



# EFDA

EUROPEAN FUSION DEVELOPMENT AGREEMENT

Task Force  
INTEGRATED TOKAMAK MODELLING

*ITM Training Session, March 2012*  
*IPP Garching*

## Creating an actor for a C(++) code

H.-J. Klingshirn

TF Leader : G. Falchetto,  
Deputies: R. Coelho, D. Coster

EFDA CSU Contact Person: D. Kalupin

This session is for everybody who wants to adapt a C(++) code for the ITM platform (“turn it into an actor”)

### What will be shown:

- How to turn a simple C code into an ITM-compatible subroutine
- How to turn this subroutine into a Kepler actor
- How to include this Kepler actor into a workflow

You can find these slides at  
[~klingshi/public/c\\_training/c-actor.pdf](https://github.com/klingshi/public/c_training/c-actor.pdf)  
(and soon on the documentation website)

- There is a C++ UAL interface to read and write CPOs
- The CPOs are implemented as classes
  - BLITZ++ library used for handling arrays

<http://www.oonumerics.org/blitz/docs/blitz.html>

- Header files: \$UAL/cppinterface
- Documentation: see UAL user guide at

[http://www.efda-itm.eu/ITM/imports/isip/public/isip\\_UAL\\_User\\_Guide.pdf](http://www.efda-itm.eu/ITM/imports/isip/public/isip_UAL_User_Guide.pdf)

- Example sources: \$UAL/cppExamples
  - Read/write (time-dependent) CPOs

- Documentation how to turn your C++ code into a Kepler actor: see  
[http://www.efda-itm.eu/ITM/html/isip\\_fc2k\\_cpp.html](http://www.efda-itm.eu/ITM/html/isip_fc2k_cpp.html)
- The procedure for building an actor is the same for a C and a C++ code
  - the difference is that for the C code you have to add a C++ wrapper

- There is no dedicated UAL interface for C
  - every C code needs a C++ part to handle I/O with CPOs
- ...but no worries, because C and C++ have by design perfect interoperability
  - you just have to be aware of some simple conventions

- Make sure your environment is set to data version 4.09a:

```
echo $DATAVERSION
```

should give “4.09a”

- Copy example:

```
cp -r ~klingshi/public/c_training $HOME/public  
cd $HOME/public/c_training/cexample
```

- Select compiler:

```
setenv OBJECTCODE linux.gnu_gw
```

# C Example Code: outline

`main()` **C++**  
 (`src/mycode_driver.cpp`)

calls

- Driver: program for running the code standalone (outside Kepler) – good for debugging
- Reads/writes CPOs from/to UAL, passes them to the actor routine

▶ `mycode_wrapper_function(<CPO objects>)`  
 (`src/mycode_wrapper.cpp, .h`)

**C++**

calls

- Main routine of actor
- “Wrapper”: copies data from CPOs into code-specific data structures
- Passes data to routine that does the actual work

`mycode_wrapper_function(<structs>)`  
 (`src/mycode_functions.c, .h`)

**C**

- Performs the actual computation (“your code”)
- Returns output data in struct
- Code-specific structs defined in `mycode_functions.h`

**ACTOR**

(From src/mycode\_wrapper.cpp)

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

Input + Output CPOs

```
void mycode_wrapper_function(ItmNs::Itm::limiter& lim,  
                             ItmNs::Itm::edge& edge)
```

```
{  
    MycodeInputData input;  
    MycodeOutputData output;
```

```
    // transfer data from CPOs to input...
```

```
    // call actual code
```

```
    mycode_function( &input, &output );
```



Call to C routine

```
    // transfer data from output to CPOs...
```

```
}
```



(From `src/mycode_functions.h`)

```
#ifndef MYCODE_FUNCTIONS_H

#define MYCODE_FUNCTIONS_H

//... some code omitted here ...

/* exported functions */

#ifdef __cplusplus
extern "C" {
#endif
    void mycode_function( MycodeInputData* input,
                        MycodeOutputData* output );
#ifdef __cplusplus
}
#endif

#endif
```

