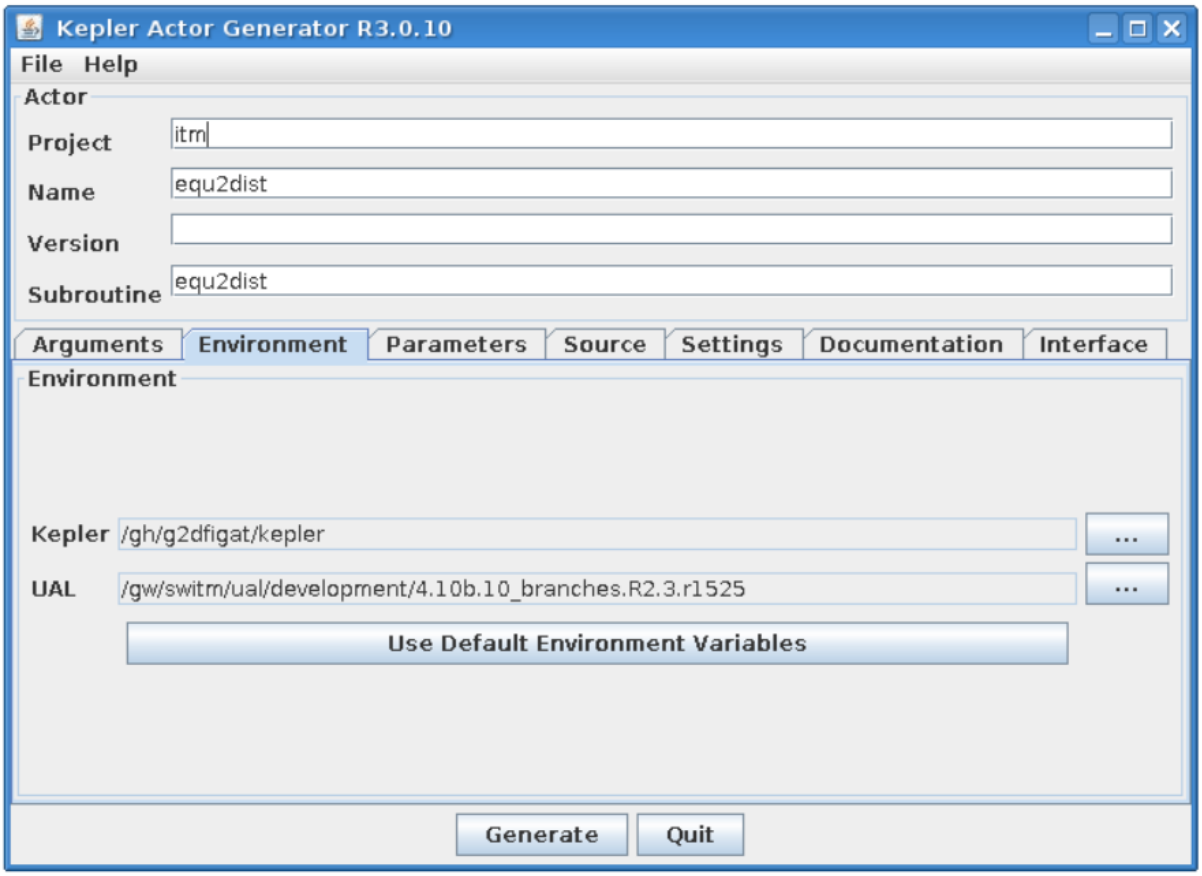
This is as simple as typing fc2k from terminal

```
shell> fc2k
```

After a while, you should see FC2K's main window

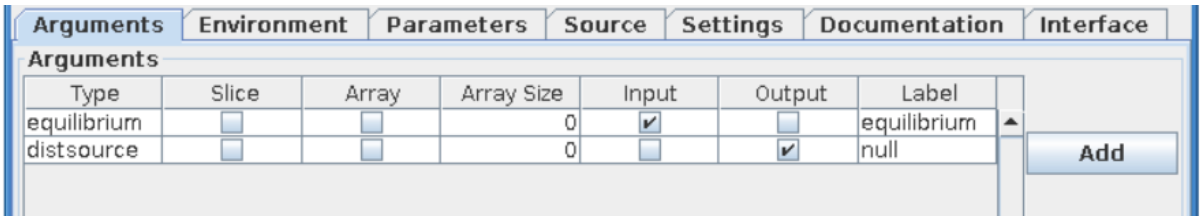
2.1.7.5 Open project cposlice2cposlicef_fc2k

1. Choose **File -> Open**
2. Navigate to $TUTORIAL_DIR/FC2K/equilibrium2distsource/.
3. Open file cposlice2cposlicef_fc2k.xml.
4. You should see new project loaded into FC2K.



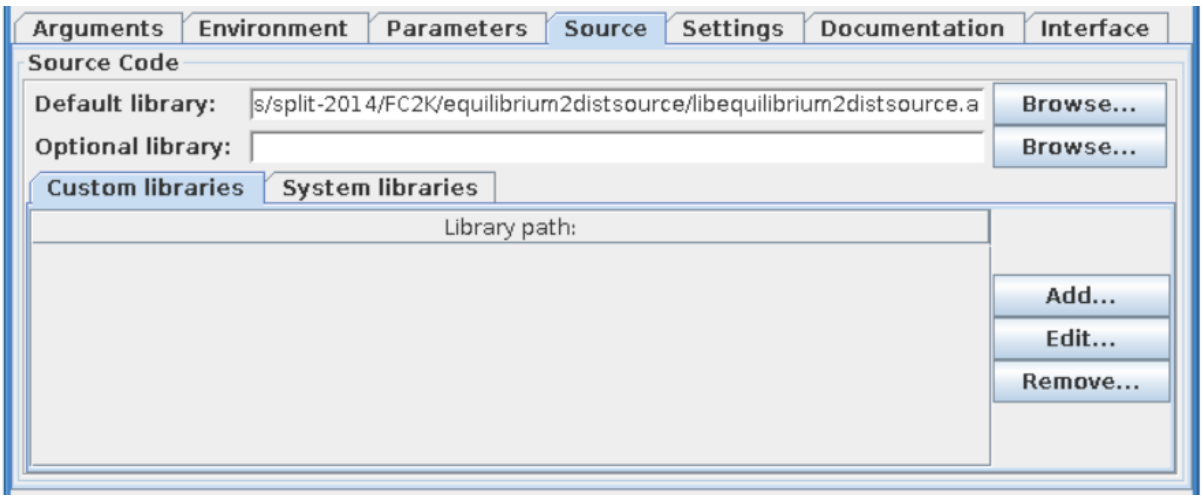
2.1.7.6 Project settings

Please take a look at the project settings.



Subroutine arguments:

- one input argument - CPO array
- one output argument - CPO array



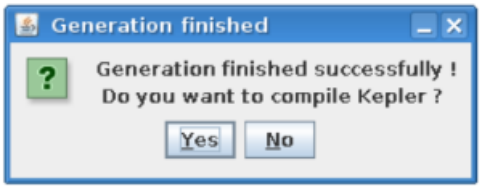
After loading parameters you can notice that library location points to location within your itmwork directory (**$ITMWORK**).

2.1.7.7 After all the settings are correct, you can generate actor

Simply press "Generate" button and wait till FC2K finishes the generation.

2.1.7.8 Confirm Kepler compilation

After actor is generated, FC2K offers to compile Kepler application. Make sure to compile it by pressing "Yes".



2.1.7.9 You can now start Kepler and use generated actor

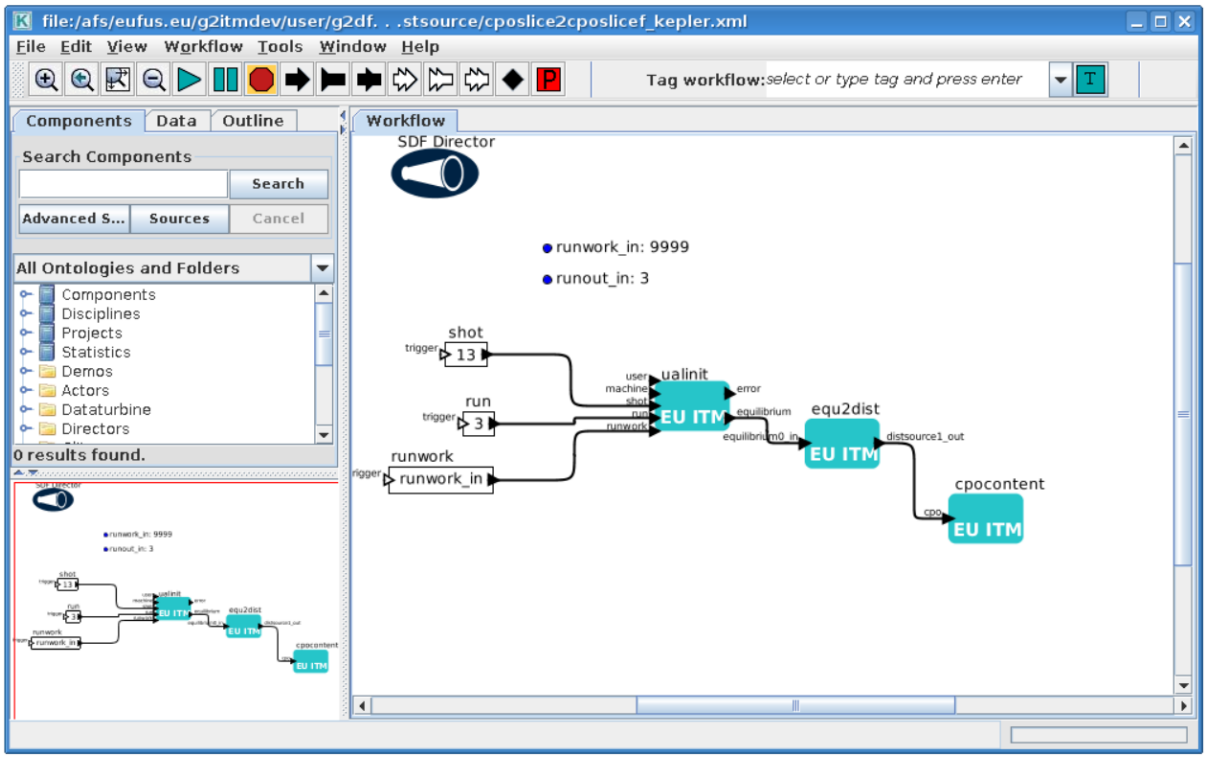
Open new terminal window and make sure that all environment settings are correctly set and execute Kepler.

```
shell> source $ITMSCRIPTDIR/ITMv1 kepler test 4.10b > /dev/null
shell> kepler.sh
```

After Kepler is started, open example workflow from the following location

```
shell> $TUTORIAL_DIR/FC2K/equilibrium2distsource/cposlice2cposlicef_kepler.xml
```

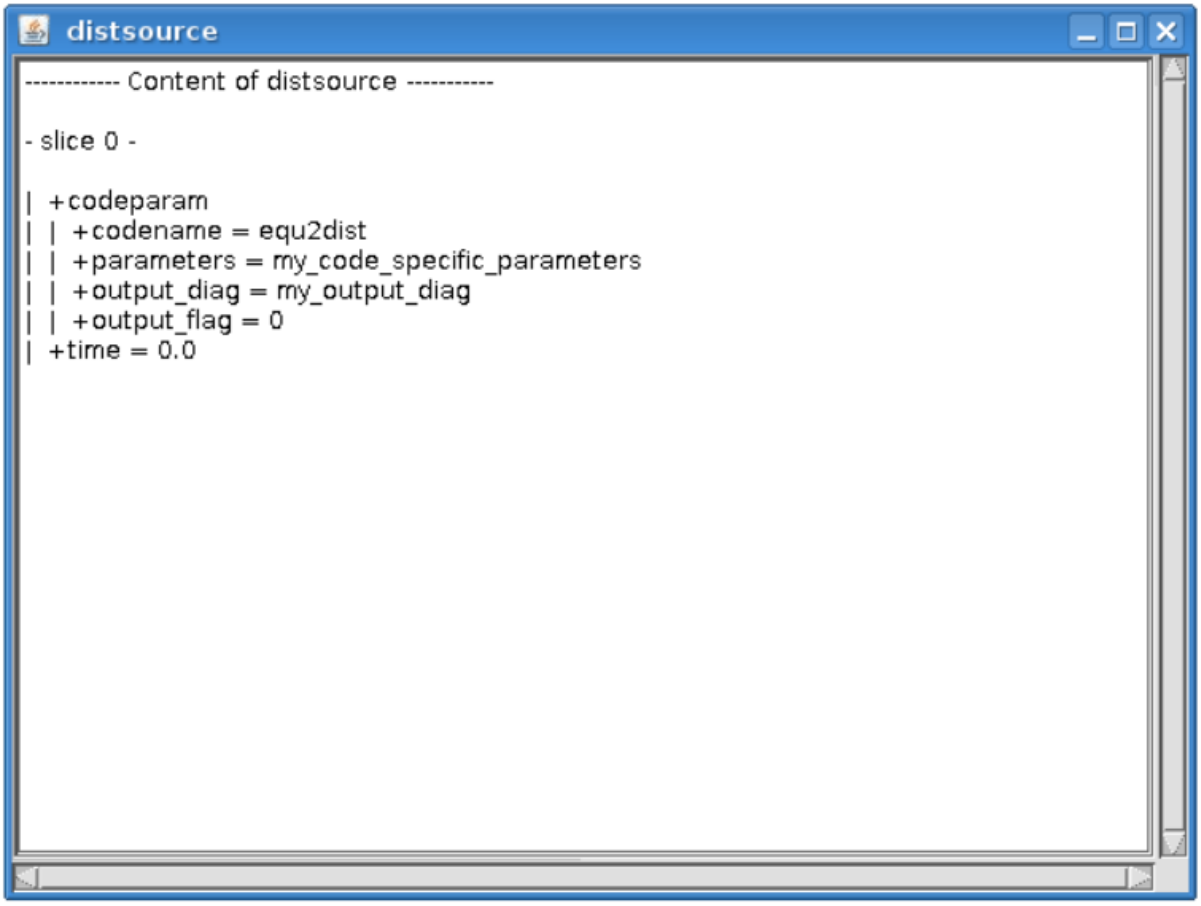
You should see similar workflow on screen.



2.1.7.10 Launch the workflow

You can start the workflow execution, by pressing “Play” button

After workflow finishes it’s execution, you should see result similar to one below:



Exercise no. 2 finishes here.

2.1.8 FC2K - Example 3 - Embedding C++ code within Kepler (no CPOs)

Exercise no. 3

Embedding simple C++ code within Kepler (no CPOs)

(approx. 30 min)

The knowledge gained After this exercise you will:

- know how to prepare C++ codes for FC2K
- know how to prepare C++ library
- know how set up Makefile
- know how start and configure FC2K tool

In this exercise you will execute simple C++ code within Kepler. In order to do this follow the instructions:

2.1.8.1 Get familiar with codes that will be incorporated into Kepler

Go to Code Camp related materials within your home directory

```
cd $TUTORIAL_DIR/FC2K/simplecppactor_nocpo
```

You can find there various files. Pay particular attention to following ones:

- simplecppactornocpo.cpp - C++ source code that will be executed from Kepler
- Makefile - makefile that allows to build library file
- simplecppactor_nocpo_fc2k.xml - parameters for FC2K application (NOTE! this file contains my own settings, we will modify them during tutorial)
- simplecppactor_nocpo_workflow.xml - example workflow

2.1.8.2 Build the code by issuing

```
shell> make clean
shell> make
```

Codes are ready to be used within FC2K

2.1.8.3 Prepare environment for FC2K

Make sure that all required system settings are correctly set

```
shell> source $ITMSCRIPTDIR/ITMv1 kepler test 4.10b > /dev/null
```

2.1.8.4 Start FC2K application

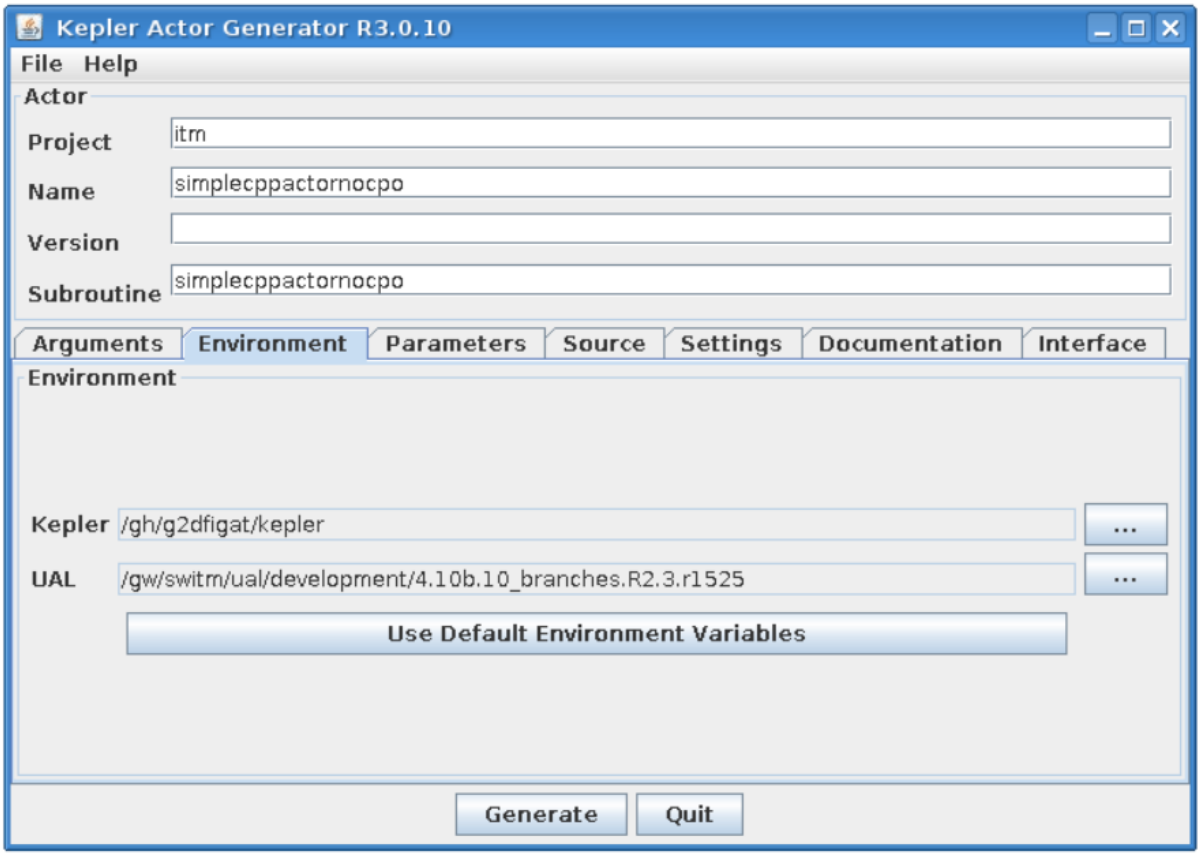
This is as simple as typing fc2k from terminal

```
shell> fc2k
```

After a while, you should see FC2K's main window

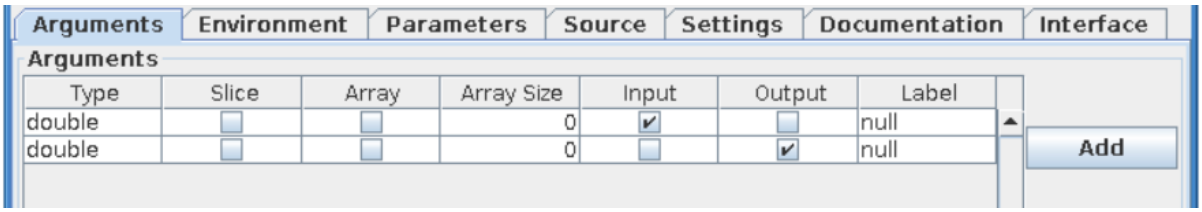
2.1.8.5 Open project simplecppactor_nocpo

1. Choose **File -> Open**
2. Navigate to $TUTORIAL_DIR/FC2K/simplecppactor_nocpo
3. Open file simplecppactor_nocpo_fc2k.xml.
4. You should see new project loaded into FC2K.



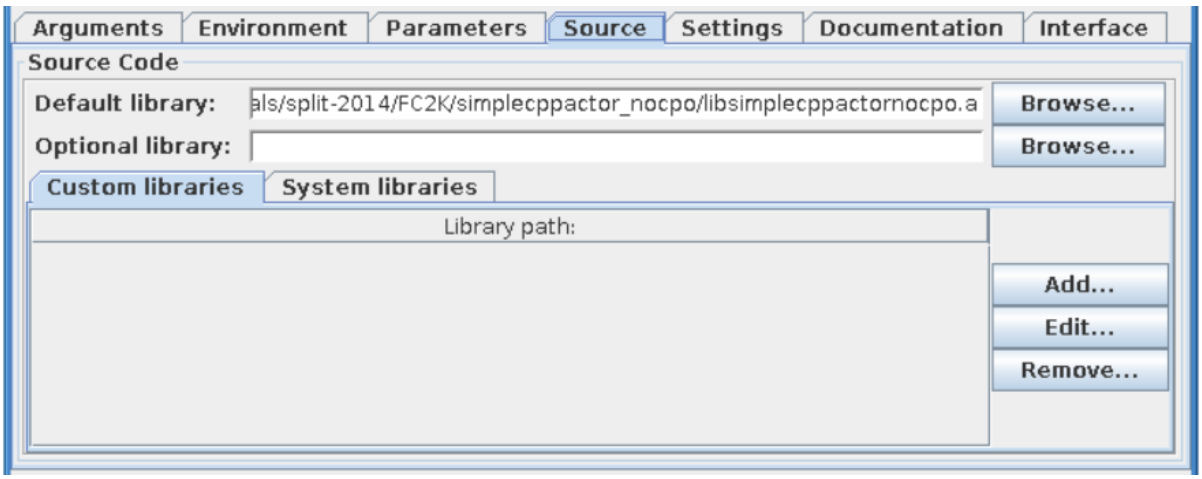
2.1.8.6 Project settings

Please take a look at the project settings.



Function arguments:

- one input argument - double
- one output argument - double



After loading parameters you can notice that library location points to location within your $TUTORIAL_DIR directory.

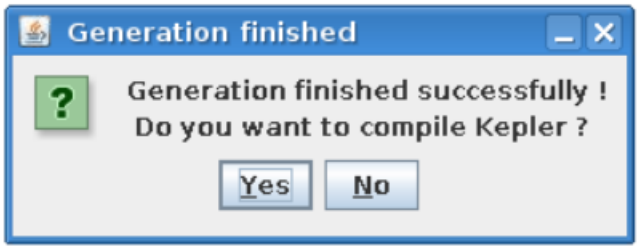
2.1.8.7 Actor generation

After all the settings are correct, you can generate actor

Simply press "Generate" button and wait till FC2K finishes the generation.

2.1.8.8 Confirm Kepler compilation

After actor is generated, FC2K offers to compile Kepler application. Make sure to compile it by pressing "Yes".



2.1.8.9 You can now start Kepler and use generated actor

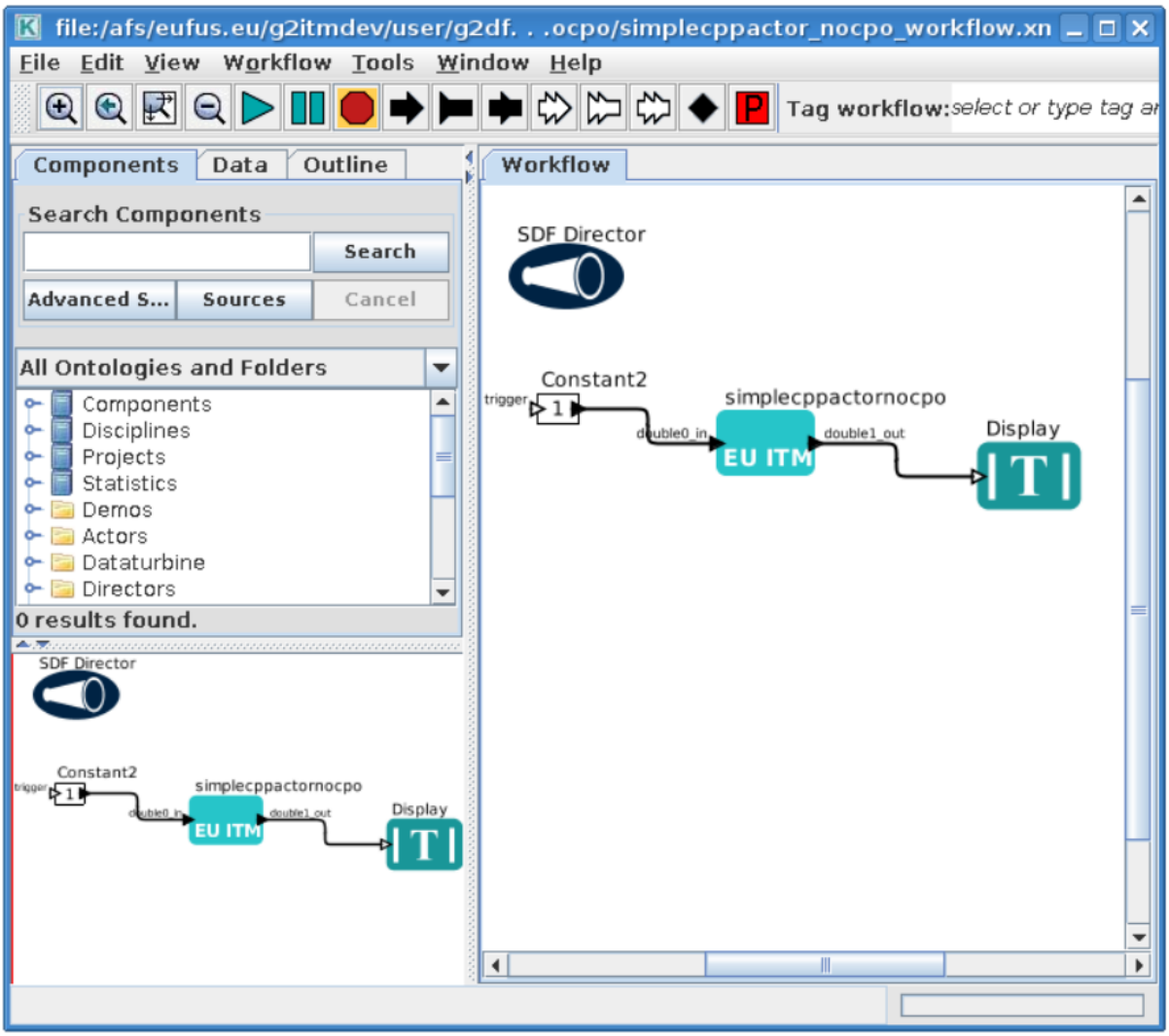
Open new terminal window and make sure that all environment settings are correctly set and execute Kepler.

```
shell> source $ITMSCRIPTDIR/ITMv1 kepler test 4.10b > /dev/null
shell> kepler
```

After Kepler is started, open example workflow from the following location

```
$TUTORIAL_DIR/FC2K/simplecppactor_nocpo/simplecppactor_nocpo_workflow.xml
```

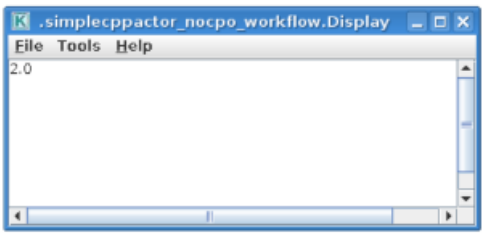
You should see similar workflow on screen.



2.1.8.10 Launch the workflow

You can start the workflow, by pressing “Play” button

After workflow finishes it’s execution, you should see result similar to one below:



Exercise no. 3 finishes here.

2.1.9 FC2K - Example 4 - Embedding C++ code within Kepler (CPOs)

Exercise no. 4

C++ code within Kepler (CPO handling)

(approx. 30 min)

The knowledge gained: After this exercise you will: - know how to prepare C++ codes for FC2K - know how to prepare C++ library - know how set up Makefile - know how start and configure FC2K tool In this exercise you will execute simple C++ code within Kepler. In order to do this follow the instructions:

2.1.9.1 Get familiar with codes that will be incorporated into Kepler

Go to Code Camp related materials within your home directory

```
shell> cd $TUTORIAL_DIR/FC2K/simplecppactor
```

You can find there various files. Pay particular attention to following ones:

- simplecppactor.cpp - C++ source code that will be executed from Kepler
- Makefile - makefile that allows to build library file
- simplecppactor_fc2k.xml - parameters for FC2K application (NOTE! this file contains my own settings, we will modify them during tutorial)
- simplecppactor_workflow.xml - example workflow

2.1.9.2 Build the code by issuing

```
shell> make clean
shell> make
```

Codes are ready to be used within FC2K

2.1.9.3 Prepare environment for FC2K

Make sure that all required system settings are correctly set

```
shell> source $ITMSCRIPTDIR/ITMv1 kepler test 4.10b > /dev/null
```

2.1.9.4 Start FC2K application

This is as simple as typing fc2k from terminal

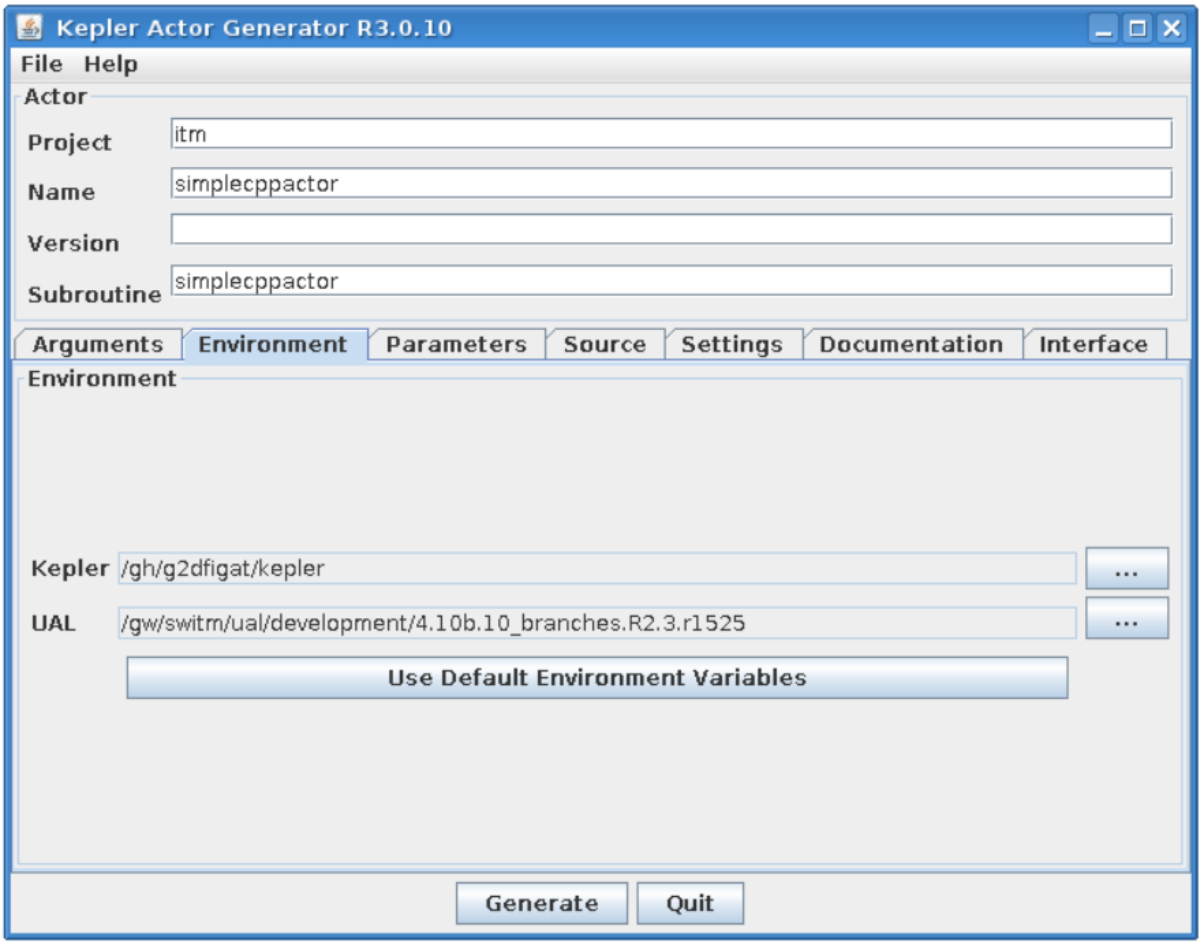
```
shell> fc2k
```

After a while, you should see FC2K's main window.

2.1.9.5 Open project simplecppactor

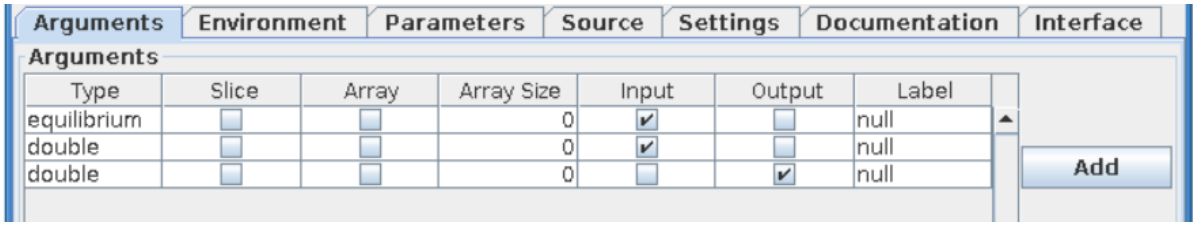
1. Choose **File -> Open**
2. Navigate to $TUTORIAL_DIR/FC2K/simplecppactor.
3. Open file simplecppactor_fc2k.xml.

