Training - Garching 09.2011

Navigate space

Accessing gateway from Mac OS X Lion

Mac OS X Lion users have to switch from NX Client to NX Player in order to access gateway.
You can get NX Player here: link

1. Kepler installation at Gateway (Garching 09.2011)

Kepler installation at Gateway

There are three Kepler sessions that will be conducted during Code Camp.

- Kepler basics - this session will cover basics of Kepler
- integration of ITM tools within Kepler - this part will cover more sophisticated material (it will require additional tools as well)
- Grid/HPC jobs submission - tutorials will cover the usage of workflows for submission of jobs into various distributed computing environments.

Each of these parts require Kepler 4.09a installation. Additionally, ITM related training requires 4.09a database and additional tools: actors, Fortran codes, C++ codes. In order to install Kepler you have to follow the instructions.

All-in-one installation

You can run an installer that will prepare everything for all sessions. This is a recommended solution even if you do not plan to participate in all of them. In order to use the all-in-one installer, please execute the following script:

```
~zokt/public/tutorials/installer.sh
```

An add-on installer

If you took part in the basic Kepler part installation time, then you only need to install an add-on containing additional data for ITM tools and GRID+HPC sessions. In this case, please execute the following script:

```
~zokt/public/tutorials/addon.sh
```

Running instructions

Please remember that during the tutorial you should **always** run in an interactive session. Before running Kepler, please execute:

```
itmgo
```

You should also load an ITM initialization script:

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
```

2. Tutorial - Introduction to Kepler (Garching 09.2011)

This page is devoted to Kepler basic training - Garching 09.2011

Navigate space

- 1. Tutorial - Introduction to Kepler - Basics (Garching 09.2011)
- 2. Tutorial - Introduction to Kepler - Loops (Garching 09.2011)
- 3. Tutorial - Introduction to Kepler - Python (Garching 09.2011)

Go back to Training page

1. Tutorial - Introduction to Kepler - Basics (Garching 09.2011)

Introduction to Kepler - Basics

Table of Contents

1. Introduction

This tutorial is designed to introduce the concept of building ITM tools based workflows within Kepler.

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.

NX connection to the Gateway

This tutorial assumes that Gateway accounts will be used for starting up Kepler application.
If you are not familiar with NX setup for the Gateway, take a look at following location NX setup

2. Requirements for the tutorial

Backing up Kepler home directory

Before you proceed with installation of the Kepler application be sure to make a backup of your Kepler home directory

```
mv ~/.kepler ~/.kepler_09_2011
mv ~/kepler ~/kepler_09_2011
mv ~/serpens ~/serpens_09_2011
```

2.1 Using ITM Kepler installation at Gateway

In order to make Kepler installation for the tutorial faster we will use preinstalled version of the Kepler that is available for Gateway users.

In order to install Kepler and ITM example workflow you have to follow instructions at following page:

Kepler installation

1. Kepler installation at Gateway (Garching 09.2011)

After you follow all the installation steps, you should see Kepler loading.

Starting Kepler

No matter which way have you used to install Kepler, make sure to export some variables before you start Kepler again.

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

3. Executing simple workflows

In order to execute workflow, you have to load workflow XML file into Kepler. During this tutorial session we will use following workflows:

- "Hello world" - it is installed as: $HOME/serpens/demo-ITM-09.2011/workflow/basic/Hello_World.xml
- "Hello world with debug" - it is installed as: $HOME/serpens/demo-ITM-09.2011/workflow/basic/Hello_World_Debug.xml
- "Simple Actors.xml" - it is installed as: $HOME/serpens/demo-ITM-09.2011/workflow/basic/simple_actors.xml
- "Input port selection" - it is installed as: $HOME/serpens/demo-ITM-09.2011/workflow/basic/input_selector.xml
- "Output port selection" - it is installed as: $HOME/serpens/demo-ITM-09.2011/workflow/basic/output_selector.xml
- "Relations" - it is installed as: $HOME/serpens/demo-ITM-09.2011/workflow/basic/relation.xml
- "If-else-simple" - it is installed as: $HOME/serpens/demo-ITM-09.2011/workflow/basic/if_else_simple.xml
- "If-else-simple-expression" - it is installed as: $HOME/serpens/demo-ITM-09.2011/workflow/basic/if_else_simple_expression.xml

3.1 Hello world workflow

3.1.1 Using existing "Hello world" workflow

After this exercise you will:

- know how to start Kepler
- know how to load simple workflow
- know how to execute workflow
- know how to animate workflow

Exercise no. 1 (approx. 10 min)

Film available: http://www.youtube.com/watch?v=1xsPH6Mnzx0
In this exercise you will execute simple Kepler workflow. In order to this follow the instructions:

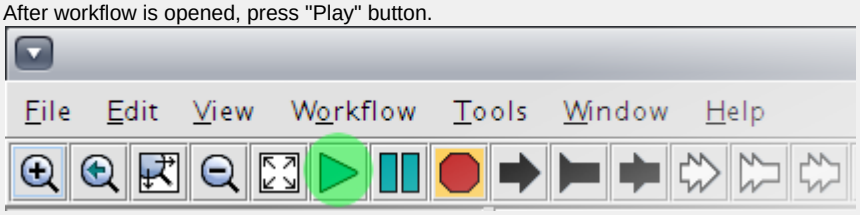
1. Start Kepler application by issuing:

