UAL TUTORIAL

DATE	COMMENT
01/10/2008	Creation by F. Imbeaux
09/2010	Current revision

1. Introduction

Warning: The knowledge of the UAL commands is not required for using the ITM framework with Kepler.
In Kepler, physics modules are wrapped into actors and the wrapping layer transparently deals with the UAL commands.
Therefore this UAL tutorial is intended only for people who want to test their ITM compliant programs OUTSIDE Kepler.

2. Create your private database

To practice the UAL, you need first to create a private database on your account.
This private database is created for a given machine (tokamak name) and data structure version, by the following command:

/afs/efda-itm.eu/project/switm/scripts/create_user_itm_dir *TokamakName DataVersion*

Example: (creates a tree for tokamak name test, allowed for testing purposes)
/afs/efda-itm.eu/project/switm/scripts/create_user_itm_dir test 4.08b

The database is created under **~my_username/public/itmdb/itm_trees**

3. Specify the working environment

Now you need to specify on which database the UAL is going to work. Use the following command:
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 public *TokamakName DataVersion*

Example: set the environment variables to work to the previously created directory:
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 public test 4.08b

3.1. Working with the public database

The public database is located at **/pfs/itmdb/itm_trees/public/**

To work with the public database, after the previous command, use:
source /afs/efda-itm.eu/project/switm/scripts/set_itm_data_env public *TokamakName DataVersion*

Example: set the environement variables to work with the public database, a tokamak named test and the 4.08b release of the UAL :
source /afs/efda-itm.eu/project/switm/scripts/set_itm_data_env public test 4.08b

Note: the public database is only readable

3.2. Working with another user's database

To read another user's database, after specifying the UAL release to use, type the following command:
source /afs/efda-itm.eu/project/switm/scripts/set_itm_data_env username *TokamakName DataVersion*

Example: to work with the database of the user named john, a tokamak named test, and the 4.08b release of the UAL:
source /afs/efda-itm.eu/project/switm/scripts/set_itm_data_env john test 4.08b

Note: the other users' databases are read only

4. Create a program

4.1. FORTRAN

We will now practice the Fortran UAL to
1) create a new (shot, run),
2) populate it with some data
3) read the data

Copy the tutorial directory to your account with the following command :

cp -r $UAL/fortranExamples .
cd fortranExamples

Let's create now a new entry in the private database just created. An entry is defined by the four items (username, machine, shot, run). Username and machine have been specified already by the previous command. We shall now compile and execute a Fortran script that creates an entry for a given shot and run (instruction euitm_create of the UAL) : create_pulse.f90

The Makefile has been configured to produce two sets of executables, one for PGI compiler, the other for G95.
To build the executables just run the command:

make

This compiles the test routines and links them to the UAL libraries
You can now try to run the create pulse function:

pgi_create_pulse

or

g95_create_pulse

This function simply creates a new entry with shot = 11 and run = 1. If you look to your database directory, you will find the new entry (3 new files with name euitm_110001)

ls ~/public/itmdb/itm_trees/test/4.08b/mdsplus/0

or

ls $MDSPLUS_TREE_BASE_0

Now we are going to populate this entry with some dummy data. Have a look to test_equilibrium_put, and execute it :

pgi_equilibrium_put

or

g95_equilibrium_put

In order to check what we have done, we can use test_equilibrium_get.f90: this program simply opens the data entry,
reads the CPO just put in, and writes to the screen the data just read:

pgi_equilibrium_get

or

g95_equilibrium_get

We have now seen the main UAL functions: create, open, get, put

A typical Kepler actor will do the following actions:
euitm_open
euitm_get (some CPOs)
do some physics calculations using the input CPOs just get and produce some output CPOs

euitm_put (the produced CPOs)

This schematic view of the Kepler actor shows that the physics calculations are in fact independent of the UAL and should not use any UAL commands. Those are simply wrapping the physics code. Tools are provided to create these wrapping layer automatically.

4.2. C++

We will now practice the C++ UAL to

1) create a new (shot, run),
2) populate it with some data
3) read the data

Create directory cppExamples in your home directory and copy there the tutorial directory with the following command:

cp -r /afs/efda-itm.eu/project/switm/ual/cppExamples cppExamples
cd cppExamples

Build the executable for the examples with the command

make

At this point you will have the following programs:

put_cpos
put_cpo_slice

get_cpos
get_cpo_slice

put_cpos creates a pulse file with shot number 123 and run number 1. If you look to your database directory, you will find the new entry (3
new files with name euitm_1230001) giving the command

ls ~/public/itmdb/itm_trees/test/4.08b/mdsplus/0

or

ls $MDSPLUS_TREE_BASE_0

The program then writes an array of equilibrium and pfsystems.
You can read the stored values with program **get_cpos.**

put_cpo_slice does a similar thing of **put_cpos** (i.e. writes an array of 200 pfsystems CPO whose associate time ranges from 0 to 199), but it does it by appending a single CPO instances at a time. You can read the content of one of the saved pfsystems CPOs with program **get_cpo_slice** which takes as argument a time. This program will select the CPO slice corresponding to the passed time (possibily performing a kinear interpolation if the time does not correspond to any saved time).