4. You should see new parameter settings loaded into FC2K.



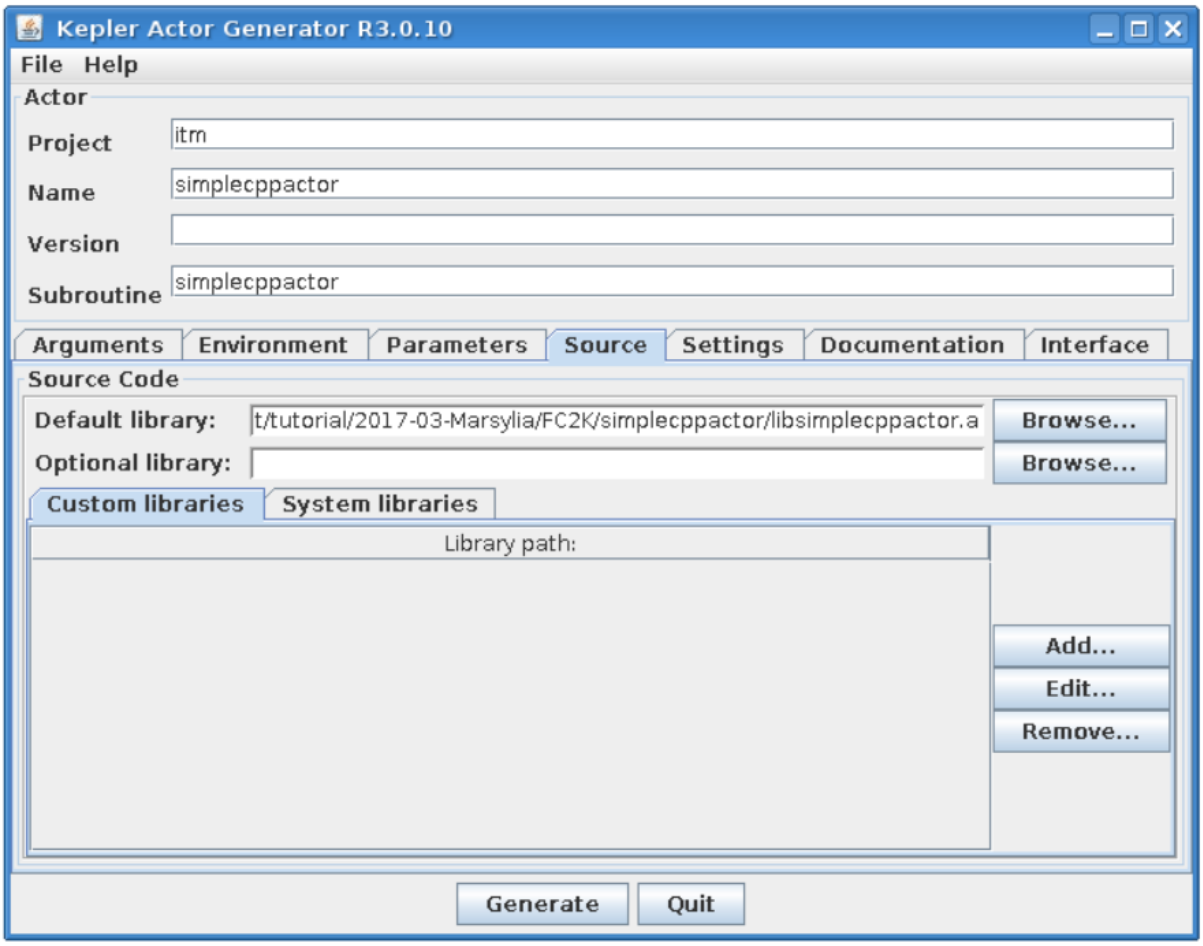
2.1.9.6 Project settings

Please take a look at the project settings.



Function arguments:

- input argument - equilibrium
- input argument - double
- output argument - double



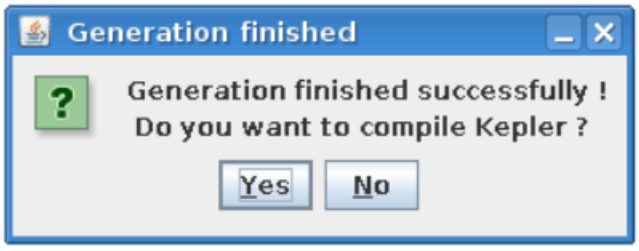
You should modify these setting, so they point to locations within you home directory. They will typically be as follows:

2.1.9.7 Actor generation

After all the settings are correct, you can generate actor Simply press “Generate” button and wait till FC2K finishes the generation.

2.1.9.8 Confirm Kepler compilation

After actor is generated, FC2K offers to compile Kepler application. Make sure to compile it by pressing “Yes”.



2.1.9.9 You can now start Kepler and use generated actor

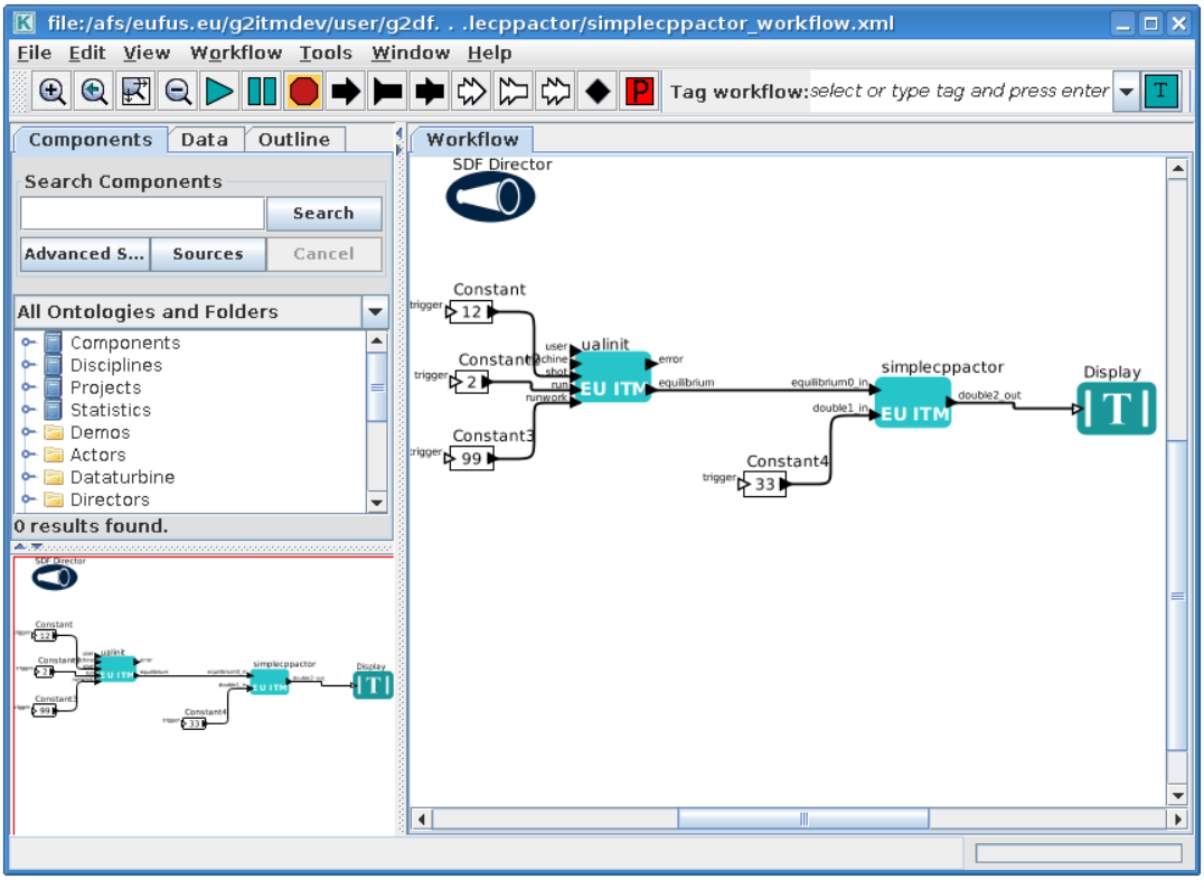
Open new terminal window and make sure that all environment settings are correctly set and execute Kepler.

```
shell> source $ITMSCRIPTDIR/ITMv1 kepler test 4.10b > /dev/null
kepler.sh
```

After Kepler is started, open example workflow from the following location

```
shell> $TUTORIAL_DIR/FC2K/simplecppactor/simplecppactor_workflow.xml
```

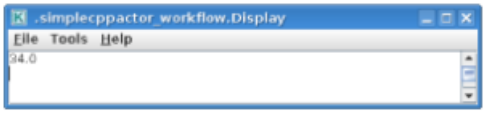
You should see similar workflow on screen.



Launch the workflow

You can start the workflow, by pressing “Play” button

After workflow finishes it’s execution, you should see result similar to one below:



Exercise no. 4 finishes here.

2.1.10 IMAS Kepler 2.1.3 (default release)

2.1.10.1 Installation of default version of Kepler (without actors)

In order to use most recent version of Kepler do following. First of all make sure you have directory imas-kepler inside your $HOME

```
 in case you already have imas-kepler inside $HOME
 you can move it to $ITMWORK/imas-kepler
> mv $HOME/imas-kepler $ITMWORK/imas-kepler

 If you don't have $HOME/imas-kepler directory, create
 it inside $ITMWORK
> mkdir $ITMWORK/imas-kepler

 create symbolic link inside $HOME
> cd $HOME
> ln -s $ITMWORK/imas-kepler
```

Then, you can load imasenv module by calling

```
> module load imasenv
```

If there is no Kepler version installed, you will be informed by message

```
WARNING: Cannot find /afs/eufus.eu/user/..../imas-kepler/2.5p2-2.1.3... Run kepler_install_
 ↪light before running kepler;
INFO: setting KEPLER=/gw/swimas/extra/kepler/2.5p2-2.1.3;
IMAS environment loaded.
Please do not forget to set database by calling 'imasdb <machine_name>' !
```

In that case, call kepler_install_light - you will see installation process running in your terminal.

```
> kepler_install_light
Warning: $KEPLER_INSTALL_PATH override by environment: /afs/eufus.eu/user/g/g2michal/imas-
 ↪kepler/2.5p2-2.1.3
mkdir: created directory ?/afs/eufus.eu/g2itmdev/user/g2michal/imas-kepler/2.5p2-2.1.3?
sending incremental file list
.ptolemy-compiled
build-area/
build-area/README.txt
build-area/build.xml
build-area/current-suite.txt
...
...
...
?gui? -> ?gui-2.5?
?common? -> ?common-2.5?
Done installing /afs/eufus.eu/g2itmdev/user/g2michal/imas-kepler/2.5p2-2.1.3.
Run `module switch kepler/2.5p2-2.1.3` to update $KEPLER to match.
Then run `kepler` to try your lightweight installation.
```

You have to switch module, to make sure that KEPLER variable points to proper location.

```
> module switch kepler/2.5p2-2.1.3
```

Once you have set version of Kepler, you can run it by typing kepler

```
> kepler
The base dir is /afs/eufus.eu/g2itmdev/user/g2michal/imas-kepler/2.5p2-2.1.3
Kepler.run going to run.setMain(org.kepler.Kepler)
JVM Memory: min = 1G,  max = 8G, stack = 20m, maxPermGen = default
adding $CLASSPATH to RunClassPath: /gw/switm/jaxfront/R1.0/XMLParamForm.jar:/gw/switm/
 ↪jaxfront/R1.0/jaxfront-core.jar:/gw/switm/jaxfront/R1.0/jaxfront-swing.jar:/gw/switm/
 ↪jaxfront/R1.0/xercesImpl.jar:/gw/swimas/core/imas/3.20.0/ual/3.8.3/jar/imas.jar
...
```

```
...
```

2.1.10.2 Installation of "dressed" version of Kepler (with actors)

In order to use most recent version of Kepler (with actors) do following. First of all make sure you have directory imas-kepler inside your $HOME

```
> mkdir $HOME/imas-kepler
```

Then, you can load imasenv module by calling

```
> module load imasenv
```

If there is no Kepler version installed, you will be informed by message

```
WARNING: Cannot find /afs/eufus.eu/user/..../imas-kepler/2.5p2-2.1.3... Run kepler_install_
 ↪light before running kepler;
INFO: setting KEPLER=/gw/swimas/extra/kepler/2.5p2-2.1.3;
IMAS environment loaded.
Please do not forget to set database by calling 'imasdb <machine_name>' !
```

You have to switch to "dressed" version of Kepler by calling

```
> module switch kepler/2.5p2-2.1.3_IMAS_3.20.0
```

```
> kepler_install_light
Warning: $KEPLER_INSTALL_PATH override by environment: /afs/eufus.eu/user/g/g2michal/imas-
 ↪kepler/2.5p2-2.1.3_IMAS_3.20.0
mkdir: created directory ?/afs/eufus.eu/g2itmdev/user/g2michal/imas-kepler?
mkdir: created directory ?/afs/eufus.eu/g2itmdev/user/g2michal/imas-kepler/2.5p2-2.1.3_IMAS_3.
 ↪20.0?
...
...
Done installing /afs/eufus.eu/g2itmdev/user/g2michal/imas-kepler/2.5p2-2.1.3_IMAS_3.20.0.
Run `module switch kepler/2.5p2-2.1.3_IMAS_3.20.0` to update $KEPLER to match.
Then run `kepler` to try your lightweight installation.
```

You have to switch module, to make sure that KEPLER variable points to proper location.

```
> module switch kepler/2.5p2-2.1.3_IMAS_3.20.0
```

Once you have set version of Kepler, you can run it by typing kepler

```
> kepler
The base dir is /afs/eufus.eu/g2itmdev/user/g2michal/imas-kepler/2.5p2-2.1.3_IMAS_3.20.0
Kepler.run going to run.setMain(org.kepler.Kepler)
JVM Memory: min = 1G,  max = 8G, stack = 20m, maxPermGen = default
...
...
```

2.1.11 IMAS Kepler 2.1.5 (release candidate)

Most recent steps for Gateway users

In order to use most recent version of Kepler do following. First of all make sure you have directory imas-kepler inside your $HOME

```
> mkdir -p $HOME/imas-kepler/modulefiles
```

Make sure to set IMAS_KEPLER_DIR variable inside .cshrc file

```
> echo "setenv IMAS_KEPLER_DIR $HOME/imas-kepler" >> ~/.cshrc
```

Now, you can load imasenv/3.21.0 module by calling

```
> module load imasenv/3.21.0
```

Note that this module uses kepler/2.5p2-2.1.5 instead of kepler/2.5p2-2.1.3

```
> module load imasenv/3.21.0
IMAS environment loaded.
Please do not forget to set database by calling 'imasdb <machine_name>' !
```

Now, you can install your personal Kepler installation (please note that since release 2.5p-2.1.5 and keplertools-1.7.0 it is possible to switch between different installations of Kepler (they will not collide).

```
> kepler_install my_own_kepler
Using IMAS_KEPLER_DIR at: /pfs/work/g2michal/imas-keplers.
Using KEPLER_SRC from KEPLER: /gw/swimas/extra/kepler/2.5p2-2.1.5.
mkdir: created directory ?/pfs/work/g2michal/imas-keplers/my_own_kepler
mkdir: created directory ?/pfs/work/g2michal/imas-keplers/my_own_kepler/.kepler?
mkdir: created directory ?/pfs/work/g2michal/imas-keplers/my_own_kepler/.ptolemyII?
mkdir: created directory ?/pfs/work/g2michal/imas-keplers/my_own_kepler/KeplerData?
Done installing /pfs/work/g2michal/imas-keplers/my_own_kepler/kepler.
?/gw/swimas/extra/keplertools/1.7.0/share/modulefiles/kepler? -> ?/pfs/work/g2michal/imas-
 ↪keplers/modulefiles/kepler/my_own_kepler?

Kepler was installed inside /pfs/work/g2michal/imas-keplers/my_own_kepler
Its module file is: /pfs/work/g2michal/imas-keplers/modulefiles/kepler/my_own_kepler
To load this environment, run: module switch kepler/my_own_kepler
To see available installations: module avail kepler
```

As you can see, your personal Kepler installations are available via modules. In order to switch to given version of Kepler you need to switch the module

```
> module switch kepler/my_own_kepler
```

Once you have set version of Kepler, you can run it by typing kepler

```
> kepler
kepler
The base dir is /marconi_work/eufus_gw/work/g2michal/imas-keplers/my_own_kepler/kepler
Kepler.run going to run.setMain(org.kepler.Kepler)
JVM Memory: min = 1G,  max = 8G, stack = 20m, maxPermGen = default
...
...
```

2.1.12 Installation based on README file

Installation instructions based on most recent version of IMAS Kepler

Detailed, up to date, instructions on how to install and switch between different installations of Kepler, can be found here

```
> git clone ssh://git@git.iter.org/imex/kepler-installer.git

> cat kepler-installer/README
```

You can also find latest documentation at following location (Gateway)

```
> cat $SWIMASDIR/extra/kepler-installer/README
```

2.2 General Grid Description and Grid Service Library

2.2.1 Resources

- GForge project page
- Linking to library: general , specific
- A tutorial talk. Note: some slides might be out of date, please refer to the documentation.

2.2.2 Documentation

- 4.09a Resources: Sources, Fortran Examples

  Documentation:

  - Release v1.2: Fortran 90 , Python , ualconnector ,
- 4.10a Resources: Sources, Fortran Examples

  Documentation:

  - Release v1.2: Fortran 90 , Python , ualconnector ,

2.2.3 Outdated documentation

This section collects information and documentation related to the general grid description.

- Some presentations:
  - A tutorial talk from 2011 ,
  - General Meeting 2011: Short overview talk and detailed presentation
- Instructions how to get a copy of the Grid Service Library
- Documentation for the EU-IM Grid Service Library: Fortran 90 , Python
- A short manual for ualconnector and VisIt

Some examples are included in the Grid Service Library distribution.

2.2.3.1 Example grids

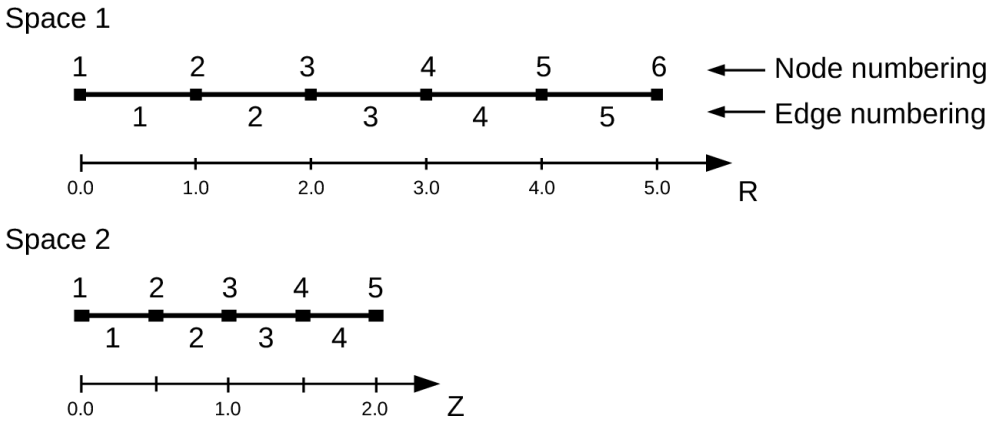
2.2.3.1.1 Example grid details

This section describes a number of example grids and gives some examples for specific constructs (object lists, subgrids).

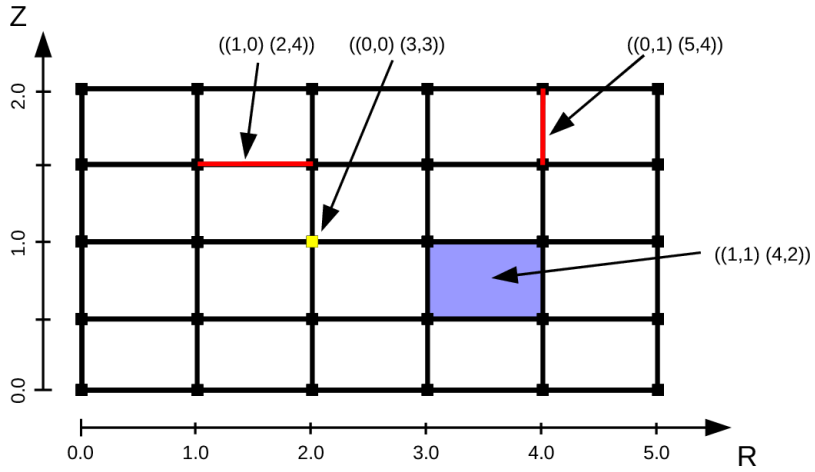
2.2.3.1.1.1 Example Grid #1: 2d structured R,Z grid

Note: the grids shown here are used in the unit tests of the grid service library implementation, i.e. the automated testing framework.

A 2d grid in (R,Z) constructed by combining two structured one-dimensional spaces. The spaces are defined as follows, they define nodes and edges as subobjects.



The whole grid then looks like this (attention, slightly differing scales in R and Z):



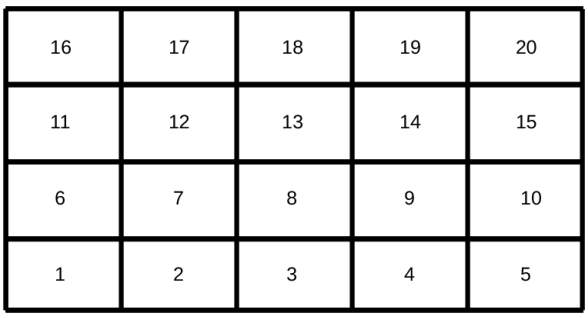
A couple of examples for object descriptor are given. Some explanations:

((1,1) (4,2)) = a 2d object (2d cell or face), implicitly created by combining the 1d object (edge) no. 4 from space 1 and the 1d object no. 2 from space 2. ((1,0) (2,4)) = a 1d object (edge), implicitly created by combining 1d object (edge) from space 1 with the 0d object (node) no. 4 from space 2. ((0,0) (2,2)) = a 0d object (node), implicitly created by combining 0d objects (nodes) no. 2 from space 1 and no. 2 from space 2.

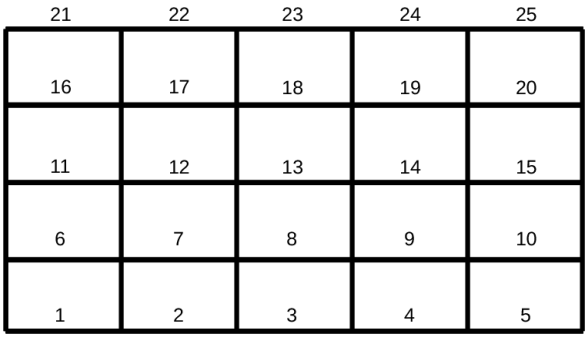
2.2.3.1.1.2 Object classes

This section shows the different object classes present in the grid. The implicit numbering of the objects in a class is obtained by iterating over all subobjects defining the objects, lowest space first.

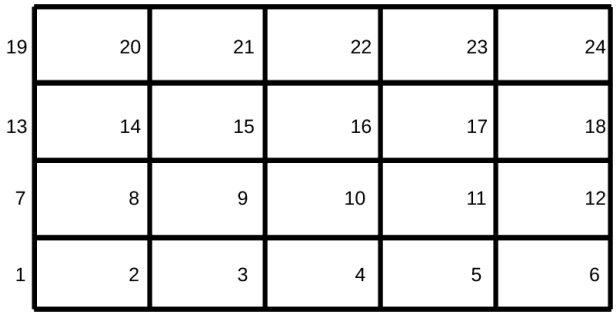
Object class (1,1): 2d cells/faces. They have the following implicit numbering:



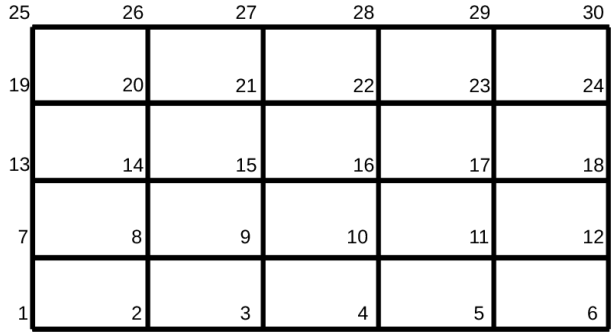
Object class (1,0): 1d edges, aligned along the R axis ("r-aligned"). They have the following implicit numbering:



Object class (0,1): 1d edges, aligned along the Z axis ("z-aligned"). They have the following implicit numbering:



Object class (0,0): 0d nodes. They have the following implicit numbering:



2.2.3.1.1.3 Example 2: B2 grid

2.2.3.1.2 Object list examples

Some examples for object lists, to explain the concept and show the notation. All examples refer to the 2d structured R,Z example grid #1 given above. Object descriptor A single object (= and object descriptor), for object with object class (1,1), object index (4,2).

```
((1,1) (4,2))
```

Explicit object lists An explicit object list is simply an enumeration of object descriptors. The ordering of the objects is given directly by their position in the list. Note that by definition, all objects in the list must be of the same class (An implementation of an explicit object list should enforce this. If you need lists of objects with differing class, have a look at subgrids).

An explicit list of 2d cells (faces), listing the four corner cells of the grid in the order bottom-left, bottom-right, top-left, top-right:

```
(((1,1) (1,1)),
((1,1) (5,1)),
((1,1) (1,4)),
((1,1) (5,4)))
```

Implicit object lists Implicit object lists use the implicit order of (sub)objects to form an efficient representation of (possibly large) sets of objects. They thus avoid explicit enumeration of individual objects as done in the explicit objects lists. The following examples demonstrate the implicit list notation. Note: the implicit list notation is used in the Python implementation of the grid service library in exactly the form given here.

Selecting all indices An implicit object list of all r-aligned edges:

```
((1,0) (0,0))
```

Object and subobject indices in the grid description start counting from 1, i.e. object no. 1 is the first object. The index 0 is special and denotes an undefined index. In this notation, it denotes all possible indices.

An implicit object list of the (z-aligned) boundary edges on the left boundary of the grid:

```
((0,1) (1,0))
```

The first entry of the index tuple denotes the first node in the r-space, the second entry denotes all edges in the z space. The implicit list denotes a total of 4 1d edges. Their implicit numbering is again given by iterating over all defining objects, lowest space first. The list therefore expands to

```
((0,1) (1,1))
((0,1) (1,2))
((0,1) (1,3))
((0,1) (1,4))
```

Selecting explicit lists of indices An implicit object list of the (z-aligned) right and left boundary edges:

```
((0,1) ([1,6],0))
```

The first entry of the index tuple denotes a list of nodes in the r-space, more specifically the first and the last (=6th) node. The second entry denotes again all edges in the z space. The implicit list then denotes a total of 8 1d edges in the following order:

```
((0,1) (1,1))
((0,1) (6,1))
((0,1) (1,2))
((0,1) (6,2))
((0,1) (1,3))
((0,1) (6,3))
((0,1) (1,4))
((0,1) (6,4))
```

Selecting ranges of indices An implicit object list of all 2d cells, except the cells on the left and right boundary.

```
((1,1) ((2,4),0))
```

The first entry of the index tuple denotes a range of edges in the r-space, more specifically the edges 2 to 4. The second entry of the index tuple denotes all four edges in the z-space. The implicit list then denotes a total of 12 2d cells in the following order:

```
((1,1) (2,1))
((1,1) (3,1))
((1,1) (4,1))
((1,1) (2,2))
((1,1) (3,2))
((1,1) (4,2))
((1,1) (2,3))
((1,1) (3,3))
((1,1) (4,3))
((1,1) (2,4))
((1,1) (3,4))
((1,1) (4,4))
```

All implementations of the grid service library define the constant GRID_UNDEFINED=0 to specify an undefined index. Use of GRID_UNDEFINED instead of 0 is advised to increase the readability of the code. The following notations are therefore equivalent ((1,0) (0,0)) = ((1,0) (GRID_UNDEFINED,GRID_UNDEFINED)) ((0,1) (1,0)) = ((0,1) (1,GRID_UNDEFINED))

2.2.3.1.3 Subgrid examples

A subgrid is an ordered list of grid objects of a common dimension. The difference to object lists is that they can contain objects of different object classes.

The subgrid concept is central to storing data on grids. To store data, first a subgrid has to be defined. The objects in the grid have a fixed order, which then allows to unambiguously store the data associated with the objects in vectors.

Technically, a subgrid is an ordered list of object lists, of which every individual list is either explicit or implicit. The ordering of the objects in the subgrid is then directly given by the ordering of the object lists and the ordering of the grid objects therein.

Subgrid example The following subgrid consists of all boundary edges of the 2d R,Z example grid #1, given as four implicit object lists.

```
((1,0) (0,1))    ! bottom edges
((0,1) (6,0))    ! right edges
((1,0) (0,5))    ! top edges
((0,1) (1,0))    ! left edges
```

Explicitly listing the objects in the order given by the subgrid gives:

```
1:  ((1,0) (1,1))    ! bottom edges
2:  ((1,0) (2,1))
3:  ((1,0) (3,1))
4:  ((1,0) (4,1))
5:  ((1,0) (5,1))
6:  ((0,1) (6,1))    ! right edges
7:  ((0,1) (6,2))
8:  ((0,1) (6,3))
9:  ((0,1) (6,4))
10: ((1,0) (1,5))    ! top edges
11: ((1,0) (2,5))
12: ((1,0) (3,5))
13: ((1,0) (4,5))
14: ((1,0) (5,5))
15: ((0,1) (1,1))    ! left edges
16: ((0,1) (1,2))
17: ((0,1) (1,3))
18: ((0,1) (1,4))
```

The number at the beginning of each line is the *local index* of the object, where local means locally in the subgrid. Note that, again, counting starts at 1.

2.2.3.2 Grid service library

2.2.3.2.1 Using the grid service library

2.2.3.2.1.1 Setting up the environment

The grid service library requires the EU-IM data structure version 4.09a (or later). Before using it you have to make sure your environment is set up properly. The following section assumes you are using csh or tcsh on the Gateway.

First, your environment variables have to be set up properly. To check them do

```
echo $TOKAMAKNAME
```

It should return

```
test
```

Also do

```
echo $DATAVERSION
```

It should return

```
4.09a
```

(or some higher version number). If either of them returns something different, run

```
source $EU-IMSCRIPTDIR/EU-IMv1 kepler test 4.09a > /dev/null
```

and check the variables again.

Second, you have to ensure your data tree is set up properly. Do

```
ls ~/public/itmdb/itm_trees/$TOKAMAKNAME/$DATAVERSION/mdsplus/0/
```

If you get something like “No such file or directory”, you have to set up the tree first by running

```
$EU-IMSCRIPTDIR/create_user_itm_dir $TOKAMAKNAME $DATAVERSION
```

and then do the previous check again.

2.2.3.2.1.2 Checking out and testing the grid service library

To be able to get the code of the grid service library, you have to be a member of the EU-IM General Grid description (itmggd) project (you can apply for this here).

Once you are a member, you can check out the code by

```
svn co https://gforge6.eufus.eu/svn/itmggd itm-grid
```

Then you can run the unit tests for the grid service library by

```
cd itm-grid
source setup.csh
```

This will setup environment variables (especially OBJECTCODE) and aliases. Then do

```
testgrid setup
```

This will set up the build system for the individual languages. It will also build and execute a Fortran program that writes a simple 2d example grid stored in an edge CPO into shot 1, run 1.

To actually run the tests do

```
testgrid all
```

This will go through the implementations in the different languages (F90, Python, ...) and run unit tests for every on of them. If all goes well, it should end with the message

```
Test all implementations: OK
```

If this is not the case, something is broken and must be fixed.

2.2.3.2.2 Example applications (outdated)

Note: this is a bit outdated. Have a look here.

2.2.3.2.2.1 Plotting 3d wall geometry with VisIt (temporary solution, not required any more)

This example plots a 3d wall representation stored in the edge CPO (in the future, this information will be stored in the wall CPO). The example data used here is generated by a preprocessing tool which is part of the ASCOT code.

1. Check out the grid service library (See above. You don't necessarily have to run the tests)
2. Change to the python/ directory and setup the environment:

```
cd itm-grid/python/; source setup.csh
```

3. Edit the file itm/examples/write_xdmf.py to use the right shot number

4. Run it (still in the python/ directory of the service library) with

```
   python26 itm/examples/write_xdmf.py

This will create two files: wall.xmf and wall.h5
```

5. Start visit with

```
   visit23

and open the wall.xmf file. Then select Plot->Mesh->Triangle and
click on the "Draw" button.
```

2.2.3.2.2.2 Using UALConnector to visualize CPOs using the general grid description

UALConnector allows you to bring data directly from the UAL into VisIt.

1. Check out the grid service library (See above. You don't necessarily have to run the tests)
2. Run UALConnector. Examples:

```
./itm-grid/ualconnector -s 9001,1,1.0 -c edge -u klingshi -t test -v 4.09a
```

```
./itm-grid/ualconnector -s 15,1,1.0 -c edge -u klingshi -t test -v 4.09a
```

3. When finished, close VisIt and terminate the UALConnector by typing 'quit'.

You don't even have to check out the service library. UALConnector is made available at

```
~klingshi/bin/itm-grid/ualconnector
```

, i.e.

```
~klingshi/bin/itm-grid/ualconnector -s 9001,1,1.0 -c edge -u klingshi -t test -v 4.09a
```

```
~klingshi/bin/itm-grid/ualconnector -s 15,1,1.0 -c edge -u klingshi -t test -v 4.09a
```

2.2.3.3 IMP3 General Grid Description and Grid Service Library - Tutorial

2.2.3.3.1 Setup your environment

```
echo $DATAVERSION
echo $TOKAMAKNAME
```

should give "4.09a" and "test". If not, run

```
source $EU-IMSCRIPTDIR/EU-IMv1 kepler test 4.09a > /dev/null
```

To copy the tutorial files:

```
cp -r ~klingshi/bin/itm-grid ~/public
```

Switch to the right version of the PGI compiler:

```
module unload openmpi/1.3.2/pgi-8.0 compilers/pgi/8.0
module load compilers/pgi/10.2 openmpi/1.4.3/pgi-10.2
```

To set up the environment:

```
cd $HOME/public/itm-grid/f90
source setup.csh
```

2.2.3.3.2 Compile & run examples

2d structured grid write example Source file is at:

```
src/examples/itm_grid_example1_2dstructured_servicelibrary.f90
```

Compile:

```
make depend
make $OBJECTCODE/itm_grid_example1_2dstructured_servicelibrary.exe
```

Run:

```
$OBJECTCODE/itm_grid_example1_2dstructured_servicelibrary.exe
```

2d structured grid read example Source file is at:

```
src/examples/itm_grid_example1_2dstructured_read.f90
```

Compile:

```
make $OBJECTCODE/itm_grid_example1_2dstructured_read.exe
```

Run:

```
$OBJECTCODE/itm_grid_example1_2dstructured_read.exe
```

2.2.3.3.3 Visualize

To visualize the data written by the example program

```
~klingshi/bin/itm-grid/ualconnector -s 9001,1,0.0 -c edge
```

To visualize a more complex dataset

```
~klingshi/bin/itm-grid/ualconnector -s 17151,899,1000.0 -c edge -u klingshi -t aug
```

Combining data from two CPOs:

```
~klingshi/bin/itm-grid/ualconnector -s 17151,898,1000.0 -c edge -s 17151,899,1000.0 -c edge -
↪u klingshi -t aug
```

CHAPTER

THREE

EUROPEAN TRANSPORT SIMULATOR (ETS)

3.1 ETS Documentation

This page contains useful information on the European Transport Simulator (ETS) including documentation, description of the pre and post-processing tools used with ETS as well as instructions on how to use ETS and tools on the EUROfusion Gateway.

3.1.1 Configuration of the ETS-5 workflow in Kepler

ETS-5 uses CPO for actor integration in Kepler and as input data to the workflow. This means that the user environment needs to be set up as ITM environment.