```
kepler
```

2. Open "Hello world" workflow by issuing: File -> Open and navigate to:

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/Hello_World.xml
```

3. After workflow is opened, press "Play" button.



Workflow should generate output within Display actor

> **Animating workflows**
>
> In Kepler it is possible to animate workflows during execution. In order to animate workflow you have to turn on animations. You can do this by choosing: ***Tools -> Animate at Runtime...***
> Demo movie for this feature can be found at following location: animation

3.1.2 Using existing "Hello world - with debug" workflow

After this exercise you will:

- know how to start Kepler
- know how to load simple workflow
- know how to execute workflow
- know how to listen to the actor

Exercise no. 2 (approx. 10 min)

Film available: http://www.youtube.com/watch?v=EVGSXC4kcks
In this exercise you will execute simple Kepler workflow with Debug information. In order to this follow the instructions:

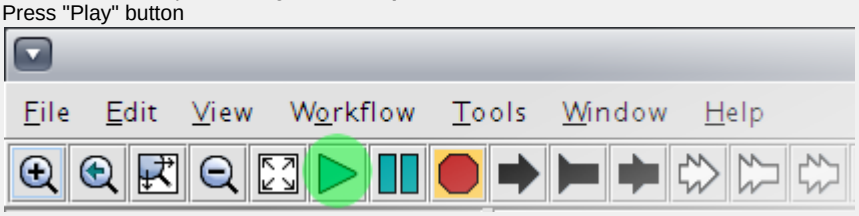
1. Start Kepler application by issuing:

```
kepler
```

