ITM gateway user's guide

Revision version: 3rd version
Date: 11 December 2008
Authors : B. Guillerminet, F. Iannone, F.Imbeaux, G. Manduchi, L. Eriksson

Figure 1: Gateway cluster at Portici (Italy)

ITM gateway user's guide 1
Introduction 3
Connection to the gateway 3
Summary: 3
NX 3
Shared directory 6
ITM framework overview 6
Summary: 6
Setting your environment 6
Data 7
Hands-on session: 8
UAL 8
Summary 8
Creating pulse file 10
Opening pulse file 10
Reading CPOs 10
Writing CPOs 11
How to deal with empty fields 11
Tests 11
Performance issues 14
Visualization 15
Post-processing tools 15
Scilab 15
IDL 16
Matlab 17
Codes 18
KEPLER 18
Summary: 18
Documentation 19
Installation 19
Running an example 19

Installing your private version of KEPLER 21
Integration of a local code 22
Integration of a remote code (Web Service) 25
Building a workflow 25
ISE: ITM Simulation Editor 28
Summary: 28
Portal 32

Introduction

The gateway is a cluster located at Portici (near Napoly in Italy). It will be used for fusion simulations in the ITM project. Several web sites are available to get for more information:

-ITM Task Force web site: http://www.efda-taskforce-itm.org/

-Gateway web site (useful for news, administration, trouble ticket …) at http://www.efda-itm.eu

-ITM portal (news, ITM applications, login, wiki, …): http://portal.efda-itm.eu/portal

Connection to the gateway

Summary:

- Get and sign the User Agreement
- Get a login
- Install NX

Use NX or ssh command for the connection. You need a login and password on the gateway (ask your ITM representative or F. Iannone at Francesco.iannone@enea.it).

Note: you have to sign the User Agreement (will be available on the ITM portal or could be obtained on request from your ITM representative).

NX

You must install the client part of NX on your computer. Download the appropriate version from http://freenx.berlios.de/ or http://www.nomachine.com/download.php .

Using the wizard or setting the parameters by hand, you must specify (Figure 1):

Parameter name	Value
Host	enea142.efda-itm.eu or enea143… or enea144
Port	22
Key	Use the default key
Network	WAN

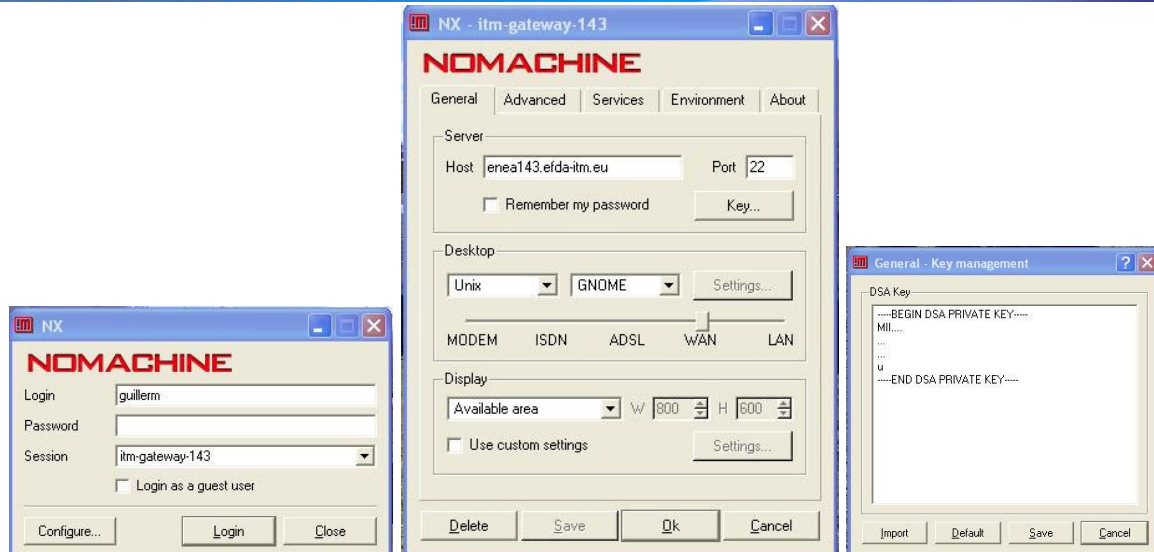


Figure 2: NX configuration

The user's name must have been previously registered on the "gateway" computer (ask Francesco Iannone)

Launch NX client with the above parameters, a window must appear as a local user.

Your own directory is located on /afs/efad-itm.eu/isip/user/xxx where xxx is your login name if you belong to the isip project (could be imp1, imp2…). The shared files and directories of the ITM project are under /afs/efda-itm.eu/project/switm (see Fig 2). Be careful: /afs is a worldwide shared directory, so don't use time consuming command like "ls –al" at the high level. To check or set the right access, you have to use the "fs .." commands (see the documentation at **Organization of Shared Data Areas**).

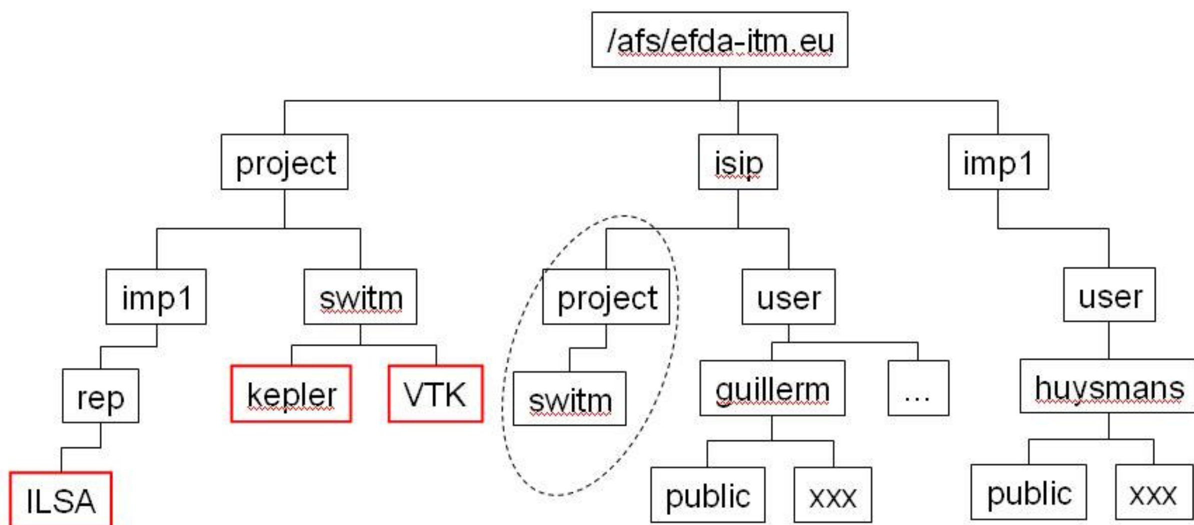


Figure 3: project and users directories

The required environment variables will be set automatically by the system. The standard shell system is tcsh, it uses several files at login (.cshrc and .login). The common cshrc is stored in /afs/efda-itm.eu/system/default.user/cshrc.common file. If you want another shell system (bash for instance), you must add the following command in your ".cshrc" file after the standard initialization:

…
/bin/bash

Password:
To change your password, you have to use the "**kpasswd**" command

You could check several tools like java:

```
java –version
java version "1.6.0_10-ea"
Java(TM) SE Runtime Environment (build 1.6.0_10-ea-b09)
Java HotSpot(TM) 64-Bit Server VM (build 11.0-b09, mixed mode)
```

Or the C compiler. The last gcc version is the version 4 which could be used with the "gcc4" command but the standard version is:

```
gcc --version
gcc (GCC) 3.4.6 20060404 (Red Hat 3.4.6-9)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.

gcc4 --version
gcc4 (GCC) 4.1.2 20070626 (Red Hat 4.1.2-14)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

The Fortran compiler is either the gnu or Portland compiler:

```
g95 --version
G95 (GCC 4.0.3 (g95 0.91!) Jan 16 2008)
Copyright (C) 2002-2005 Free Software Foundation, Inc.

G95 comes with NO WARRANTY, to the extent permitted by law.
You may redistribute copies of G95
under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING

pgf90 -show
pgf90-Warning-No files to process

Known Variables:
ACTTPVAL
AS              Name of assembler program
              =as
ASARGS          Arguments to assembler
…
```

Shared directory

A directory is automatically created for each user to share its contents with others. It is called "public" in your home directory (see Fig 2). Access is "read/write" for you and read only for the others.
Example:

drwxr-xr-x 2 guillerm isip 2048 Jan 15 16:05 public

So put in your public directory the files and documents you want to share with the others users.

ITM framework overview

Summary:

- Get ITMv1
- set the environment variables: source ITMv1
- get a Fortran example from $UAL/fortranExamples and try it
- try to use the interactive tools: vtk, idl, scilab and matlab

The ITM framework provides the necessary tools to define and run a fusion simulation. Several tools are used to set up the parameters and run a simulation (ISE), others are used to import experimental data (exp2ITM), one is for composing workflows (KEPLER) and a few others to integrate the codes (FC2K, WS2K). One does not need to master all the tools to build and run a simulation. An important point is the data storage and access. The experimental and simulation data are stored in the ITM database (see ITM database). In order to use easily all these tools, we provide a set of environment variables (see Settings your environment).

Setting your environment

Most of the environment variables are set automatically during the login session but it could be useful to change some of them if you want to use a non-standard version, for any purpose (testing a new version, using a previous version).
The scripts are gathered in /afs/efda-itm.eu/project/switm/scripts directory and you must set up your path in order to access it easily:
>set path = ($path /afs/efda-itm.eu/project/switm/scripts)
In this directory, we have:

/afs/efda-itm.eu/project/switm/scripts>ls
create_public_itm_dir
set_itm_data_env_sh
create_public_itm_model_dir
set_itm_env
create_user_itm_dir
set_itm_data_env
getactor
rmactor
putactor
README
ITMv1

In green, these are the procedures for the data (location and directories) and in blue, the scripts for KEPLER. ITMv1 is used to set all the variables, copy it in your home directory and update it for your data storage (if any, see below) and your private KEPLER version.

Part of ITMv1 which must be tuned to your local directories:

1. KEPLER. If you have a private version of KEPLER, set the KEPLER variable to the installed version (here $HOME/Kepler)
   # setenv KEPLER /afs/efda-itm.eu/project/switm/kepler
   #
   setenv KEPLER_PUBLIC /afs/efda-itm.eu/project/switm/kepler/4.06d/kepler
   setenv KEPLER $HOME/kepler
   #
2. Data storage. If you have a local data storage ($HOME/public/itmdb/….), you have to specify it:
   # ----------------------------------
   # variables for DATA (UAL & storage)
   # ----------------------------------
   # change test to the machine name
   source /afs/efda-itm.eu/project/switm/ual/set_itm_data_env $USER test 4.06d
   source /afs/efda-itm.eu/project/switm/ual/set_itm_env 4.06d

Data

Data organisation and UAL on the Gateway. Version 4.06d storage and UAL have been installed on the Gateway. The HTML documentation of the data structure is at the usual place, http://www.efda-taskforce-itm.org/ Expert area, ISIP, Data Structure Page.

The UAL is located at **/afs/efda-itm.eu/project/switm/ual/** on the Gateway. In the fortranExamples and cppExamples folders, you will find basic examples how to use the UAL (CREATE,OPEN,GET,PUT). If you are interested, open the file "getting_started" in those folders, it contains a very simple UAL tutorial. We stress again that using the UAL is for "advanced users" only, for those who want to test their programs OUTSIDE KEPLER (see below for the UAL).

We have also set up the new database organisation. A database entry is defined by (user,machine,shot,run), where user is "public" or any userid (allows for private databases). Machine is the tokamak name. We suggest to use a "test" machine name for testing, thus avoiding confusion with real experimental data and real shot numbers. In the "getting_started" files mentioned above, there are the instructions for initialising your private database and setting the environment variables properly. Always use UAL/OPEN or UAL/CREATE with treename = "euitm" and the system does the rest.

The public database is now located in **/pfs/itmdb/itm_trees/public**. Under this directory, you will find the various machine names.
There is one example in the public database : Machine : test Shot : 1 Run :1 It is an example with a few CPOs, contains physical data used as input to the ETS prototype (July 2008) :

- topinfo
- equilibrium (1 time slice)
- coreprof, coretransp, neoclassic (2 time slices)

The corresponding files are located in
/pfs/itmdb/itm_trees/public/test/4.06d/mdsplus/0/euitm10001*

For the moment, if you want to use them as input, it is required to copy "manually" the three pulse files euitm10001* to your private database ~your_username/public/itmdb/itm_trees/test/4.06d/mdsplus/0. No need to copy any model file. Later on, we will provide a UAL command that allows to switch from one (user,machine) to another.

Hands-on session:

The script are reachable if you have updated your path. Otherwise, add the path in front of the command (either /afs/efda-itm.eu/project/switm/ual or /afs/efda-itm.eu/project/switm/scripts).

Creation of your private data storage:

create_user_itm_dir test 4.06d

Copy data:

cp /pfs/itmdb/itm_trees/public/test/4.06d/mdsplus/0/euitm10001*
$HOME/public/itmdb/itm_trees/test/4.06d/mdsplus/0

Setting the path and environment variables:

set_itm_data_env $USER test 4.06d
set_itm_env

UAL

Summary:

- This part is only for stand-alone code: without KEPLER or for testing purpose
- Based on your favourite language, go directly to $UAL/fortranExamples or …
- Copy locally the example, copy it and run it

The data access is done through the Universal Access Layer (UAL) for a code. We emphasize the knowledge and the use of the UAL is not mandatory for a user: it is only useful to know the UAL access when you want to run your code outside of the ITM framework, for testing your code for instance. The data server could be a MDS+ server, an in-memory server or files.

- The Universal Data Access layer allows data retrieval in ITM simulations.
- It represents the unique interface to data.
- It provides a structured view of data organizated in Consistent Physical Objects (CPOs).
- It is available in multiple languages : F90, C++, Java, (Matlab, Scilab)
- Each CPO has a tree organization. CPO fields can be time-dependent or not. Conceptually, every pulse file contains an array of CPOs, where the time-independent fields hold always the same value, regardless the current time.
- The « time » field is therefore always defined in CPOs containing time-dependent fields. It contains the time to which that CPO instance refers.

- Even though time-independent fields are stored in single instance in the underlying database, the get() routine will return an array of structures, with as many instances as the number of stored samples.

A standalone program will need to:

- Open or create a ITM pulse file;
- Read/write CPOs: every CPO is represented by an (array of) language – dependent structures reflecting the agreed CPO hierarchy;
- Close the ITM pulse file.
-

However, a module running in the Kepler framework does not need to read/write CPOs. In fact it will receive the specified (array of) CPO instance(s) as argument, and it will be up to the Kepler framework to read and write CPO instances. Therefore a correct approach in the development of ITM components is:

- Implement the analysis algorithms in a module which receives the CPO instances as argument, using the Kepler-compatible argument organization;
- Develop a data wrapper which reads and writes CPOs in the pulse file, calling the module above as a subroutine.

The UAL and MDS+ libraries are both installed on the "gateway" computer. You could check their locations using the following commands:

env |grep MDS
MDS_PATH=/afs/efda-itm.eu/project/switm/mdsplus/mdsplus/tdi
MDSPLUS_DIR=/afs/efda-itm.eu/project/switm/mdsplus/mdsplus

cd /afs/efda-itm.eu/project/switm/ual>ls

lowlevel
cppExamples
fortranExamples
cppinterface
fortraninterface
remote
cppTest

set_itm_data_env
create_public_itm_dir
include
create_public_itm_model_dir
jar
set_itm_data_env_sh
create_user_itm_dir
javainterface
set_itm_env
javaTest
TestUAL
db
lib
xml

The directories for the examples are enlightened in **blue** and several important procedures are in **green**.

Creating pulse file

- New pulse files are created by a call to euitm_create(). The arguments are:
  - Shot Number
  - Run Number
  - Reference Shot Number
  - Reference run Number.
- Shot and run number are combined together to identify the actual pulse file. Currently the run number is limited to 4 digits, but this restriction will be removed in an incoming version.
- Reference Shot and run numbers identify the reference pulse file. When a Data item is not found in the currently opened pulse file, it is searched by the UAL in the reference pulse file. This allows to reduce the space needed in a sequence of runs, but may slow down data retrieval due to the fact that different pulse files need to be open during data access.

Opening pulse file

- Existing pulse files are open by a call to euitm_open(). The arguments are:
  - Shot Number
  - Run Number.
- Shot and run number are combined together to identify the actual pulse file. Currently the run number is limited to 4 digits, but this restriction will be removed in an incoming version.
- There is no need to specify reference shot and number, since they are stored in the pulse file itself.
- Both euitm_open() and euitm_create() return an integer identifier which is used by the subsequent calls to specify the pulse file. It is therefore possible to deal with more than one open pulse file at a time.
- Pulse files are closed by a call to euitm_close().

Reading CPOs

- Several CPO instances can be stored for each CPO type in the pulse file.
- Routine get() will return an array of as many CPO instances as the number of stored time samples. Its arguments are:
  - File idx: the integer file identifier returned by either euitm_create() or euitm_open().
  - Path name: the name of the CPO type (equilibrium, mhd, …)
  - The CPOinstance array. Recall thet field « time » of each CPO instance will hold the corresponding time sample.
- If we are interested not in the whole CPO array, but only on the CPO instance which is closest to a given time, we can use routine get_slice() with the following arguments:
  - File idx
  - Path name: the name of the CPO type (equilibrium, mhd, …)
  - Returned CPO instance
  - Time value
  - Interpolation mode (3=Interpolation, 1=Closest Sample, 2=Previous Sample).

Writing CPOs

- Routine put() will store an array of CPO instances, each associated with a given time, in the pulse file. The required arguments are:
  - File idx: the integer file identifier returned by either euitm_create() or euitm_open().
  - Path name: the name of the CPO type (equilibrium, mhd, …)
  - The CPOinstance array.
- The array of CPOs will replace previous data. If instead a single CPO instance (a new time slice) has to be appended to an existing CPO array, routine put_slice() has to be used. NB this function is working only for adding a new time slice AFTER the existing time slices of the CPO. Inserting a time slice is not allowed.
- The arguments of put_slice are:
  - File idx
  - Path name: the name of the CPO type (equilibrium, mhd, …)
  - CPO instance
- Observe that there is no need to specify a time argument, since it is already defined in the CPO « time » field.

How to deal with empty fields

- Several CPO fields may be empty
  - during a PUT, the UAL will detect the empty fields and skip them
  - during a GET, the UAL will build the CPO structure leaving fields empty
- For Arrays, an (language-dependent) empty array is used
  - in Fortran 90, the non-emptiness of arrays and strings can be tested by :

  if associated(cpo%variable) then
- For scalar values, it is necessary to adopt a convention:
  - For integer values: EMPTY_INT = -999999999;
  - For floating point values EMPTY_DOUBLE = -9.0E40;   (-9.0D40 in F90)

- In practice, no values are written in the pulse file. These values are returned by the UAL when no data are found in the corresponding field.

Tests

In the directory /afs/efda-itm.eu/project/switm/ual/, we provide several directories to serve as examples: **cppExamples** for the C/C++ codes, **javaTest** for Java programs and **fortranExamples** for the Fortran codes.

C/C++

Under cppExamples, we have:

/afs/efda-itm.eu/project/switm/ual/cppExamples>ls
```
get_cpos          get_cpo_slice.o  put_cpos          put_cpo_slice.o
get_cpos.cpp      get_cpos.o       put_cpos.cpp      put_cpos.o
get_cpo_slice     getting_started  put_cpo_slice
get_cpo_slice.cpp Makefile         put_cpo_slice.cpp
```

For instance, the getcpos.cpp could be used as a sample for C/C++ ITM data access. The main program is (see the file with an editor to display the contents of the various routines):

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

```
/*
This sample program will open an existing pulse file (shot 123, run 1, created by program put_cpos_cxx) and will
read the stored (array of) equilibrium CPOs.
It will then output the content of some fields of the equilibrium CPOs.
*/

//This routine writes an array of pfsystems CPOs in the database, filling some fields of the CPOS
int main(int argc, char *argv[])
{
   //Class Itm is the main class for the UAL. It contains a set of field classes, each corresponding to a CPO defined in the UAL
   //The parameters passed to this creator define the shot and run number. Thesecond pair of arguments defines the reference shot and run
   //and is used when the a new database is created, as in this example.
   //All the UAL classes belong to the ItmNs namespace
   ItmNs::Itm itm(123,1,123,0);

   itm.open(); //Open the database

   //Read the whole array of pfsupplies CPOs
    itm._pfsystemsArray.get();

    //field array of inner class _equilbriumArray is a blitz array containing the array of CPOs
    //(objects of class _equilibrium)
    cout << "Method get() for pfsupplies CPO returned " << itm._pfsystemsArray.array.extent(0) << " CPO instances\n";

    //Print the contents of fields time and pfsupplies.voltage.value
    for(int i = 0; i < itm._pfsystemsArray.array.extent(0); i++)
    {
       cout << "\nCPO " << i << ": time = " << itm._pfsystemsArray.array(i).time << "profiles_1d.F_dia = " << itm._pfsystemsArray.array(i).pfsupplies.voltage.value <<"\n";
    }

    //Dump of the whole CPO array
    cout << itm._pfsystemsArray.array;

    //Dump only the first slice of the CPO array
    cout << itm._pfsystemsArray.array(0);
}
```

By the way, several editors are available (nedit, emacs …).

FORTRAN

The directory “fortranExamples” contains FORTRAN samples and the Makefiles (g95 & Portland compilers).