- C routines have to be declared extern "C" for the C++ compiler
- C++ standard defines the `__cplusplus` preprocessor directive to indicate compilation by a C++ compiler
- Use this to make definitions in C header files compatible with C++

# C Example: compiling and running standalone version

- To compile, run:

```
cd $HOME/public/c_training/cexample  
make
```

This will run the commands specified in `Makefile` to

- Compile `mycode_functions.c` with the C compiler
- compiles `mycode_wrapper.cpp` with the C++ compiler
- zip the resulting object files (\*.o) into the library archive file `libmycode.a`
- It also builds the standalone program `mycode_driver`  
→ **Linking the program has to be done with the C++ compiler!**
- To run the standalone program:  

```
./mycode_driver
```

## C Example: building the actor

- Run FC2K:

```
fc2k
```

- Fill in the required information in FC2K:

- Open the provided configuration file: File → Open →  
`$HOME/public/c_training/cexample/  
MyCode-fc2k-configuration.linux.gnu_gw.xml`
- Set the Kepler path: `$HOME/kepler`
- On tab “Source”, select the library:  
`$HOME/public/c_training/cexample/libmycode.a`

- Click “Generate”

- This will compile the actor and install it in your Kepler environment

# C Example: building the actor

Kepler Actor Generator V4.4e <2>

File Help

Actor

Project: IMP3

Name: MyCode

Subroutine: mycode\_wrapper\_function


Argument | HasReturn | HasParameters | Source

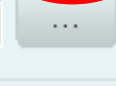
Arguments

Type	Single Slice	Is Array	Array Size	Input	Output	Label
limiter	<input type="checkbox"/>	<input type="checkbox"/>	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Limiter CPO Input
edge	<input checked="" type="checkbox"/>	<input type="checkbox"/>	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Edge CPO Output