2. Open "Hello world debug" workflow by issuing: File -> Open and navigate to:

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/Hello_World_Debug.xml
```

3. After workflow is opened, "Right-click" **Expression** actor and choose "Listen to actor"
4. Press "Play" button



Workflow should generate output within Display actor and should print debug information generated by Expression actor

3.1.3 Building "Hello world" from the scratch

After this exercise you will:

- know how to start Kepler
- know how to build simple workflow
- know how to connect elements
- know how to add elements to the workflow
- know how to search for the actors within Kepler's library

Exercise no. 3 (approx. 15)

Film available: http://www.youtube.com/watch?v=DXXYnuDjnWw

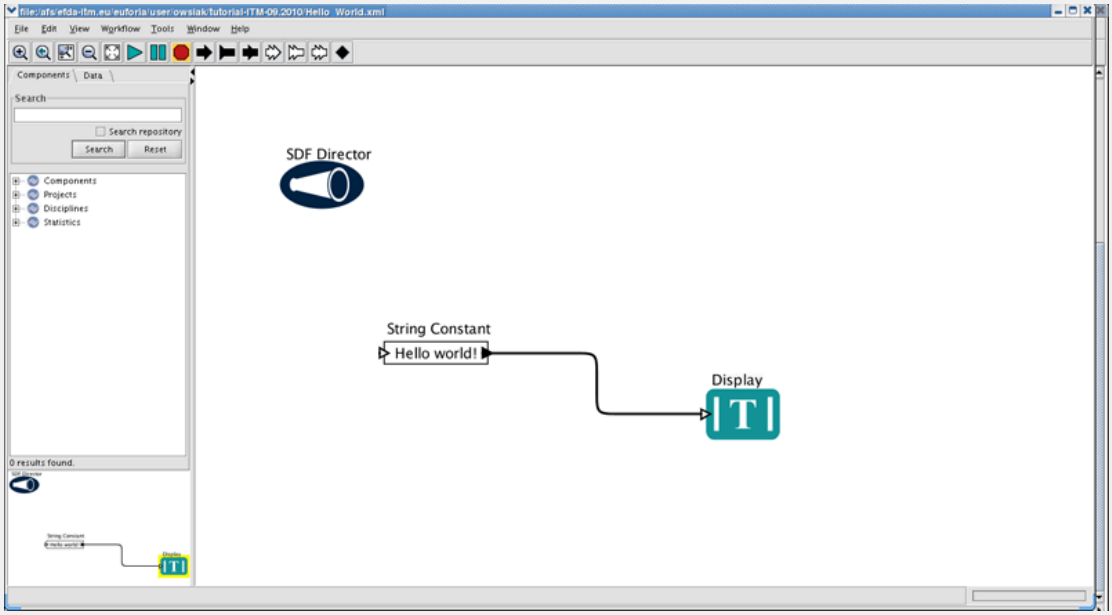
In this exercise you will build simple "Hello World" workflow and execute it. In order to get this task done, follow the instructions:

1. Start Kepler application by issuing:

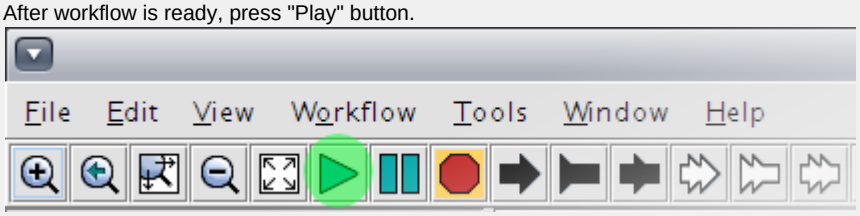
```
kepler
```

2. Type in "SDF" into "Search" field and press "Search" button
3. Drag and Drop ***SDF director*** into workflow
4. Right-click on ***SDF director*** and choose "Configure Director"
5. Set number of iterations to "1"
6. Type in "String" into "Search" field and press "Search" button
7. Add "String Constant" actor to the workflow
8. Right-click "String Constant" actor and choose "Configure Actor"
9. Type "Hello world!" into "value" field
10. Commit changes
11. Type in "Display" into "Search" field and press "Search" button
12. Add "Display" actor into workflow
13. Connect "String Constant" actor with "Display" actor

Intermediate results



14. After workflow is ready, press "Play" button.



Workflow should generate output within Display actor

3.2 Basic actors, explained - String Constant, Constant, Expression

After this exercise you will:

- know the difference between Constant and String Constant
- know how to use Expression actor

Exercise no. 4 (approx. 20 minutes)

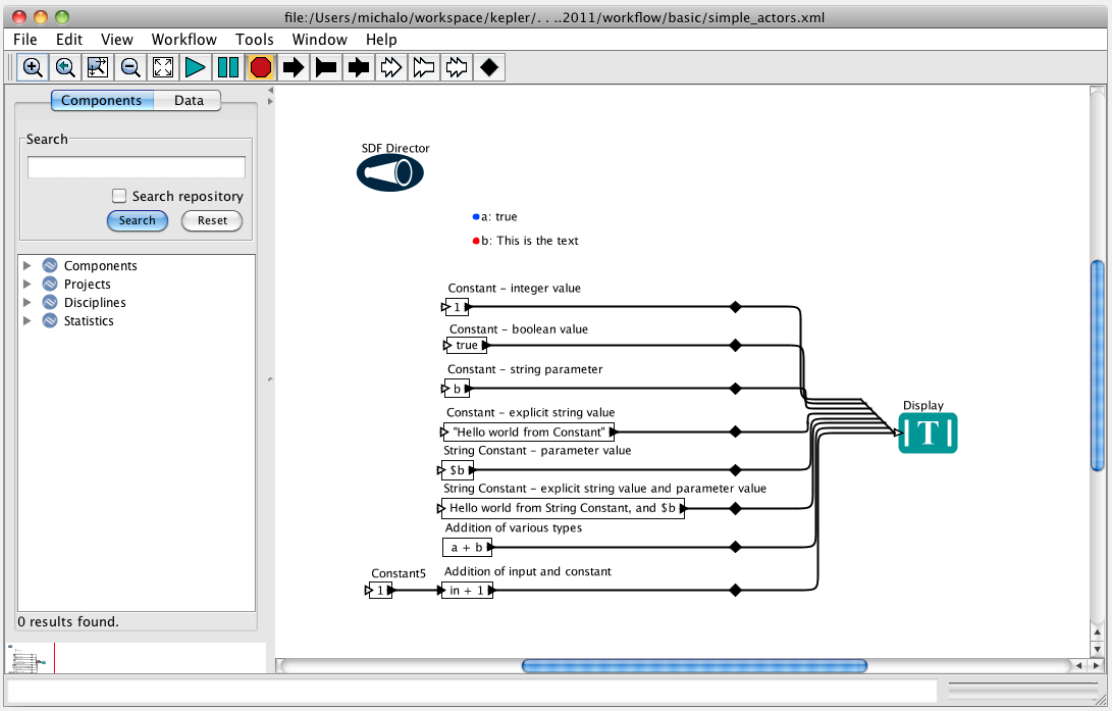
1. Start Kepler application by issuing:

```
kepler
```

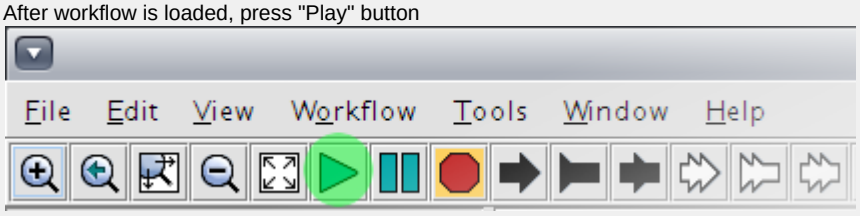
2. Load example workflow from following location

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/simple_actors.xml
```

3. You should see workflow similar to one below



4. After workflow is loaded, press "Play" button



3.3 Using DDF Boolean Select and Select in order to determine input for processing

After this exercise you will:

- know how to determine which input data should be processed
- know how to choose between DDF Boolean Select and Select actor

Exercise no. 5 (approx. 20 minutes)

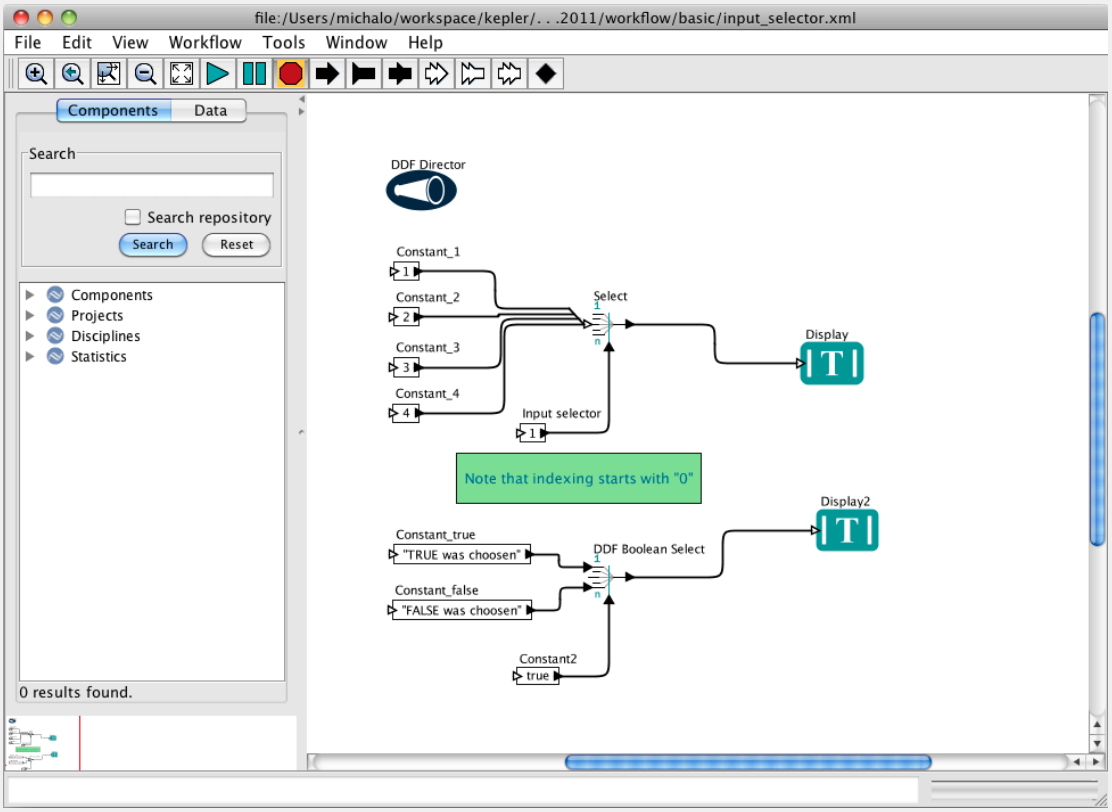
1. Start Kepler application by issuing:

```
kepler
```