```
/afs/efda-itm.eu/project/switm/ual/fortranExamples>ls -l
total 12
-rwxr-xr-x  1 imbeaux 333  613 Oct  2 11:27 create_pulse.f90
-rw-r--r--  1 imbeaux 333 3585 Oct  2 11:27 getting_started
-rwxr-xr-x  1 imbeaux 333  655 Oct  2 11:27 make_g95
```

```
-rwxr-xr-x  1 imbeaux 333  648 Oct  2 11:27 make_pgi
-rwxr-xr-x  1 imbeaux 333 1738 Oct  2 11:27 test_equilibrium_get.f90
-rwxr-xr-x  1 imbeaux 333 2911 Oct  2 11:27 test_equilibrium_put.f90
```

Example of a Fortran program test_equilibrium_get.f90:

```
program test
! This program is for practicing the UAL GET command
! euitm_get gets all time slices of a CPO, replacing any previous data for thatCPO
! it can be used both for time-dependent and time-independent CPOs

use euITM_schemas
use euITM_routines
implicit none

integer,parameter :: DP=kind(1.0D0)

type (type_equilibrium),pointer :: cpotest(:) => null()
! For time-independent CPOs :
type (type_vessel) :: cpotest2

real(DP) :: scalans
character(len=132)::stringans,stringans2
integer :: idx, shot, run, status
integer :: numDims,dim1,dim2,dim3
character(len=5)::treename
shot =11 ! Your choice
run = 1  ! Your choice
treename = 'euitm'   ! Mandatory

 write(*,*) 'Open shot in MDS !'
 call euitm_open(treename,shot,run,idx)

write(*,*) 'Reading the results :'
call euitm_get(idx,"equilibrium",cpotest)   ! This does the memory allocation automatically
write(*,*) 'This CPO has ',size(cpotest),' time slices'   ! check the number oftime slices present
in the CPO

! Printout a few variables, just to show how it works
write(*,*) 'datainfo%provider = ',cpotest(1)%datainfo%dataprovider
write(*,*) 'codeparam%codename = ',cpotest(1)%codeparam%codename

write(*,*) 'time at time slice 1 = ',cpotest(1)%time
write(*,*) 'time at time slice 2 = ',cpotest(2)%time
write(*,*) 'li = ',cpotest(:)%global_param%li
write(*,*) 'Profiles_1d%pprime @ time slice 1 = ',cpotest(1)%Profiles_1d%pprime
write(*,*) 'Profiles_1d%pprime @ time slice 2 = ',cpotest(2)%Profiles_1d%pprime

write(*,*) 'Profiles_2d%psi_grid @ time slice 1 = ',cpotest(1)%Profiles_2d%psi_grid
write(*,*) 'Profiles_2d%psi_grid @ time slice 2 = ',cpotest(2)%Profiles_2d%psi_grid

call euitm_deallocate(cpotest) ! For a clean deallocation of the CPO variable
end
```

The corresponding **Makefile** is (using the portland Fortran compiler):

afs/efda-itm.eu/project/switm/ual/fortranExamples>cat make_pgi

```
F77=pgf90
F90=pgf90
CC=gcc
COPTS = -r8  -fPIC -Mnosecond_underscore
LIBS =   -L/afs/efda-itm.eu/project/switm/ual/lib  -lUALFORTRANInterface_pgi
INCLUDES = -I/afs/efda-itm.eu/project/switm/ual/include/amd64_pgi

all: pgi_equilibrium_put pgi_equilibrium_get pgi_create_pulse

pgi_equilibrium_put: test_equilibrium_put.f90
	$(F90) $(COPTS)   -o $@ $^ ${INCLUDES} $(LIBS)

pgi_equilibrium_get: test_equilibrium_get.f90
	$(F90) $(COPTS)   -o $@ $^ ${INCLUDES} $(LIBS)

pgi_create_pulse: create_pulse.f90
	$(F90) $(COPTS)   -o $@ $^ ${INCLUDES} $(LIBS)

clean:
	rm *.o  pgi_equilibrium_put pgi_equilibrium_get pgi_create_pulse
```

For the g95 FORTRAN compiler, the UALFORTRANInterface_pgi must be replaced by UALFORTRANInterface_g95 and the include directory is …/include/amd64_g95 .

The execution requires defining the shared FORTRAN library. To run test, type:

test

If the shared FORTRAN library is not available, update the LD_LIBRARY_PATH variable with:

```
setenv LD_LIBRARY_PATH /afs/efda-itm.eu/project/switm/ual/lib:$LD_LIBRARY_PATH
```

Language mixing, particularly allocatable arrays, is an issue for communicating complex datastructure between several programs. The UAL server is in charge of this data transfer and you could check the cross-compatibility with programs written in different languages.

Performance issues

As shown in the previous example, the data storage could be in the ITM database or in memory depending on the “enabling memory” instruction. Benchmarks give the following values for the data access:

CPO Size	1K	10K	1M
File			
Memory			

We estimate roughly the performance speed-up with a C program writing 40000 time slices of a CPO (poloïdal field coil in our benchmark). It takes about 3s to write and read them using the parallel file system (pfs) and less than 1s with the in-memory version. Notice: pfs is

already very fast compare to the other file system (pfs is around 800MB/s, afs is roughly 5MB/s).

Visualization

Various visualization tools will be available to monitor the data during a simulation or to plot them as a post-processing tool. The visualization library is based on VTK which has been installed on the gateway.
To play with VTK, you could type (see Figure 4):

cd /afs/efda-itm.eu/project/switm/vtk/VTK/Examples/Tutorial/Step5/Java>
java Cone5

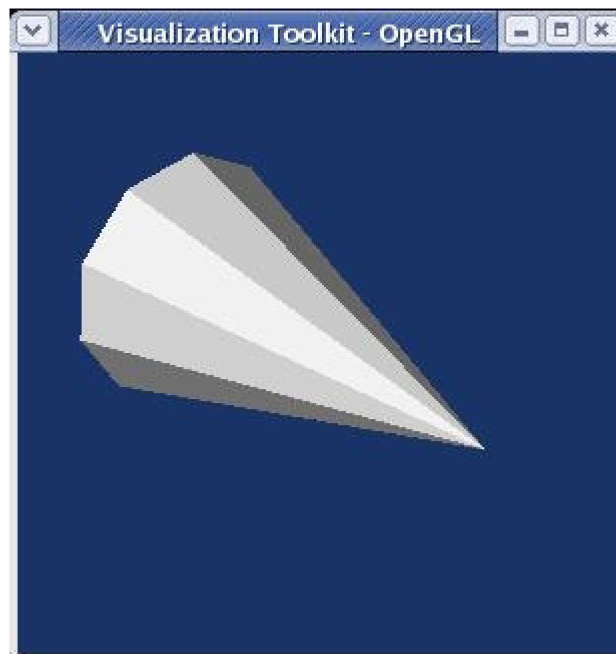


Figure 4: VTK example

Post-processing tools

Several post-processing tools are available on the gateway like Scilab, Visit, Matlab, IDL. Notice: only open source package are promoted by the ITM but several commercial tools are widely used in the fusion community, so they are provided on the gateway but for a very limited number of users, typically two.

Scilab

To use Scilab, simply type:

scilab

In the scilab window, you could define a vector by x=[1,2,3] and plot it (see Figure 5).

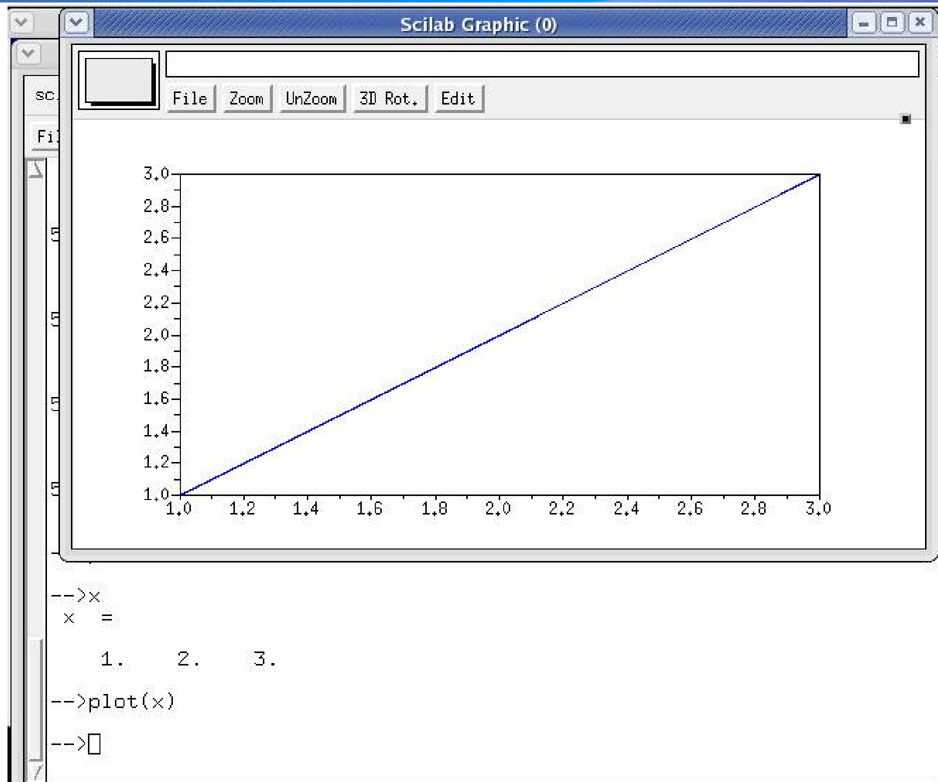


Figure 5: simple Scilab session

Notice: Scilab is currently unable to access the UAL data. This work is in progress.

IDL

IDL is available on the gateway using the “idl” command (see Fig 6). No access to the UAL is provided with IDL. This access is foreseen for later.

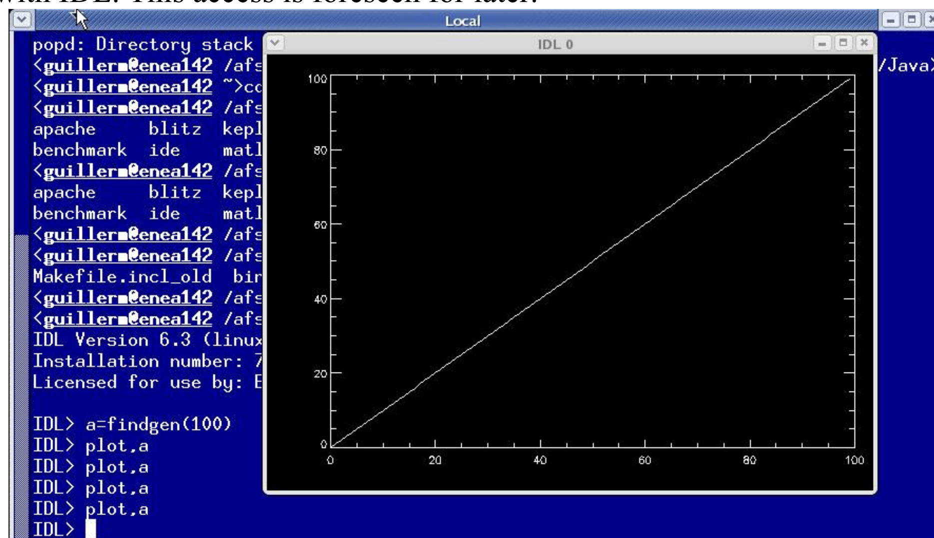


Figure 6: simple IDL session

Matlab

Matlab could used on the gateway with the mentioned restrictions previously. The UAL data access is available for Matlab and is installed at /afs/efda-itm.eu/project/switm/matlabitm/4.06d together with several examples.

To use it, you must load the Java-UAL library and add the path to the m functions :