To do so login on the EUROfusion Gateway and type the following commands:

```
>module purge
>module load cineca
>module load itmenv/ETS_4.10b.10_v5.5.0
>source $ITMSCRIPTDIR/ITMv2.sh JET
>export ITM_KEPLER_DIR=$ITMWORK/my_keplers
>export _JAVA_OPTIONS=-Dsun.java2d.xrender=false
>export I_MPI_FABRICS=shm
>export _JAVA_OPTIONS="-Xss20m -Xms4g -Xmx8g -Dsun.java2d.xrender=false"
```

The command 'module load itmenv/ETS...' loads the itmenv environment and in particular in the case above the ETS / Kepler version 5.5.0 To load a different version just change the number e.g. v5.4.0

The $ITMSCRIPTDIR/ITMv2.sh JET command will set up your local database folder to 'JET'. This means that any simulation done with ETS will be saved in the JET folder (even if you are simulating TCV!!). If you would like to simulate any other Tokamak, type again the command and change JET with e.g. AUG.

The remaining commands are JAVA options for running Kepler and setting of MPI useful for running parallel actors.

If it is the first time you run ETS then you will need to install your first 'dressed' Kepler version which corresponds to Kepler with all the WPCD actors embedded in it.

This can be done by executing the following command

```
>install_kepler.sh ets_v550 trunk/ETS_4.10b.10_v5.5.0/central "dressed central kepler v5.5.0"
>switch_to_kepler.sh ets_v550
```

For loading the Workflow+tools (import data, postprocessing):

```
>svn co https://gforge6.eufus.eu/svn/keplerworkflows/tags/ETS_4.10b.10_v5.5.0
```

The above command requires enabled access to GFORGE (if you are a 'simple' user you might need to apply for access to GFORGE). Typing the command above will check out and store the ETS_worflow.xml and useful python scripts in the directory from where it is issued.

Plotting routines such as kplots can be found under

```
>cd $KEPLER
```

You are now ready to start ETS!!

The latest version of ETS-5 is v5.5.0 (09/12/2019)

To launch Kepler and load the ETS-5 workflow just type

```
>kepler.sh
```

or if you like to see the the log messages printed on the scree while ETS runs

```
>kepler.sh -nolog
```

Once the Kepler canvas opens, chose the option 'load workflow' from the File menu and select the workflow you would like to load. The recommendation is to use the ETS workflow released with the version release procedure and then upload your parameter settings via the parameter file. See the option - running ETS with autoGui

A video showing how to run and set up ETS-5 can be viewed here

https://www.youtube.com/watch?v=dv427_XOFf4&t=87s

3.1.2 ETS releases

ETS release 5.5.0 is installed on the Gateway.

Quick installation instructions (to update your environment) are available here (password protected areas):

https://portal.eufus.eu/twiki/bin/view/Main/Installation_of_latest_kepler_release

Detailed instructions are available here:

https://portal.eufus.eu/twiki/bin/view/Main/User_Guide_accessing_JET_data

List of modifications (as compared to the previous release) is available here:

https://portal.eufus.eu/twiki/bin/view/Main/Updates_550

3.2 ETS workflows in KEPLER

The ETS workflow is used for 1-D transport simulation of a tokamak core plasma.

ETS workflows in KEPLER:

- use actors and composite actors from the WPCD / IMAS fusion library
- complex, but clearly structured workflow, which offers user friendly interface for configuring the simulation

- allow for easy modifications (connecting new modules, or reconnecting parts of the workflow) through an easy graphical interface
- provide users with all updates through the version control system
- still in active development tool (ETS-6)

Starting the workflow: If you have the workflow already installed, there are several ways to execute it:

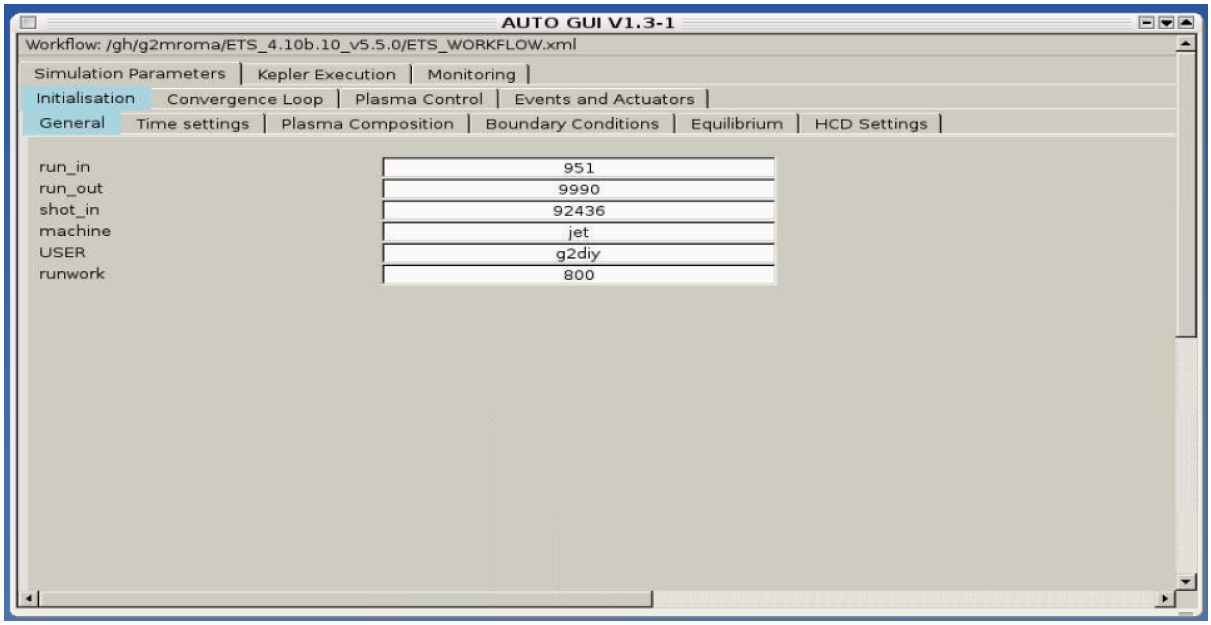
- For execution via kepler GUI:

```
>kepler.sh workflow_path/workflow_name.xml
```

- for executution via autoGui

```
>autoGui
```

once the GUI opens select load workflow after which a parameter file can be loaded. You can create a parameter file by loading the standard workflow released with the Kepler version and then chosing the option from the top menu 'save parameter file'. The use of autoGui is strongly recommanded as workflows are large xml files while parameter files are small and do not take all your disk space. Moreover parameter files can be loaded in any version of ETS-5 by opening the standard workflow included in the release.



3.2.1 Configuring the ETS run

3.2.1.1 Workflow parameters

3.2.1.1.1 General Parameters

- USER - your userid
- MACHINE - machine name (database name) for which comutations are done
- SHOT_IN - input shot number

- RUN_IN - input run number
- RUN_OUT - output run number
- RUNWORK - work directory number (typically 800)

3.2.1.1.2 Time resolution

Start and End time:

- TBEGIN - Computations start time
- TEND - Computations end time

3.2.1.1.3 Transport

- NRHO - number of radial points for transport equations

3.2.1.1.4 Equilibrium

- NPSI - number of points for equilibrium 1-D arrays
- NEQ_DIM1 - number of points for equilibrium 2-D arrays, first index
- NEQ_DIM2 - number of points for equilibrium 2-D arrays, second index
- NEQ_MAX_NPOINTS - maximum number of points for equilibrium boundary

3.2.1.1.5 Numerics

- NUMERICAL_SOLVER - choice of the numerics solving transport equations (RECOMENDED SELECTION: 3 or 4)
- EXPLICIT HYPER DIFFUSIVITY - Constant diffusivity used in the stabilization scheme needed to deal with stiff transport models
- IMPLICIT HYPER DIFFUSIVITY - Same as above used in the implicit part of the solver
- MINIMUM TIME STEP - Minimum time step allowed in the transport solver
- MAXIMUM TIME STEP - Maximum time step allowed in the transport solver

3.2.1.1.6 Equilibrium

- NPSI - number of points for equilibrium 1-D arrays
- NEQ_DIM1 - number of points for equilibrium 2-D arrays, first index
- NEQ_DIM2 - number of points for equilibrium 2-D arrays, second index
- NEQ_MAX_NPOINTS - maximum number of points for equilibrium boundary

Time step:

- right click on the box BEFORE THE TIME EVOLUTION

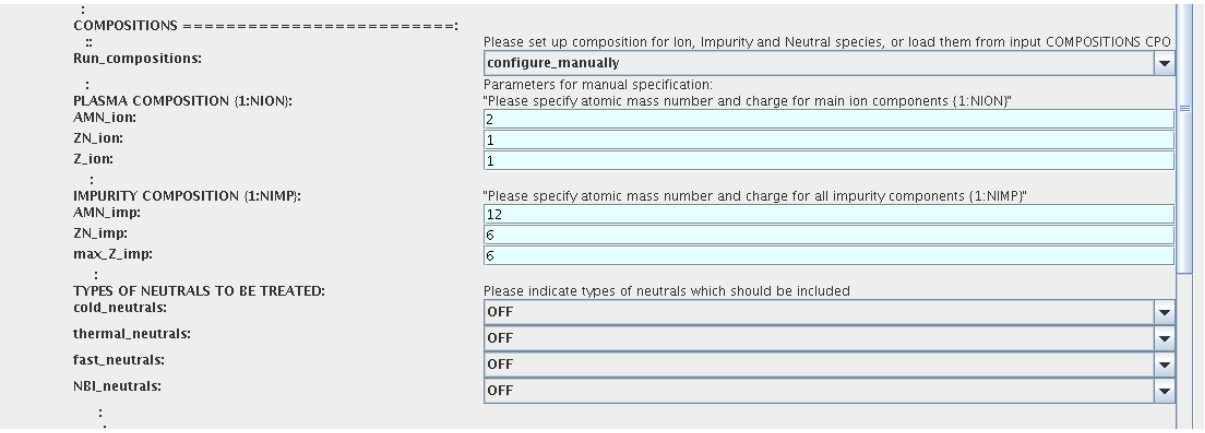
- select Configure actor
- TAU :specify value of the time step in [s]
- TAU_OUT : specify value of the output time interval in [s]
- Commit

3.2.1.2 Ion, Impurity and Neutral Composition

Before starting the run you need to define types of main ions, impurity (optional) and neutrals (optional) to be included in simulations.

To define plasma composition:

- right click on the box BEFORE THE TIME EVOLUTION
- select **Configure actor**
- choose one of modes for setting Run_compositions
  - from_input_CPO - will pick up the COMPOSITIONS structure of the COREPROF CPO saved to the input shot;
  - configure_manually - will force the composition from the values specified below
- specify values of atomic mass (AMN_ion), nuclear charge (ZN_ion) and charge (Z_ion , from the first ion to the last [1:NION] , separated by commas
- (optional) specify values of atomic mass (AMN_imp), nuclear charge (ZN_imp) and maximal ionization state (max_Z_imp) for impurity ions, from the first to the last [1:NIMP] , separated by commas
- (optional)for neutrals activate, by switchen them to **ON**, the types which shall be followed by neutral solver
- press **Commit**



3.2.1.3 Equations to be solved and boundary conditions

3.2.1.3.1 Main Plasma

Before starting the run you need to select the type and value of the boundary conditions for all equations. Please note that the value should correspond to the type. All equations allow for following types of boundary conditions:

- OFF - equation is not solved, initial profiles will be kept for whole run
- value - edge value should be specified
- gradient - edge gradient should be specified
- scale_length - edge scale length should be specified
- generic - generic form: a1*y´ + a2*y = a3 of the boundary condition is assumed, 3 coefficients (a1, a2, a3) should be provided
- value_from_input_CPO - equation is solved, edge value evolution will be red from input shot
- profile_from_input_CPO - equation is not solved, profile evolution will be red from input shot

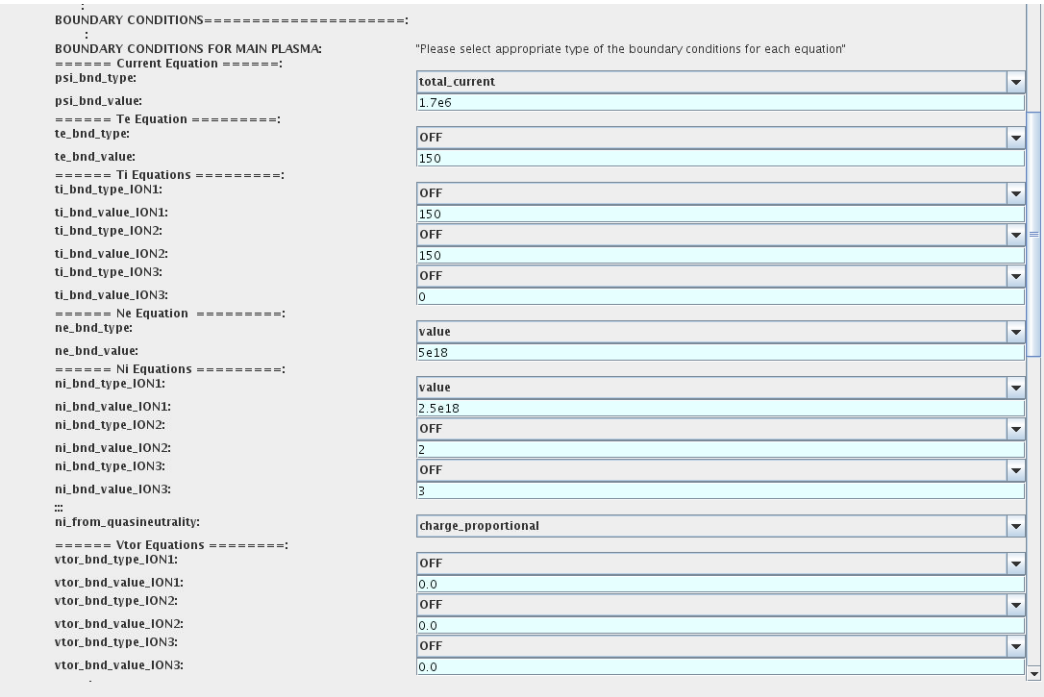
The particular equation will be activated if the boundary condition type for it is other than *OFF*

To set up boundary conditions:

- right click on the box BEFORE THE TIME EVOLUTION
- select **Configure actor**
- select appropriate boundary condition for each equation
- specify values for boundary conditions corresponding to the type and to the ion component
- **Commit**

The workflow will not allow the user all particle components (ions[1:NION]+electrons) to be run predictively. At least one of them shall be set to OFF (this component will be computed from quasi-neutrality condition).

!!! If electron density is solved, all ions with ni_bnd_type=OFF will be computed from the quasineutrality condition and scaled proportional to specified *ni_bnd_value* or inversely proportional to their charge, *charge_proportional*. This is defined by option: *ni_from_quasineutrality*.

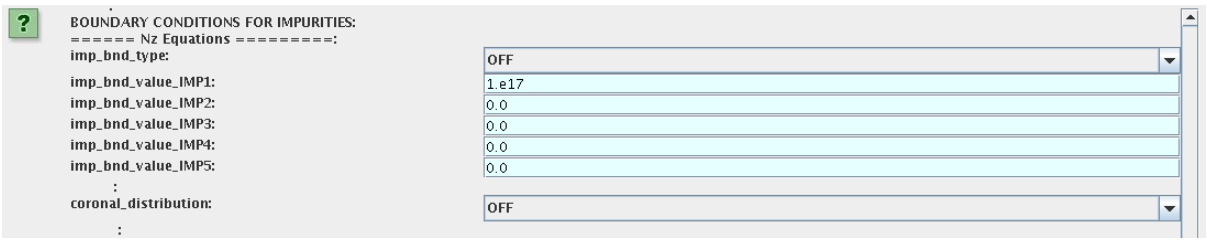


3.2.1.3.2 Impurity

You can set up the boundary conditions for impurity ions in a similar way as for main ions. !!! Note, that at the moment only types: *OFF*; *value* and *value_from_input_CPO* are accepter by impurity solver.

To set up boundary conditions:

- right click on the box BEFORE THE TIME EVOLUTION
- select **Configure actor**
- select appropriate boundary condition for each impurity species (OFF-equation is not solved)
- specify values for boundary density of each impurity component [1:MAX_Z_IMP], separated by commas
- **Commit**



Interface for impurity boundary condition has additional option, *coronal_distribution*, that allow to pre-set the edge values or entire profiles of individual ionization states from coronal distribution. In tis case only single value is required to be specified for each impurity boundary value. The options are:

- OFF - the boundary values for impurity densities will be as they are specified above;
- boundary_conditions - the boundary densities will be renormalized with corona, using the first element from above as a total density

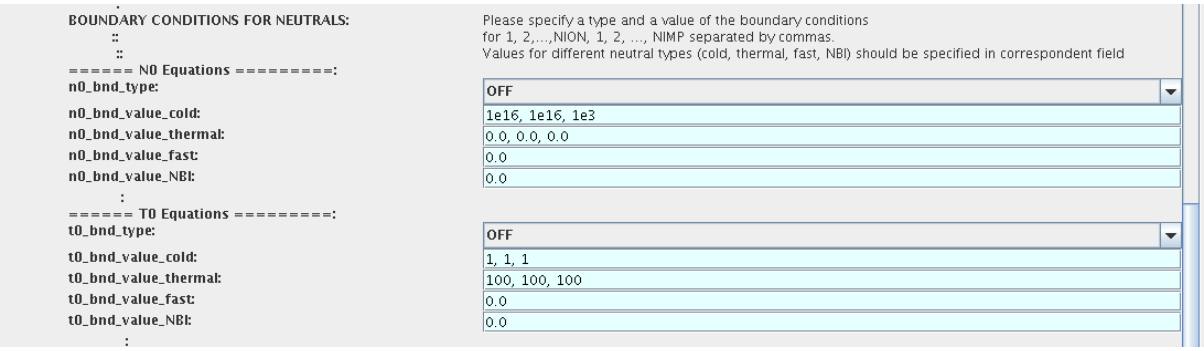
- boundary_conditions_and_profiles - the boundary densities and starting profiles will be renormalized with corona, using the first element from above as a total density

3.2.1.3.3 Neutrals

Note, that ALL values should be specified in the order: {*1, 2, 3 ...NION, 1, 2, 3, ...NIMP*}

To set up boundary conditions:

- right click on the box BEFORE THE TIME EVOLUTION
- select **Configure actor**
- select appropriate boundary condition for each neutral species (OFF-equation is not solved)
- specify values for boundary density and temperature of each neutral component [1, 2, 3 ...NION, 1, 2, 3, ...NIMP], separated by commas
- **Commit**



3.2.1.3.4 Input profiles interpolation

You are going to start the ETS run from some input shot, which might contain some conflicting rho grids saved to different CPOs. Thus there is a choice for the user to decide on the grid on which the starting profiles should be load by the worflow,

Interpolation_of_input_profiles.

To define the interpolation grid select:

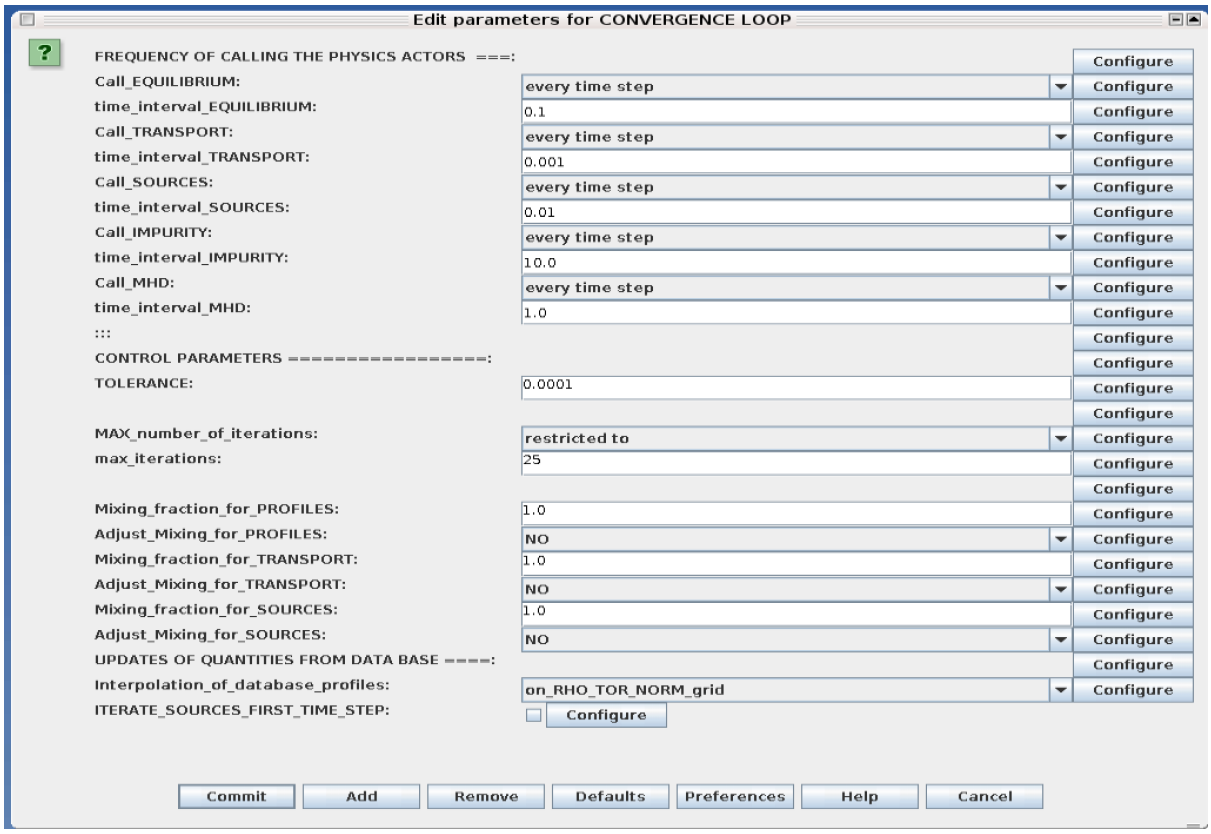
- on_RHO_TOR_grid - interpolate input profiles based on the grid specyfied in [m];
- on_RHO_TOR_NORM_grid - interpolate input profiles based on normalised rho grid [0:1]

3.2.1.4 Convergence loop

ETS updates input from different physics actors in a sequence, which is finished by solving the transport equations. Ther are possible none-linear couplings between different parts of the system. These nonelinearities are treated by the ETS using iterations. The decision to step in time is made by the ETS based on the criteria that the maximum relative deviation of main plasma profiles is lower than some

predefined tolerance. There is a number of settings and switches in the ETS that are used by the iterative scheme. To edit them do:

- right click on the box CONVERGENCE LOOP
- select **Configure actor** to edit settings
- choose your settings
- **Commit**



Switches in the field *FREQUENCY OF CALLING THE PHYSICS ACTORS* define how many times the actors of a certain cathegory (equilibrium, transport, etc.) should be called.

Switches and parameters in the field *CONTROL PARAMETERS* define how iterations are done

- Tolerance - defines the maximum relative error of profiles change compared to previous iteration. If it is achieved the time steping is done.

For highly none-linear case the required precision can be achieved faster by the iterative scheme if only fraction of the new solution is mixed to the previous state. The following scheme is adopted by the ets to reduce none-linearities in profiles, transport coefficients and sources:

```
Y = (Amix * Y+) + ((1-Amix)*Y-)
```

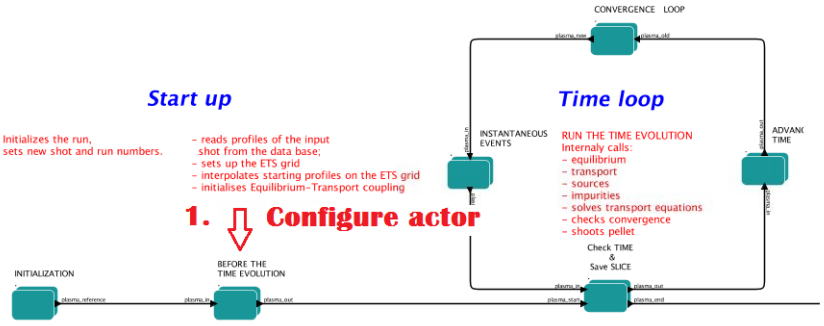
where Amix is the mixing fraction You can activate the mixing of profiles, transport coefficient and sources by selecting the corresponding *Mixing_fraction_...* to be between [0:1] You also can activate the authomatic ajustment of this fraction by selecting: *Ajust_Mixing_for_...* to *YES*

3.2.1.5 Equilibrium

3.2.1.5.1 Initialization Settings

Before starting the run you need to set up your initial equlibrium. There are several options to do it: if your input shot contains the consistent equilibrium with all necessary parameters - you can start immediately from it; if your input shot contains the equilibrium but it is not consistent or some parameters are missing you can check it automatically; if your input equilibrium is corrupt or not present - you can define the starting equlinbrium by tree moment description. To select your starting equilibrium please do:

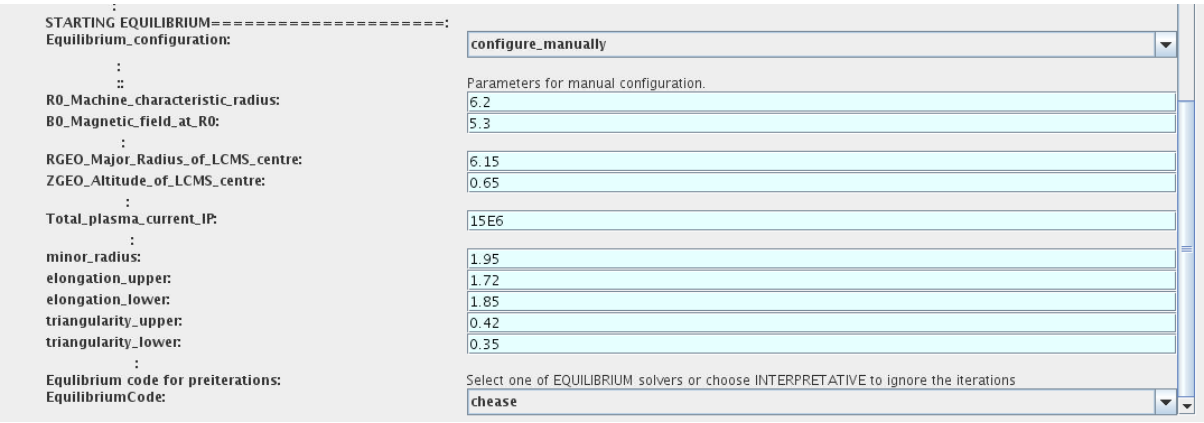
- right click on the box BEFORE THE TIME EVOLUTION
- select **Configure actor** to edit settings
- Select your settings or specify values
- **Commit**



SETTINGS:

- Equilibrium_configuration - select configure_manually if you like to specify configuration below; select from_input_CPO if all quantities should be picked up from the input CPO
- R0_Machine_characteristic_radius - Characteristic radius of the machine, here B0 is measured [m]
- B0_Magnetic_field_at_R0 - Magnetic field measured at the position R0 [T]
- RGEO_Major_Radius_of_LCMS_centre - R coordinate of the geometrical centre of the LCMS [m]
- ZGEO_Altitude_of_LCMS_centre - Z coordinate of the geometrical centre of the LCMS [m]
- Total_plasma_current_IP - plasma current within the LCMS [A]
- Minor_radius - minor radius of the LCMS [m]
- Elongation - elongation of the LCMS [-]
- Triangularity_upper - upper triangularity of the LCMS [-]
- Triangularity_lower - lower triangularity of the LCMS [-]

- Equilibrium code - select one of available equilibrium solvers to check the consistency between starting equilibrium and current profile; use INTERPRETATIVE if you trust your input data (in this case the check will be ignorred).

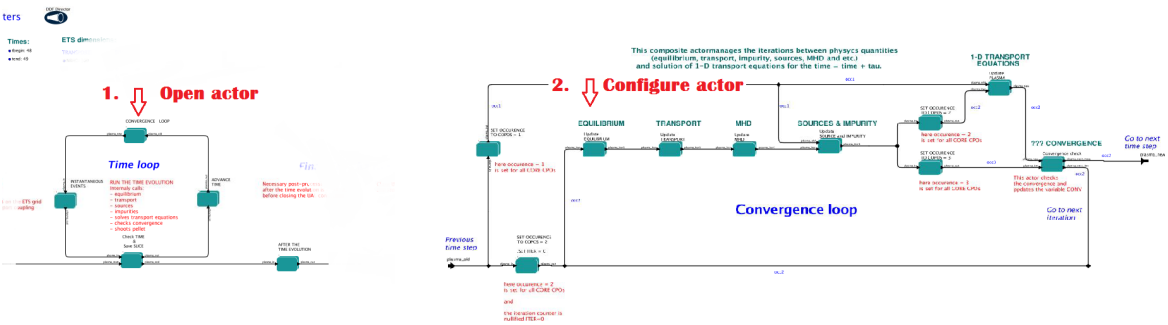


Please note, that different equilibrium solvers might require slightly different input. Thus it is a user responsibility to check that the information inside input shot/run is enough to run selected equilibrium solver.

3.2.1.5.2 Run Settings

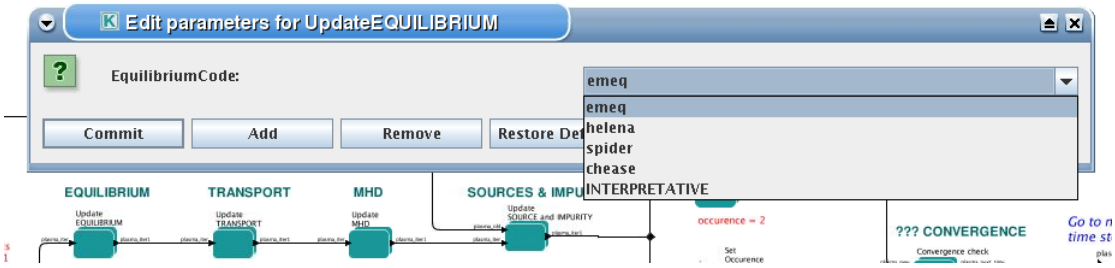
There are several equilibrium solvers connected to the ETS. You can select the one of them.Therefore please do:

- right click on the box CONVERGENCE LOOP
- select **Open actor**
- right click on the box EQUILIBRIUM
- select **Configure actor** to edit settings
- choose your equilibrium solver
- **Commit**



INTERPRETATIVE means that the ETS will not update the equilibrium, instead it will be using the initial equilibrium.

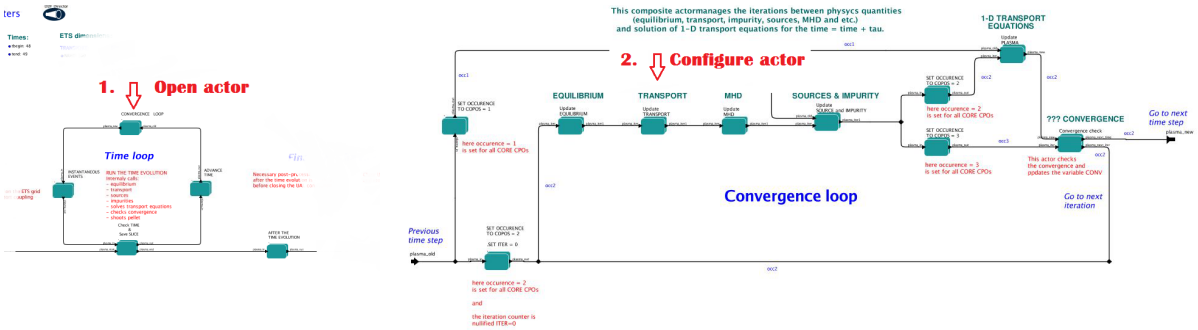
Please note, that it is better to select the same code as you used for pre-iterrations. Because outputs of different equilibrium solver are not necessary done with the same resolution. Therefore the routine saving the information to the data base might brake due to uncompatible sizes of some signals.