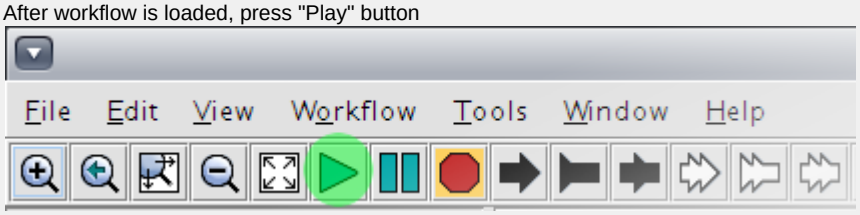
2. Load example workflow from following location

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/input_selector.xml
```

3. You should see workflow similar to one below



4. After workflow is loaded, press "Play" button



5. DDF Boolean Select vs. Select in a nutshell

DDF Boolean Select vs. Select

- DDF Boolean Select can choose between two values
- Select can choose between multiple values
- DDF Boolean Select uses "true", "false" to determine input port
- Select uses integer values (port index) to determine input port. Indexing starts with 0
- Both, DDF Boolean Select and Select, can use any type of input (e.g. String, integer, boolean, etc.)

3.4 Using Boolean Switch and Switch in order to determine output for processing

After this exercise you will:

- know how to determine where output data will be sent
- know how to choose between Boolean Switch and Switch actor

Exercise no. 6 (approx. 20 minutes)

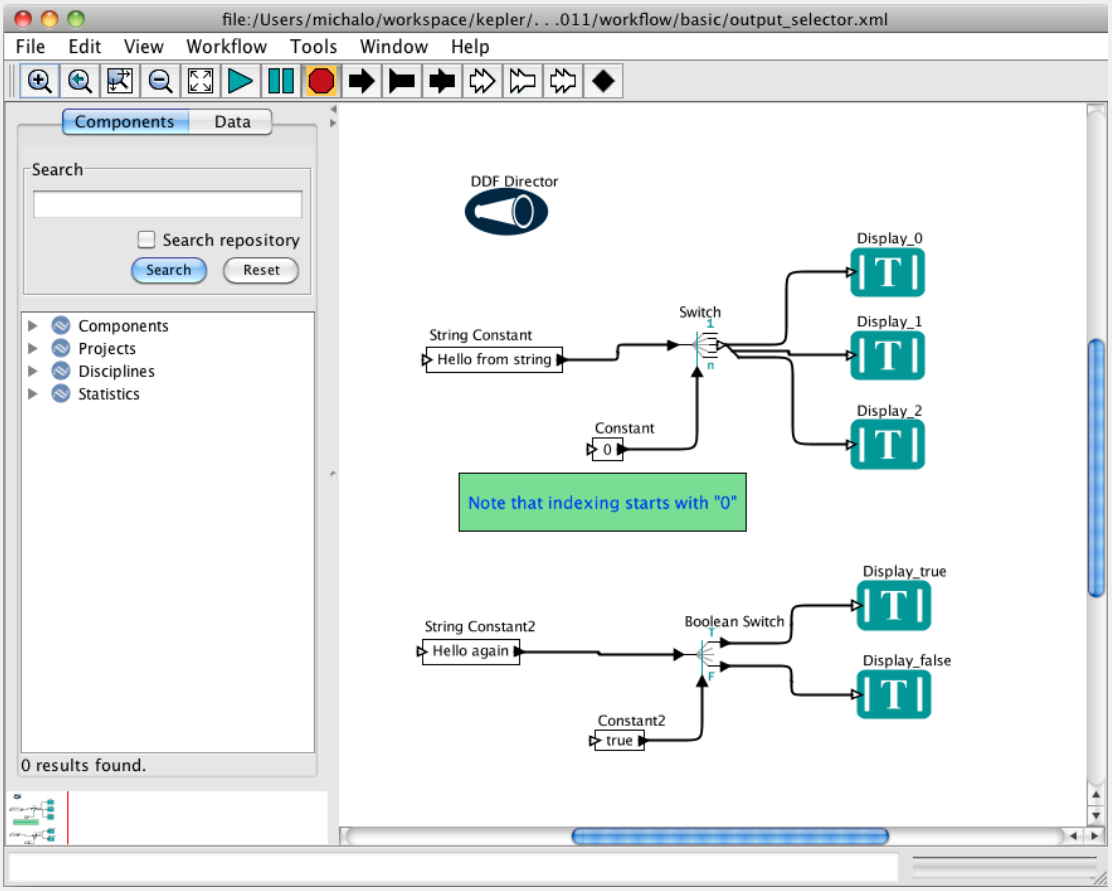
1. Start Kepler application by issuing:

```
kepler
```

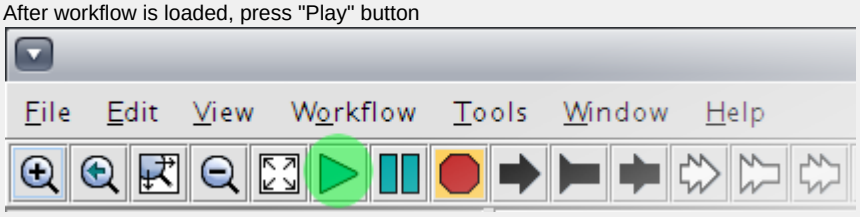
2. Load example workflow from following location

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/output_selector.xml
```

3. You should see workflow similar to one below



4. After workflow is loaded, press "Play" button



5. Boolean Switch vs. Switch in a nutshell

DDF Boolean Select vs. Select

- Boolean Switch can choose between two output ports (these ports are referred as **true/false**)
- Select can choose between multiple output ports
- Boolean Switch uses "true", "false" to determine output port
- Switch uses integer values (port index) to determine output port. Indexing starts with 0
- Both, Boolean Switch and Switch, can use any type of input/output (e.g. String, integer, boolean, etc.)

3.5 Using Relations for splitting and combining data flow

After this exercise you will:

- know how what Relation is
- know how to use Relations in order to split/combine data

Exercise no. 7 (approx. 20 minutes)