```
>> javaaddpath('/afs/efda-itm.eu/project/switm/matlabitm/4.06d/ualjava.j
>> ar')
>> addpath('/afs/efda-itm.eu/project/switm/matlabitm/4.06d')
```

Example :

```
function [out,t] = test_get_ip(shot, run)
import ualmemory.javainterface.*
expIdx = openBD('euitm', shot, run);
cpo=CPO_get(expIdx, 'magdiag');
for i=1:size(cpo(:))
 out(i)=cpo(i).ip.value;
 t(i)=cpo(i).time;
end
```

Use in Matlab (see figure 7):

```
>> [y, t]=test_get_ip(58094,1);
>> plot(t,y)
```

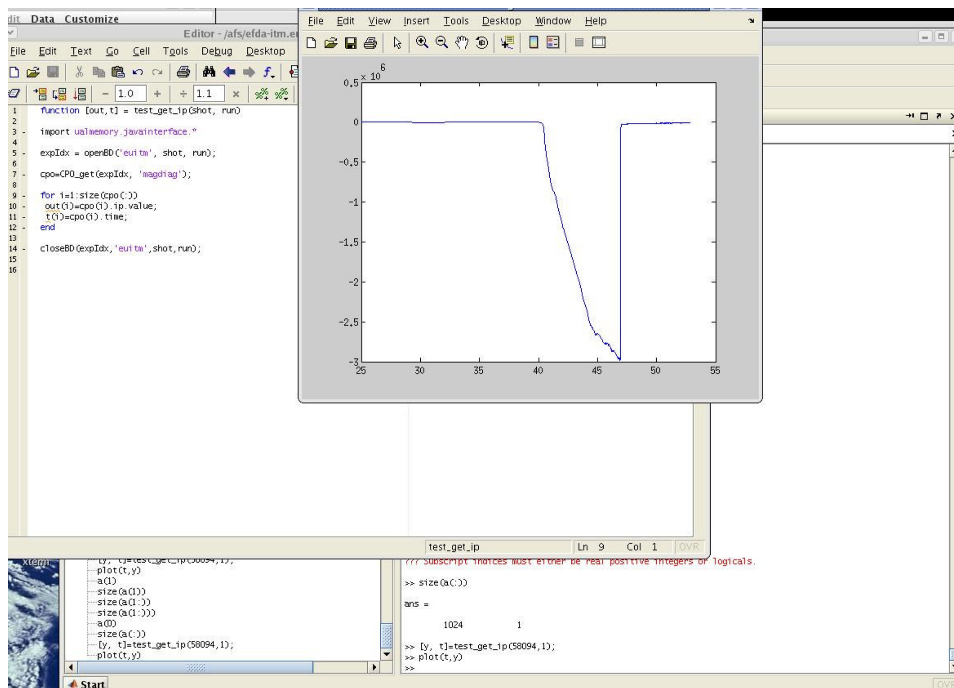


Figure 7: simple Matlab session

Codes

Before any workflow composition, the user must run separately the involved codes. Several codes will be used to demonstrate the UAL access. The 1st is SOLOVIEV which fills the "equilibrium" CPO (Consistent Physical Object).
You could get it from the /afs/efda-itm.eu/project/imp1/rep/soloviev directory (Note: temporary unavailable):

cp /afs/efda-itm.eu/project/imp1/rep/soloviev/*.f90 .

Compile it with a command line or use the Makefile which is provided in this directory.

The execution output yields:

```
./soloviev
 Open shot in MDS !
  cpsurf :  0.11111112
  radius :  0.
  BM     :  0.93417233
  alfa   :  1.471974
  delta  :  0.18448305
  R0, RM :  2.6180341 2.8025172
  psib   :  0.6793598
  P0     :  133961.27
  Q0     :  1.2
 nchi= 31
 Put full Equilibrium CPO
Remaining memory: 800 bytes allocated at line 181 of soloviev.f90
Remaining memory: 6944 bytes allocated at line 39 of soloviev.f90
Remaining memory: 8568 bytes allocated at line 125 of soloviev.f90
Remaining memory: 248 bytes allocated at line 156 of soloviev.f90
Remaining memory: 408 bytes allocated at line 126 of soloviev.f90
Remaining memory: 168 bytes allocated at line 127 of soloviev.f90
Remaining memory: 800 bytes allocated at line 180 of soloviev.f90
Remaining memory: 248 bytes allocated at line 157 of soloviev.f90
Remaining memory: 800 bytes allocated at line 178 of soloviev.f90
Remaining memory: 800 bytes allocated at line 179 of soloviev.f90
Remaining memory: 800 bytes allocated at line 182 of soloviev.f90
```

The pulse 11 has been updated.

KEPLER

Summary:

- Use the standard version of KEPLER
- Install your private version
- Update your code to the ITM structure (subroutine with CPOs as input or output)
- Develop new Fortran actors and integrate them in KEPLER with fc2k
- Develop your own workflow and enjoy the life

Kepler is used to compose a workflow and run it. This framework must be available together with the additional tools to integrate easily ITM codes.

Documentation

The official KEPLER web site contains many tutorials and user's guide:

http://www.kepler-project.org/
http://users.sdsc.edu/~altintas/KeplerTutorial/
For the Ptolemy II part, consult: http://ptolemy.eecs.berkeley.edu/publications/

Installation

The shared version of KEPLER is already installed on the "gateway" computer. Adding new actors in the shared version is restricted to some users. You could use this version to compose your workflow but it is better to have your own copy of KEPLER if you are willing to add new actors (your code for instance): see the paragraph named "Installing your private version of KEPLER".

Running an example

It's better to play with examples to master KEPLER before the integration of new actors (codes or anything else). To launch KEPLER (see Fig 8), you must type:

kepler

Note: several environment variables must be set in order to launch KEPLER and FC2K without errors. Copy locally ITMv1 (from /afs/efda-itm.eu/project/switm/scripts) and amend it for your own usage (KEPLER location could be private and "set_itm_data_env" must point possibly on your own data).

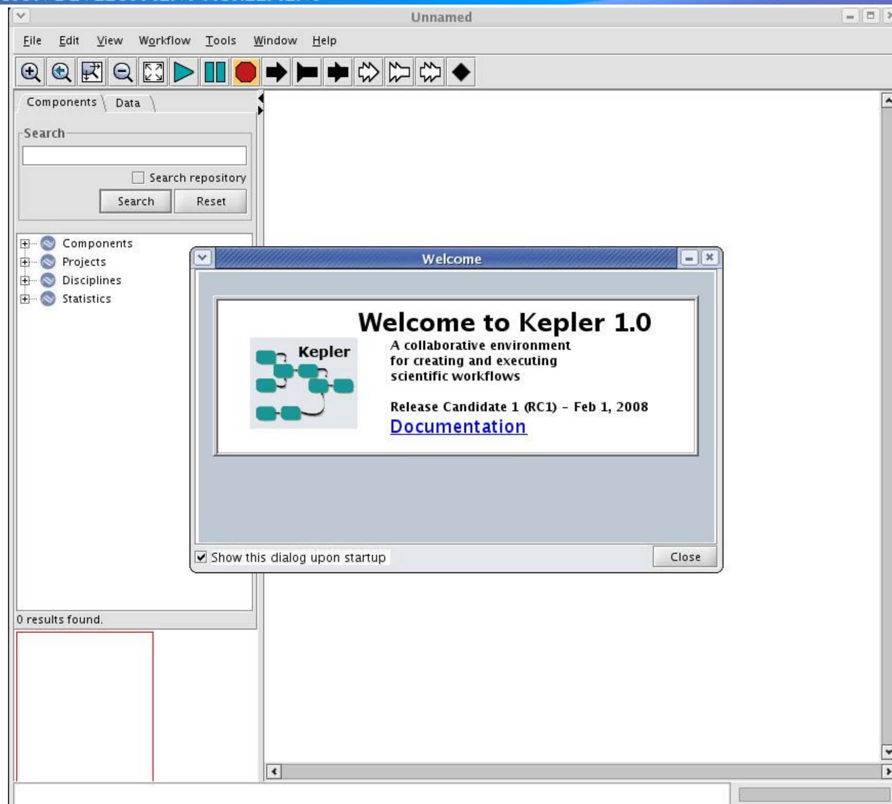


Figure 8: Kepler initial GUI

Several demos are available using the File menu and choosing demos under "open file":

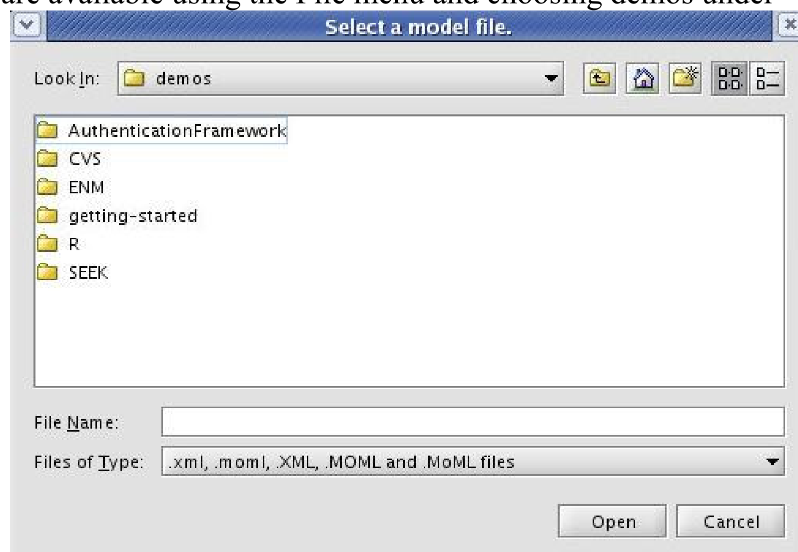
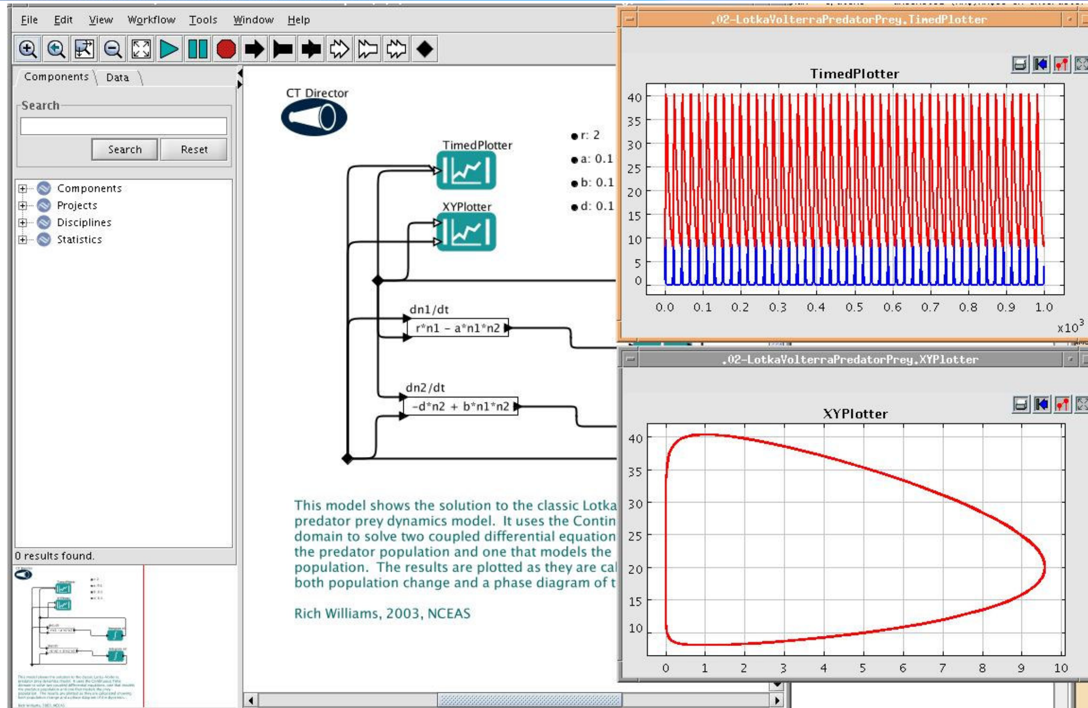


Figure 9: selecting an example in the "demos" directory

Under demos, choose "getting-started" (see Fig 9) and one example ("LotkaVolterraPredatorPrey.xml" for example). After running, the screen looks like:



Installing your private version of KEPLER

To have your own copy of KEPLER, you must follow the 2 following steps:

1. copy KEPLER (and Ptolemy) in your own directory
2. set the environment variables

Copy KEPLER

Type the commands:

cd "your directory" (where you want to install Kepler)

Note: we will create automatically 2 directories named kepler

tar xvf /afs/efda-itm.eu/project/switm/kepler/4.06d/kepler.tar

Setting the environment variables

setenv KEPLER "your directory"/kepler

Instead of setting these variables for each session and window, you have better to update your private copy of ITMv1 (see below in red)

Example: (use it with "**source …**")