3.2.1.6 Transport

The settings for TRANSPORT can be done inside the CONVERGENCE LOOP composite actor. Therefore please do:

- right click on the box CONVERGENCE LOOP
- select **Open actor**
- right click on the box TRANSPORT
- select **Configure actor** to edit settings
- choose your settings
- press **Commit**



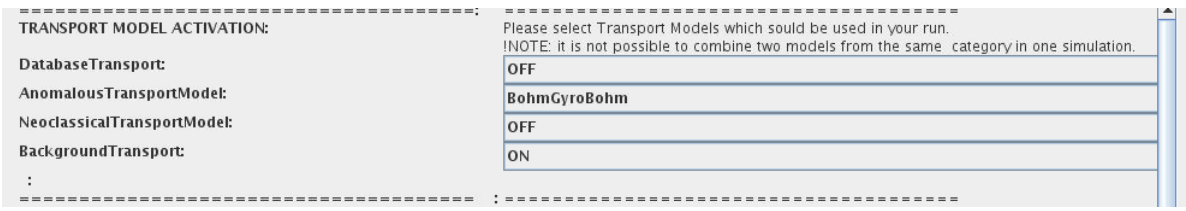
3.2.1.6.1 Transport models

ETS constructs the total transport coefficients from the combination of Anomalous transport (model choice), Neoclassical transport (model choice), Database transport (transport coefficients be saved to the input shot) and Background transport (Transport coefficients defined through the GUI interface)

D_tot = D_DB*M_DB + D_AN*M_AN + D_NC*M_NC + D_BG*M_BG

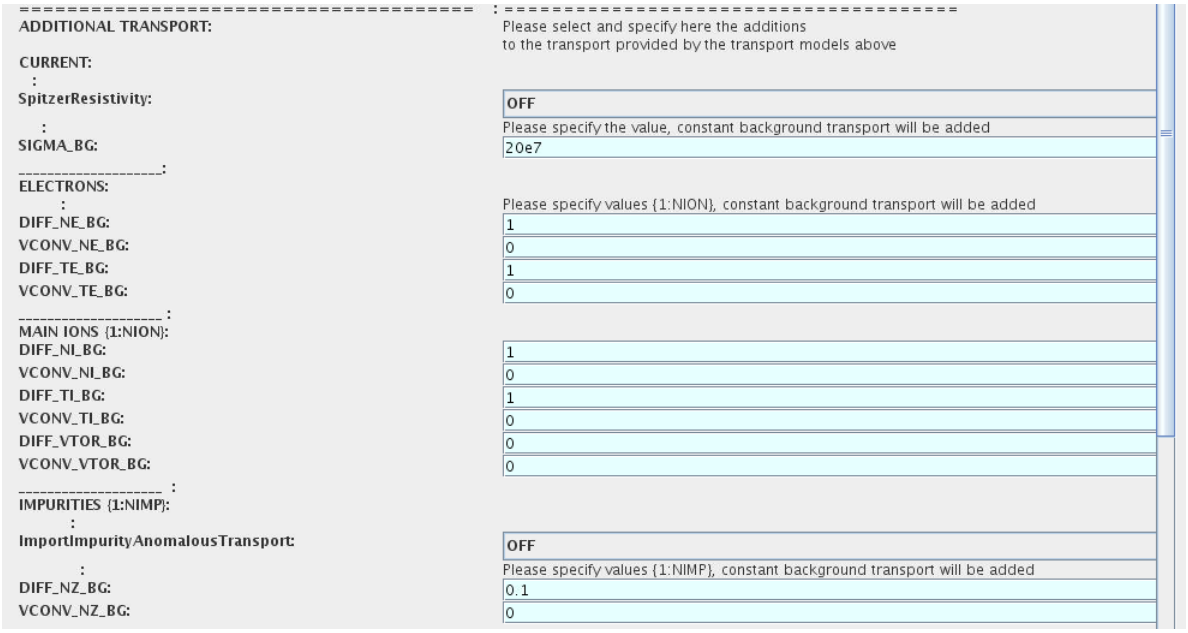
You should choose from the list of available models in each cathegory or switch it **OFF**

Individual multipliers for all channels shall be specified on the lower level through the code parameters of Transport Combiner



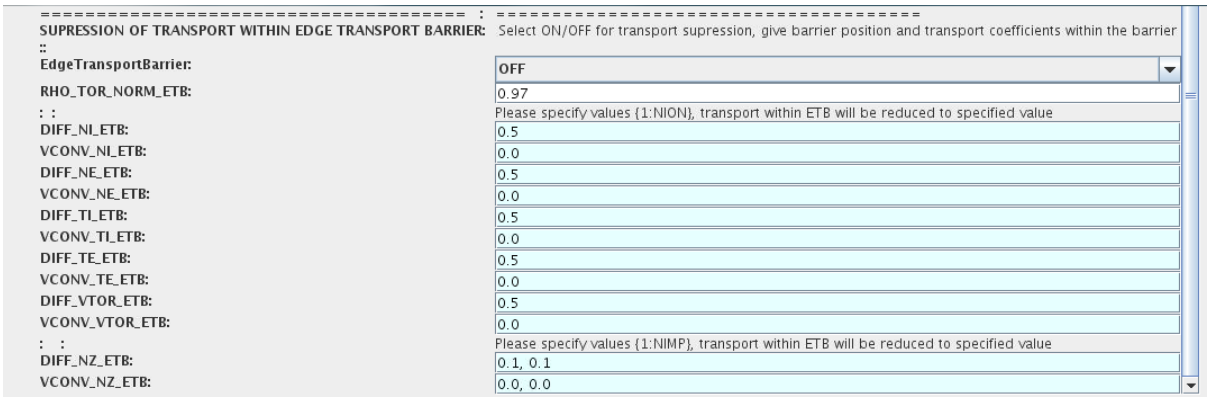
3.2.1.6.2 Background transport

You can add the constant background level for each coefficient (ion and impurity coefficients are expected to be the strings of [1:NION] and [1:NIMP] elements respectively, separated by commas)



3.2.1.6.3 Edge transport barrier

In this section you can artificially supress the transport outside of specified *RHO_TOR_NORM_ETB*. Total transport coefficients for all transport channels (ne, ni, nz, Te, Ti,...) will be reduced to constant values specified below (ion and impurity coefficients are expected to be the strings [1:NION] and [1:NIMP] respectively)

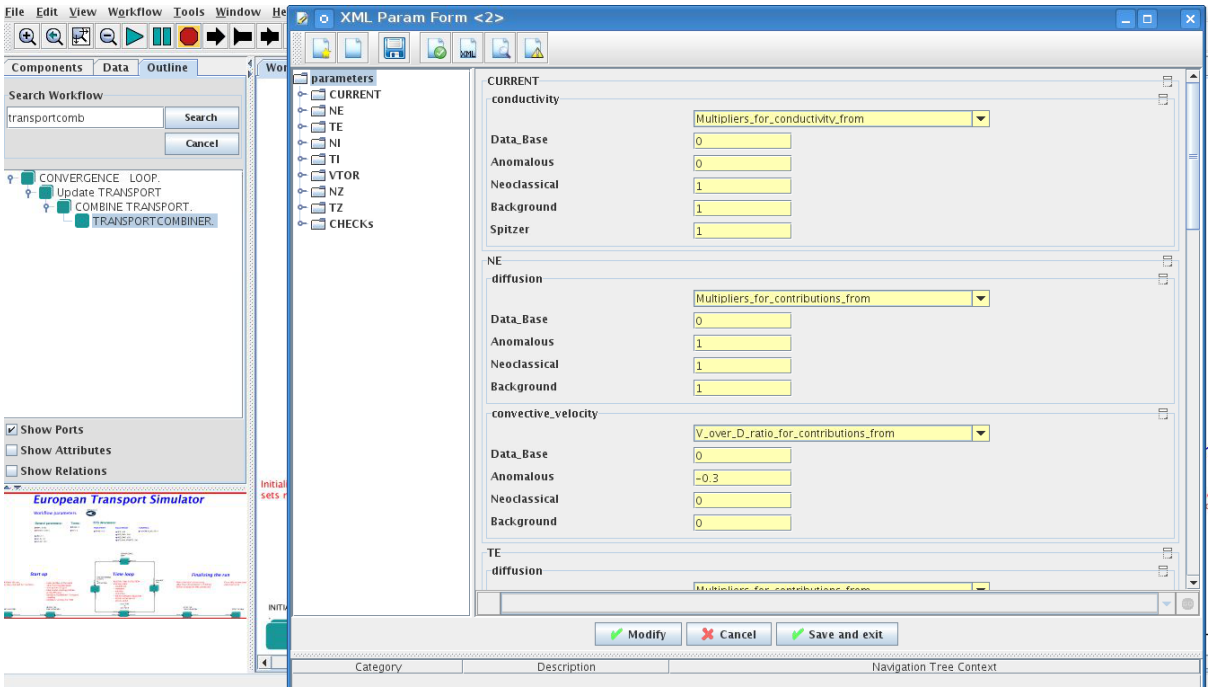


3.2.1.6.4 Total transport coefficients

The fine tuning of of transport coefficients can be done through editing the XML code parameters of the **transport combiner** actor:

- In Outline browse for transportcombiner

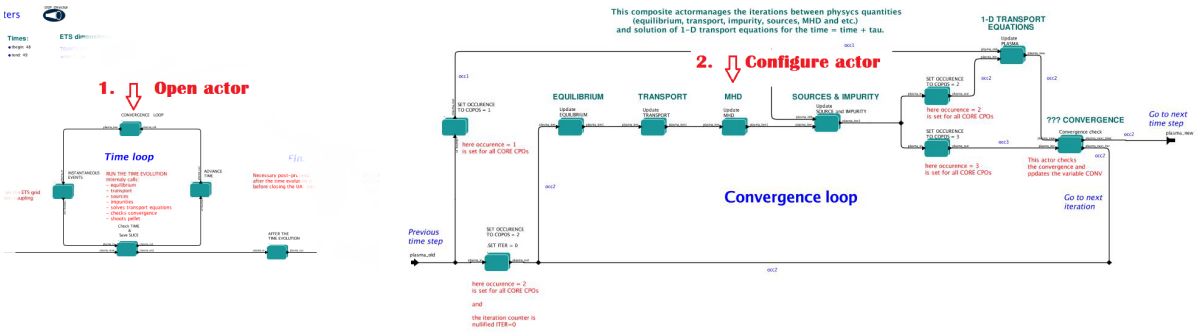
- select **Configure actor**
- click **Edit Code Parameters**
-
  - If you select **OFF** contributions from all transport models to this channel will be nullified;
  - If you select **Multipliers_for_contributions_from** the transport channel will be activated, and the total transport coefficient will be combined from active tranport models. You gust need to specify multiplier against each channel;
  - For convective velocity there is an additional option **V_over_D_ratio_for_contributions_from**. With this option selected the combiner will ignore the convective components provided by transport models. The convective velocity will be determined from the diffusion coefficient by applying fixed V/D ratio (for inward pinch the values should be negative!).
- **Save and exit**
- **Commit**



3.2.1.7 MHD

The settings for MHD type of events can be done inside the CONVERGENCE LOOP composite actor. Therefore please do:

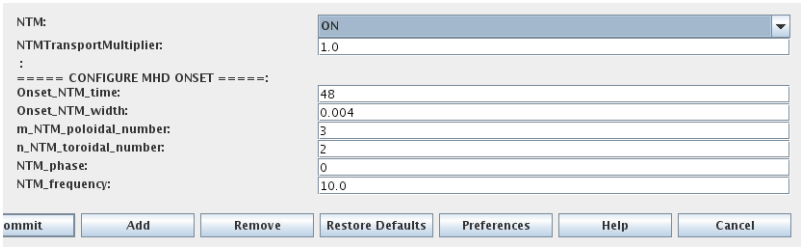
- right click on the box CONVERGENCE LOOP
- select **Open actor**
- right click on the box MHD
- select **Configure actor** to edit settings
- choose your settings
- **Commit**



At the moment ETS allows only for NTM to be activated.

User can ajust the following NTM settings:

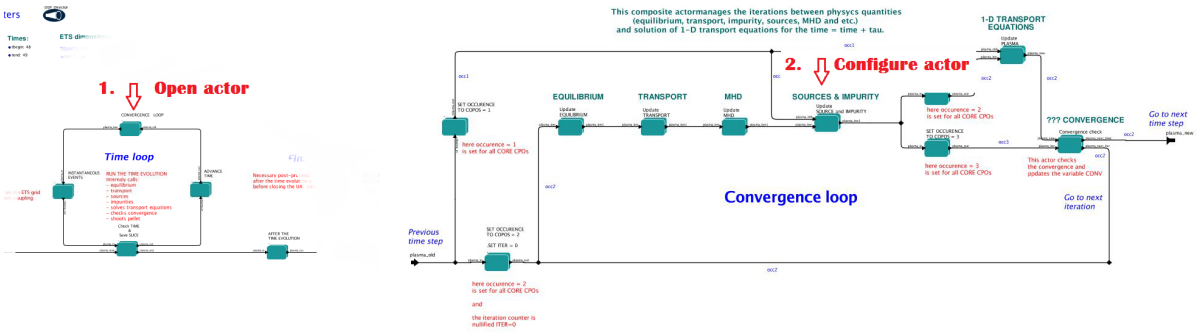
- NTM – **ON** means that ETS will add the NTM driven transport to the total transport coefficient; **OFF** -ignored
- NTMTransportMultiplier – the transport contrinution from NTM will be multiplied with this value
- Onset_NTM_time - activation time for the NTM mode
- Onset_NTM_width - starting width of the mode
- m_NTM_poloidal_number
- n_NTM_toroidal_number
- NTM_phase
- NTM_frequency



3.2.1.8 Sources and impurity

The settings for SOURCES AND IMPURITY can be done inside the CONVERGENCE LOOP composite actor. Therefore please do:

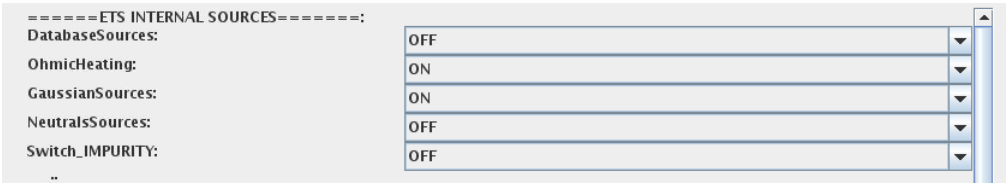
- right click on the box CONVERGENCE LOOP
- select **Open actor**
- right click on the box SOURCES AND IMPURITY
- select **Configure actor** to edit settings
- choose your settings
- **Commit**



3.2.1.8.1 Analytical & Impurity sources

There is a number of sources developed by WPCD, which are actors or internal routines of the transport solver. You can activate them by selecting **ON / OFF** in front of corresponding source:

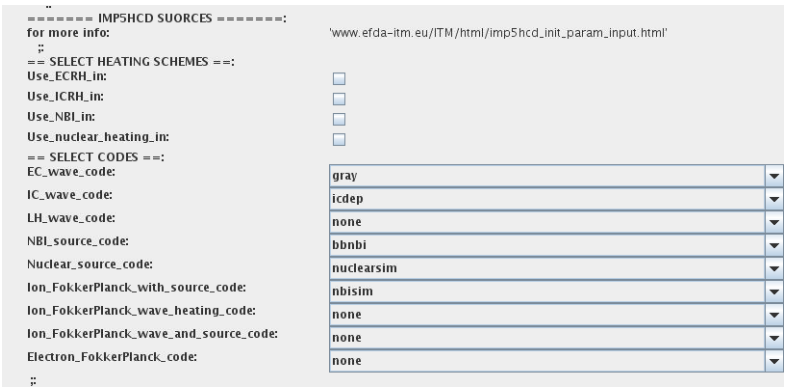
- Database Sources – **ON** - ETS will pick up the evolution of source profiles saved to your input shot/run; **OFF** -ignored
- Ohmic Heating – **ON** - ETS will compute Ohmic heating internaly; **OFF** -ignored
- Gaussian Sources – **ON** - ETS will add sources from the Gaussian source actor (you can configure heat and particle deposition profiles by editing the code parameters of the actor); **OFF** -ignored
- Neutral Sources – **ON** - Fluid neutrals will be solved according to the boundary conditions specified on ¨Before_time_evolution¨ composite actor interface; **OFF** -ignored
- Switch_IMPURITY – **ON** - Impurity density and radiative sources will be computed; **OFF** -ignored; **INTERPRETATIVE** – profiles of impurity density will be read from input shot/run



3.2.1.8.2 HCD sources

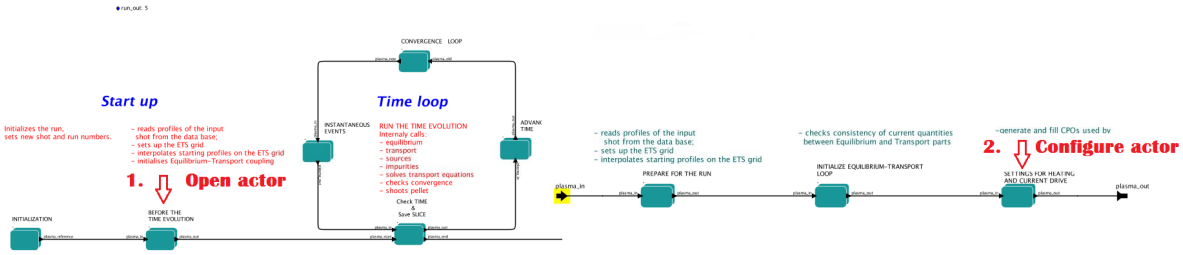
There is a number of sources developed by WPCD, that are incorporated by the ETS workflow.

For the HCD sources please activate the type of heating source, by ticking the box in front of it, and select the code to simulate it.

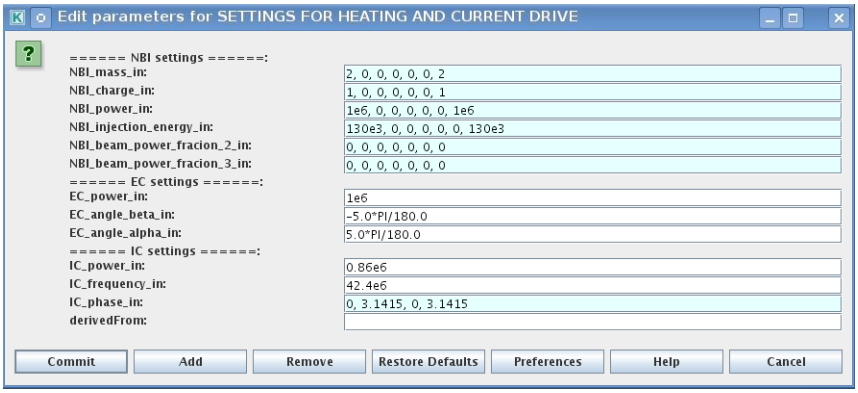


You also need to configure initial HCD settings. Therefore please:

- right click on the box BEFORE THE TIME EVOLUTION
- select **Open Actor**
- right click on the box SETTINGS FOR HEATING AND CURRENT DRIVE
- select **Configure actor**
- edit the stettings
- **Commit**



Please note that settings for NBI are done independent for each PINI. Therefore, for NBI settings, please insert the values separated by commas. The number of the element in the array corresponds to the number of activated PINI. Maximum accepted number of PINIs = 16.



3.2.1.8.3 Power control

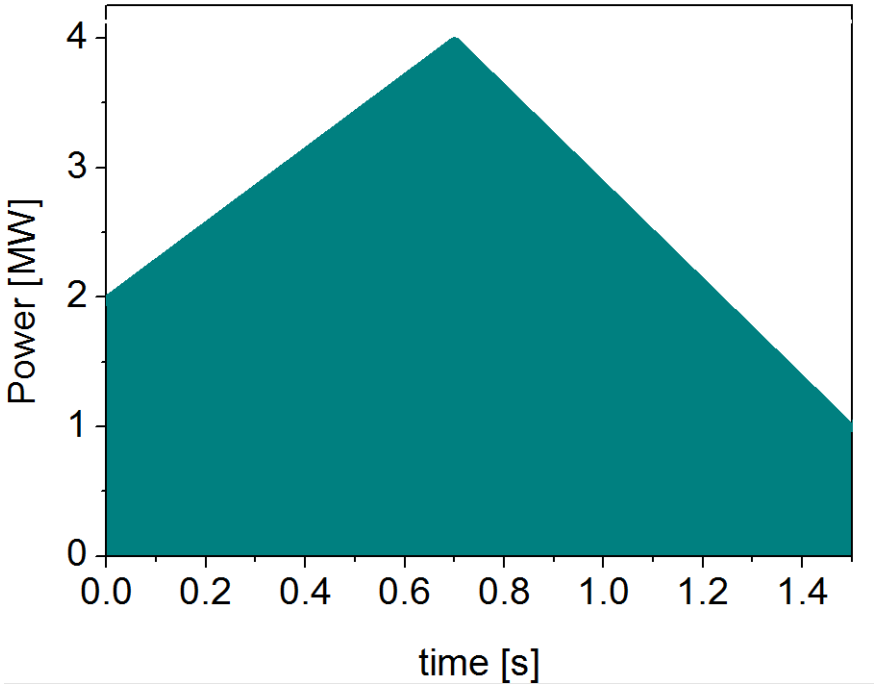
You also can activate the power control for the IMP5HCD sources.

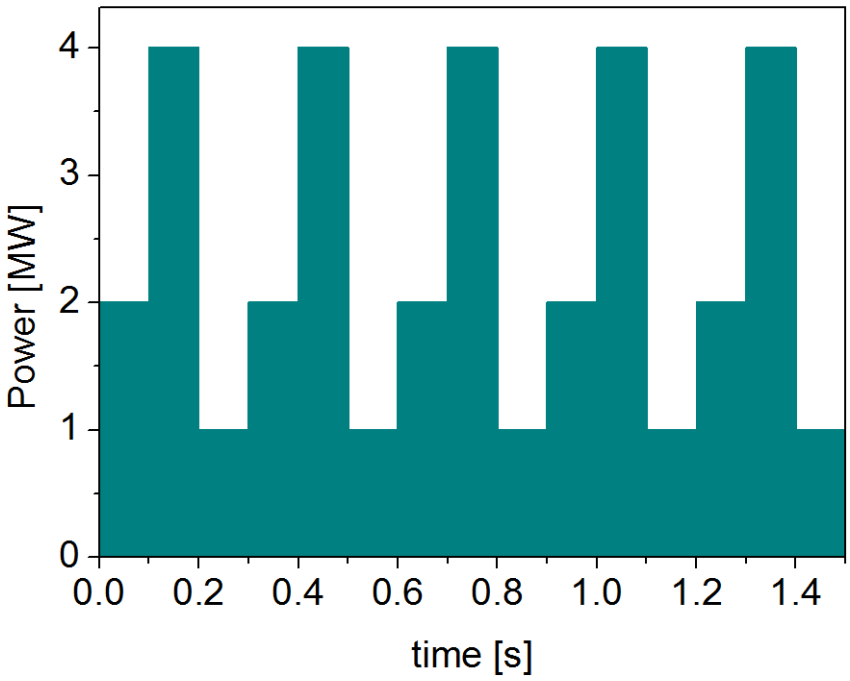
If the POWER_CONTROL is not **OFF**, there are two modes of operation: **specific** and **frequency**

For **specific** you should specify the time sequence separated by commas and the corresponding power sequence (where first power level corresponds to the first time, second to second and etc.). Linear interpolation will be done between the sequence points. For example: if you give the power **sequence** = 2e6,4e6,1e6 and **times** = 0.0, 0.7, 1.5 (s) the delivered power would be:

For **frequency** you should specify the power levels sequence separated by commas, start and end time of the power control and the frequency of switching between these levels. For example: if you give the power **sequence** = 2e6,4e6,1e6 and **frequency** = 10 (Hz) **tstart** = 0.0 (s) **tend** = 1.5 (s) the delivered power would be:

== POWER CONTROL ==:	
NBI_power_control:	specific
Times_NBI:	0, 100
Power_NBI_P1:	2E6, 2E6
Power_NBI_P2:	0
Power_NBI_P3:	0
Power_NBI_P4:	0
Power_NBI_P5:	0
Power_NBI_P6:	0
Power_NBI_P7:	2E6, 2E6
Power_NBI_P8:	0
Power_NBI_P9:	0
Power_NBI_P10:	0
Power_NBI_P11:	0
Power_NBI_P12:	0
Power_NBI_P13:	0
Power_NBI_P14:	0
Power_NBI_P15:	0
Power_NBI_P16:	0
tstart_NBI_control:	48
tend_NBI_control:	49
frequency_NBI_control:	100
ECRH_power_control:	OFF
Times_ECRH:	48
Power_ECRH:	3e6,0e6,2e6
tstart_ECRH_control:	48
tend_ECRH_control:	49
frequency_ECRH_control:	100
ICRH_power_control:	OFF
Times_ICRH:	0
Power_ICRH:	0
tstart_ICRH_control:	47
tend_ICRH_control:	55
frequency_ICRH_control:	100





3.2.1.8.4 Total power

Profiles of the total source for each channel are obtained from the the individual contributions (Data Base, Gaussian, Neutrals, Impurity and HCD) as a summ of all activated sources multiplied with coefficients specified on the interface of the composite actor.

S_tot = S_DS*DSM + S_GS*GSM + S_Neu*NeuSM + S_IMP*IMPSM + S_HCD*HCDSM

The fine tuning of of sources can be done through editing the XML code parameters of the source combiner actor:

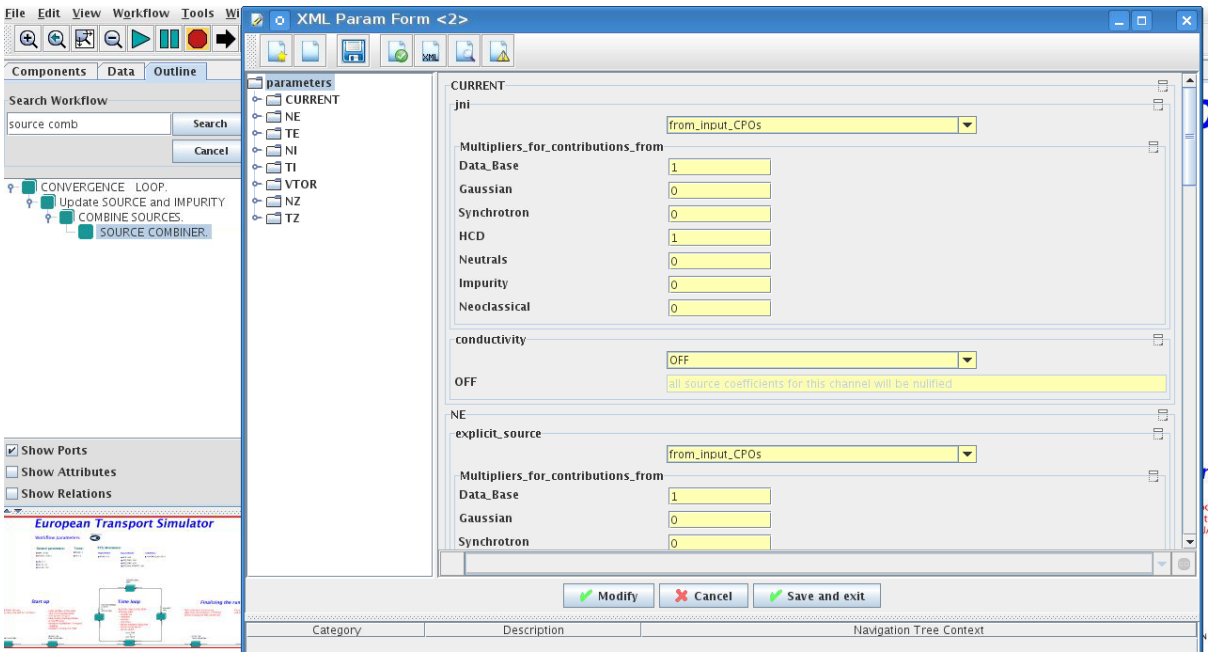
- In the Outline browse for source combiner
- select **Configure actor**
- click **Edit Code Parameters**
- If you like the sources to the particular equation being activated - select **from_input_CPOs**, and then, put the multipliers against each contribution; if you select **OFF** contributions from all sources to this channel will be nullified.
- save and exit
- **Commit**

3.2.1.9 Instantaneous events & Actuators

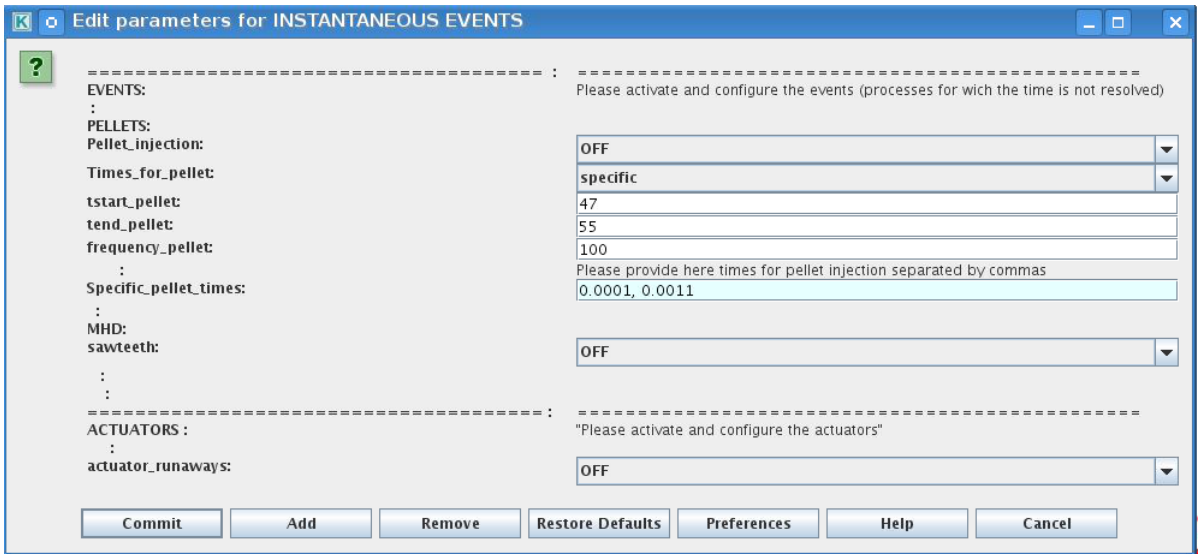
At the moment, user can swith **ON** and **OFF** two types of events: PELLET and SAWTOOTH

3.2.1.9.1 Pellet

At the top level of the workflow you can configure times for pellet injection

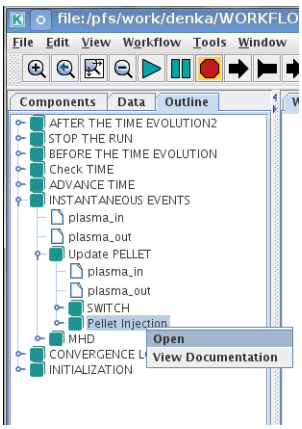


- right click on the box INSTANTANEOUS EVENTS & ACTUATORS
- select **Configure actor** to edit settings
- Select Pellet_injection equal **ON** if you like to use pellet in your simulation
- Select mode of operation:
  - Times_for_pellets equals **specific** – pellets will be shut at exact times specified in array times_pellet
  - Times_for_pellets equals **frequency** – pellets will be shut from tstart_pellet until tend_pellet with a frequency_pellet
- **Commit**

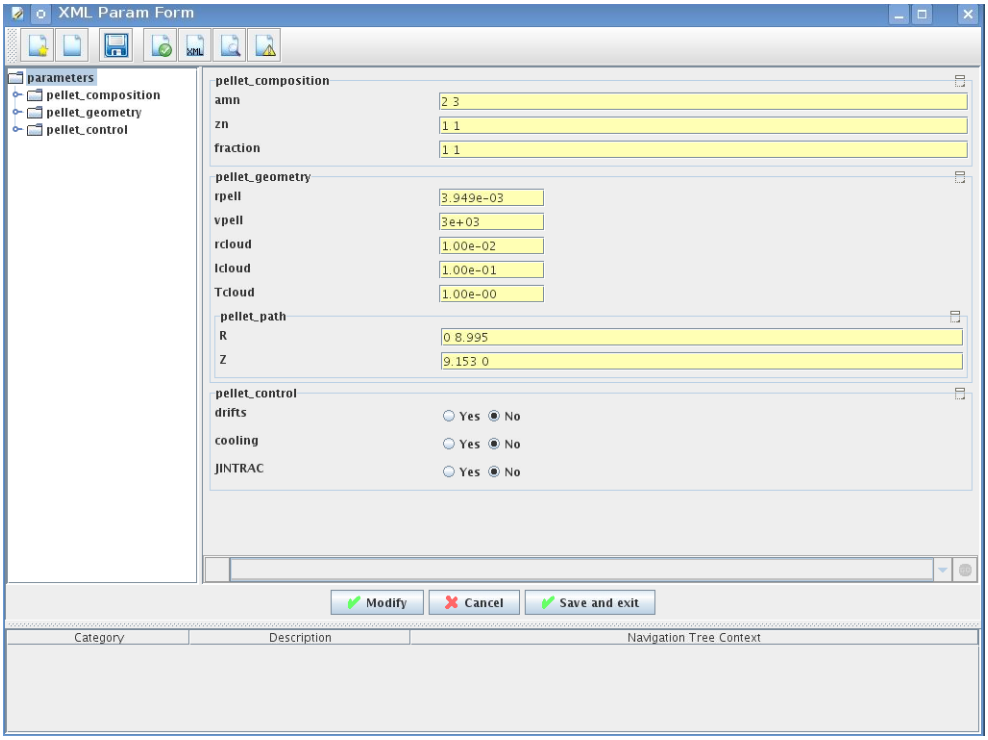


Parameters of individual pellet need to be configured through the code_parameters of the PELLET actor. To access it go to **Outline** on the right upper corner and open the following:

- right click on the actor PELLET



- select **Configure actor**
- click **Edit Code Parameters**
- edit parameters and click **save and exit**
- **Commit**



amn – atomic mass number: array of elements separated by space (1:nelements) [-]

zn – nuclear charge: array of elements separated by space (1:nelements) [-]

fraction – fraction of each element in the pellet, based on the number of atoms: array of elements separated by space (1:nelements) [-]

rpell – radius of the pellet [m]

vpell – velocity of the pellet [m/s]

rcloud – radius of the pellet cloud [m], radial extension of the cloud = 2*rp0

lcloud – length of the pellet cloud along the field line [m]

Tcloud – temperature of the pellet cloud [eV]

Pellet path is specified by two points, for which R and Z coordinated should be specified

R – R coordinates of the pivot and second points of the pellet path, separated by space [m]

Z – Z coordinates of the pivot and second points of the pellet path, separated by space [m]

Control switches allow to activate:

- drifts - YES - will activate radial displacement of deposition profile, same for all path points
- cooling - YES - will activate cooling of the other side of the plasma due to parallel heat transport (essential for large pellets, which might cross the same flux surface twice)
- JINTRAC - YES - will provide temperature reduction consistent with the model used in JETTO

3.2.1.9.2 Sawtooth

At the top level of the workflow you can switch ON/OFF possible MHD events

- right click on the box INSTANTANEOUS EVENTS & ACTUATORS
- select **Configure actor** to edit settings
- Select SAWTOOTH **ON** if you like to use them in your simulation
- **Commit**

3.2.1.9.3 Actuators

At the top level of the workflow you can switch ON/OFF actuator for Runaway Indicator (Runin) - this is **ON** by default. It only gives warning messages, and has no effect on the simulation results.

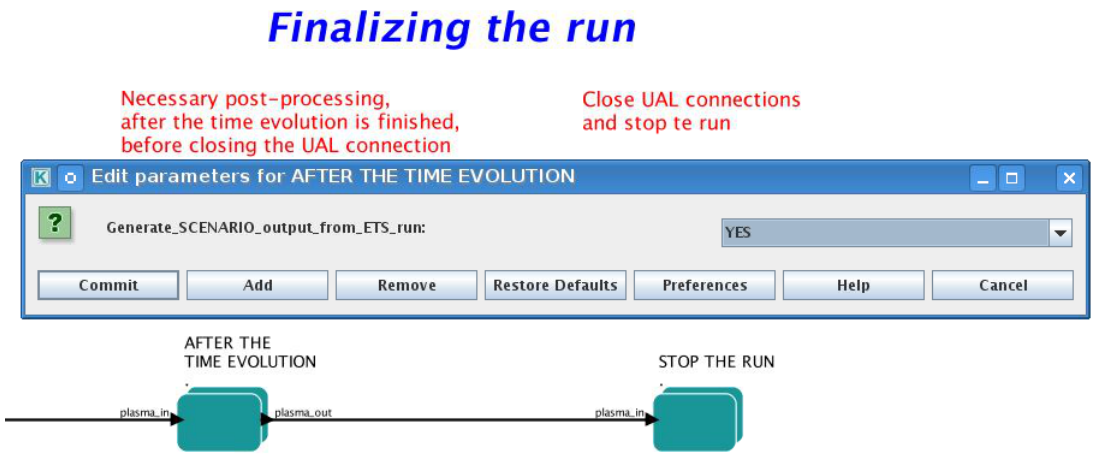
- right click on the box INSTANTANEOUS EVENTS & ACTUATORS
- select **Configure actor** to edit settings
- Select actuator_runaways **OFF** if you'd like **not** to use Runaway Indicator in your simulation
- **Commit**

3.2.1.10 Scenario output

You can summarize the ETS run by activating the output to SCENARIO CPO (as post-processing of the run).

To activate the SCENARIO output:

- right click on the box AFTER THE TIME EVOLUTION
- select **Configure actor**
- select Generate_SCENARIO_output_from_ETS_run equal **YES**
- **Commit**



3.2.1.11 Visualization during the run

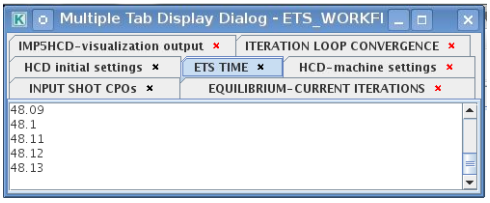
There is a number tools visualizing the ETS run.