2 # args

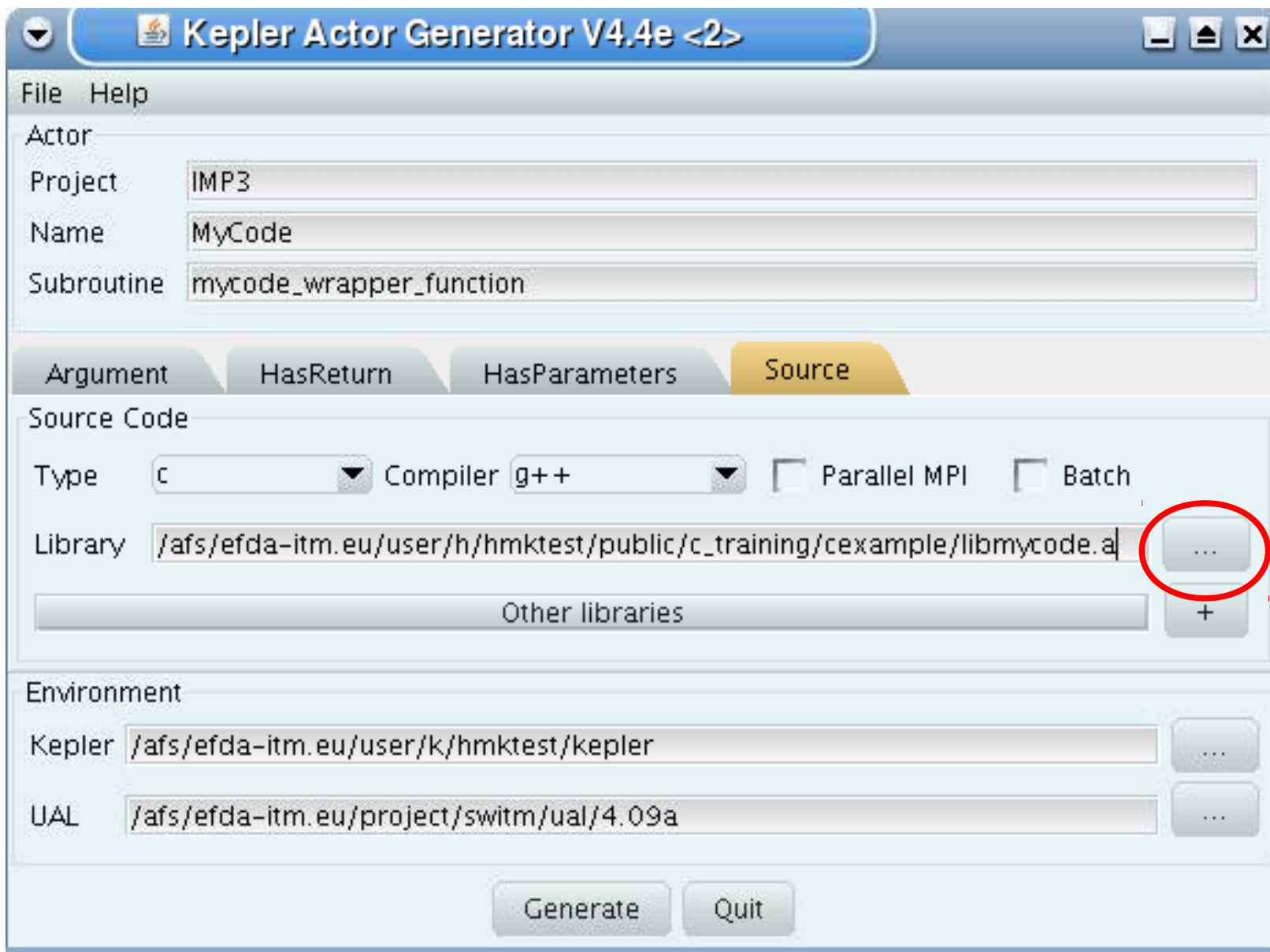
Environment

Kepler: /afs/efda-itm.eu/user/h/hmktest/kepler 

UAL: /afs/efda-itm.eu/project/switm/ual/4.09a 

Generate Quit

# C Example: building the actor

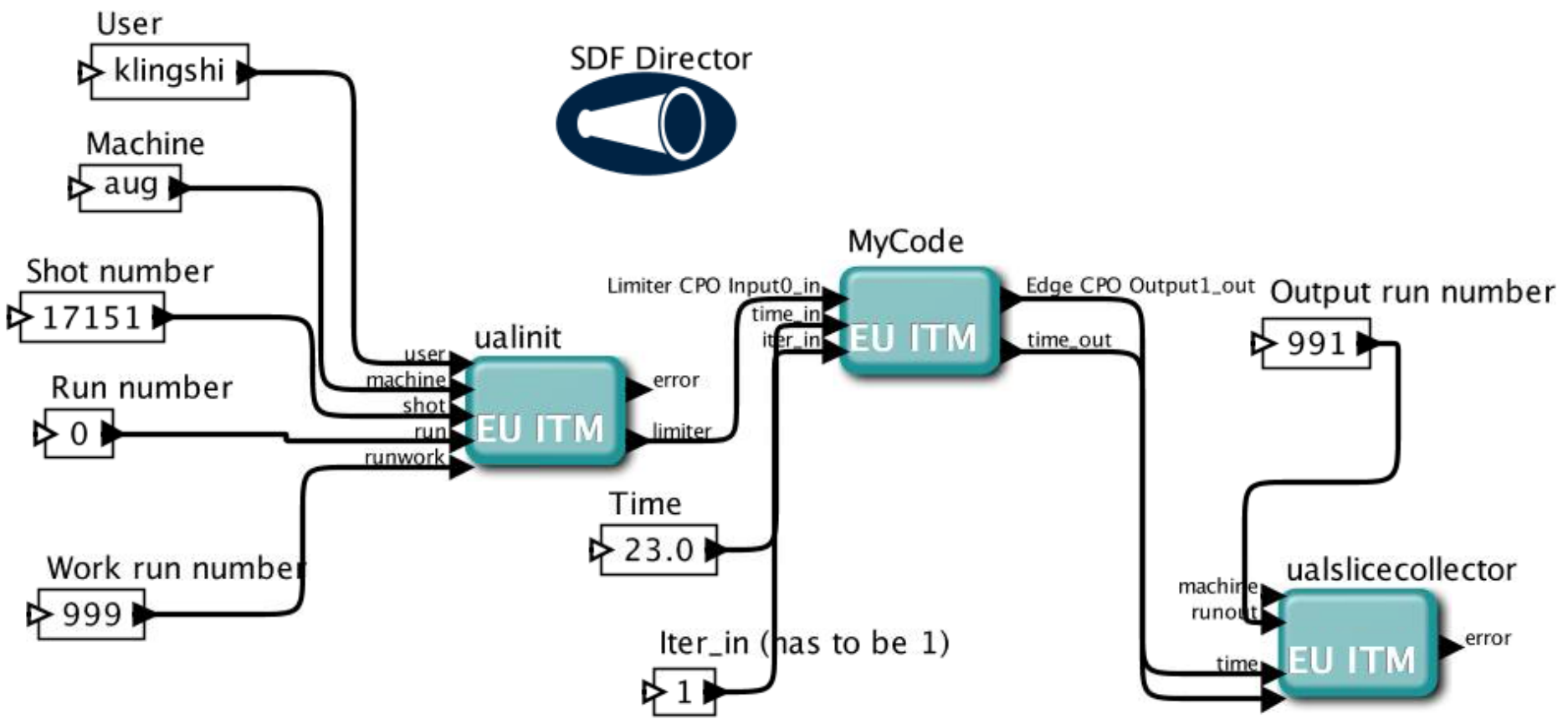


# Try the new actor in a workflow

- Run Kepler
- Open example workflow:  
`$HOME/public/c_training/cexample/MyCode-Kepler-workflow.xml`
- Click “Play” - the workflow should run
- After it finished executing, you should have output CPOS.

Check for files:

```
ls $HOME/public/itmdb/itm_trees/$TOKAMAKNAME/  
$DATAVERSION/mdsplus/0/euitm_17151*
```



# Development cycle: updating the actor

- Every time you change your code you have to rebuild your actor
  - run FC2K, load configuration, click “Generate”
- You can also do this automatically by running

```
fc2k MyCode-fc2k-configuration.$OBJECTCODE.xml
```
- In the example:

```
make update-kepler-actor
```

(have a look at the Makefile for an example how to set this up)



```
$HOME/public/c_training/cppexample/mycppfunction.cpp
```

```
void mycppfunction(  
    ItmNs::Itm::summary & sum,  
    ItmNs::Itm::antennas & ant,  
    ItmNs::Itm::equilibriumArray & eq,  
    int & x,  
    ItmNs::Itm::limiter & lim,  
    ItmNs::Itm::coreimpur & cor,  
    ItmNs::Itm::ironmodelArray & iron,  
    double * y,  
    char * str,  
    param & codeparam)
```

Described in detail at:

[http://www.efda-itm.eu/ITM/html/isip\\_fc2k\\_cpp.html](http://www.efda-itm.eu/ITM/html/isip_fc2k_cpp.html)

- Building the actor:  
`cd $HOME/public/c_training/cppexample`  
`setenv OBJECTCODE linux.gnu_gw`  
`make`  
`fc2k` → set up fields as before
- This example also shows how to use code parameters  
More details on code parameters at:

[http://www.efda-itm.eu/ITM/html/itm\\_code\\_parameters.html](http://www.efda-itm.eu/ITM/html/itm_code_parameters.html)