```
#!/bin/tcsh
#
# -----------------------
# ITM framework v1: ITMv1
# -----------------------
# Author: B.Guillerminet
# Date:  17 October 2008
#
# --------------------
# variables for KEPLER
# --------------------
#
# standard variables
```

```
# setenv PTII /afs/efda-itm.eu/project/switm/ptolemy/ptII7.0
# setenv KEPLER /afs/efda-itm.eu/project/switm/kepler
#
# setenv KEPLER /afs/efda-itm.eu/project/switm/kepler/4.06d
setenv KEPLER $HOME/itm_v1.0/kepler
#
# overwrite the "kepler" command
alias kepler 'pushd $KEPLER;ant run-dev;popd'
#
# update the LD_LIBRARY_PATH for the UAL shared libraries
# must be updated later with the private actors directory: ACTORS
setenv LD_LIBRARY_PATH $KEPLER/lib:$LD_LIBRARY_PATH
#
# ----------------------------------
# variables for DATA (UAL & storage)
# ----------------------------------
# change test to the machine name
source /afs/efda-itm.eu/project/switm/ual/set_itm_data_env $USER test 4.06d
source /afs/efda-itm.eu/project/switm/ual/set_itm_env 4.06d
setenv runs_path $euitm_path
#
# ------------------
# variables for FC2K
# (use KEPLER, PTII,
# UAL, FC2K and later
# ACTORS)
# ------------------
#
setenv FC2K /afs/efda-itm.eu/project/switm/ihm/fc2k/KeplerJniActorGen_v4
# update the PATH to access the GUI tools (FC2K)
set path = ( $FC2K/bin  $path )
#
# -----------------
# variables for ISE
# -----------------
#
alias ise 'pushd /afs/efda-itm.eu/project/switm/ise/ISEv1.0.3;./ise.sh;popd'
#
```

Integration of a local code

A graphical tool (GUI) has been developed for the integration of codes written in C/C++ or FORTRAN. The code must be changed in a subroutine and integrated in a shared (*.so) or absolute (*.a) library.

- To adapt a code to run in Kepler, it is first necessary to identify which CPOs it needs as input and output.
- The structure of the CPOs can be found at the ITM-TF website: http://www.efda-taskforce-itm.org/. Follow the link to ISIP’s data structure page.

A simple example will show you how to transform your FORTRAN code in an actor which could be included in a workflow. Let us start with “cpo2ip.f90” program (see the FORTRAN paragraph above) which get the “equilibrium cpo” and write an integer array. The part which

must be implemented in the workflow, is the computation part ("**subroutine cpo2ip**"). The "equilibrium" input & array output are managed by the tool.

So, here are the steps to follow:

1. create a directory, for instance "**mkdir myactor**"
2. create a file (**cpo2ip.f90** for instance) in this directory which contains only the computation part. See below the contents of cpo2ip.f90:
3. create a Makefile which build a library ("**libcpo2ip.a**" for instance)
4. **make**
5. integrate your code in KEPLER using the FC2K tool: type **fc2k**
6. integrate it in a workflow using the KEPLER design tool (see above)

<u>Example</u>:

cpo2ip.f90:

```
subroutine cpo2ip(equi_in, ip)
!-----------------------------------------------------------------------
  use euitm_schemas
  use euITM_routines
  implicit none
  integer,parameter :: DP=kind(1.0D0)

  type (type_equilibrium),pointer :: equi_in(:)
  integer :: ip(20)
  integer :: i

  write(*,*) 'time deb',equi_in(:)%time,size(equi_in)
! --------------------------------------------
! print a few existing fields
! --------------------------------------------
  write(*,*) 'size(dataprovider): ', size(equi_in(1)%datainfo%dataprovider)
  if (size(equi_in(1)%datainfo%dataprovider) /= 0) then
    write(*,*) 'dataprovider: ',equi_in(1)%datainfo%dataprovider(1)
  endif
  do i=1,20
    ip(i) = 2*i
  enddo
  write(*,*) 'ip:',ip
  return
end subroutine cpo2ip
```

Makefile:

```
F90=pgf90
COPTS= -r8  -Mnosecond_underscore -fPIC
LIBS= -L/afs/efda-itm.eu/project/switm/ual/lib -lUALFORTRANInterface_pgi
INCLUDES= -I/afs/efda-itm.eu/project/switm/ual/include/amd64_pgi