3.2.1.11.1 Multiple Tab Display

The display appeares automaticaly when the ETS workflow is launched. It displays diagnostic text messages from the workflow on following topics:

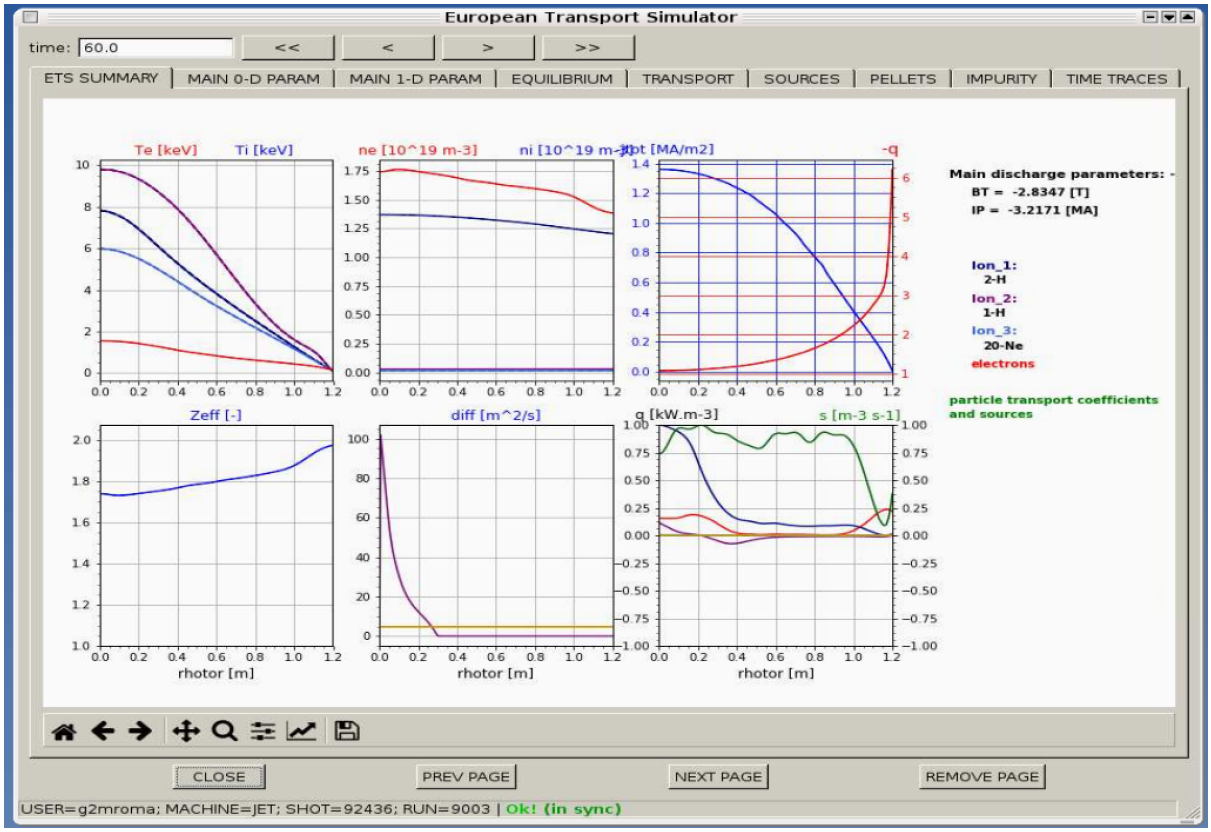
- Input data statement
- Iterations to check the initial convergence between EQUILIBRIUM and CURRENT
- Time evolution
- Convergence of iteratinos within the time step
- HCD settings
- Power used by HCD actors durung the run

Also the error messages from execution of the workflow will be displayed here.



3.2.1.11.2 ETSviz

ETSviz is a python visualization tool with a graphical interphase that shows during the run the calculated kinetic profiles evolution, particle energy sources and sinks, equilibrium evolution and other useful infoirmnation. ETSviz appears automatically during the ETS run. If you would like to launch ETSviz you can find the script in $KEPLER/kplots


European Transport Simulator
time: 60.0
<<
<
>
>>
ETS SUMMARY
MAIN 0-D PARAM
MAIN 1-D PARAM
EQUILIBRIUM
TRANSPORT
SOURCES
PELLETS
IMPURITY
TIME TRACES
Te [keV]
Ti [keV]
ne [10^19 m-3]
ni [10^19 m-3]
Jtot [MA/m2]
-q
Zeff [-]
diff [m^2/s]
q [kW.m-3]
s [m-3 s-1]
rhotor [m]
Main discharge parameters: -
BT = -2.8347 [T]
IP = -3.2171 [MA]
Ion_1:
2-H
Ion_2:
1-H
Ion_3:
20-Ne
electrons
particle transport coefficients and sources
CLOSE
PREV PAGE
NEXT PAGE
REMOVE PAGE
USER=g2mroma; MACHINE=JET; SHOT=92436; RUN=9003 | Ok! (in sync)

3.2.2 List of Actors

3.2.2.1 Equilibrium actors

Code name	Code Category	Contact persons	Short description
chease	Grad-Shafranov solver	Olivier Sauter	Chease is a fixed boundary Grad-Shafranov solver based on cubic hermitian finite elements see H. Lütjens, A. Bondeson, O. Sauter, Computer Physics Communications 97 (1996) 219-260
emeq	G-S solver	Rui Coelho	fix-b equilibrium
spider	G-S solver	5. Fable	ASTRA fix-B equilibrium
helena	G-S solver	7. Huijsmans	fix-B equilibrium
spider_imp12	G-S dolver	18. Coelho	ASTRA fix-b equilibrium

3.2.2.2 Core transport actors

Code name	Code Category	Contact persons	Short description
BohmGB	Bohm/gyro-Bohm transport coefficients	1. Taroni	Analytical model
CDBM	CDBM transport coefficients	13. Honda	Analytical model
Weiland	Transport coefficient from fluid turbulence	Pär Strand	Fluid model
GLF23	Transport coefficient from drift wave turbulence	7. Stabler (GA)	Gyrokinetic model
RITM	Transport coefficient from drift wave turbulence	Pär Strand	Gyrokinetic model
MMM	Transport coefficient from drift wave turbulence	PPPL	Gyrokinetic model
EDWM/EDWMZ	Transport coefficient from drift wave turbulence	Pär Strand	multi-ion model
nclass	Neoclassical transport coefficients	Pär Strand	Neoclassical model
neos	Neoclassical	Olivier Sauter	Neoclassical model

3.2.2.3 Heating and current drive actors

Code name	Code Category	Contact persons	Short description
gray	EC/waves	Lorenzo Figini	GRAY is a quasi-optical ray-tracing code for electron cyclotron heating & current drive calculations in tokamaks. Code-parameter documentation can be found
travis	EC/waves	Nikolai Marushchenko and Lorenzo Figini	Travis is a ray-tracing code for electron cyclotron heating & current drive calculations in tokamaks.
Torray-FOM	EC/waves	Egbert Westerhof	Torray-FOM is a ray-tracing code for electron cyclotron heating & current drive calculations in tokamaks.
bbnbi	NBI/source	Seppo Sipila	Calculate the deposition rates of neutrals beam particles, i.e. the input source for Fokker-Planck solvers (not the heating and current drive). Note that the number of markers generated by BBNBI is described by the kepler variable number_nbi_markers_in.
nemo	NBI/source	Mireille Schneider	Calculate the deposition rates of neutrals

3.2.2.4 Events actors

Code name	Code Category	Contact persons	Short description
pelletactor	pellet	Denis Kalupin	
pellettrigger	pellet	Denis Kalupin	
sawcrash_slice	sawteeth	Olivier Sauter	
sawcrit	sawteeth	Olivier Sauter	
runaway_indicator	runaway	Roland Lohneroch Gergo Pokol	Indicating the presence of runaway electrons: 1) Indicate, whether electric field is below the critical level, thus runaway generation is impossible. 2) Indicate, whether runaway electron growth rate exceeds a preset limit. This calculation takes only the Dreicer runaway generation method in account and assumes a velocity distribution close to Maxwellian, therefore this result should be considered with caution. The growth rate limit can be set via an input of the actor. Limit value is set to \(10^{12} \) particle per second by default. (This growth rate generates a runaway current of approximately 1kA considering a 10 seconds long discharge.)

3.2.2.5 Non-physics actors

The ETS uses the following list of non-physics actors: addECant, addICant, backgroundtransport, calculateRHO, changeocc, changepsi, changeradii, checkconvergence, controlAMIX, coredelta2coreprof, correctcurrent, deltacombiner, emptydistribution, emptydistsource, emptywaves, eqinput, etsstart, fillcoreimpur, fillcoreneutrals, fillcoreprof, fillcoresource, fillcoretransp, fillequilibrium, fillneoclassic, filltoroidfield, gausiansources, geomfromcpo, hcd2coresource, ignoredelta, ignoreimpurity, ignoreneoclassic, ignoreneutrals, ignorepellet, ignoresources, ignoretransport, IMP4dv, IMP4imp, importimptransport, itmimpurity, itmneutrals, merger4distribution, merger4distsource, merger4waves, nbifiller, neoclassic2coresource, neoclassic2coretransp, parabolicprof, plasmacomposition, PowerFromArray, PowerModulation, profilesdatabase, readjustprof, sawupdate_slice, scaleprof, sourcecombiner, sourcedatabase, transportcombiner, transportdatabase, wallFiller and waves2sources.

3.3 Turbulent Flux Quantities in Transport Models

3.3.1 Overview

In conventional transport modelling, all quantities appearing in the equations are 1-D, in some radial coordinate (poloidal flux, normalised radius, etc). In general any monotonic radial coordinate is acceptable. In the TF-EU-IM, the toroidal flux radius is standard. All we need from the radial coordinate is the transformation to get to V, the volume enclosed by the flux surface, which is fundamental to the governing equations, which are conservation laws.

What we have to do is to take a measured result, which is a time-averaged fluctuation-based transport flux and turn it into 1-D quantities suitable to modelling. This is done using the flux surface average, explained in conventions. The transport equations themselves constitute a mean field approximation to the 3-D conservation laws. For the fundamentals encountered in transport modelling see R Hazeltine and J Meiss, Plasma Confinement (Addison-Wesley, 1992) chapter 8. For the special properties of transport driven by small-scale pressure driven ExB microturbulence see B Scott, "The character of transport caused by ExB drift turbulence," Phys Plasmas 10 (2003) 963-976.

For ambipolarity we follow the rules for dynamical alignment, which follows the physics of how electron fluctuations determine the ExB velocity fluctuations, which then advect all species. Magnetic flutter nonlinearities act independently of this, but in our modelling they are used solely for heat fluxes since the averaged particle transport due to magnetic flutter and the current cancels, leaving the parallel ion velocity which we neglect for this purpose. The reference for dynamical alignment is B Scott, "Dynamical alignment in three species tokamak edge turbulence," Phys Plasmas 12 (2005) 082305.

Note: there are now auxiliary actors provided for this purpose: IMP4DV, which does the D/V conversion and enforces ambipolarity assuming absence of impurities, and IMP4imp, which subsequently enforces ambipolarity for the set of main ion and impurity species. The IMP4DV actor should be invoked directly after the transport model actor in the workflow chain, if the model produces only fluxes or if the coefficients have to be modified with the flux given. Ambipolarity is done using IMP4imp if the coreimpurity CPO is used in the workflow.

3.3.2 Particle Flux as an Example

The mean field equation governing particle balance is the transport equation for electrons,

$$\frac{\partial}{\partial t}\langle n\rangle + \langle \vec{\nabla}\cdot \tilde{n}\tilde{\vec{v}}_E\rangle = S$$

in which the tilde symbol over the n and v denotes fluctuating quantities and we neglect all transport processes except ExB eddy diffusion. The ExB velocity is given by

$$\vec{v}_E = \frac{c}{B^2}\vec{B} \times \vec{\nabla}\phi$$

where ϕ is the electrostatic potential.

The angle brackets denote the flux surface average, and we will use the property that the flux surface average of a divergence of a vector is the volume derivative of the flux surface average of a contravariant volume component of the vector, in this case

$$\langle \vec{\nabla} \cdot \vec{\Gamma} \rangle = \frac{\partial}{\partial V} \langle \Gamma^V \rangle$$

where Γ is the particle flux whose flux-surface averaged volume component is

$$\langle \Gamma^V \rangle = \langle \widetilde{n}\widetilde{v}_E^V \rangle$$

This is converted to expression in terms of the radial coordinate (rho‘ using the fact that both V and ρ are flux quantities whose gradients are parallel to each other. We have

$$\frac{\partial}{\partial V} = \frac{1}{V'_\rho}\frac{\partial}{\partial \rho} \qquad \Gamma^\rho = \frac{1}{V'_\rho}\Gamma^V \qquad V'_\rho = \frac{\partial V}{\partial \rho} \qquad g^{VV} = (V'_\rho)^2 g^{\rho\rho}$$

so we can write the transport equation as

$$\frac{\partial n}{\partial t} + \frac{1}{V'_\rho}\frac{\partial}{\partial \rho} V'_\rho \langle \Gamma^\rho \rangle = S,$$

where we have replaced $\langle n \rangle$ with n following the assumptions of the 1-D version of mean field transport theory.

With all quantities now expressed in terms of flux quantities, we are free to characterise the transport flux $\langle \Gamma^\rho \rangle$ in an arbitrary way, so long as only flux quantities appear. The flux expansion within the flux surface as well as expansion or contraction of surfaces of constant ρ is treated using the metric coefficient $g^{\rho\rho}$ which is dimensionless. This way we can characterise transport in terms of an effective diffusivity and an effective frictional slip velocity which are given in SI units. By convention both of these are done solely via $g^{\rho\rho}$ for convenience, also reflecting that the effective velocity is actually marking off-diagonal diffusive elements. Our convention for this follows the ETS code and is given by

$$\langle \Gamma^\rho \rangle = \langle g^{\rho\rho} \rangle \left(n V_{\text{eff}} - D_{\text{eff}} \frac{\partial n}{\partial \rho} \right)$$

So despite the special spatial distribution of any particular transport process (ie, the underlying instability or nonlinear free energy access), the flux-surface averaged flux itself and its expression in terms of diffusion and frictional slip are identical characterisations.

3.3.3 Metric Coefficients

Transport modellers want the Ds and Vs as physical quantities in SI units. In general the fluxes are (magnetic) flux surface averaged quantities, which implies the existence of metric elements in the conversion. In our case we need $\langle g^{\rho\rho} \rangle$ where ρ is the toroidal flux radius in meters, so the metric elements are dimensionless. In the equilibrium CPO, this is gm3 under equilibrium%profiles_1d in the structure.

Note this is different from the ASTRA code which casts the Vs as proper velocities, i.e., with one factor of grad-rho given by $\langle \sqrt{g^{\rho\rho}} \rangle$ which is gm7 under equilibrium%profiles_1d in the structure. The units are the same and the informational content is the same, but this difference has to be taken into account in any transport modelling and benchmarking.

3.3.4 Heat Fluxes

The heat flux is treated in a similar way, with transport equation

$$\frac{3}{2}\frac{\partial p_e}{\partial t} + \frac{1}{V'_\rho}\frac{\partial}{\partial \rho}V'_\rho\langle q_e^\rho\rangle = Q_e + \sum_{\text{ions}} T_{ei},$$

for electrons, with T_{ei} giving the species transfer and Q_e the source. For ExB transport the heat flux has a advective (also called convective) and a conductive piece given by

$$q_E = q_{E\text{cond}} + (3/2)T\Gamma_E$$

which appears with a 3/2 due to the Poynting cancellation. For magnetic flutter transport the advective piece appears with the usual factor,

$$q_m = q_{m\text{cond}} + (5/2)T\Gamma_m$$

Here the forms are given for each species and E and m refer to the ExB eddy and magnetic flutter channels, respectively. For reasons given below we are neglecting the magnetic flutter piece Γ_m for the time being, and then the flutter piece merely adds to the heat diffusivity.

The forms of these due to the fluctuations are then

$$\langle q^\rho\rangle = (3/2)\langle\widetilde{p}\widetilde{v}_E^\rho\rangle + \langle\widetilde{q}_\parallel\widetilde{b}^\rho\rangle$$

which breaks into advective and conductive pieces according to linearisation of the pressure fluctuations

$$\langle q^\rho_{\text{cond}}\rangle = (3/2)n\langle\widetilde{T}\widetilde{v}_E^\rho\rangle + \langle\widetilde{q}_\parallel\widetilde{b}^\rho\rangle \qquad \langle q^\rho_{\text{adv}}\rangle = (3/2)T\Gamma = (3/2)T\langle\widetilde{n}\widetilde{v}_E^\rho\rangle$$

hence the density fluctuation piece is accounted for by the particle flux. Neglect of the magnetic flutter advective piece (and particle flux) is the same as neglect of the $\widetilde{u}_\parallel\widetilde{b}^\rho$ nonlinearity (in the delivery of the results, not in the turbulence computations themselves).

The total conductive flux is then represented by

$$\langle q^\rho_{\text{cond}}\rangle = \langle g^{\rho\rho}\rangle\left(nTY_{\text{eff}} - n\chi_{\text{eff}}\frac{\partial T}{\partial\rho}\right)$$

with χ and Y giving the heat diffusion and frictional slip pieces for each species, respectively (these are in diff_eff and vconv_eff in the CPO for each quantity).

Operationally, the turbulence module communicates the diff_eff and vconv_eff due to each transport channel for each species to the transport solver, and the metric coefficients are used by both modules. The two modules can be on arbitrarily different grids, which communicate through standard interpolation. This despite the fact that transport at the micro-level is angle dependent (in general, it can be 3-D in the time average if the sources are 3-D). The effective transport is 1-D so long as parallel sound transit within the flux surface remains fast compared to the local transport time. This breaks down anyway in the edge, so the fact that the volume is a problematic coordinate and the flux surface average is a problematic operation on open field lines doesn't enter.

3.3.5 Ds and Vs from Turbulence Codes to Transport Solvers

To serve the results from turbulence codes to transport solvers, we have to turn the fluxes (results) into diffusivities and effective velocities (coefficients, Ds and Vs for short), which represent more information than is at hand. Transport solvers must work with Ds and Vs because they use implicit schemes.

The matrix must be diagonally dominant; hence one cannot simply use the Vs. Fluxes which are zero and/or negative should be given with positive diffusivities for the solvers to work. We need a set of rules to provide this.

Considering the particle and heat transport fluxes for a given species, we convert the gradient in to a logarithmic derivative and express the flux in terms of a specific flux, which has units of velocity,

$$F = \frac{1}{n}\langle g^{\rho\rho}\rangle^{-1}\langle\Gamma^{\rho}\rangle = V_{\text{eff}} - D_{\text{eff}}\frac{\partial \log n}{\partial \rho}$$
$$G = \frac{1}{nT}\langle g^{\rho\rho}\rangle^{-1}\langle q^{\rho}_{\text{cond}}\rangle = Y_{\text{eff}} - \chi_{\text{eff}}\frac{\partial \log T}{\partial \rho}$$

wherein the conductive part of the heat flux (without the $3\Gamma/2$ enters.

The choice of what to do with the Ds and Vs is somewhat arbitrary. The needs of implicit transport solvers is for a positive D regardless of the value or sign of either flux. We decide this by putting a limit on the effective Prandtl number or its inverse: the larger specific flux is taken to be entirely diffusive, with the effective velocity set to zero. Furthermore, to address cases with very small or negative gradients, we use proxy variables for the scale lengths to calculate the provisional diffusivities before using the Prandtl number limitation to turn these into actual diffusivities. Finally, the rest of the flux is asigned to the effective velocity, so that the D and V formula reflects the actual specific flux.

The Prandtl number limitation is expressed as follows. If the smaller specific flux is within a factor of 5 of the larger, then both are purely diffusive and the effective velocities are both zero. If not, then the D ratio is set to 5, with the result that the smaller D, having been corrected, is accompanied by the corresponding V, which is now nonzero. The specific flux with the larger D will be returned with a V which is zero.

The rationale is that the turbulent mixing by the ExB velocity affects all processes, but that linear forcing can shift the average phase shift of the fluctuations such that the effective flux can be small or negative. The simplest example is adiabatic electrons, for which the ion heat flux is robust but the particle flux is zero. In most situations the specific heat flux will be the larger, and hence the familiar situation is that of a D and V for the particle flux but a D (the chi) only for the conductive heat flux.

The full algorithm starting with the specific fluxes appears as

$$L_n^{-1} = \max\left(\frac{1}{R}, \left|\frac{\partial \log n}{\partial \rho}\right|\right) \quad L_T^{-1} = \max\left(\frac{1}{R}, \left|\frac{\partial \log T}{\partial \rho}\right|\right)$$
$$D' = |F|\, L_n \quad \chi' = |G|\, L_T$$
$$D = \max\left(D', \frac{1}{5}\chi'\right) \quad \chi = \max\left(\chi', \frac{1}{5}D'\right)$$
$$V = \left(F + D\frac{\partial \log n}{\partial \rho}\right) \quad Y = \left(G + \chi\frac{\partial \log T}{\partial \rho}\right)$$

and all four elements are set. Note that the channels are done in parallel except for the Prandtl correction, in which the Max's are taken sequentially. For the provisional diffusivities, absolute values are used to ensure positive values which are needed by transport solvers.

Note how in the end the actual gradients are used. If the gradients are moderate then their actual values are used, and if the Prandtl correction is not invoked, then both channels are diagonal. In any case the full relation is used to get the effective velocities (V and Y) so having set the rules to handle the arbitrariness of the diffusivities (D and chi) to guarantee reasonable diagonal dominance in a transport solver, the D's and V's agree with the fluxes themselves.

If there are more than two specific fluxes per species to consider, then we treat each scale length separately as above and use N-way maxima in the Prandtl correction for the N channels.

3.3.6 Ambipolarity

There remains the issue of ambipolarity of the D and V for particle flux. For a pure singly charged plasma the ion and electron Ds and Vs should be equal. Even if the turbulence model is gyrokinetic or gyrofluid, in which case the gyrocenter charge density is not zero but is equal to the generalised vorticity (polarisation), the quantities given to a transport solver should follow the rules for a fluid representation. However, transport modelling usually applies ambipolarity rules to the electrons after computing the ions, while the action of turbulence is actually the other way around: Dynamical alignment refers to the process by which (1) electron parallel dynamics controls the electrostatic fluctuations, then (2) the resulting ExB velocity advects all species equally. So we correct the particle fluxes by assuming the electrons determine the D according to the above procedure and then (1) the fluctuations in the flux-inducing part of the spectrum for the logarithmic densities are the same, and (2) the D's are the same. Then the V's are solved for again, by taking

$$D_z = D_e = D \qquad V_z = V_e + D\frac{\partial \log b_z}{\partial \rho} \qquad b_z = n_z/n_e$$

This is better than the transport modelling convention but will give them the same information in a different way, and they will compute ambipolar particle fluxes (radial transport of charge is zero).

3.3.7 Statistical Character

Turbulence has a statistical character, so convergence to a mean is not monotonic and when within one std dev of the mean there is no further convergence. The diffusivity for ExB turbulence is comparable to

$$D_E = \langle(\widetilde{v}_E)^2\rangle \big/ \langle(\varpi)^2\rangle^{1/2} \qquad \varpi_E = \frac{c}{B}\nabla_\perp^2\widetilde{\phi}$$

where ϖ_E is the ExB vorticity fluctuation, and these angle brackets denote the ensemble average. To get an ensemble average over a statistical quantity in practice, one must do some sort of finite-time running averaging.

For transport modelling, the transport coefficients derived from a turbulence code should always be given in terms of *running exponential averages.*

3.4 Running Exponential Average

3.4.1 Overview

In conventional transport modelling, turbulent fluxes are modelled in terms of processes which are diffusive in the local relaxation sense, with the average flux given by a diffusion coefficient and an effective pinch velocity. The equations are of dominantly parabolic character, which means in practice that an iterate will move monotonically towards the solution in parameter space.

This is not the case for turbulence. Convergence is statistical, which is something different than a diffusive relaxation. If turbulence is stationary, it is meant only that the mean of a distribution of iterates is stationary, not the iterates themselves. The standard deviation can be significant, of order unity compared to the mean, of any distribution of iterates.

This makes for a noisy signal if the output of a turbulence code is used for transport coefficients in a workflow. A sound way to overcome the attendant problems is to use a moving average. Even an average over a moving window can be as noisy as the original signal, however. What works better is a weighted average over recent past values. A method to get this is called a running exponential average, which is

essentially the same thing as a convolution integral over an exponential memory decay times the past signal. It turns out to be very easy to obtain this without saving past values.

The original reference for the following is S W Roberts, "Control Chart Tests Based on Geometric Moving Averages," Technometrics 1 (1959) 239-250, cited by all the good WWW resources, including the Wikipedia page on Moving Averages and the NIST Statistical Handbook online.

3.4.2 Definition

Consider a process $p(\vec{u})$ which is a functional of dependent variables $\vec{u}$. Measure p at discrete time intervals t_n, with values $p_n = p(t_n)$ and interval length $\tau = t_n - t_{n-1}$. The moving exponential average $A_n = A(p_n)$ on the n-th interval is defined as

$$A_n = \epsilon p_n + (1-\epsilon)A_{n-1} \qquad \text{with} \qquad \epsilon = \alpha\tau$$

in which the small parameter ϵ is given in terms of the interval τ and an inverse time constant α.

In the first instance p is measured there is no A so the first value of A is simply set to p since it can be assumed that the initial state for p has persisted for infinite previous time up to the initial time point.

3.4.3 Differential Equation

The equivalent differential equation is found by forming the relevant finite difference,

$$A_n - A_{n-1} = \epsilon(p_n - A_{n-1})$$

which we can also cast as

$$(1-\epsilon)(A_n - A_{n-1}) = \epsilon(p_n - A_n)$$

Taking the limit $\tau \to 0$ is the same as taking $\epsilon \to 0$ so both of these expressions become equivalent to

$$\frac{\partial A}{\partial t} = \alpha(p - A)$$

whose solution is given below.

3.4.4 Equivalence to Past-Time Convolution Integral

The solution of the above differential equation is given by the method of undetermined coefficients,

$$\frac{\partial A}{\partial t} + \alpha A = \alpha p \;\; e^{-\alpha t}\frac{\partial}{\partial t}\left(e^{\alpha t}A\right) = \alpha p \;\; \frac{\partial}{\partial t}\left(e^{\alpha t}A\right) = \alpha p e^{\alpha t}$$

We may integrate this over all past time, to find

$$A(t) = \int_{-\infty}^{t} \alpha dt' p(t') e^{-\alpha(t-t')}$$

This is a convolution integral over the kernel $e^{-\alpha(t-t')}$ and the signal $p(t')$. The time constant α^{-1} is just the memory decay time, while if p is constant then the integral yields unity times p. This is the same as the normalisation with the $(1-\epsilon)$ factor in the average formula above, which is needed since the interval is of finite size.

Hence the running exponential average is operationally the same as a memory decay integral over past time. The elegant feature is the need to keep only the current value of A, as it already contains all that is needed of the past time evolution of p.

3.4.5 notes

Some properties of the running exponential average and how to choose its main time-memory parameter:

- The $(1-\epsilon)$ factor is needed for normalisation
- if $p =$ constant then $A = p$ for all t
- the integral with $\alpha dt'$ yields unity
- the ϵ and $(1-\epsilon)$ factors add to unity
- therefore set the first value of A to the first value of p
- in choosing the memory decay time $\alpha^{-1} \ldots$
- one should have $\alpha\tau_{cor} \ll 1$
- best results are for $\alpha\tau_{sat} \sim 1$
- some trial/error required; edge turbulence likes $\alpha^{-1} = 200L_{\|}/c_s$

In these expressions τ_{cor} and τ_{sat} are the correlation and saturation times of the turbulence, respectively.

CHAPTER

FOUR

EQUILIBRIUM AND MHD STABILITY WORKFLOW (EQSTABIL)

4.1 Workflow rationale

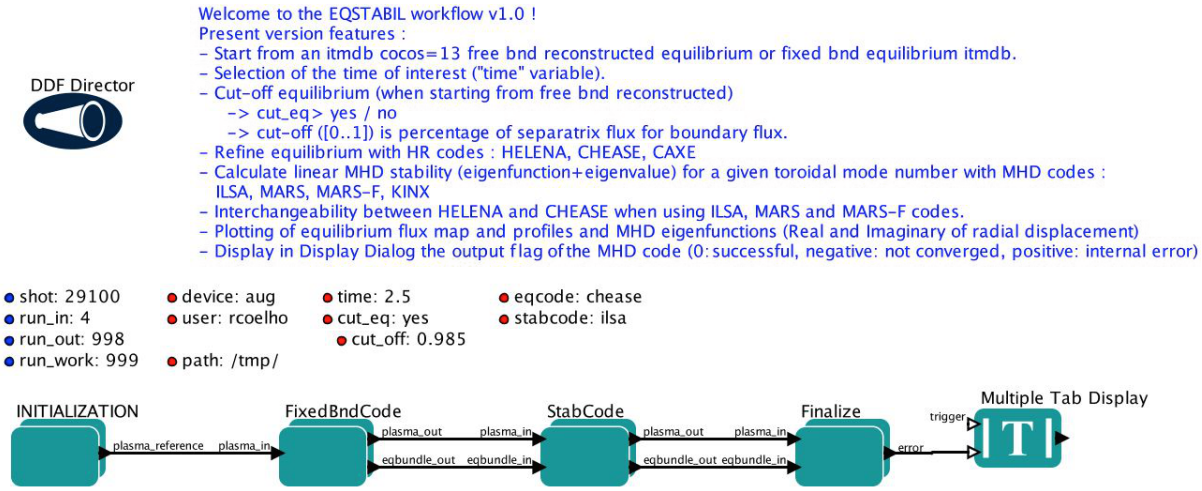
The EQSTABIL workflow is a Kepler workflow aimed at performing linear MHD stability analysis of tokamak plasma equilibria for a single or multiple toroidal mode numbers when executed. The high resolution equilibrium actors consider axisymmetric toroidal static plasmas with isotropic pressure and the linear MHD stability models stem from single fluid ideal/resistive MHD with compressibility.

The workflow is meant for straightforward stability calculations of any plasma scenario, reading from a pre-existent WPCD database shot/run/time entry. Therefore,

- Equilibrium data need to be read from experimental databases and stored locally on the platform where EQSTABIL is run. Alternatively the equilibrium IDS could be the output of another workflow (e.g. EQRECONSTRUCT or ETS)
- It is not meant for parametric studies in a single workflow execution e.g. process several time slices or scan over resistive wall position or number of poloidal harmonics. Dedicated runs for such cases are necessary, storing each run on a dedicated output shot/run_out database entry. The workflow may be subject to upgrades/revisions to accomodate new features that facilitate/enhance user experience so stay tuned for News and Recent activity.

4.2 Workflow organization & design

The top level layout of the workflow is shown below.



The workflow is organised in four sequential steps :

4.2.1 Initialization

Composite actor used to initialize the workflow. It reads from the IMAS database that is specified by local variables (user, device, shot, run_in) and for the closest time sample to local variable *time* . If the user reads the input data from some other user database, the output data will however be written on his/her own database with shot/run_out id.

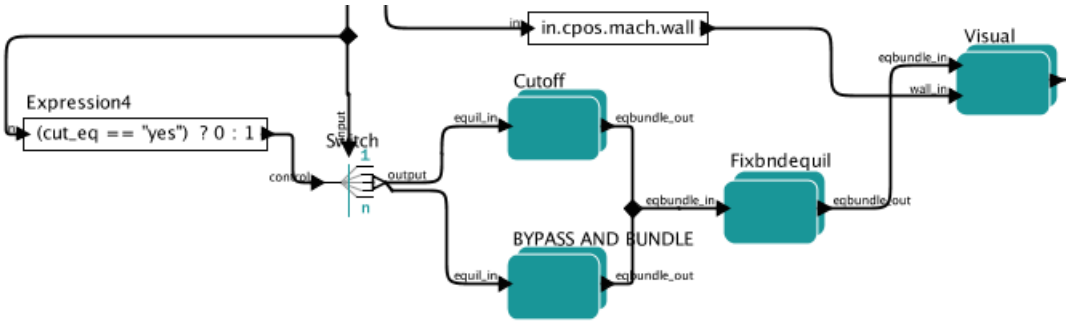
-> The workflow local variable *device* **must** be the same as the environment variable TOKAMAK-NAME. In case the two do not match the workflow stops execution. The user must close the workflow, set imasdb with the correct device name and run the workflow.

-> Validity checks (void/not void) are made on the input equilibrium and core_profile IDSs (MARSGW actor can use core_profile for density profile). If the equilibrium IDS is not considered valid the workflow stops. If the core_profile IDS is not considered valid, the workflow continues to run but the user can still have the option to stop it before executing the chosen MHD code.

At the exit of the Composite actor, a Plasma_reference bundle (list of Kepler variables, mimicking the ETS bundle is returned. This facilitates the future coupling of the workflow to the ETS.

4.2.2 FixedBndCode

Composite actor that prepares/calculates the equilibrium to be passed later to the MHD stability codes. This composite actor is composed of 3 main steps:



4.2.2.1 Redefining the plasma boundary (Cutoff)

This is deemed necessary when the input equilibrium (reconstructed/predictive equilibrium) as a separatrix as plasma boundary since at this moment none of the flux coordinates based equilibrium codes handles/returns plasmas with a separatrix.

If the input equilibrium does not contain a Psi(R,Z) equilibrium mapping the cut-off is not possible and thus the workflow execution will be stopped.

Redefining the plasma boundary is done by setting cut_eq: yes and places the new plasma boundary at a flux surface corresponding to cut_off (in percentage) of the input boundary flux.

The plasma profiles are also cut-off accordingly. An equilibrium bundle exits the actor containing occurrence=1 for the cut-off input equilibrium and occurrence=2 for the original equilibrium. If cut_off:no then both occurrences contain the original equilibrium.

A plot of the original + cut_off equilibrium summary is shown. Closing the plot window leads to the second stage.

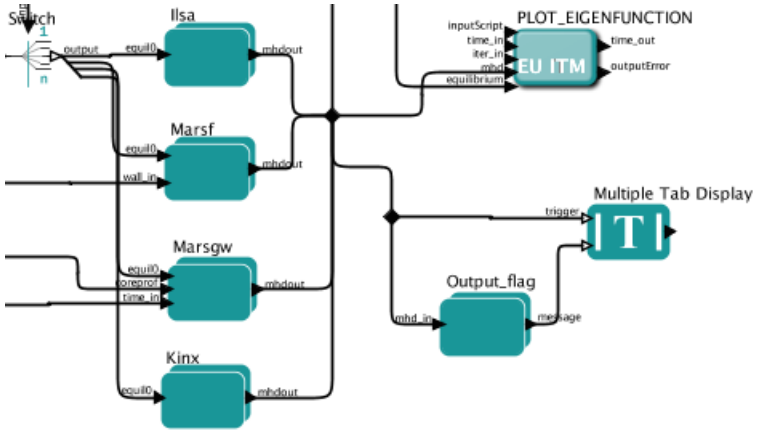
4.2.2.2 Calculation of Equilibrium (Fixbndequil)

Calculation of the high resolution equilibrium with 3 possible codes (CAXE, CHEASE, HELENA). The cut-off equilibrium (or original one) is passed to the equilibrium codes. The output HR equilibrium is added to the equilibrium bundle such that in the end one has occurrence=0 for the HR, occurrence=1 for the cut-off input equilibrium (or the original if not cut_off is requested) and occurrence=2 for the original equilibrium.

4.2.2.3 Visualization (Visual)

Visualization part. This part plots the (R,Z) flux map of the HR equilibrium and the most relevant profiles. The figures are saved automatically on closing the windows at the *path* indicated in the top level accordingly Kepler variable.

4.2.3 StabCode



Composite actor for the MHD stability calculation using 4 possible linear MHD stability codes (ILSA, KINX, MARS, MARS-F). After execution of the stability code is completed, plotting of the radial component of the displacement vector eigenfunction in the plasma domain is shown (real and imaginary parts). In case multiple toroidal mode numbers are set (ILSA or KINX), one plot window per each toroidal eigenmode is returned. A Copy in EPS format of each window is stored on the path defined by Kepler variable *path*

The Multiple Tab display window will also display the output flag of the code execution i.e. if the output is valid and the result can be used or not. The plasma bundle, on exit, is updated with the MHD cpo from the stability code.

4.2.4 Finalize

Composite actor to wrap up the final plasma bundle, with the equilibrium IDS containing 3 occurrences and one occurrence of the MHD IDS.

N.B. Only a single time slice of equilibrium and MHD IDSs is written, the remaining plasma bundle IDSs are written “as is” (whatever time slices).

4.3 Actors involved

Name	Location	Description
Check_Device	INITIALIZATION	Checks if the *device* Kepler variable coincides with the environment variable TOKAMAKNAME. If not the run stops.
SELECT_TIME_CORE/EQ	INITIALIZATION	Selects time slice of IDSs matching/closest to the requested time in *time* Kepler variable
Check Coreprof/Equil Time and Flag	INITIALIZATION	Checks the output_flag of the input IDSs to know if they are valid and prints the actual time stamp retrived from both IDSs (if time = -1 and output_flag is negative then the IDS is not valid). If the equilibrium is considered invalid a message in displayed on the Multi Tab Display window and workflow execution is stopped. If the core_profile is considered invalid a message is displayed on the Multi Tab Display window but the workflow will continue since some of the MHD codes handle plasma density internally as code parameter and their execution is not affected.
Cutoff	FixedBndCode	Performs the cut-off of the input equilibrium if requested and

4.4 Setting up Workflow and Actor parameters

4.4.1 Setting workflow parameters

The workflow has basic settings in order to work.

- **shot** : the shot number on the user database (or from another user) where to read the reference equilibrium from (shot/run_in pair)
- **run_in** : the run number where the reference equilibrium is (shot/run_in pair)
- **run_work** : placeholder run for the temporary Kepler IDSs
- **run_out** : run number where the final results of the run will be stored (user running the workflow/shot/run_out). Since the input equilibrium can be a reconstruction that goes beyond the separatrix, 3 occurrences of the equilibrium are saved (original eq., cut equilibrium inside separatrix and corresponding high resolution equilibrium).
- **user** : username. Reading from someone else database is possible but the run_out will naturally be written to personal database only.
- **device** : device database where the input reference data is. MUST BE the same as env variable TOKAMAKNAME
- **time** : time slice (in equilibrium IDS) to be analysed in case the input shot/run_in contains many time slices.
- **path** : temporary folder where to dump the plots generated. Also used to store output files (used by HELENA/ILSA only)
- **cut_eq** :
  - yes : cut the input equilibrium (necessary if high resolution equilibrium code cannot handle separatrix plasma equilibria)
  - no : input equilibrium is used "as is".
- **cut_off** : float]0,1], specifies the percentage of the separatrix flux that will define the poloidal flux of the new plasma boundary.
- **eqcode** : chease/caxe/helena. The equilibrium code to be used
- **stabcode** : ilsa/kinx/marsgw/marsf. The MHD stability code to be used

The user can always prevent the workflow from proceeding to the calculation of the high resolution equilibrium after the cut-off stage by Pressing the STOP button in Kepler GUI before closing the plot window with the summary of the equilibrium.

4.4.2 Setting actor parameters

Actor parameters are set on the actors themselves (not passed by the workflow). To access the actors codeparam the easiest route is to :

1. Click on "Outline" Tab (below the "Pause" button)
2. Type the name of the actor and press "Search" (or Enter)
3. On the final item in the chain of the actor composite, right click and press "Configure". A pop-up panel appears

4. Click on “Edit Code Parameters” and a new window appears
5. Edit the code parameters and Press “Save & Exit”
6. Press “Commit” and setting is completed

4.5 EQSTABIL Tutorial

Tutorial on using EQSTABIL workflow is available in `PDF`.

CHAPTER

FIVE

THE EQRECONSTRUCT WORKFLOW

5.1 1. Workflow rationale

The EQRECONSTRUCT workflow is a Kepler workflow aimed at performing the reconstruction of the plasma equilibrium from diagnostic data. The workflow can perform both a single time reconstruction and over a defined time range with a user defined sampling rate. The workflow design facilitates the integration of a variety of plasma reconstruction equilibrium codes, all using the same input data from a user defined IMAS database. In addition, during all workflow stages (including initialization and finalization), the experimental and modeling data are cast under the same conceptual data bundling as used by the ETS, HCD and EQSTABIL workflow. This deliberate choice greatly facilitates the interfacing to any of such workflows. In fact, elements of the EQSTABIL workflow can and were seamlessly integrated in the workflow namely the stage for high resolution equilibrium actors for axisymmetric toroidal static plasmas with isotropic pressure.

The workflow includes built-in visualization plugin options to visualize the equilibrium reconstruction (and high resolution calculation) during the run execution. This allow for an immediate inspection of the results. Some fundamental data verification is performed on the input and processed data to ensure a "safe landing" of the workflow in case any problems are identified.

The workflow is presently targeting primarily straightforward magnetics only plasma reconstruction calculations in any plasma scenario. Interferometry, polarimetry and Motional Stark Effect assisted reconstructions are also possible since no workflow changes are necessary. Future versions of the workflow will incorporate kinetic data (thermal/fast pressure).

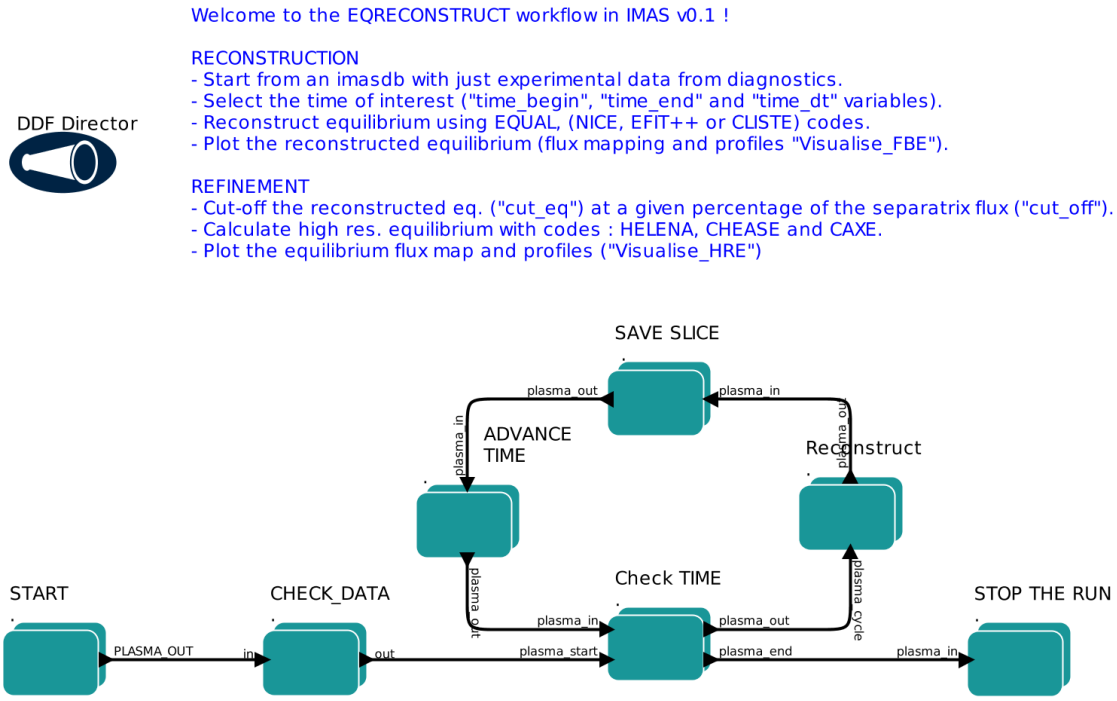
The workflow is not meant for running use cases that require strong user intervention during the run execution e.g. setting different code parameters at different time steps (though possible by pausing the workflow, changing the code parameters and resuming) for deselecting given diagnostic channels or changing regularization coefficients.

5.2 2. Workflow organization & design

The top level layout of the workflow is shown below.

As shown in the workflow layout, the workflow execution typically follows the following steps (further detailed below):

- START (set up input imasdb database and simulation time range)
- CHECK_DATA (verify data consistency)
- Check TIME (continue simulation if time < time_end)



- Reconstruct (calculate reconstructued equilibrium and high resolution equilibrium)
- SAVE SLICE (save time slice on database)
- ADVANCE TIME (advance time to next time step)
- STOP THE RUN (end the simulation and stop)

5.2.1 I - START

Composite actor used to initialize the workflow. It reads experimental data from an ITM database and assembles the plasma bundle. The database details e.g. _user, device, shot, run*in* are configurable in the actor when double clicking on the actor (see Figure below).

5.2.2 II - CHECK_DATA

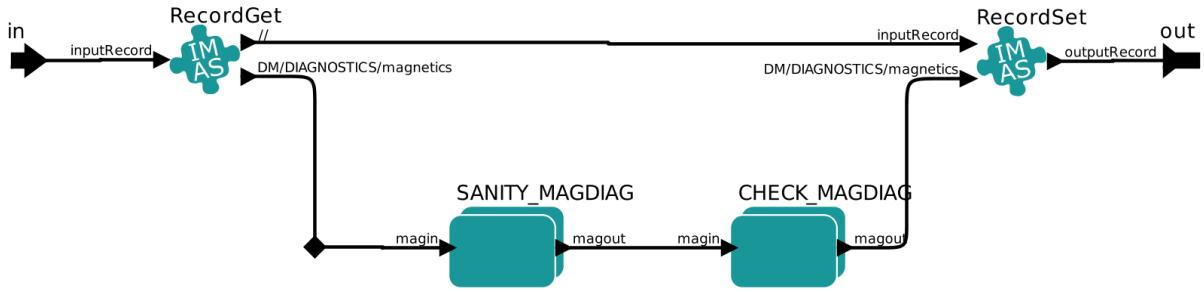
In this composite actor a basic sanity check is performed on the input data and appropriate action is taken e.g. if there is no magnetics sensor data it is pointless for the workflow to proceed and execution is immediately stopped. Additional checking and action includes for instance flagging as invalid any sensor data with flatline signal (see Figure below for the sequence of 2 steps)

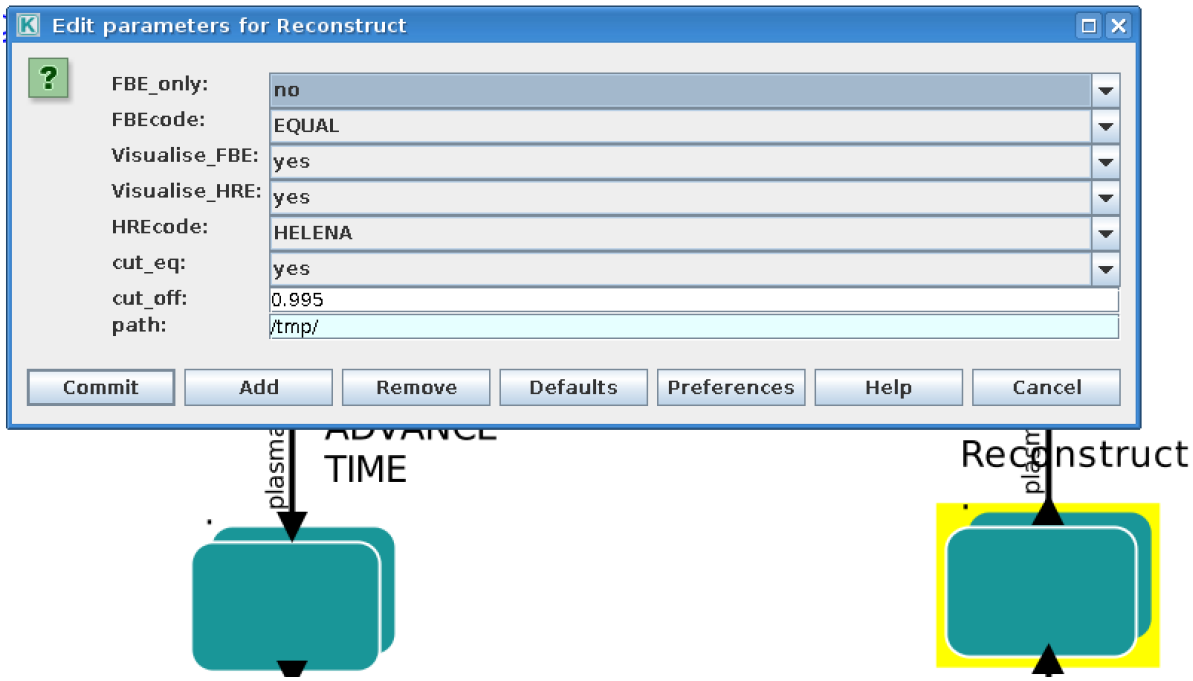
5.2.3 III - Reconstruct

In this composite actor the actual calculation of the reconstructed equilibrium and if requested of the equivalent high resolution equilibrium (with cut-off plasma boundary with X-point removed) takes place. The user can easily gain access to several options for workflow execution by double clicking on the actor (see Figure below).

Among the several options the user can choose :

Here we do a SANITY CHECK on the Magnetics data.
If it shows a "flatline" signal then for sure it should be
ignored and the signal and errors are instead set to -9e40





- To perform plasma equilibrium reconstruction only (FBE_only = yes/no)
- Which code to use to perform the equilibrium reconstruction (FBEcode)
  - EQUAL
  - NICE
  - EFIT++
  - CLISTE
- Which code to use for high resolution equilibrium (HREcode)
  - HELENA
  - CHEASE
  - CAXE
- **If cutting the equilibrium to be piped to the high resultuion calculation** is necessary (cut_eq = yes/no) and if so at what percentage of the normalised separatrix flux (0<cut_off<1)
- **To visualise the reconstruction and high resolution results during** workflow execution (Visualise_FBE, Visualise_HRE=yes/no)

When the user chooses to cut the boundary to perform the high resolution equilibrium calculations:

- A new plasma boundary is determined from the calculated 2D flux map
- The plasma profiles are also cut accordingly (the plasma is not artifically "shrank")
- The total toroidal plasma current is *not* recalculated (equilibrium code should be set to use the boundary poloidal magnetic flux as boundary confition)
- A plot of the original + cut_off equilibrium summary is shown if _Visualise*HRE=yes*.

When the user chooses to visualise any of the calculated equilibria (_Visualise*FBE=yes or* _Visualise*HRE=yes*):

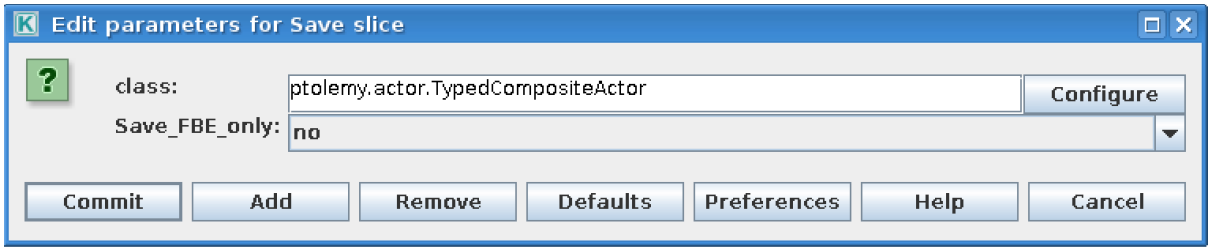
- A window showing the 2D poloidal flux map and radial profiles of Pressure, Toroidal averaged current density and q-profile is displayed for 4 seconds.
- Corresponding image files are saved at the filesystem path indicated by the user selected *path* variable (**START** actor setup).

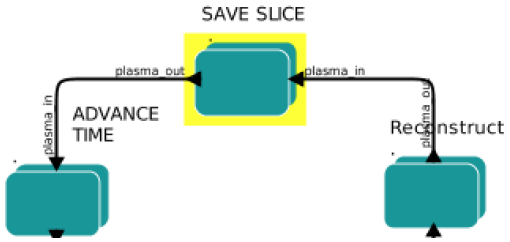
5.2.4 IV - SAVE SLICE

In this composite actor the calculated equilibria are saved at each time step. Depending on whether the user choses to calculate also the high resolution equilibrium and if the Save_FBE_only parameter is set to "yes" or "no", a different number of occurrences of the equilibrium IDS can be stored (see Figure below).

The purpose of saving at each time slice several versions of the equilibrium is to grant extra flexibility. If the user decides to calculate the high resolution equilibrium associated to the reconstructed plasma equilibrium, it might be worth store all 3 stages of the calculated equilibrium. This is managed by using multiple *occurrences* of the equilibrium IDS.

High resolution equilibrium is stored as occurrence=0, the cut boundary "precursor equilibrium" as occurrence=1 and the equilibrium reconstruction as occurrence=2. To control the imasdb saving option please refer to the SAVE SLICE parameter details.





5.3 3. Installing and running the workflow

Establish the IMAS environment by typing

```
module purge
module load cineca
module load imasenv

export KEPLER_DIR=$ITMWORK/imas_kepler
```

if it is the first time you go through this process you will need to create the imas_kepler directory

```
mkdir $ITMWORK/imas_kepler
```

(the one below is the latest version of the dressed kepler containing all the actors for EQRECONSTRUCT, EQSTABIL and ETS-6)

```
module switch kepler/2.5p4-3.0.6_dressed_3.25
kepler_install my_2.5p4-3.0.6_dressed_3.25
kepler_load my_2.5p4-3.0.6_dressed_3.25
```

Once you have installed kepler you do not need to repeat this operation and it will be enough to execute the kepler_load instruction.

Now you need to check out the workflow by typing (only for first time users)

```
svn co --username g2mroma https://gforge6.eufus.eu/svn/eqstabil/tags/imas_3.25.0_4.4.0/
↪workflows eqstabil_workflow
```

Create the database folder with the name of the device you wish to run the equilibrium for

```
imasdb JET
```

Retrieve the data for magnetic-only equilibrium by launching IMASviz or TCV2IDS

Import the following IDSs

magnetics, pf_active, (pf_passive), (iron_core), wall, tf

Note that iron_core is only needed for JET and WEST and pf_passive is only desirable (not supported by all equilibrium reconstruction actors)

You are now ready to launch Kepler by typing

```
kepler
```

load the EQRECONSTRUCT workflow from your eqstabil_workflow directory

5.4 4. Setting up the Workflow and Actor parameters

5.4.1 I - Setting the workflow parameters

The workflow parameters in the **START** actor are as follows:

- **shot** : the shot number on the user database (or from another user) where to read the reference equilibrium from (shot/run_in pair)
- **run_in** : the run number where the reference equilibrium is (shot/run_in pair)
- **run_work**: placeholder run for the temporary Kepler CPOs
- **run_out**: run number where the final results of the run will be stored (user running the workflow/shot/run_out). Since the input equilibrium can be a reconstruction that goes beyond the separatrix, 3 occurrences of the equilibrium are saved (original eq., cut equilibrium inside separatrix and corresponding high resolution equilibrium).
- **user**: username. Reading from someone else database is possible but the run_out will naturally be written to personal database only.
- **device** : device database where the input reference data is. MUST BE the same as the device set once running "*imasdb*" command otherwise the run_out data will end on the wrong database path.
- **time_begin**: starting time for the run (in *seconds*).
- **time_end**: ending time for the run (in *seconds*).
- **time_dt** : time step (constant value) while moving from _time*begin* to _time*end*.

The workflow parameters in the **Reconstruct** actor are as follows:

- **FBE_only**: Set to "*yes*" if addressing only the plasma equilibrium reconstruction. If set to "*no*" the high resolution equilibrium is also calculated.
- **FBEcode**: Choice for equilibrium reconstruction code to be used.
- **Visualise_FBE**: Set to "*yes*" to get a plot of the reconstructed equilibrium at every step.
- **Visualise_HRE**: Set to *yes* to get a plot of the high resolution equilibrium derived from the reconstructed equilibrium at every step.
- **HREcode**: Choice for high resolution equilibrium code to be used.
- **cut_eq**:
  - yes: cut the input equilibrium (necessary if high resolution equilibrium code cannot handle separatrix plasma equilibria)
  - no: input equilibrium is used "as is".
- **cut_off**: float]0,1], specifies the percentage of the separatrix flux that will define the poloidal flux of the new plasma boundary.

- **path**: temporary folder where to dump the plots generated. Also used to store output files (used by HELENA).

The workflow parameters in the **SAVE SLICE** actor are as follows:

- **Save_FBE_only**:
  - yes : only occurrence=0 is saved. If the user set _FBE*only=yes* then the equilibrium reconstruction is saved, otherwise the high resolution equilibrium is stored.
  - no : occurrences = 0,1,2 are saved. Only meaningful if the user set _FBE*only=no*. High resolution equilibrium is stored as occurrence=0, the cut boundary "precursor equilibrium" as occurrence=1 and the equilibrium reconstruction as occurrence=2

The user can always stop the workflow by Pressing the STOP button in Kepler canvas.

5.4.2 II - Setting actor parameters

Actor parameters are set on the actors themselves (not passed by the workflow). To access the actors codeparam the easiest route is to :

1. Click on "Outline" Tab (below the "Pause" button in the KEPLER canvas)
2. Type the name of the actor and press "Search" (or Enter)
3. On the final item in the chain of the actor composite, right click and press"Configure". A pop-up panel appears
4. Click on "Edit Code Parameters" and a new window appears
5. Edit the code parameters and Press "Save & Exit"
6. Press "Commit" and setting is completed

5.5 6. News and Recent activity

8th March 2019: JET version of the workflow tested successfully on test/84600/28 database. Only EQUAL + HELENA codes included. Successful run from t=49s to 53s with both EQUAL and HELENA being executed and the corresponding data stored on the IMAS database (run=33).

CHAPTER

SIX

CODES

6.1 IMASviz

The IMASViz code is used for IMAS visualisation.

6.2 IMASgo!

IMASgo! is an OMFIT module for the mapping of experimental data and machine descriptions into an IMAS database. IMASgo! builds upon the OMFITprofiles, KineticEFIT and TRANSP modules. Any tokamak for which the above modules have been configured can use IMASgo! to map its data into IMAS. At present IMASgo! can write IDSs on the ITER server and on the WPCD Gateway. An account on one of these servers is necessary to complete the workflow in IMASgo! IMASgo! workflow starts with reading the equilibrium from the Tokamak database; kinetic measurements such as Thomson Scattering, ECE, Charge Exchange will be then mapped on the equilibrium and fitted with a variety of available methods. The last step is to export the experimental data, the fitted profiles and the machine description of NBI and RF antennas in the IMAS database. Below are the step by step instructions on how to launch the IMASgo! module and perform the mapping for JET. Some of the files used by IMASgo! for the mapping of the NBI / ICRH machine data are only accessible from inside the JET network hence this example has been run on Heimdall

On an Heimdall terminal type

```
$ module purge
$ module load omfit
$ omfit
```

The OMFIT framework will be launched Click on continue to OMFIT. From the *File* drop menu select *Import module ...*

1. Select the *IMASgo* module from the list of available modules.
2. Double click on *IMASgo*. The module will be loaded in OMFIT and ready to be launched.
3. Double click on *IMASgo* in the *View1* list of loaded modules.
4. A GUI will be displayed. In this case the device choses is JET and the pulse and *Times* to be mapped appear in the next two fields at the top of the GUI.
5. From the drop menu Operation chose equilibrium from PPF It is possible to choose amongst different EFIT++ option: run EFIT++, load the chain 1 magnetics only EFIT, load the pressure constraint equilibrium in the *Equilibrium Source DDA* drop menu and selecting the PPF UID and sequence number.

6. Click on *Generate equilibrium*.
7. Once the step is completed click on the tab *Profiles* and select *1D fits* from the drop menu *Workflow*.
8. Click the *Fetch* tab and select which diagnostics you would like to upload data from.
9. Click on the *fetch* and map all data buttons.
10. In the Slice tab select the time averaging and click *slice all data*.
11. In the *select* tab you can visualise the sliced profiles and deselect profiles in advance of the the fitting step. Once the selected profiles are ok move to the fit tab and select the fitting method for all the data that need fitting. Chose the default fitting parameters or modify them. Then click on *fit 1D* and *plot*.
12. Once all the data are fitted move to the *postfit tab* and *calculate* the derived quantities.
13. The *postfit* tab allows also for manipulation of the profile in order to meet certain constraints e.g. separatrix values. The *plot* tab allows to have a final look at the data before they are saved in the database.
14. Click on the *Machine* tab and click on *Generate Machine Description*. This step will provide the data for the NBI and ICRH IDSs.
15. The final step is to export the data.
16. Click on the *Export* tab and on *Generate OMAS*. This will save the data in memory into the OMAS datastructure.
17. Then set up the server, shot number, run number and hit *save ODS to IMAS*. An entry will be created in the user's IMAS database on the Gateway for JET/92436/107 as well as the same entry for the ITM database (CPOs).

Two videos showing an example of use of IMASgo! to fetch and map data of JET pulse 92054 (NBI only) and run ETS with the same data are available on YouTube at

https://www.youtube.com/watch?v=8bPSjEy2dNk&t=8s

https://www.youtube.com/watch?v=dv427_XOFf4&t=287s

6.3 How to turn a C++ code into a Kepler actor

This document is based on material provided by Yann Frauel and describes how to make your C++ code EU-IM compliant and how to turn it into a Kepler actor.

6.3.1 Adapt your C++ function

You must include the header file UALClasses.h:

```
#include "UALClasses.h"
```

The function arguments that are arrays or strings must be declared as pointers, as usual. All other arguments must be passed by reference (i.e. they must be declared with an ampersand):

```
void mycppfunction(double * vector, char * string, int & scalar)
```

The function arguments that are CPOs must be declared with types ItmNs::Itm::cpo_type or ItmNs::Itm::cpo_typeArray. The first form is for time-independent CPOs or a single slice of a time-dependent CPO. The latter is for a complete time-dependent CPO. Note that in all cases, the CPO is considered as a single object, not an array, so it must be passed by reference as mentioned above:

```
void mycppfunction(
ItmNs::Itm::limiter & lim,
ItmNs::Itm::coreimpur & cor,
ItmNs::Itm::ironmodelArray & iron)
```

The syntax is identical for input and output arguments. For output CPOs, do not forget to use the usual methods to assign strings and allocate arrays:

```
lim.datainfo.dataprovider.assign("test_limiter");
iron.array.resize(3);
iron.array(j).desc_iron.geom_iron.npoints.resize(3);
```

Otherwise, the content of CPOs is accessed as usual:

```
cout << lim.datainfo.dataprovider << endl;
cout << iron.array(j).desc_iron.geom_iron.npoints(i);
```

6.3.2 How to use code parameters

The code parameters are passed as the last argument with ItmNs::codeparam_t& type:

```
void mycppfunction(..., ItmNs::codeparam_t & codeparam)
```

Each field of the param structure is a vector of 132-byte strings, not necessarily terminated by 0-character! (This does not follow C/C++ standards and should be changed in the future.)

6.3.3 Compile your function as a library

You need to include the header directories for the UAL and Blitz:

```
-I$(UAL)/include -I$(UAL)/lowlevel -I$(UAL)/cppinterface/ -I/afs/efda-
itm.eu/gf/project/switm/blitz/blitz-0.9/include/
```

Same for linking:

```
-L$(UAL)/lib -lUALCPPInterface -lUALLowLevel -L/afs/efda-
itm.eu/gf/project/switm/blitz/blitz-0.9/lib -lblitz
```

Additionally, you must compile with the -fPIC option.

6.3.4 Full example

We want to generate an actor that has three different types of actors as inputs and three different types of actors as output. Additionally, we have an integer as input/output, a vector of doubles as output and a string as output. We also want to use code parameters. Content of mycppfunction.cpp:

```
#include "UALClasses.h"

typedef struct {
      char **parameters;
      char **default_param;
```

```
      char **schema;
} param;

void mycppfunction(
      ItmNs::Itm::summary  SUM,
      EU-IMNS::EU-IM::ANTENNAS & ANT,
      EU-IMNS::EU-IM::EQUILIBRIUMARRAY & EQ,
      INT & X,
      EU-IMNS::EU-IM::LIMITER & LIM,
      EU-IMNS::EU-IM::COREIMPUR & COR,
      EU-IMNS::EU-IM::IRONMODELARRAY & IRON,
      DOUBLE * Y,
      CHAR * STR,
PARAM & CODEPARAM)
{

      /* DISPLAY FIRST LINE OF PARAMETERS */
      COUT &LT< codeparam.parameters[0] << endl;
      cout << codeparam.default_param[0] << endl;
      cout << codeparam.schema[0] << endl;
      /* display content of inputs */
      cout << "x=" << x << endl;
      cout << sum.time << endl;
      cout << sum.datainfo.dataprovider << endl;
      cout << ant.datainfo.dataprovider << endl;
      cout << eq.array(0).datainfo.dataprovider << endl;
      for (int k=0; k<3; k++) {
           for (int i=0; i<4; i++) {
                cout << eq.array(k).profiles_1d.psi(i)<< " ";
           }
           cout << endl;
      }
      /* fill limiter CPO */
      lim.datainfo.dataprovider.assign("test_limiter");
      lim.position.r.resize(5);     // allocate vector
      for (int i=0; i<5; i++) {
            lim.position.r(i)=(i+1);
      }
      /* fill coreimpur CPO */
      cor.datainfo.dataprovider.assign("test_coreimpur");
      cor.flag.resize(3);          // allocate vector
      for (int i=0; i<3; i++) {
            cor.flag(i)=(i+1)*10;
      }
      cor.time=0; // don't forget to fill time for time-dependent CPOs
      /* fill ironmodel CPO */
      iron.array.resize(3);        // allocate slices
      for (int j=0; j<3; j++) {
            char s[255];
            sprintf(s,"test_ironmodel%d",j);
            iron.array(j).datainfo.dataprovider.assign(s); // allocate vector
            iron.array(j).desc_iron.geom_iron.npoints.resize(3);
            for (int i=0; i<3; i++) {
                  iron.array(j).desc_iron.geom_iron.npoints(i)=j*i;
            }
            iron.array(j).time=j;       // fill time for time-dependent CPOs
      }
      /* assign value to non CPO outputs */
      x=5;
      for (int i=0; i<10; i++) {
            y[i]=i;
      }
      strcpy(str,"This is a test string");
}
```

Content of Makefile:

```
CXXFLAGS=-g -fPIC -I$(UAL)/include -I$(UAL)/lowlevel -I$(UAL)/cppinterface/
-I$SWEU-IMDIR/blitz/blitz-0.9/include/
LDFLAGS=-L$(UAL)/lib -lUALCPPInterface -lUALLowLevel -L/afs/efda-
itm.eu/gf/project/switm/blitz/blitz-0.9/lib -lblitz
```

```
libmycppfunction.a: mycppfunction.o
      ar -rvs libmycppfunction.a mycppfunction.o
mycppfunction.o: mycppfunction.cpp
clean:
      rm mycppfunction.o libmycppfunction.a
```

6.3.5 How to fill the FC2K window

First tab (Argument):

- set number of input and output arguments (combined)
- select type of arguments from drop-down menu
- tick if argument is a single time slice
- tick if argument is array (not for pointers)
- if necessary define size of arrays
- tick if argument is input argument
- tick if argument is output argument (multiple ticks possible)

The fields Kepler, Ptolemy, and UAL are automatically filled with the values which you set by running the `EU-IMv1 script`.