all: libcpo2ip.a
libcpo2ip.a:cpo2ip.f90
	$(F90) $(COPTS) -c cpo2ip.f90 ${INCLUDES} $(LIBS)
	ar -rv libcpo2ip.a cpo2ip.o
clean:
```

rm *.a *.o

Make your library (libcpo2ip.a) before to launch FC2K. To launch the graphical tool to integrate the code in KEPLER, type:

fc2k (meaning Fortran & C to Kepler)

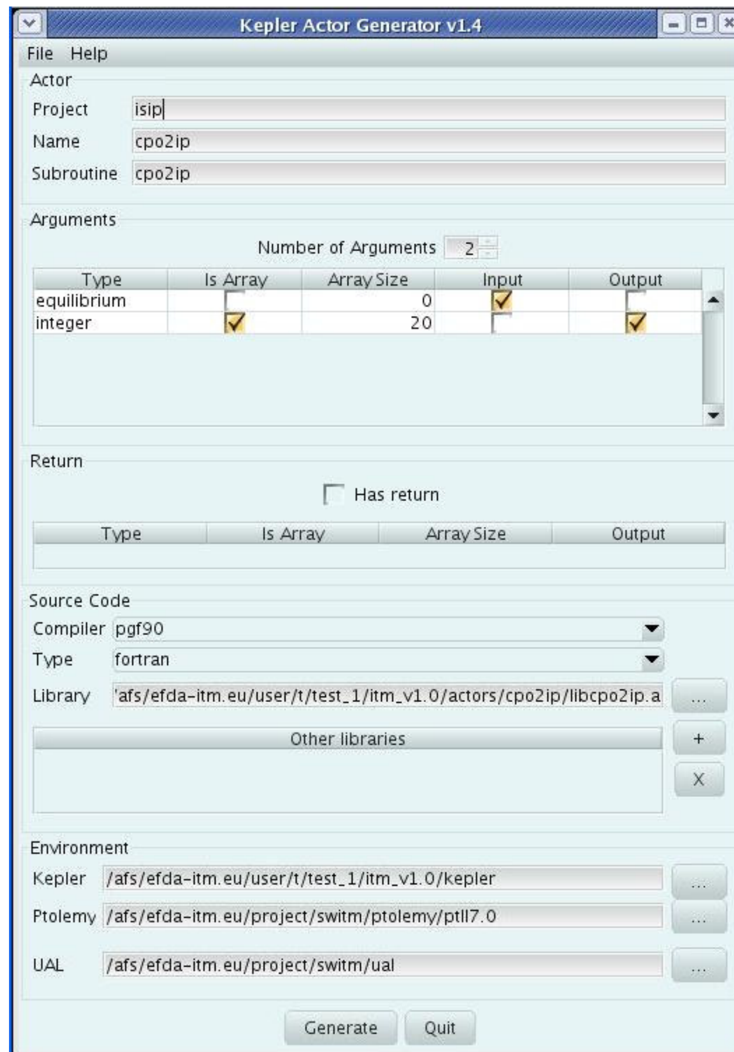


Figure 10: Graphical tool for Fortran/C code insertion

In this form (Figure 10), you must fill:

- **Project**: this name is free and it is used to classify the actors. We recommend using IMP1, 2 … 5 for the various codes and ISIP for the general usage actors (used here).
- **Name**: It will appear as the actor name in the workflow. It's free but it is better to use the code name (cpo2ip in Fig 9)
- **Routine name**: name of your Fortran/C routine. Here we have "cpo2ip"
- Specify the arguments. Here we have 2 arguments, one Equilibrium CPO in input and one integer array of 20 values in output
- **Compiler**: use pgf90 for the Fortran codes (or g95) and gcc for C and C++
- **Library**: define the shared (*.so) or absolute (*.a) library where is your code. In our example, we specify the path and libtest1itm.a for the library.

Other libraries: add the libraries which are required for the linkage (*.so or *.a). The UAL library does not to be specified here since it is already defined in LD_LIBRARY_PATH variable.

- **Kepler:** specify the path of KEPLER (your private version or the standard version)
- **Ptolemy:** specify the path to your private version of Ptolemy (ptII7.0) if you have one.
- **UAL:** specify the path of the UAL location. The default value is the standard UAL location.
- The "Generate" button allows to create the actor and to include it in the Kepler catalogue.

The description could be saved in a XML file for further uses ("Saved as" in the File menu). You could check your new actor by launching KEPLER (see below in Fig 10).

Integration of a remote code (Web Service)

Will be described later

…

Building a workflow