Second tab (HasReturn):

- specify return parameters (type, array, size)

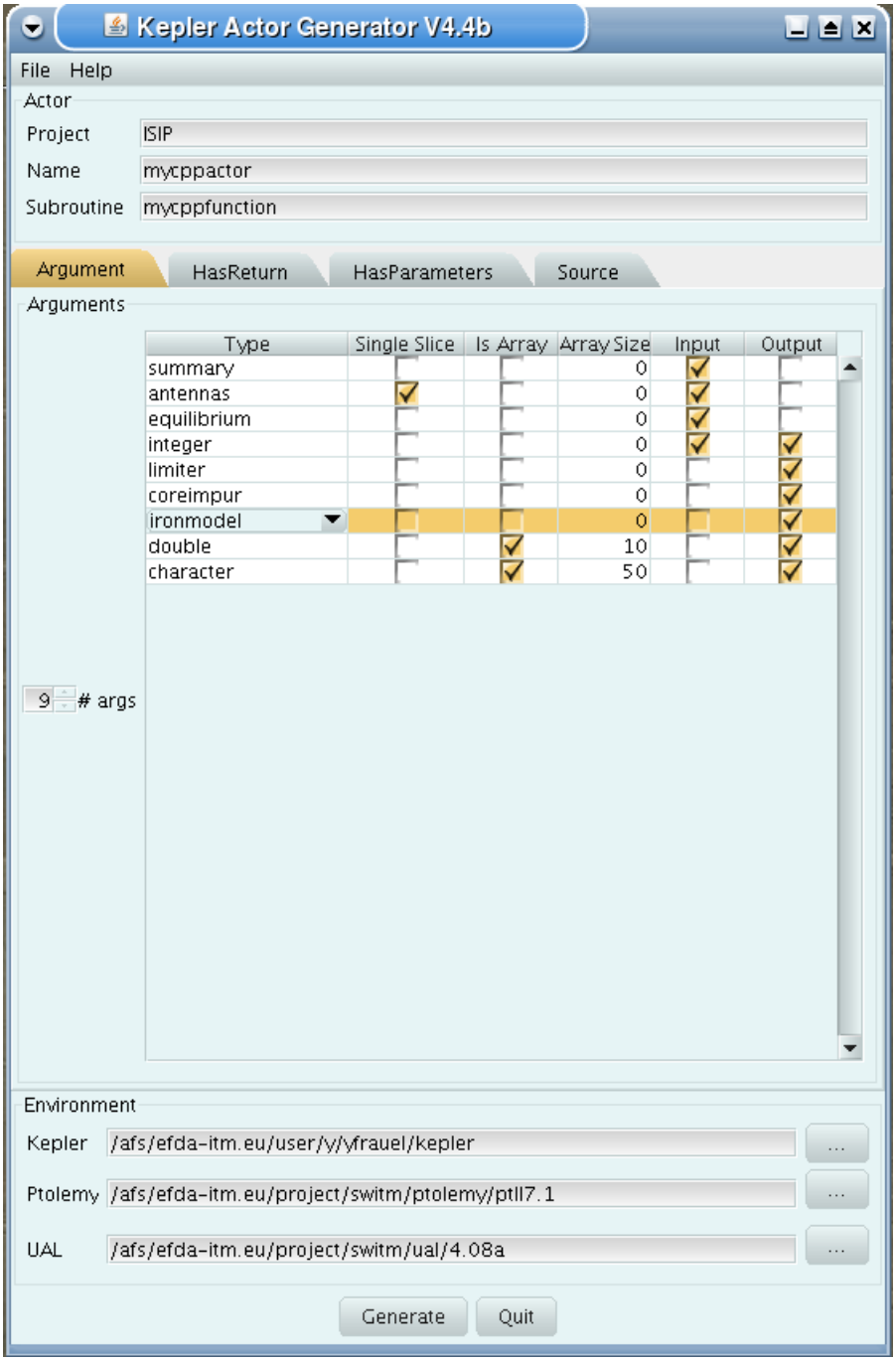
Third tab (HasParameters):

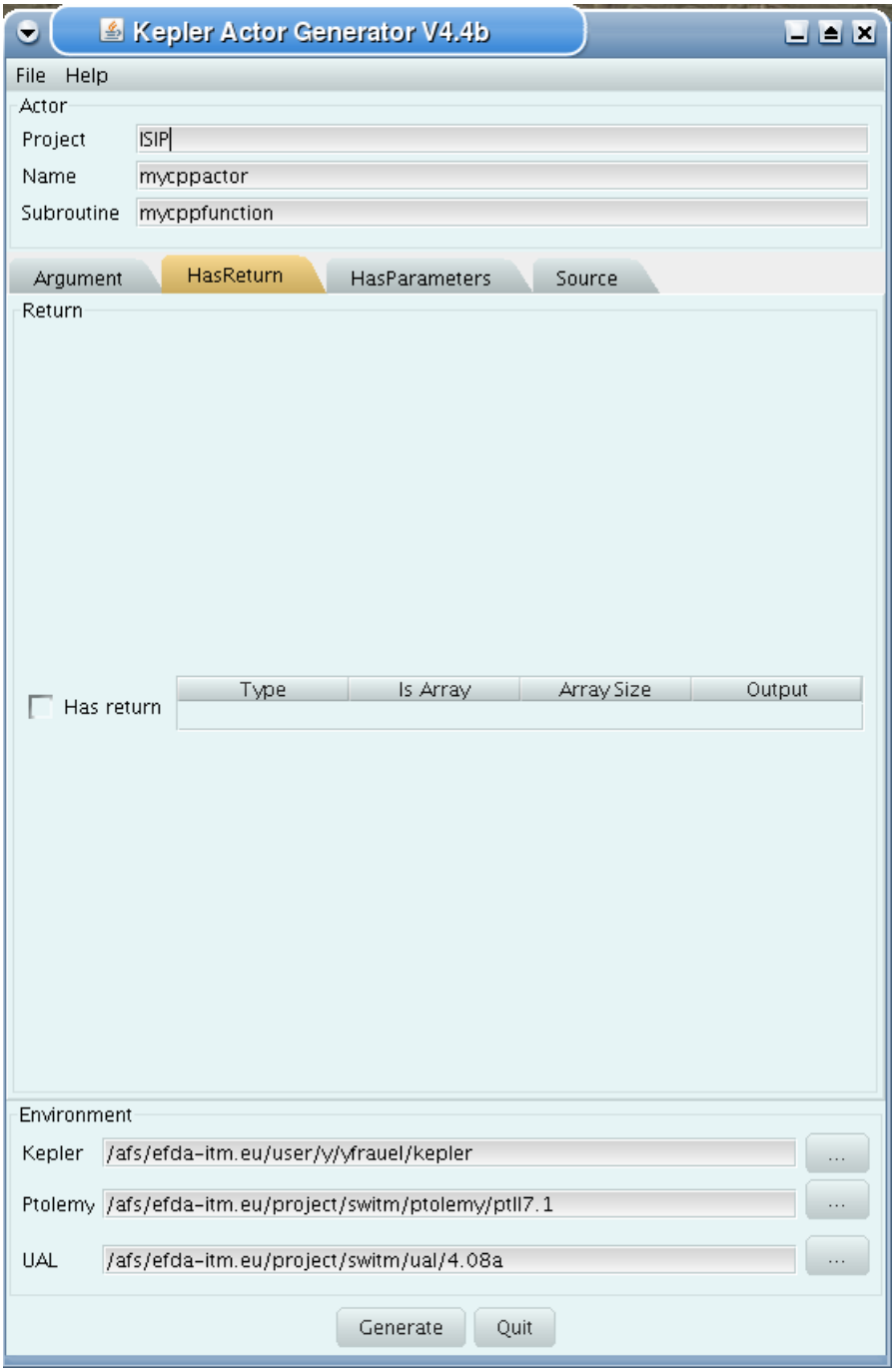
- tick if subroutine uses code specific parameters
- specify (or browse for) XML code parameter input file
- specify (or browse for) XML default code parameter file
- specify (or browse for) W3C XML schema file (XSD)

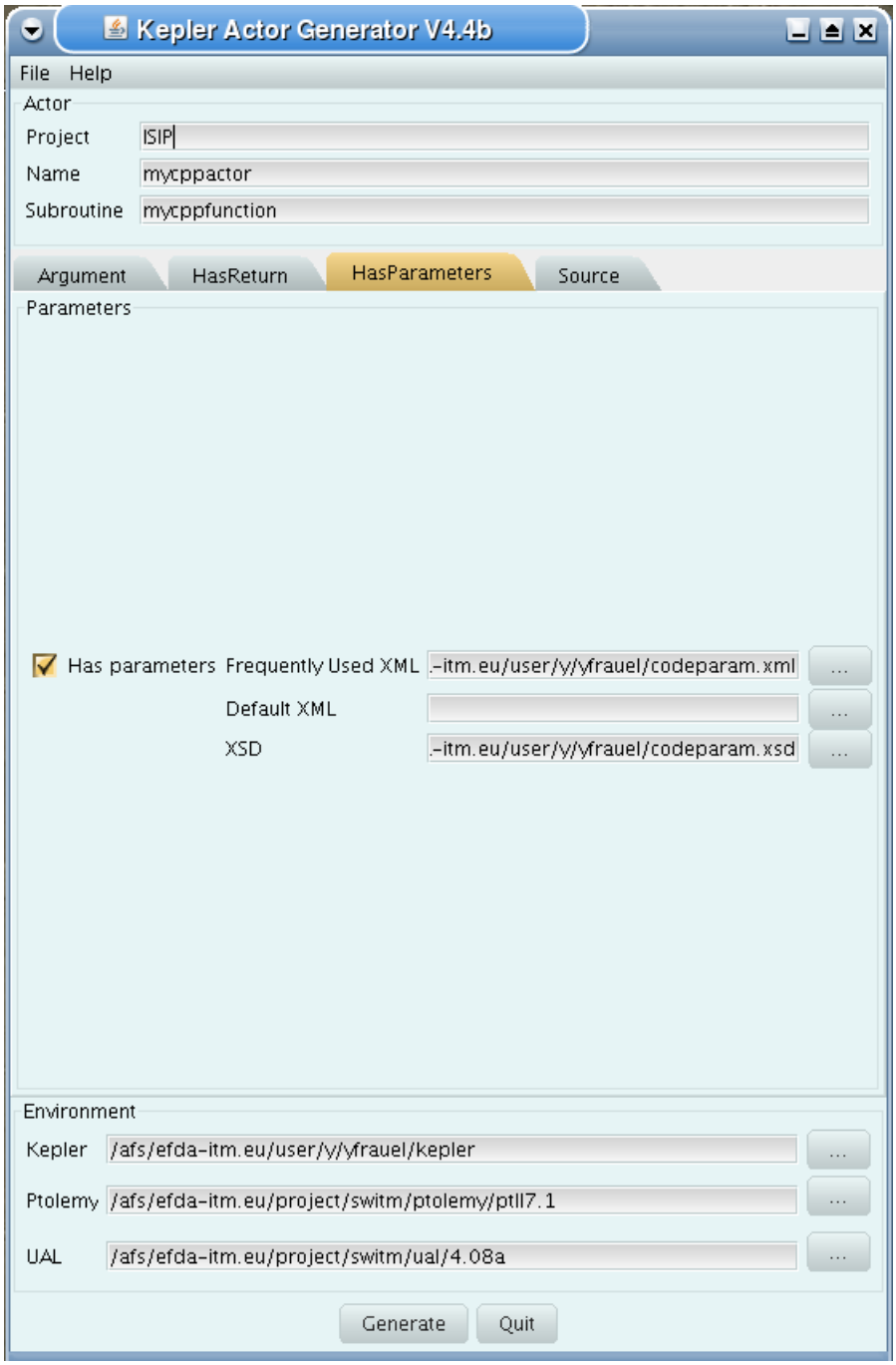
For information on code specific parameters, please see *How to handle code specific parameters*.

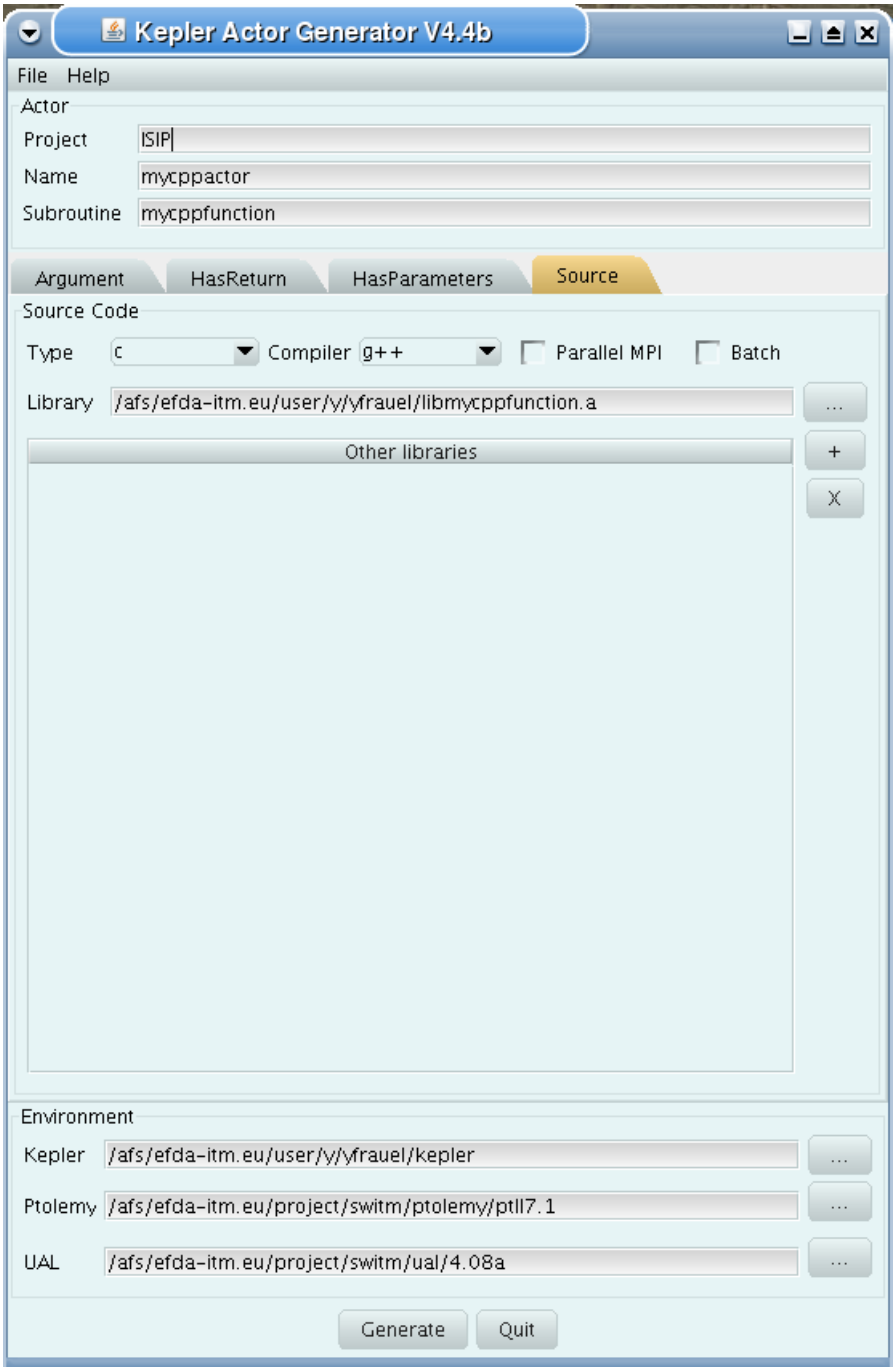
Fourth tab (Source):

- specify programming language of source code
- select appropriate compiler
- tick Parallel MPI if code module is using MPI
- tick Batch if code module shall be run in batch mode rather than interactively when running Kepler workflows
- specify (or browse for) library file containing the code module
- specify (or browse for) other libraries required by the code module


Kepler Actor Generator V4.4b
File Help
Actor
Project ISIP
Name mycppactor
Subroutine mycppfunction
Argument HasReturn HasParameters Source
Arguments
Type Single Slice Is Array Array Size Input Output
summary 0
antennas 0
equilibrium 0
integer 0
limiter 0
coreimpur 0
ironmodel 0
double 10
character 50
9 # args
Environment
Kepler /afs/efda-itm.eu/user/y/yfrauel/kepler
Ptolemy /afs/efda-itm.eu/project/switm/ptolemy/ptII7.1
UAL /afs/efda-itm.eu/project/switm/ual/4.08a
Generate Quit


Kepler Actor Generator V4.4b
File Help
Actor
Project ISIP
Name mycppactor
Subroutine mycppfunction
Argument HasReturn HasParameters Source
Return
Has return
Type Is Array Array Size Output
Environment
Kepler /afs/efda-itm.eu/user/y/yfrauel/kepler
Ptolemy /afs/efda-itm.eu/project/switm/ptolemy/ptII7.1
UAL /afs/efda-itm.eu/project/switm/ual/4.08a
Generate Quit


Kepler Actor Generator V4.4b
File Help
Actor
Project ISIP
Name mycppactor
Subroutine mycppfunction
Argument HasReturn HasParameters Source
Parameters
Has parameters Frequently Used XML .-itm.eu/user/y/yfrauel/codeparam.xml
Default XML
XSD .-itm.eu/user/y/yfrauel/codeparam.xsd
Environment
Kepler /afs/efda-itm.eu/user/y/yfrauel/kepler
Ptolemy /afs/efda-itm.eu/project/switm/ptolemy/ptII7.1
UAL /afs/efda-itm.eu/project/switm/ual/4.08a
Generate Quit


Kepler Actor Generator V4.4b
File Help
Actor
Project ISIP
Name mycppactor
Subroutine mycppfunction
Argument HasReturn HasParameters Source
Source Code
Type c Compiler g++ Parallel MPI Batch
Library /afs/efda-itm.eu/user/y/yfrauel/libmycppfunction.a
Other libraries
Environment
Kepler /afs/efda-itm.eu/user/y/yfrauel/kepler
Ptolemy /afs/efda-itm.eu/project/switm/ptolemy/ptII7.1
UAL /afs/efda-itm.eu/project/switm/ual/4.08a
Generate Quit

6.4 Plasma equilibrium and MHD list of codes

The following list lists the codes and modules which are part of WPCD tasks and their responsible officers.

6.4.1 Free boundary equilibrium codes

- CEDRES++, S. Brémond, CEA
- CLISTE, P. Mc Carthy, DCU
- CREATE-NL, M. Mattei, ENEA Frascati
- EFIT++, L. Appel, CCFE
- EQUAL, W. Zwingmann, EC
- EQUINOX, B. Faugeras, CEA
- FIXFREE, E. Giovannozzi, ENEA Frascati

6.4.2 Fixed boundary equilibrium codes

- CAXE, S. Medvedev, EPFL
- CHEASE, O. Sauter, EPFL
- HELENA, C. Konz, IPP

6.4.3 Linear MHD stability codes

- KINX, S. Medvedev, EPFL
- ILSA, C. Konz, IPP
- MARS, G. Vlad, ENEA Frascati
- MARS-F, D. Yadykin, Chalmers

6.4.4 Sawtooth Crash Modules

- SAWTEETH, O. Sauter, CRPP

6.4.5 ELM Modules

6.4.6 NTM Modules

- NTMETS, S. Nowak

6.4.7 Numerical Tools

- PROGEN, C. Konz, IPP
- JALPHA, C. Konz, IPP

6.5 Heating, current drive (H&CD) and fast particles list of codes

The following list lists the codes and modules which are part of WPCD tasks and their responsible officers.

6.5.1 Electron heating codes

6.5.1.1 EC wave codes

- TORAY-FOM, E. Westerhof, FOM
- TORBEAM, E. Poli, IPP-Garching
- GRAY, L. Figini, ENEA-CNR
- TRAVIS, N. B. Marushchenko, IPP-Greifswald

6.5.1.2 Combined electron Fokker-Planck codes

- RELAX, E. Westerhof, FOM

6.5.1.3 Wave codes for ion cyclotron heating

- TORIC, R. Bilato, IPP-Garching
- EVE, R. Dumont, CEA (Cadarache)
- LION, O. Sauter, CRPP
- Cyrano, E. Lerche, ERM/KMS
- ICCOUP, T. Johnson, VR

6.5.1.4 Fokker-Planck codes for ion cyclotron heating

- RFOF, T. Johnson, VR
- StixRedist, E. Lerche and D. Van Eester

6.5.1.5 NBI sources for Fokker-Planck codes

- BBNBI (Beamlet-based NBI module of ASCOT), J. Varje, TEKES
- NEMO, M. Schneider, CEA (Cadarache)

6.5.1.6 Nuclear sources (input for Fokker-Planck codes)

- Nuclearsim, T.Johnson, VR
- AFSI Ascot Fusion Source Integrator, J. Varje, Aalto

6.5.1.7 NBI Fokker-Planck codes

- RISK, M. Schneider, CEA (ITER)
- NBISIM, T. Johnson, VR

6.5.1.8 Runaway electrons

- Runaway Indicator (Runin), G. Pokol, et al (BME): Runin has been developed to provide an indication for when to expect runaway tail formation. The source code is stored in the *OSREP project <https://github.com/osrep>*.
- Runaway Fluid (Runafluid), G. Pokol, et al (BME): Porpose of Runafluid is to provide a non-inductive current due to runaway electrons using computationaly cheap analytical estimates of ruraway electron growth rates and transport. The source code is stored in the *OSREP project <https://github.com/osrep>*.

6.5.1.9 Advanced codes

(The following codes include either the synergy between IC and NBI heating, or include both wave field and Fokker-Planck solver)

- ASCOT, S. Sipila and J. Varje, Aalto
- SPOT, M. Schneider, CEA (Cadarache)

6.5.1.10 Codes for fast ion-MHD interactions

- LIGKA, P. Lauber, IPP-Garching
- MARS, G. Vlad, ENEA-Frascati
- HYMAGYC, G. Vlad, ENEA-Frascati
- HMGC, C. Di Troia, ENEA-Frascati
- LEMAN, W.A. Cooper, EPFL-CRPP

6.6 Transport list of codes

- ASPOEL
- BIT1
- CARRE
- COS

- EIRENE
- EIRENE2
- EMC3-EIRENE
- ERO
- ETS
- METIS4EU-IM
- SOLPS
- SOLPS6

CHAPTER

SEVEN

CONVENTIONS

7.1 Standard Machine Names

The following machine names are suggested:

- aug
- ftu
- iter
- jet
- mast
- tcv
- tore_supra
- west

7.2 Physics Conventions

The EU-IM-TF has agreed on a variety of conventions to facilitate the integration of the code modules across EFDA. In the following the most important conventions are explained in detail to remove confusion and avoid ambiguity. For more physical detail than that represented here see F Hinton and R Hazeltine, Rev Mod Phys 48 (1976) 239-308, or R Hazeltine and J Meiss, Plasma Confinement (Addison-Wesley, 1992).

7.2.1 Coordinate System

There are generally two choices for defining a right-handed coordinate system in a toroidal geometry with the following coordinates:

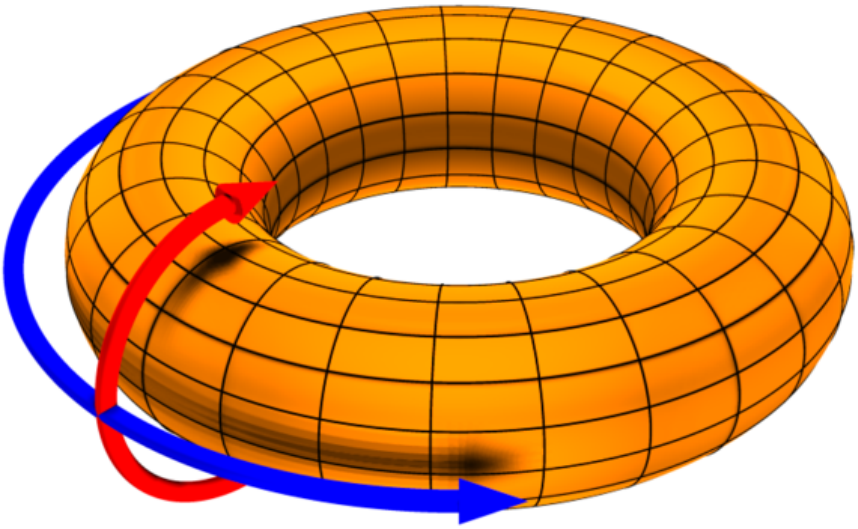
- major radius R
- vertical heights Z
- toroidal angle ϕ

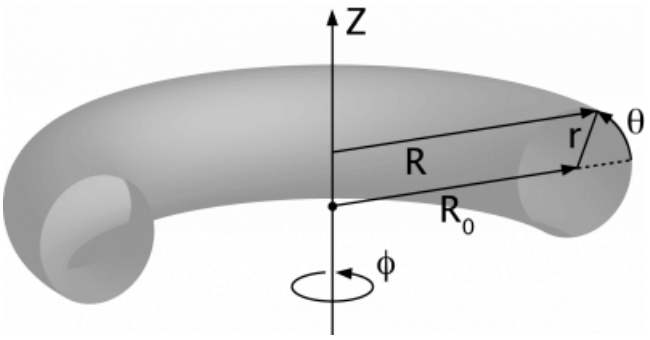
Remaining consistent with ITER, the EU-IM-TF has chosen to adopt the right-handed system

$$(R, \phi, Z)$$

i.e. R is to the right, Z is upwards, and ϕ points into the plane on the right-hand side of the torus (i.e. mathematically positive). Looking from above, the toroidal angle is counter-clockwise, i.e. mathematically positive.

The following figures demonstrate the orientation of the toroidal angle ϕ and the poloidal angle θ:





source:

http://www-fusion.ciemat.es/fusionwiki/index.php/Toroidal_coordinates

http://en.wikipedia.org/wiki/Toroidal_and_poloidal

7.2.2 Representation of the Magnetic Field and Current

Generally, the magnetic field is described in terms of two scalar fields as it is divergence free. If the field is also axisymmetric then MHD equilibrium demands these are functions of each other. In the

EU-IM-TF the relevant quantities are $F_{\rm dia}$ and Ψ and the representation is

$$\mathbf{B} = F_{\rm dia}\nabla\phi + (2\pi)^{-1}\nabla\Psi \times \nabla\phi$$

where the factor of 2π is to have Ψ one and the same with the poloidal flux in Webers (see below).

The current given by Ampere's law is

$$\mu_0\mathbf{J} = \nabla F_{\rm dia} \times \nabla\phi - (2\pi)^{-1}(R^2\nabla\cdot R^{-2}\nabla\Psi)\nabla\phi$$

The respective covariant toroidal components are useful forms:

$$B_\phi = F_{\rm dia} \qquad\qquad \mu_0 J_\phi = -(2\pi)^{-1}(R^2\nabla\cdot R^{-2}\nabla\Psi)$$

where the latter is often expressed in terms of the "delta-star" operator, $\Delta^* = R^2\nabla\cdot R^{-2}\nabla$. These are not the toroidal field and current but the toroidal field and current multiplied by R respectively. The total plasma current I_p is the integral of J_ϕ/R over the poloidal cross section (usually, but not always, over the closed flux surface region only).

7.2.3 Poloidal and Toroidal Fluxes

The toroidal flux Φ is the integral of B_ϕ/R over the region enclosed by the flux surface. Due to axisymmetry it is also a volume integral

$$\Phi = \oint d^3V(2\pi R^2)^{-1}F_{\rm dia}$$

All volume integrals are understood as integration over the region enclosed by the flux surface. They are therefore flux quantities (pure functions of Ψ). The units of Φ are volt-seconds, or Webers (Wb).

The poloidal flux is Ψ due to the construction of $\mathbf{B}$. The factor of 2π ensures this is not Wb per radian (the more usual quantity ψ used as a covariant toroidal component of the magnetic potential is in Wb/radian; the factor of 2π results from integration over one angular circuit). Note that the poloidal flux Ψ and its equivalent per radian ψ are often used equivalently in the literature.

7.2.4 Safety Factor

The magnetic pitch parameter is defined in terms of the flux components:

$$q \equiv d\Phi d\Psi$$

which is a flux quantity. This definition is the same as saying the magnetic pitch is given as the number of toroidal cycle a magnetic field line traverses per unit poloidal cycle. It is also called the local safety factor for MHD stability reasons (here, "local" means local to a given flux surface). Equivalent relations often seen depend on the definition of coordinates. These are given for straight field line coordinates, below.

7.2.5 Signs

With the above definition of the toroidal coordinate system and the magnetic field, the following sign relationships ensue (where increasing and decreasing refer to going from the magnetic axis to the separatrix on the outboard midplane):

Table 7.1: Sign Relations

B_{tor}	I_p	Ψ	Φ	safety factor q
positive	positive	decreasing	increasing	negative
positive	negative	increasing	increasing	positive
negative	positive	decreasing	decreasing	positive
negative	negative	increasing	decreasing	negative

7.2.6 COCOS - toroidal coordinate conventions

16 different fundamental coordinate conventions (COCOS) has been identified for toroidal systems. These are described by O. Sauter and S. Yu. Medvedev, Computer Phys. Commun. 184 (2013) 293.

The current EU-IM convention (described above) is number 13, while the ITER convention is 11.

7.2.6.1 Equilibrium COCOS transformation library and actor

A Fortran library has been developed for transforming the equilibrium cpo between different COCOS. The source is found in

```
https://gforge6.eufus.eu/svn/numerical_tools/tags/COCOStransform_v1_1
```

and the actor is

```
https://gforge6.eufus.eu/svn/kepleractors/tags/4.09a/imp12/COCOStransformequil.tar
```

(also available from: ~sauter/public/ACTORS/4.09a)

Inputs:

- Equilibrium_in : input cpo
- COCOS_in : COCOS of the input equilibrium (if the COCOS is not stored in Equilibrium_in)
- COCOS_out : Requested COCOS for the Equilibrium_out
- Ipsign_out : Requested sign for output Ip; -9 if just wants IP_in transformed to new equilibrium, +1 or -1 if a specific sign in output is desired
- B0sign_out : Requested sign for output B0

Output:

- Equilibrium_out : Output cpo

7.2.7 The Flux Surface Average

In general, the flux surface average is the operation which annihilates the magnetic derivative $\mathbf{B} \cdot \nabla$ and acts as an identity operator on any flux quantity. It can be proved that this results in a volume derivative of a volume integral (alternatively one starts with the latter property and then proves the former, as the above Ciemat reference does). The flux surface average of a scalar and divergence of a vector are given by

$$\langle G \rangle = \frac{\partial}{\partial V} \oint d^3V G \qquad \langle \nabla \cdot \mathbf{G} \rangle = \frac{\partial}{\partial V} \langle \mathbf{G} \cdot \nabla V \rangle$$

where $\mathbf{G} \cdot \nabla V$ is the contravariant volume component of the vector $\mathbf{G}$. It follows that the flux surface average is an angle average weighted by the volume element $\sqrt{g}$

$$\langle G \rangle = \oint d\phi \oint d\theta \sqrt{g} G \Big/ \oint d\phi \oint d\theta \sqrt{g}$$

for any choice of toroidal and poloidal angle as well as radial coordinates, where g is the determinant of the covariant metric tensor components in those coordinates. Note in general G is not an axisymmetric quantity so the integration is actually over both angles.

For more detail see the above references.

7.2.8 The Toroidal Flux Radius as the Radial Coordinate

The EU-IM-TF has decided to use the toroidal flux radius $\rho_{\rm tor}$ defined by

$$\Phi = \pi B_0 \rho_{\rm tor}^2$$

where B_0 is the reference (vacuum) magnetic field value. Note that $\rho_{\rm tor}$ is a positive quantity which has units of meters. For several applications the volume radius $\rho_{\rm vol}$ is also used. It is a normalised radius going from 0 to 1 and is defined as

$$V = V_{\rm LCFS} \rho_{\rm vol}^2$$

where LCFS refers to the last closed flux surface. Both should be defined in the equilibrium CPO (as well as `volume` $\equiv V$ itself).

7.2.9 Toroidal and Parallel Current

These are not equivalent, despite the often-seen experimental practice of considering them so. The toroidal current given in Amperes depends on some convention applied to J_ϕ given above, which is not a flux quantity. The EU-IM-TF has decided on this definition of the toroidal current as a flux quantity:

$$\texttt{jphi} \equiv \langle J^\phi \rangle / \langle 1/R \rangle$$

This uses the contravariant toroidal component of $\mathbf{J}$ which is a pure divergence

$$J^\phi = \mathbf{J} \cdot \nabla \phi = J_\phi / R^2 = -\nabla \cdot (2\pi\mu_0 R^2)^{-1} \nabla \Psi$$

Hence the flux surface average invokes the often-used quantity $\langle g^{\rho\rho}/R^2 \rangle$ in the form

$$\langle J^\phi \rangle = -(2\pi\mu_0)^{-1} \frac{1}{V'_\rho} \frac{\partial}{\partial \rho} V'_\rho \langle g^{\rho\rho}/R^2 \rangle \frac{\partial \Psi}{\partial \rho}$$

Here, $V'_\rho \equiv \partial V / \partial \rho_{\rm tor}$ explicitly using the toroidal flux radius as the radial coordinate.

The parallel current is different from this due to the finiteness of the poloidal current and magnetic field. Generally the correction is $O(\epsilon^2/q^2)$ which is usually a few percent (but not in a spherical tokamak). Using the representations for $\mathbf{B}$ and $\mathbf{J}$ given above we find

$$\mathbf{J} \cdot \mathbf{B} = -(2\pi\mu_0)^{-1} F_{\rm dia}^2 \nabla \cdot \frac{1}{F_{\rm dia} R^2} \nabla \Psi$$

Since $F_{\rm dia}$ is a flux quantity the flux surface average behaves as for `jphi` and we use a factor of B_0 to provide the correct units, yielding

$$\texttt{jparallel} \equiv -(2\pi\mu_0 B_0)^{-1} \frac{F_{\rm dia}^2}{V'_\rho} \frac{\partial}{\partial \rho} \frac{V'_\rho}{F_{\rm dia}} \langle g^{\rho\rho}/R^2 \rangle \frac{\partial \Psi}{\partial \rho}$$

This form has been chosen due to the natural use of the flux surface average $\langle \mathbf{J} \cdot \mathbf{B} \rangle$ in neoclassical theory and the magnetic flux diffusion equation (see the Hinton and Hazeltine reference above).

7.2.10 Straight Field Line Coordinates

A variety of modules in the EU-IM-TF use straight field line coordinate systems to represent the closed flux surface region. To guarantee consistency with the definition of the poloidal flux and the magnetic field representation given above, a standard definition of the coordinate volume element follows. This is the same sense as the usage of the term "Jacobian" in the CPOs (note many papers use the inverse volume element as the "Jacobian" by contrast). Here, "straight field line coordinates" refers to the use of the right-handed coordinate system (Ψ, θ, ζ) with the poloidal flux Ψ, the straight field line angle θ, and the toroidal angle $\zeta = -\phi$. Therefore, θ has the same orientation as the poloidal angle θ in toroidal coordinates, while the toroidal angle ζ is in the opposite direction of ϕ. This is standard usage generally in terms of "flux coordinates" (see Hazeltine and Meiss, above).

Note here that while the toroidal angle is the geometric one in the orientation sense of flux coordinates, the poloidal angle is not geometric. This results from the demand that the field lines be straight in the coordinate plane (θ, ζ). The definition of this property is given by the specification of the ratio of contravariant components of the magnetic field as a flux quantity, which is one and the same with the pitch parameter ("local safety factor"):

$$q = q(\Psi) = -B^{\zeta}/B^{\theta} = B^{\phi}/B^{\theta}$$

where the minus sign appears by consistency with the primary definition in terms of the flux components as given above. This represents a magnetic differential equation for the poloidal angle:

$$B^{\theta} = B^{\phi}/q = F_{\mathrm{dia}}/qR^2$$

Due to the choice of "natural" coordinates (with Ψ, not ρ_{tor}) this relation is close to the definition of the volume element $\sqrt{g}$ and, equivalently, the Jacobian J

$$J \equiv \sqrt{g} \qquad J^{-1} = \nabla\Psi \cdot \nabla\theta \times \nabla\zeta = \nabla\Psi \times \nabla\phi \cdot \nabla\theta$$

Note the ordering of $\nabla\Psi$ and $\nabla\phi$.

The components of the magnetic field are then

$$\begin{aligned} B^{\theta} &= \mathbf{B} \cdot \nabla\theta = (2\pi)^{-1} \nabla\Psi \times \nabla\phi \cdot \nabla\theta = (2\pi J)^{-1} \\ B^{\zeta} &= \mathbf{B} \cdot \nabla\zeta = -B_{\phi}/R^2 = -F_{\mathrm{dia}}/R^2 \\ B^{\Psi} &= \mathbf{B} \cdot \nabla\Psi = 0 \end{aligned}$$

With these relations the following relationship between the Jacobian and pitch parameter ("local safety factor") holds

$$J = (2\pi)^{-1} qR^2/F_{\mathrm{dia}}$$

This is the quantity labelled `jacobian` in the equilibrium CPO.

7.2.11 Plasma Betas

Out of the many definitions of plasma betas, the EU-IM has agreed to adhere to the following definitions: Following Wesson (p. 116), the poloidal beta is defined as an integral over the poloidal cross section

$$\beta_{\mathrm{p}} = \frac{2\mu_0}{B_{\mathrm{a}}^2} \frac{\int_A p \mathrm{dS}}{\int_A \mathrm{dS}}$$

where $A = A(\Psi)$ is the poloidal cross section enclosed by the flux surface Ψ, $B_\mathrm{a} = \frac{\mu_0 I}{l}$ is the flux surface averaged poloidal magnetic field, $I = I(\Psi)$ the toroidal plasma current inside the flux surface Ψ and $l = \oint \mathrm{dl}$ the length of the poloidal perimeter of flux surface Ψ. This definition yields a one-dimensional profile $\beta_\mathrm{p} = \beta_\mathrm{p}(\Psi)$ stored in profiles_1d%beta_pol in the equilibrium CPO. The overall poloidal beta $\beta_\mathrm{p}(\Psi = \Psi_\mathrm{bd})$ is stored in global_param%beta_pol.

The toroidal beta is defined as

$$\beta_\mathrm{tor} = \frac{2\mu_0}{B_0^2} \frac{\int_\Omega p \mathrm{dV}}{\int_\Omega \mathrm{dV}}$$

with B_0 the vacuum magnetic field as stored in global_param%toroid_field%b0. The integral is carried out over the entire plasma volume and the result stored in global_param%beta_tor.

The normalized plasma beta is defined as

$$\beta_\mathrm{N} = 100 \frac{aB_0}{10^{-6} I_\mathrm{p}} \beta_\mathrm{tor}$$

with I_p the total plasma current (following Y.-S. Na et al., PPCF 44 (2002), 1285) and a is the minor radius. It is stored in global_param%beta_normal.

7.2.12 Internal Inductance

The definition of the internal inductance follows J.A. Romero et al., NF 50 (2010), 115002. The magnetic energy contained inside the flux surface Ψ is

$$W_\mathrm{mag} = \frac{1}{2\mu_0} \int_\Omega B_\mathrm{p}^2 \mathrm{dV}$$

where B_p is the poloidal component of the magnetic field. The (unnormalized) internal inductance is then defined as

$$L_\mathrm{i} = \frac{2W_\mathrm{mag}}{I^2}$$

where $I = I(\Psi)$ is the toroidal plasma current enclosed by the flux surface Ψ. The normalized internal inductance, as stored in profiles_1d%li is defined as

$$l_\mathrm{i} = \frac{2L_\mathrm{i}}{\mu_0 \bar{R}}$$

with the surface averaged major radius

$$barR = \frac{\int_A R \mathrm{dS}}{\int_A \mathrm{dS}} = \frac{V(\Psi)}{2\pi A(\Psi)}$$

The overall internal inductance $l_\mathrm{i}(\Psi = \Psi_\mathrm{bd})$ is stored in global_param%li.

7.2.13 Poloidal Angle Dimension in Equilibrium CPO

The following entries in the equilibrium CPO are defined along the poloidal dimension (as dim2 in the case of a flux surface equilibrium, i.e. radial coordinate psi in dim1 and poloidal angle in dim2):

```
coord_sys%jacobian(:,:)
coord_sys%g_11(:,:)
coord_sys%g_12(:,:)
coord_sys%g_13(:,:)
coord_sys%g_22(:,:)
coord_sys%g_23(:,:)
coord_sys%g_33(:,:)
profiles_2d%position
profiles_2d%grid
profiles_2d%psi_grid(:,:)
profiles_2d%jphi_grid(:,:)
profiles_2d%jpar_grid(:,:)
profiles_2d%br(:,:)
profiles_2d%bz(:,:)
profiles_2d%bphi(:,:)
```

The EU-IM-TF has decided not to repeat the first poloidal point (with poloidal angle $\theta = 0$, which is identical to $\theta = 2\pi$. This option was chosen to facilitate Fourier transforms along the poloidal direction. To that purpose it is required that the dimension dim2 be equidistant in the poloidal angle θ (going from $\theta = 0$ to $\theta = (ndim2 - 1)/ndim2 * 2\pi$ where ndim2 is the number of poloidal grid points), whatever the choice of this angle is.

7.3 Numerical and computational conventions

7.3.1 Standardized Variable Types

To ensure that physics modules produce identical results on various computer architectures and to avoid issues with double precision versus single precision interfaces, the EU-IM-TF has agreed on a set of standardized variable types. It is recommended that these types be used throughout all EU-IM modules, but at least for the interface definitions. The Fortran90 module defining the type standards itm_types.f90 is hosted by the project itmshared . To check out the relevant files please do

```
svn checkout https://gforge6.eufus.eu/svn/itmshared/trunk/src/itm_types target_dir
```

For Fortran90, the following standard types have been defined

```
INTEGER,  PARAMETER :: EU-IM_I1 = SELECTED_INT_KIND (2)        ! Integer*1
INTEGER,  PARAMETER :: EU-IM_I2 = SELECTED_INT_KIND (4)        ! Integer*2
INTEGER,  PARAMETER :: EU-IM_I4 = SELECTED_INT_KIND (9)        ! Integer*4
INTEGER,  PARAMETER :: EU-IM_I8 = SELECTED_INT_KIND (18)       ! Integer*8
INTEGER,  PARAMETER :: R4 = SELECTED_REAL_KIND (6, 37)   ! Real*4
INTEGER,  PARAMETER :: R8 = SELECTED_REAL_KIND (15, 300) ! Real*8
```

To implement these types in your code, please add the following line to your modules

```
use itm_types
```

(More information about the EU-IM libraries.)

7.3.2 Standardized Physical Constants

To avoid discrepancies in simulations from using different definitions of the physical constants, the EU-IM-TF has agreed upon a set of standardized physical constants (all in SI units except for temperatures) based on the NIST recommendations . It is recommended that these constant be used throughout all EU-IM modules. The Fortran90 module defining the standardized physical constants itm_constants.f90 is hosted by the project itmshared . To check out the relevant files please do

```
svn checkout https://gforge6.eufus.eu/svn/itmshared/trunk/src/itm_constants target_dir
```

7.3.3 Invalid Data Base Entries

The EU-IM data base does not allow for setting data base entries directly to invalid in case they should not be set. Since the Universal Access Layer (UAL) always pulls out complete CPOs, i.e. complete data structures, of which not all fields may be filled, the problem arose of how to identify those fields which have not been filled. In the case of arrays, this is simply done by not associating the corresponding pointer. In the case of scalars, however, unique values for floats and integers had to be defined to identify empty fields. These values identify invalid data base entries and can be tested through comparison. The values for invalid data base entries in Fortran90 are defined below:

```
INTEGER,  PARAMETER :: itm_int_invalid = -999999999
REAL(R8), PARAMETER :: itm_r8_invalid = -9.0D40
```

They have been found to be safely out of any physical range for the affected fields such that no accidental confusion with real values may occur. The Fortran90 module defining these values itm_types.f90 is hosted by the project itmshared . To check out the relevant files please do

```
svn checkout https://gforge6.eufus.eu/svn/itmshared/trunk/src/itm_types target_dir
```

The module also includes three functions of type boolean itm_is_valid_int4 , itm_is_valid_int8 , and itm_is_valid_real8 which are overloaded under the interface itm_is_valid to check whether a data base entry has been filled. Example:

```
if (itm_is_valid(equilibrium%global_param%i_plasma)) then
  write(*, *) 'Plasma current Ip = ', equilibrium%global_param%i_plasma
end if
```

7.3.4 Enumerated datatypes/Identifiers

This section concerns how to specify the origin of data in certain types of CPOs. The specification is performed using the datatype identifier. The following specifies the conventions of the allowed enumerated datatypes.

- cocos_identifier.xml
- coordinate_identifier.xml
- coredelta_identifier.xml
- coreneutral_identifier.xml
- coresource_identifier.xml
- coretransp_identifier.xml
- distsource_identifier.xml
- fast_particle_origin_identifier.xml
- fast_thermal_filter_identifier.xml
- fokker_planck_source_identifier.xml
- pellet_shape_identifier.xml

- species_reference_identifier.xml
- wall_identifier.xml
- wave_identifier.xml

7.3.4.1 Example: How to fill coresource/values/sourceid

When filling in an enumerated datatype, like coresource/values/sourceid, it is recomended to use the parameters and functions built into the fortran modules associated with each such datatype. These modules are available as part of the UAL package. As an examples we may include the coresource_identifier:

```
use coresource_identifier, only: fusion, get_type_name, get_type_description__ind
```

Here the value of the integer-parameter fusion is the Flag for fusion reactions in the *coresource_identifier* structure (i.e. fusion=5). Once we know the Flag we may get the Id using the function Id=get_type_name(Flag) and the Description using the function Description=get_type_description__ind(Flag). These function are available for every datatype.

Below you have an example of how to use these functions:

```
program coresource_example use euitm_schemas, only: type_coresource use
  coresource_identifier, only: fusion, get_type_name,
  get_type_description__ind use write_structures, only: open_write_file,
  write_cpo, close_write_file use deallocate_structures, only:
  deallocate_cpo implicit none

  type (type_coresource) :: coresource
  integer :: idx, i

  character*128 :: filename
  integer :: shot, run

  data filename / &
       & 'coresource.cpo' &
       & /

  allocate(coresource%values(1))
  allocate(coresource%values(1)%sourceid%id(1))
  allocate(coresource%values(1)%sourceid%description(1))
  coresource%values(1)%sourceid%flag = fusion
  coresource%values(1)%sourceid%id = get_type_name(fusion)
  coresource%values(1)%sourceid%description =
  get_type_description__ind(fusion)

  call open_write_file(1, filename)
  call write_cpo(coresource, 'coresource')
  call close_write_file

  call deallocate_cpo(coresource)

end program coresource_example
```

This example program, and similar examples for other enumerated datatypes, are available in:

```
https://gforge6.eufus.eu/svn/itmshared/trunk/src/itm_constants/examples
```

7.3.5 Grid Types in Equilibrium CPO

Equilibria may be represented in a variety of different ways depending on which EU-IM module has calculated them and which module shall use them. To avoid ambiguity and to allow modules to check

which type of equilibrium is stored in the equilibrium CPO, a unique grid identifier is stored in profiles_2d%grid_type. The grid identified currently consists of 4 strings (at 132 chars) with the following structure (array indices in Fortran notation):

Position	Content
grid_type(1)	integer identifier for grid type
grid_type(2)	string identifier for grid type
grid_type(3)	integer identifier for poloidal angle
grid_type(4)	string identifier for poloidal angle

7.3.5.1 Grid Type Identifier

The currently allowed values (integer and string) for the identifier of the grid type are listed below:

Integer Values	String Value	Description
1	rectangular	Regular grid in (R, Z). 'EFIT-like grid'
2	inverse	Regular grid in Ψ, θ. 'flux surface grid'.
3	irregular	Irregular grid. All fields in profiles_2d are given as (ndim1, 1) degenerate 2D matrices, i.e. as lists of vertices (for triangles or quadrilaterals).

7.3.5.1.1 Poloidal Angle Identifier

The currently allowed values (integer and string) for the identifier of the poloidal angle are listed below:

Integer Values	String Value	Description
1	straight field line	straight field line angle θ as defined in Straight Field Line Coordinates
2	equal arc	Poloidal angle θ defined by equal arc lengths along flux surfaces
3	polar	Poloidal angle θ in toroidal coordinates as defined in Coordinate System

7.3.6 Standardized EU-EU-IM Plasma Bundle

The EU-IM has agreed on a standardized way to bundle CPOs and control parameters inside KEPLER.

Field names			*Type*	*Description*
time			real	The synthetic time of the simulation, or for time-dependent workflows; the end of the present time step. For example, consider a time dependent workflows, where physics quantities are update one after the other. Thus, while the physics quantities are updated the various fields below (e.g. the CPOs) may be describe at different time points. In such workflows the this "time"-field describe the time at the end of the present time step. Units: (s)
CONTROL	tau		real	time-step (s)
	tau_out		real	time interval for saving output (s)
	ETS	amix	real	mixing factor
		amix_tr	real	mixing factor for profiles

Continued on next page

Table 7.2 – continued from previous page

Field names			*Type*	*Description*
		sigma_source	integer	option for origin of plasma electrical conductivity: 0: plasma collisions; 1: transport module; 2: source module
		solver_type	integer	choice of numerical solver
		conv_rec	real	required fractional convergence
CPOS	MHD	equilibrium	cpo	see type and fortran descriptions
		toroidfield	cpo	see type and fortran descriptions
		mhd	cpo	see type and fortran descriptions
		sawteeth	cpo	see type and fortran descriptions
	CORE	coreprof	cpo	see type and fortran descriptions
		coretransp	cpo	see type and fortran descriptions
		coresource	cpo	see type and fortran descriptions
		coreimpur	cpo	see type and fortran descriptions
		coreneutral	cpo	see type and fortran descriptions
		corefast	cpo	see type and fortran descriptions
		coredelta	cpo	see type and fortran descriptions
		compositionc	cpo	see type and fortran descriptions
		neoclassic	cpo	see type and fortran descriptions
	EDGE	edge	cpo	see type and fortran descriptions
	HCD	waves	cpo	see type and fortran descriptions
		distsource	cpo	see type and fortran descriptions

Continued on next page

Table 7.2 – continued from previous page

Field names			*Type*	*Description*
		distribution	cpo	see type and fortran descriptions
	MACH	vessel	cpo	see type and fortran descriptions
		wall	cpo	see type and fortran descriptions
		nbi	cpo	see type and fortran descriptions
		antennas	cpo	see type and fortran descriptions
		ironmodel	cpo	see type and fortran descriptions
		pfsystems	cpo	see type and fortran descriptions
	DIAG	fusiondiag	cpo	see type and fortran descriptions
		scenario	cpo	see type and fortran descriptions
	EVENTS	pellets	cpo	see type and fortran descriptions

CHAPTER

EIGHT

AMNS

8.1 Scientific Rationale and Main Objectives

The EU-IM has a broad need for data relating to atomic, molecular, nuclear and surface data (AMNS). In particular, AMNS data are needed in several of the EU-IM modelling projects. A consistent approach, taking into account the specific requirements of the EU-IM while maintaining the work aligned with other European efforts in this area, is therefore required. As a consequence the AMNS tasks are implemented as Tasks under the TF leadership and has the following scope:

- Coordination of the work in the four different sub areas.
- Supply of data not presently residing in easily accessible data bases.
- Identify any Intellectual Property Rights (IPR) protection needs in view of a broader collaboration with ITER partners.
- Provide software for delivery of AMNS data to EU-IM-TF codes.

8.2 EU-IM contact person

David Coster

8.3 AMNS tasks

The AMNS work is divided into two broad areas:

- The maintenance and development of the AMNS library (and the associated AMNS CPO) to provide access to AMNS data in the various languages used by the codes within the Work Package
- The addition to the AMNS database of AMNS data needed by the codes within the Work Package

8.4 AMNS Documentation

The AMNS library is meant to be called by Work Package codes if the codes need data for Atomic, Molecular, Nuclear or Surface processes. The calling sequence is described in more detail below, but the basic idea is: (1) initialize the package; (2) request data for a particular reaction by initializing a "table" for that reaction; (3) (repeatedly) requesting data for that reaction as a fuction of plasma or other parameters; (4) finishing with the table; and (5) finishing with the AMNS library.

The actual AMNS data is provided by CPOs stored under the "amns" tokamak and will first be searched for in the user's database, and if not found there, the system will default to obtaining the data from the public AMNS database. Multiple versions of the AMNS data are possible: in 4.09a and 4.09b this was done via a mysql database; in 4.10a and later this is done by having an index block stored in shot 0, run 1 of the AMNS CPO.

Some presentations:

- Nuclear reactions (pdf), by V. Kiptily
- Simulations of the edge plasma: the role of atomic, molecular and surface physics (pdf), by D. P. Coster, S. Gori, X. Bonnin, D.Reiter, A.Kukushkin, P. Krstic, P. Strand, L.-G. Eriksson, Contributors to the EFDA TF EU-IM
- Atomic, Molecular, Surface and Nuclear (AMSN) data for the EU-IM-TF (pdf), presented by D.P. Coster (IMP3 Leader) at the ADAS workshop, based on the talk given by Lars-Goran Eriksson at the EU-IM General Meeting, 2008-09
- EU-IM AMNS Interface (pdf), by D.P. Coster

Some papers:

- **Simulations of the edge plasma: the role of atomic, molecular and surface** physics (pdf), by D.P. Coster, X. Bonnin, D. Reiter, A. Kukushkin, S. Gori, P. Krstic, P. Strand, L.-G. Eriksson and Contributors to the EFDA-TF-EU-IM
- 4. Tskhakaya, D. Coster and ITM-TF contributors, Contrib. Plasma Phys., 54 (4-6), 399–403 (2014)

The present coding for the AMNS project is done in the gforge amnsproto project.

8.4.1 AMNS User Interface

This section discusses the user interface to the AMNS subsystem.

The AMNS library is made available via a module - available versions can be found by executing

```
module avail amns
```

The include and library locations are specified via the "pkg-config" system. To display the available package names do

```
pkg-config --list-all | grep amns
```

Doxygen information about the user interface can be found here.

The AMNS library can be called from

1. Fortran
2. C
3. Python
4. Java (in development)
5. Matlab (in development)

The various bindings for the different languages are given below, but make use of a set of standard concepts which are described first.

8.4.1.1 AMNS 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:

1. amns_handle_type, used for the database wide routines
2. 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:

1. amns_error_type, used to indicate if an error occured and, if so, what the error was
2. amns_reaction_type, used to indicate the requested reaction
3. amns_set_type, used to set an AMNS internal parameter
4. amns_query_type, used to query an AMNS internal parameter
5. amns_answer_type, used to contain the answer from an AMNS query
6. amns_version_type, used to specify the AMNS version
7. amns_reactants_type, used to specify the reactants to a reaction
8. amns_reactant_type, a sub-component of amns_reactants_type used to characterize the individual reactants

The definitions of these data types can be found at the doxygen documentation for the AMNS User routines

8.4.1.2 AMNS User Interface Data Reactions

The currently available reactions specified in reaction_typex%string in the call to EUIM_AMNS_SETUP_TABLE are

1. RC: Recombination (acd)
2. EI: Electron Impact Ionisation (scd)
3. CX: CX recombination coeffts (ccd)
4. BR: Recomb/brems power coeffts (prb)
5. LR: Line radiation (plt)
6. ZE: Effective Charge (zcd)
7. ZE2: Effective Square Charge (ycd)
8. EIP: Effective Ionisation Potential (ecd)
9. some nuclear reactions
10. Cross-sections of different processes
11. ...

The actual reactions are listed in the AMNS section.

8.4.1.3 AMNS User Interface Data Queries

The currently available queries for query%string in the call to EU-IM_AMNS_QUERY is

1. version: Return the version information

The currently available queries for query%string in the call to EU-IM_AMNS_QUERY_TABLE are

1. source: source (origin) of the data
2. no_of_reactants: number of tractants involved
3. index: Not sure what this is
4. filled: whether the data table has been filled ("Filled" or "Empty")
5. reaction_type: reaction type
6. reactants: nuclear charges of reactants
7. version: information about the version
8. state_label: label for the charge state (if appropriate)
9. result_unit: units of the result
10. result_label: description of the result

8.4.1.4 AMNS User Interface Data Setting Options

The currently setting options for set%string in the call to EU-IM_AMNS_SET is

1. NONE

The currently available setting options for set%string in the call to EU-IM_AMNS_SET_TABLE is

1. nowarn: deactivate warning when extrapolating

8.4.1.5 FORTRAN AMNS User Interface

The fortran interface to the AMNS subsystem is based on a standardised set of calls to the AMNS library. The details of what lies behind these calls is the responsibility of the AMNS data providers and does not need to be understood by the users of the AMNS data.

The code modules devloped for the AMNS project are hosted in gforge as the project amnsproto.

8.4.1.5.1 AMNS User Interface: Fortran Calls

The 9 calls to the AMNS system are:

1. EU-IM_AMNS_SETUP, initialization call for the AMNS package

```
subroutine EU-IM_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
```

2. EU-IM_AMNS_QUERY, query routine for the AMNS package

```
subroutine EU-IM_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
```

3. EU-IM_AMNS_SET, set a parameter for the AMNS package

```
subroutine EU-IM_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
```

4. EU-IM_AMNS_FINISH, finalization call for the AMNS package

```
subroutine EU-IM_AMNS_FINISH(handle, error_status)
  optional error_status
  type(amns_handle_type), intent(inout) :: handle
  type(amns_error_type), intent(out) :: error_status
```

5. EU-IM_AMNS_SETUP_TABLE, initialization call for a particular reaction

```
subroutine EU-IM_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
```

6. EU-IM_AMNS_QUERY_TABLE, query routine for a particular reaction

```
qsubroutine EU-IM_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
```

7. EU-IM_AMNS_SET_TABLE, set a parameter for a particular reaction

```
subroutine EU-IM_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
```

8. EU-IM_AMNS_FINISH_TABLE, finalization call for a particular reaction

```
subroutine EU-IM_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
```

9. EU-IM_AMNS_RX, get the rates associated with the input args for a particular reaction

```
interface EU-IM_AMNS_RX
  module procedure EU-IM_AMNS_RX_1, EU-IM_AMNS_RX_2, EU-IM_AMNS_RX_3
end interface

subroutine EU-IM_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 EU-IM_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 EU-IM_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
```

8.4.1.5.2 AMNS User Interface Example (Fortran)

An example of the use of the code can be found in the (fortran minimal example):

```
program minimal
  use itm_types
  use amns_types
  use amns_module

  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
  real (kind=R8) :: te=100.0_R8, ne=1e20_R8, rate

  call EU-IM_AMNS_SETUP(amns)                                    ! set up the AMNS system
  allocate(species%components(4))                              ! set up reactants
  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) /)
  xx_rx%string='CX'                                            ! set up reaction
  call EU-IM_AMNS_SETUP_TABLE(amns, xx_rx, species, amns_rx)   ! set up table
  call EU-IM_AMNS_RX(amns_rx, rate, te, ne)                      ! get results
  write(*,*) 'Rate = ', rate
  call EU-IM_AMNS_FINISH_TABLE(amns_rx)                          ! finish with table
  call EU-IM_AMNS_FINISH(amns)                                   ! finish with amns

end program minimal
```

8.4.1.5.3 AMNS User Interface Example Fortran Makefile

An example Makefile demonstrating the use of the AMNS routines:

```
obj/minimal: src/minimal.f90
        ifort -g -o $@ $< ${shell eval-pkg-config --cflags --libs \
      amns-amd64_intel_12 itmtypes-amd64_intel_12 ual-amd64_intel_12}
```

Other examples can be found (here):

8.4.2 C AMNS User Interface

The C interface to the AMNS subsystem is based on a standardised set of calls to the AMNS library. The details of what lies behind these calls is the responsibility of the AMNS data providers and does not need to be understood by the users of the AMNS data.

The code modules devloped for the AMNS project are hosted in gforge as the project amnsproto.

8.4.2.1 AMNS User Interface: C Calls

The 9 calls to the AMNS system are:

1. EU-IM_AMNS_SETUP, initialization call for the AMNS package

```
void EU-IM_AMNS_C_SETUP(void **handle_out, amns_error_type *error_status);
```

2. EU-IM_AMNS_QUERY, query routine for the AMNS package

```
void EU-IM_AMNS_C_QUERY(void *handle_in, amns_query_type *query,
                        amns_answer_type *answer, amns_error_type *error_status)
```

3. EU-IM_AMNS_SET, set a parameter for the AMNS package

```
void EU-IM_AMNS_C_SET(void *handle_in, amns_set_type *set, amns_error_type *error_status);
```

4. EU-IM_AMNS_FINISH, finalization call for the AMNS package

```
void EU-IM_AMNS_C_FINISH(void **handle_inout, amns_error_type *error_status);
```

5. EU-IM_AMNS_SETUP_TABLE, initialization call for a particular reaction

```
void EU-IM_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);
```

6. EU-IM_AMNS_QUERY_TABLE, query routine for a particular reaction

```
void EU-IM_AMNS_C_QUERY_TABLE(void *handle_rx_in, amns_query_type *query,
                              amns_answer_type *answer, amns_error_type *error_status);
```

7. EU-IM_AMNS_SET_TABLE, set a parameter for a particular reaction

```
void EU-IM_AMNS_C_SET_TABLE(void *handle_rx_in, amns_set_type *set,
                            amns_error_type *error_status);
```

8. EU-IM_AMNS_FINISH_TABLE, finalization call for a particular reaction

```
void EU-IM_AMNS_C_FINISH_TABLE(void **handle_rx_inout, amns_error_type *error_status);
```

9. EU-IM_AMNS_RX, get the rates associated with the input args for a particular reaction

```
void EU-IM_AMNS_C_RX_0_A(void *handle_rx_in, double *out,
                         double arg1, amns_error_type *error_status);
void EU-IM_AMNS_C_RX_0_B(void *handle_rx_in, double *out,
                         double arg1, double arg2, amns_error_type *error_status);
void EU-IM_AMNS_C_RX_0_C(void *handle_rx_in, double *out,
                         double arg1, double arg2, double arg3, amns_error_type *error_s
tatus);

void EU-IM_AMNS_C_RX_1_A(void *handle_rx_in, int nx, double *out,
```

```
                        double *arg1, amns_error_type *error_status);
void EU-IM_AMNS_C_RX_1_B(void *handle_rx_in, int nx, double *out,
                        double *arg1, double *arg2, amns_error_type *error_status);
void EU-IM_AMNS_C_RX_1_C(void *handle_rx_in, int nx, double *out,
                        double *arg1, double *arg2, double *arg3, amns_error_ty
pe *error_status);

void EU-IM_AMNS_C_RX_2_A(void *handle_rx_in, int nx, int ny,
                        double *out, double *arg1, amns_error_type *error_status);
void EU-IM_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 EU-IM_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 EU-IM_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 EU-IM_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 EU-IM_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 EU-IM_AMNS_C_SETUP_REACTANTS(void **reactants_handle_out, char string_in[reaction_
 ↪length],
                                  int index_in, int n_react
ants);
void EU-IM_AMNS_C_SET_REACTANT(void *reactants_handle_in, int reactant_index, amns_reactant_
 ↪type *reactant_in);
void EU-IM_AMNS_C_GET_REACTANT(void *reactants_handle_in, int reactant_index, amns_reactant_
 ↪type *reactant_out);
void EU-IM_AMNS_C_FINISH_REACTANTS(void **reactants_handle_inout);
```

8.4.2.2 AMNS User Interface Example (C)

An example of the use of the code can be found in the (c minimal 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;

  EU-IM_AMNS_CC_SETUP(AMNS_HANDLE, &ERROR_STAT)
  printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
  EU-IM_AMNS_CC_SETUP_REACTANTS(REACTANTS_HANDLE, "", 0, 4)
  EU-IM_AMNS_CC_SET_REACTANT(reactants_handle, 1, SPECIES1)
  EU-IM_AMNS_CC_SET_REACTANT(reactants_handle, 2, SPECIES2)
  EU-IM_AMNS_CC_SET_REACTANT(reactants_handle, 3, SPECIES3)
  EU-IM_AMNS_CC_SET_REACTANT(reactants_handle, 4, SPECIES4)
  EU-IM_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);
  EU-IM_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=%e\n", rate);
  EU-IM_AMNS_CC_FINISH_TABLE(AMNS_CX_HANDLE, &ERROR_STAT)
  printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
  EU-IM_AMNS_CC_FINISH_REACTANTS(REACTANTS_HANDLE)
  EU-IM_AMNS_CC_FINISH(AMNS_HANDLE, &ERROR_STAT)
  printf("error = %s: %s\n", error_stat.flag ? "true" : "false", error_stat.string);
  return 0;
}
```

8.4.2.3 AMNS User Interface Example C Makefile

An example Makefile demonstrating the use of the AMNS routines:

```
obj/minimal: src/minimal.c
        gcc -g -o $@ $< ${shell eval-pkg-config --cflags --libs\
          amns-ifort itmconstants ual-amd64_intel_12}
```

Other examples can be found (here):

8.4.3 Python AMNS User Interface

The Python interface to the AMNS subsystem is based on a standardised set of calls to the AMNS library. The details of what lies behind these calls is the responsibility of the AMNS data providers and does not need to be understood by the users of the AMNS data.

The code modules devloped for the AMNS project are hosted in gforge as the project amnsproto.

8.4.3.1 AMNS User Interface: Python Calls

The Python interface creates

1. Amns (class)
   (a) finalize (method)
   (b) get_table (method)
   (c) query (method)
   (d) set (method)
2. Table (class)
   (a) data (method)
   (b) finalize (method)
   (c) query (method)
   (d) set (method)
3. Reactants (class)
   (a) add (method)
   (b) test (method)
   (c) value (method)

8.4.3.2 AMNS User Interface Example (Python)

An example of the use of the code can be found in the (python minimal example):

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
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("CX", r)
print "table.no_of_reactants", table.no_of_reactants
print table.data(np.array([100.0]), np.array([1e20]))
amnsdb.finalize()
```

Other examples can be found (here):

8.4.4 AMNS CPO

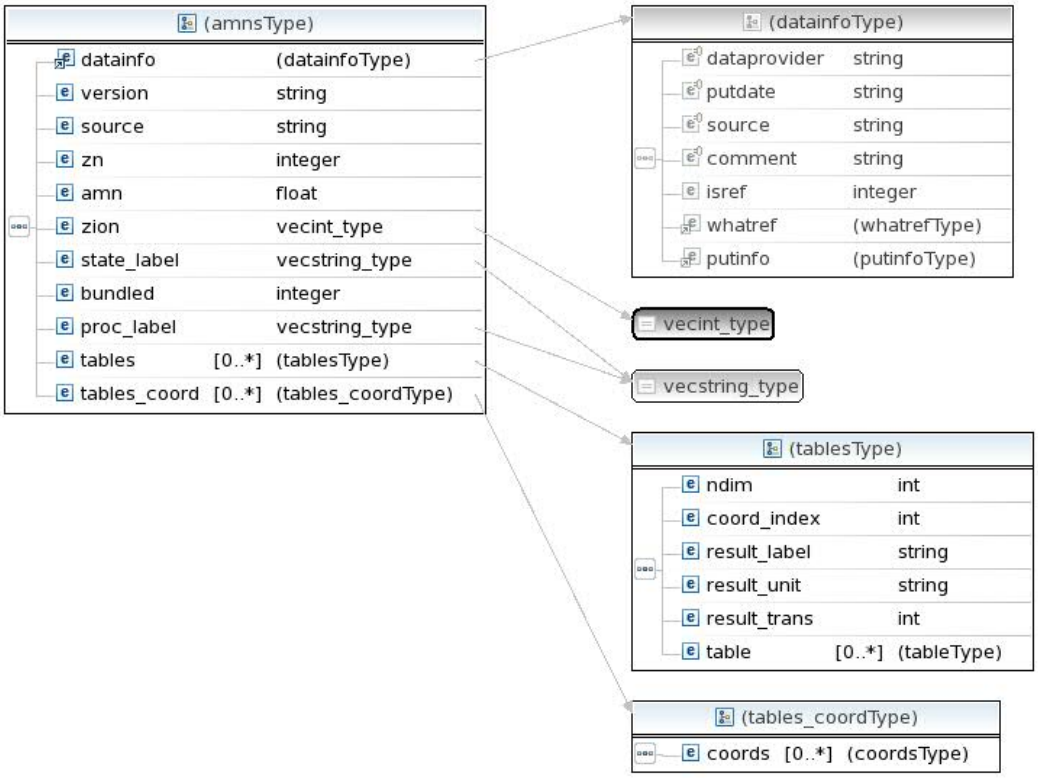
The current (4.08b) data structure for AMNS data in the standard tree view can be browsed here (Browse)

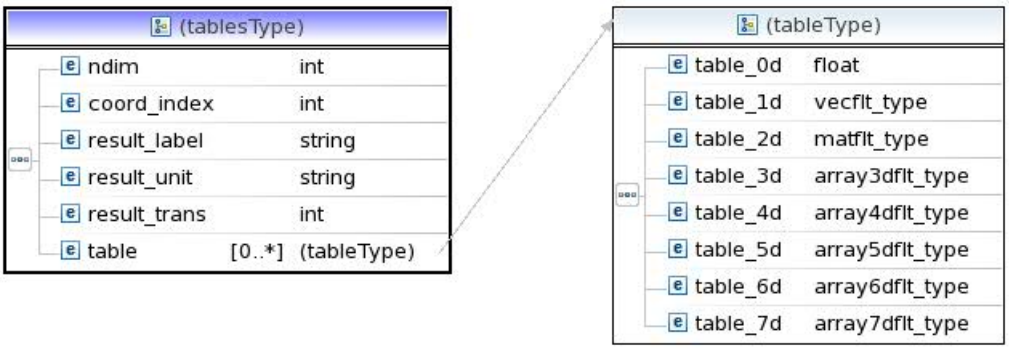
We are currently considering a revision of the AMNS data structure that makes use of arrays-of-structures (not available earlier)

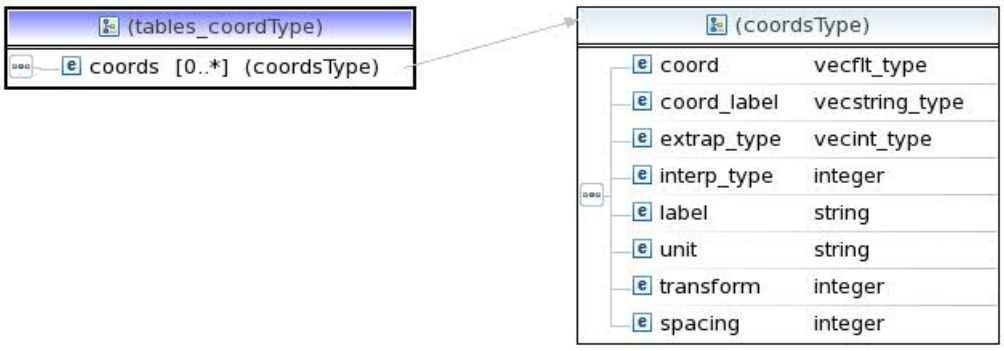
At the top level we would have

with the definition of tables

and the tables of coordinates


(amnsType)
datainfo (datainfoType)
version string
source string
zn integer
amn float
zion vecint_type
state_label vecstring_type
bundled integer
proc_label vecstring_type
tables [0..*] (tablesType)
tables_coord [0..*] (tables_coordType)
(datainfoType)
dataprovider string
putdate string
source string
comment string
isref integer
whatref (whatrefType)
putinfo (putinfoType)
vecint_type
vecstring_type
(tablesType)
ndim int
coord_index int
result_label string
result_unit string
result_trans int
table [0..*] (tableType)
(tables_coordType)
coords [0..*] (coordsType)


(tablesType)
ndim int
coord_index int
result_label string
result_unit string
result_trans int
table [0..*] (tableType)
(tableType)
table_0d float
table_1d vecflt_type
table_2d matflt_type
table_3d array3dflt_type
table_4d array4dflt_type
table_5d array5dflt_type
table_6d array6dflt_type
table_7d array7dflt_type


(tables_coordType)
coords [0..*] (coordsType)
(coordsType)
coord vecflt_type
coord_label vecstring_type
extrap_type vecint_type
interp_type integer
label string
unit string
transform integer
spacing integer

CHAPTER

NINE

USING THE WPCD WORKFLOWS

Use of the WPCD workflows is available via the EUROfusion Gateway, the JET analysis cluster (FREIA) and the ITER IO HPC

9.1 Indices and tables

- genindex
- modindex
- search