The generated actors could be easily used in a workflow. In KEPLER, use the search option with the project name for instance IMP1 or ITM. All the actors under the IMP1 or ITM will be displayed. Using the previously generated actors, we search "cpo" (see Fig 11):

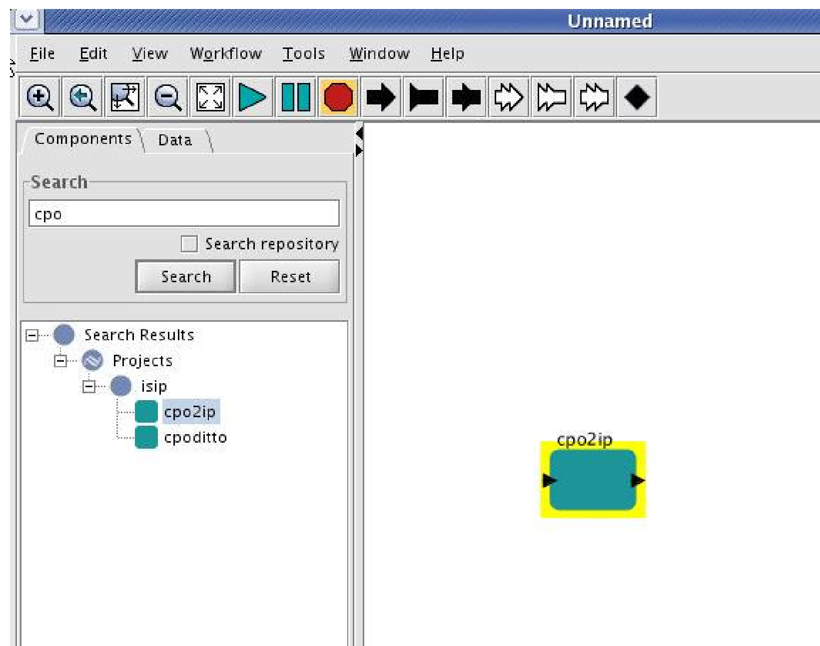


Figure 11: search and copy "cpo2ip"

To add your code to a workflow, you have simply drag and drop it to the main screens and connect the input and out ports. For "cpo2ip", 1 input ("equilibrium" CPO) and 1 output ("equilibrium CPO") ports have been defined and you must connect them to appropriate actors.

Two additional actors are mandatory for the design of a workflow, **UALinit** and **UALcollector**. UALinit is in charge to retrieve the parameters from a database and to store them in memory during the simulation. UALcollector write the data of the simulation in the database.

The workflow with UALinit and UALcollector is shown in the Fig 12.

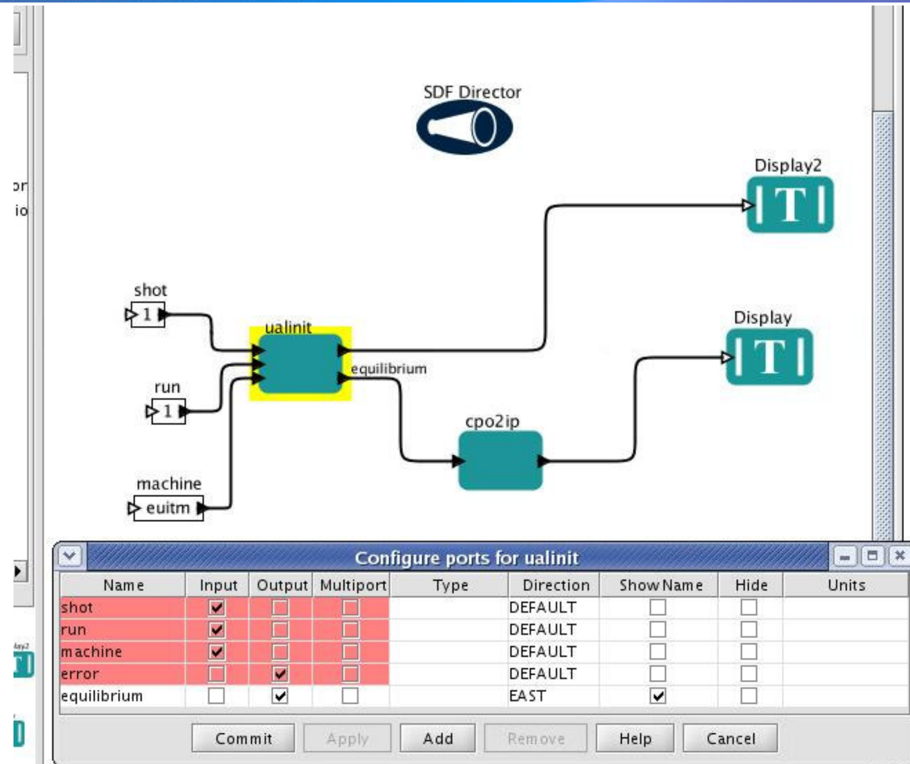


Figure 12: Configuring UALinit

In UALinit, we must add an output port ("equilibrium") which must be linked to "cpo2ip" (see Fig 12).

The required input parameters are set up by adding "constant" actors (see Fig 13).

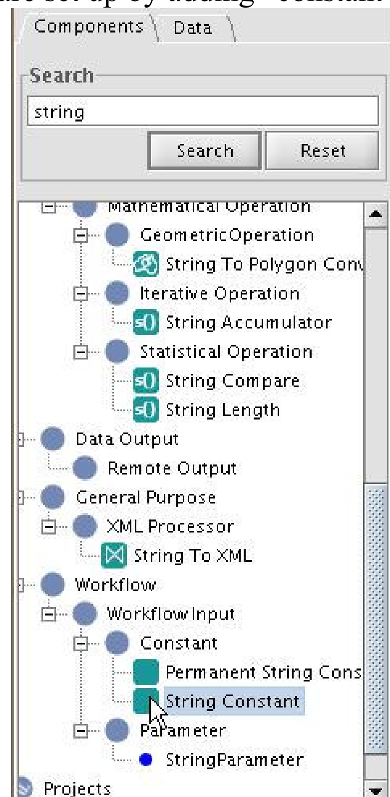


Figure 13: Constant actors (input to UALinit: machine name)

A “display” actor is used to visualize the simulation output.

Double clicking on any actor, allows to define its parameters (for instance, you can enter the pulse number and the others arguments). You must then “commit” (fig. 14).

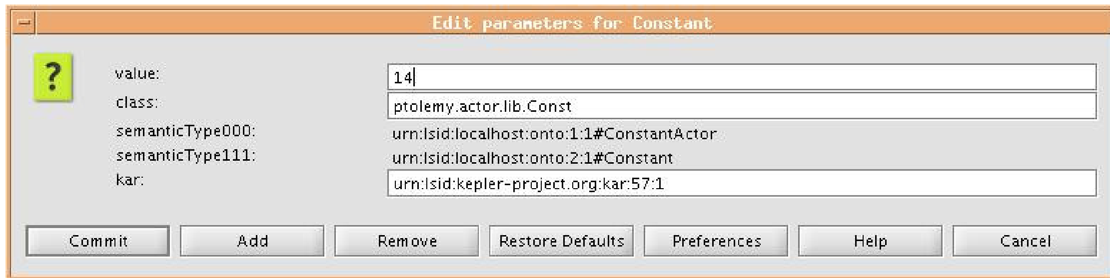


Figure 14: Changing the constant value

A director must be added to conduct the workflow. Use the simple “SDF” director and change the number of iterations to 1, then commit (see the workflow for the cpo2ip program in Fig 15).

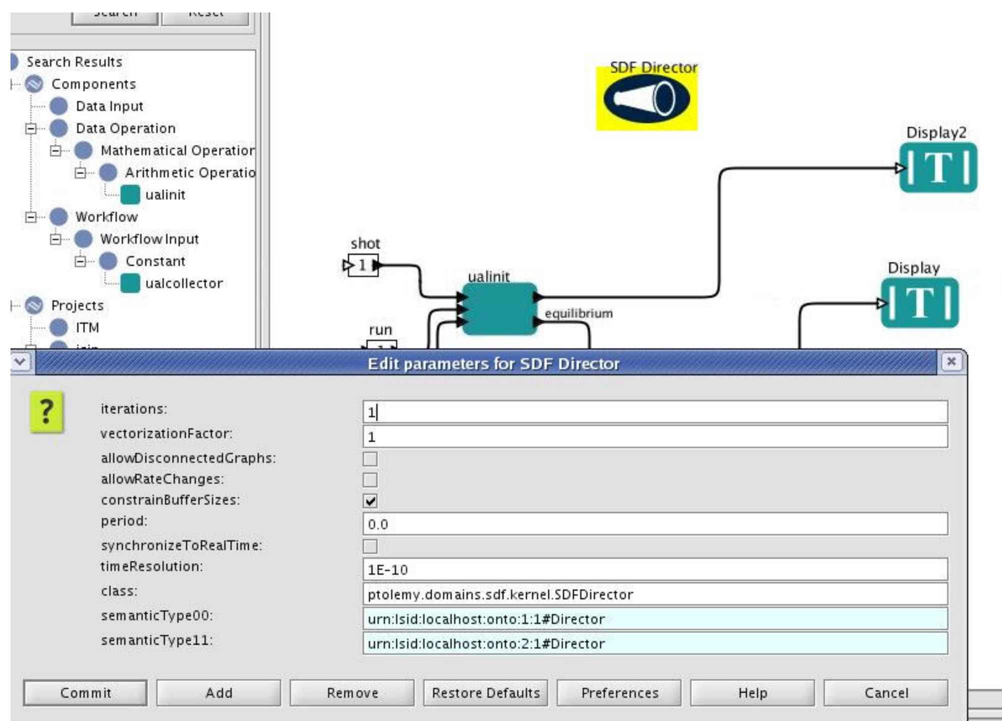


Figure 15: Comprehensive workflow for the “cpo2ip” actor

Great! You could RUN the workflow by pushing the “RUN” button (see Fig 16). It will launch KEPLER and display the outputs during the simulation.

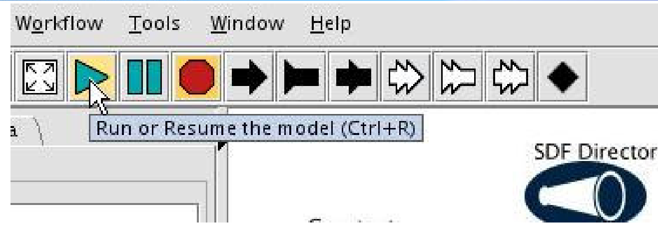


Figure 16: Button to start the simulation

We could use this kind of actor to plot data as in the following example (see figure 17):

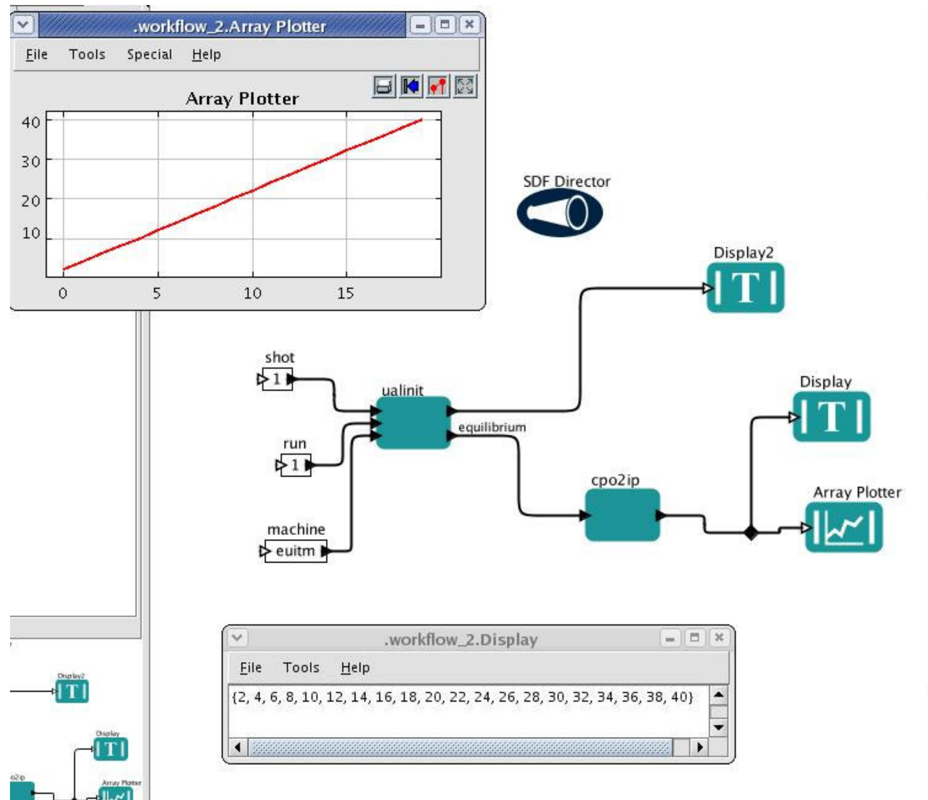


Figure 17: Cpo2Ip used to extract and plot internal data

ISE: ITM Simulation Editor

Summary:

- launch ISE and look at your favourite shot/pulse
- display a few values
- define a workflow and run it from ISE

The ISE version 1.0.3 has been released on 14th October.
ISE is used to set up the parameters, run & monitor a simulation.

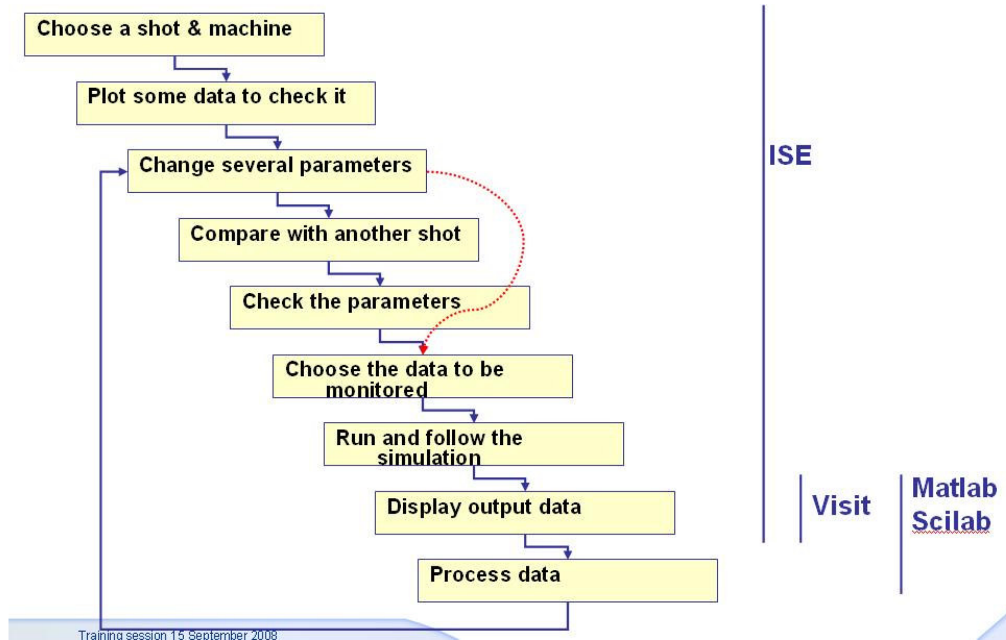


Figure 18: Use of ISE in a simulation

You can launch ISE by typing:

ise

ISE has several area, one which displays the datastructure, one for setting/editing the parameters and the last for monitoring/overview of the simulation (see figure 19):

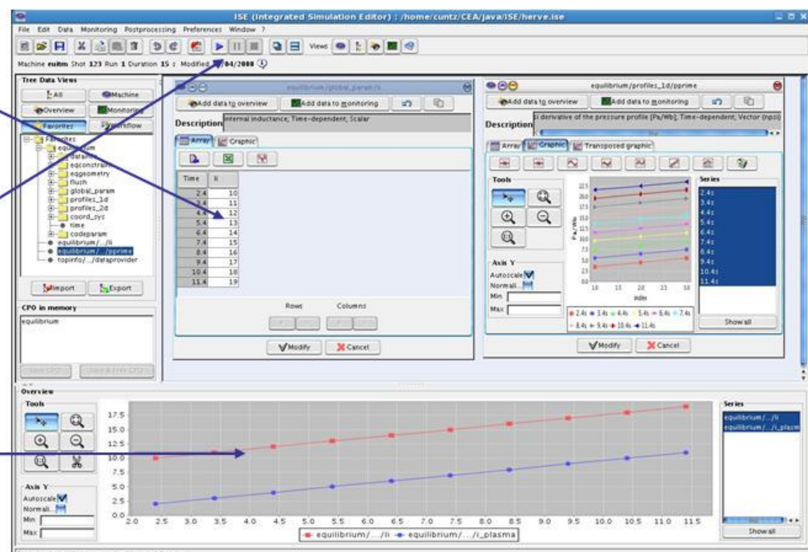


Figure 19: Overview of ISE

The 1st thing is to create a new study: define the machine name, shot and run number. It uses your $UAL environment variables to find and open the data:

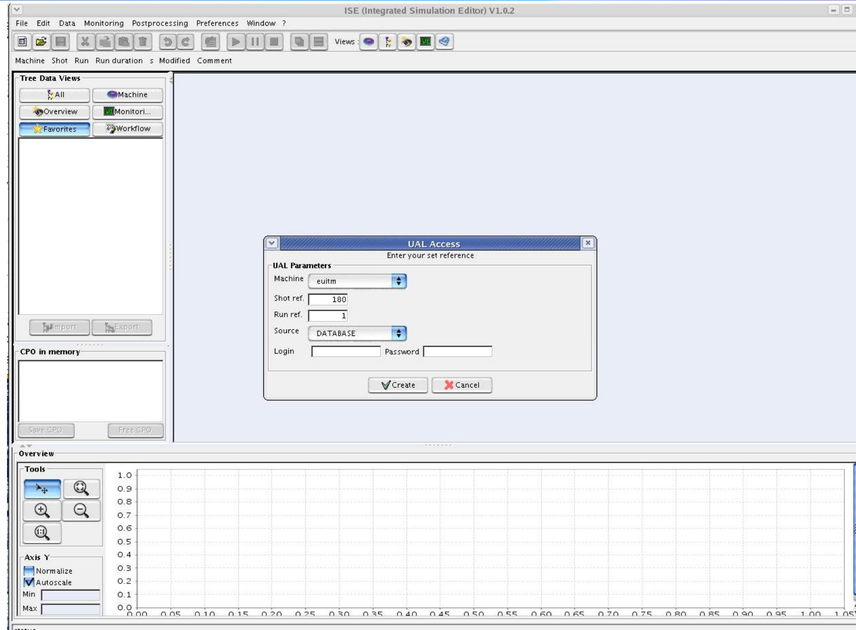


Figure 20: Creating a study

Then, it will show the data structure (see figure 21):

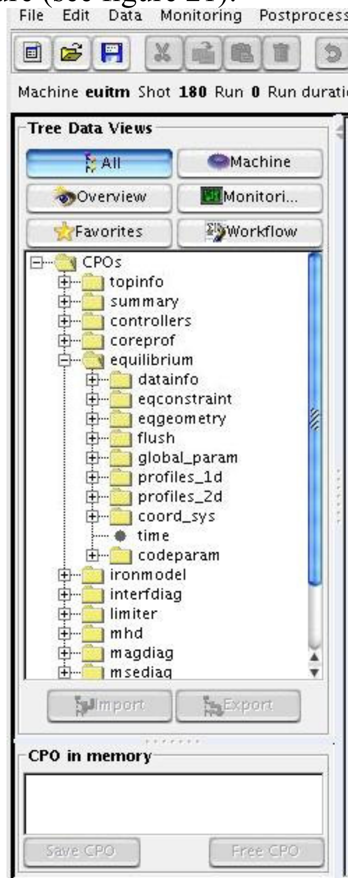


Figure 21: Datastructure

EFDA Task Force
Integrated Tokamak Modelling
EUROPEAN FUSION DEVELOPMENT AGREEMENT

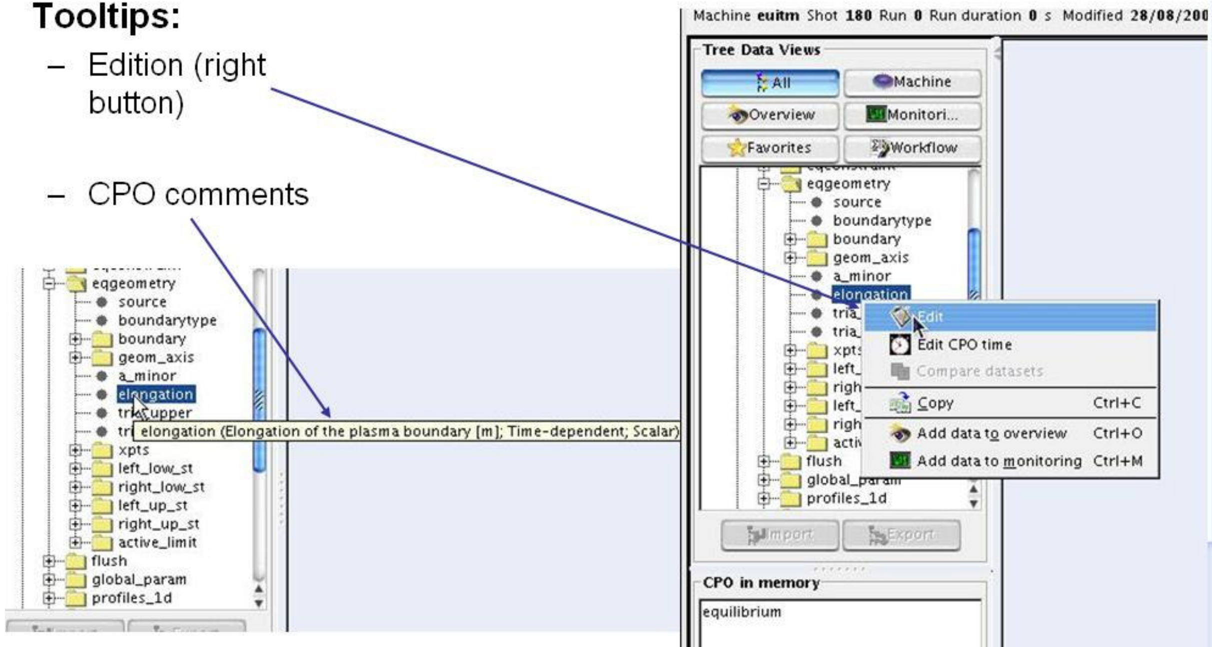


Figure 22: Parameter edition

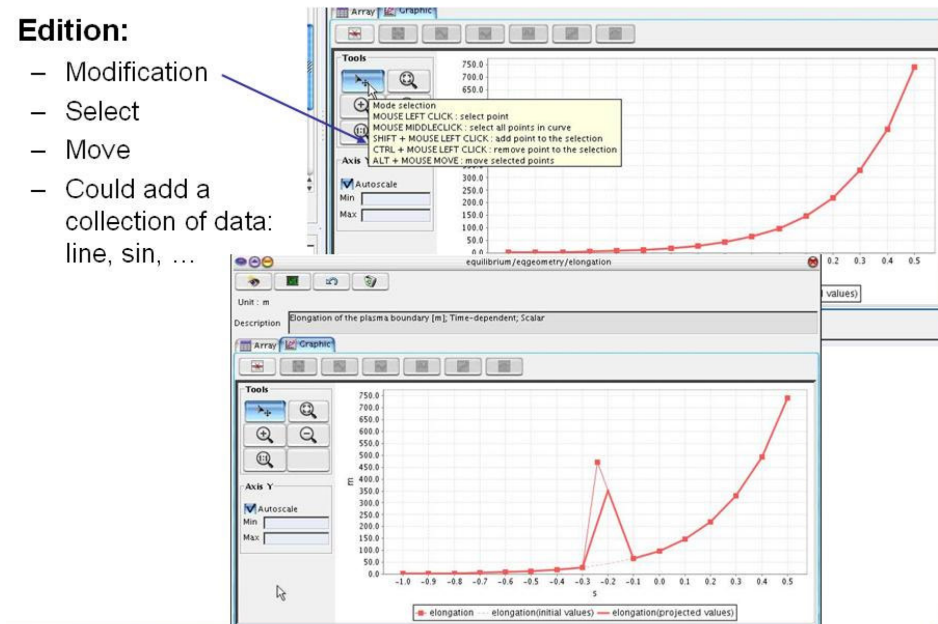


Figure 23: Graphical parameter edition

EFDA Task Force
Integrated Tokamak Modelling
EUROPEAN FUSION DEVELOPMENT AGREEMENT

Selection

Edition (switch to expert mode)

– Will be addressed in the workflow + code developer sessions

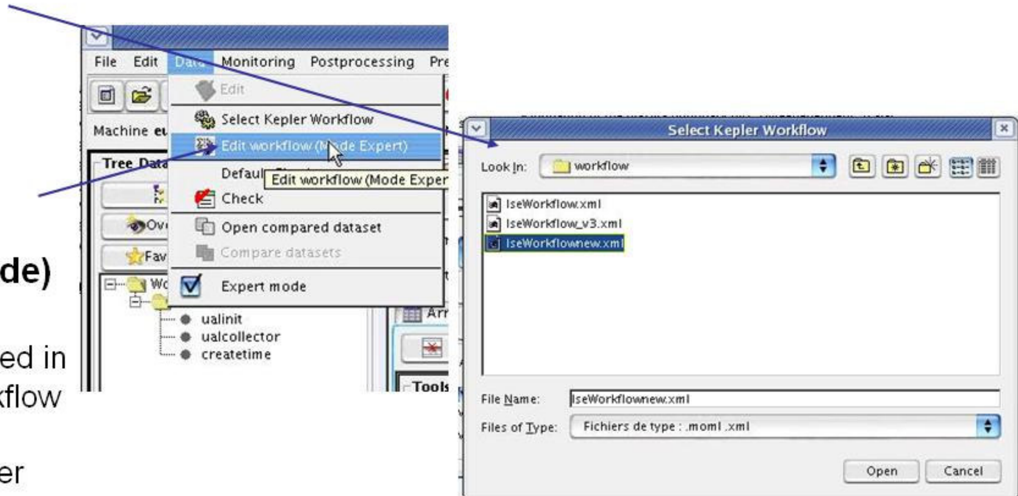


Figure 24: Workflow edition

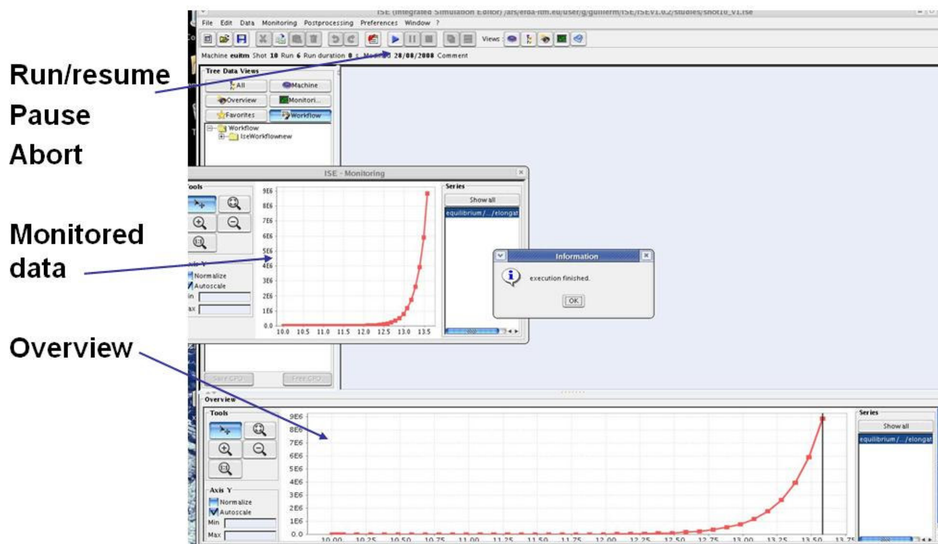


Figure 25: running a simulation

Portal

It is available since a few days at http://portal.efda-itm.eu/portal

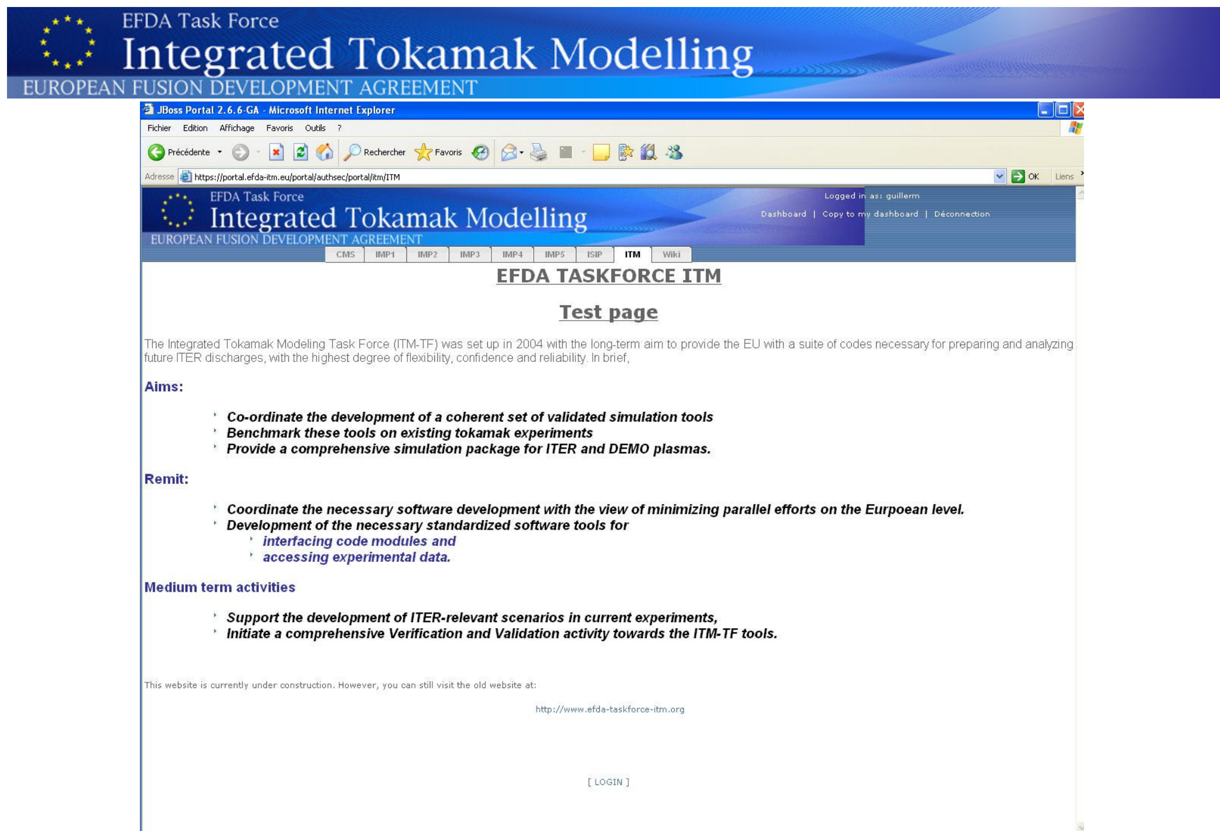


Figure 26: ITM portal