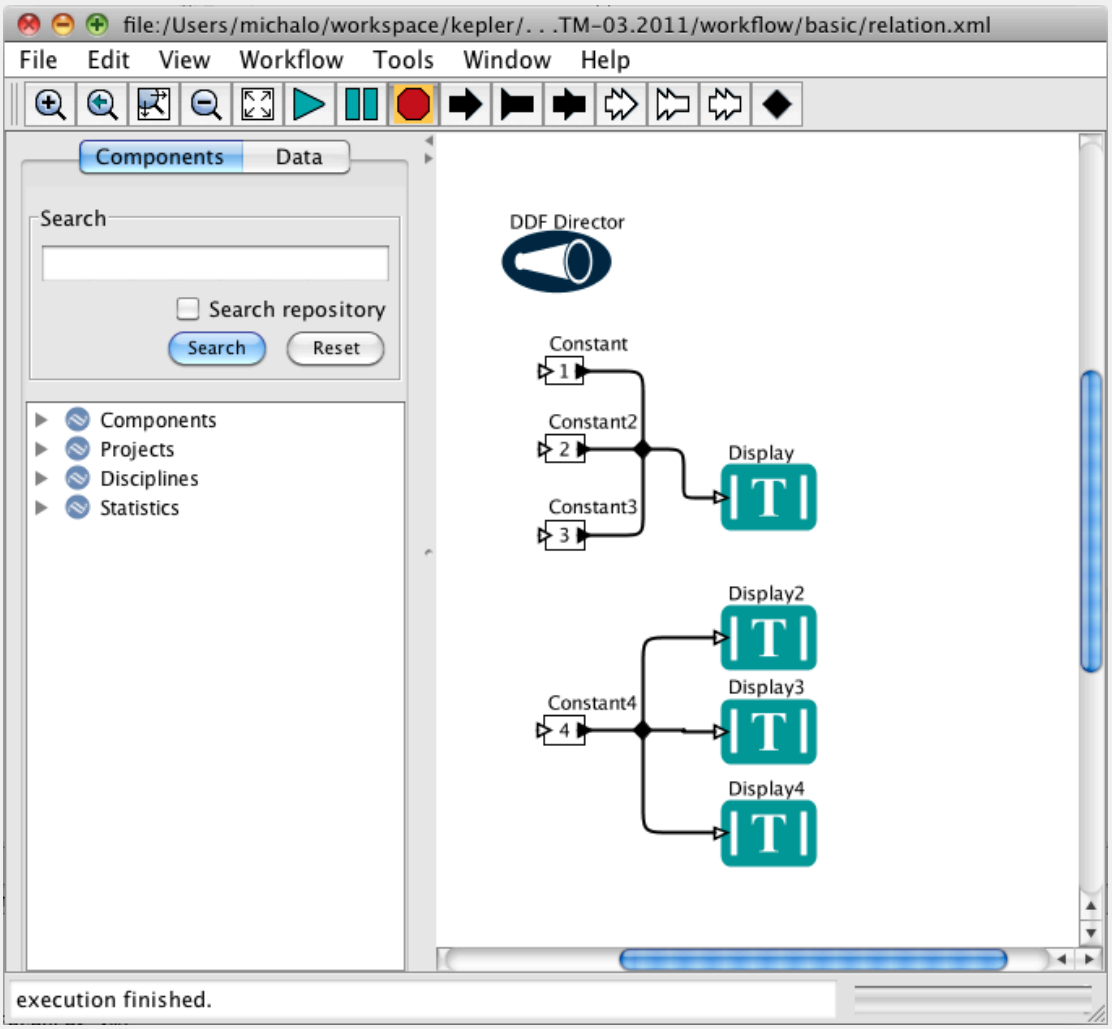
1. Start Kepler application by issuing:

```
kepler
```

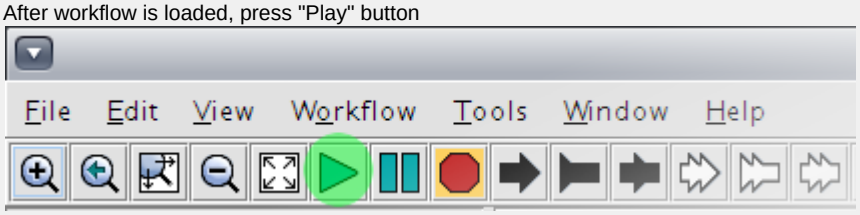
2. Load example workflow from following location

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/relation.xml
```

3. You should see workflow similar to one below



4. After workflow is loaded, press "Play" button



3.6 Relations, Paths and Synchronization

After this exercise you will:

- know how to add elements into workflow
- know hot to use expressions
- know how to synchronize workflow's execution
- know how to use parametrs
- know how to use relations

Exercise no. 8 (approx. 20 minutes)

Film available: http://www.youtube.com/watch?v=OCO9L5MzUrM

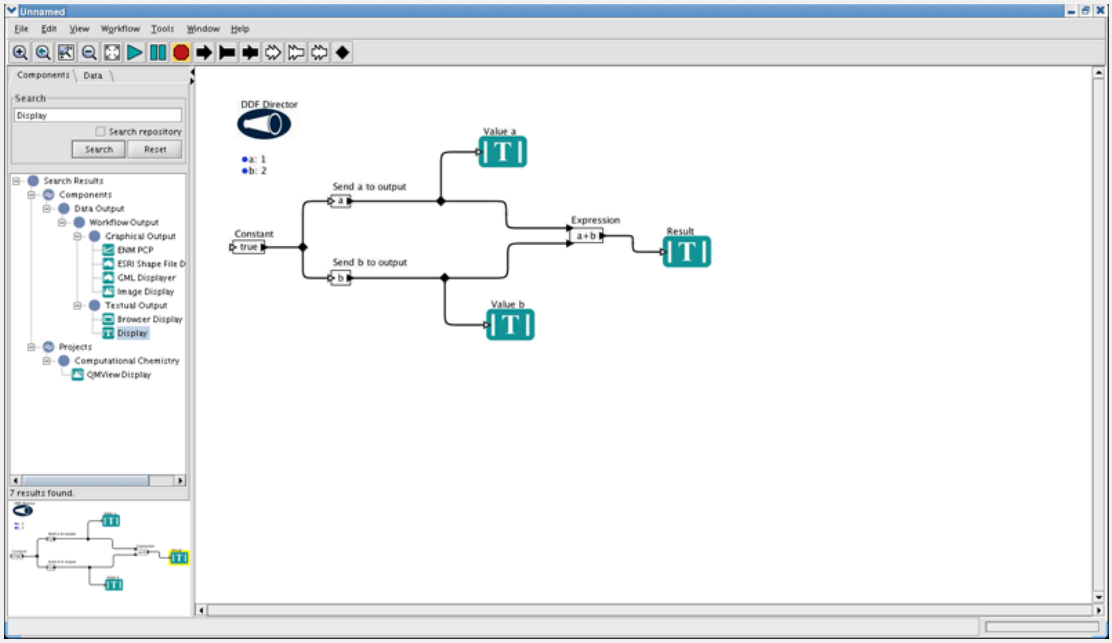
1. Start Kepler application by issuing:

```
kepler
```

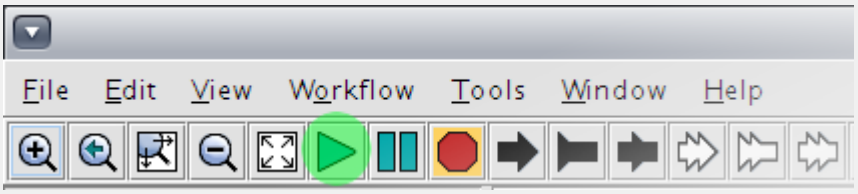
2. Add **DDF** director into workflow
3. Add **Constant** into workflow and set it's value to "true" (double click **Constant** and enter "true")
4. Set **Constant** "firingCountLimit" to "1" (Right click -> Configure Actor -> firingCountLimit Text Field)
5. Add **Relation** next to the **Constant**
6. Connect **Relation** and **Constant::output**
7. Add **Parameter** and rename it to "a" (Right click -> Customize name)
8. Set value of **a** to "1" (double click **a**)
9. Add **Parameter** and rename it to "b" (Right click -> Customize name)
10. Set value of **b** to "2" (double click **b**)
11. Add **Constant** into workflow and rename it to "Send a to output"
12. Set **Send a to output** value to "a"
13. Add **Constant** into workflow and rename it "Send b to output"
14. Set **Send b to output** value to "b"
15. Connect **Send a to output::trigger** with **Relation**
16. Connect **Send b to output::trigger** with **Relation**
17. Add **Relation** to workflow and connect it with **Send a to output**
18. Add **Relation** to workflow and connect it with **Send b to output**
19. Add **Display** to workflow and connect relation connected to **Send a to output**
20. Set **Display** "Display name" (Right click -> Customize Name) to "Value a"
21. Add **Display** to workflow and connect it with other relation
22. Set **Display** "Display name" (Right click -> Customize Name) to "Value b"
23. Add **Expression** to the workflow
24. Add input port **input_a** to the **Expression** (Right click -> Configure Ports -> Add, select checkbox "in")
25. Add input port **input_b** to the **Expression** (Right click -> Configure Ports -> Add, select checkbox "in")
26. Connect **Expression::input_a** with relation bound to **Send a to output**
27. Connect **Expression::input_b** with relation bound to **Send b to output**
28. Add **Display** to the workflow and set it's "Display name" to "Result"
29. Connect **Result::input** with *Expression::output"

30. Set **Expression** value to "a+b" (Double click **Expression**)
    At this point your workflow should be similar to the one below

Intermediate results



31. Execute workflow



Simple modification in order to make Kepler workflow fail

32. Set **Expression** value to "a/b"
33. Set **b** value to "0"

3.7 If-else workflow

3.7.1 Using existing "if-else" workflow

After this exercise you will:

- know how to use different paths for data flow
- know how to split workflow execution path
  V know how to use **Boolean Switch** actor

Exercise no. 9 (approx. 10 minutes)

Film available: http://www.youtube.com/watch?v=rr03bekyiDU

In this exercise you will execute simple Kepler workflow. In order to this follow the instructions:

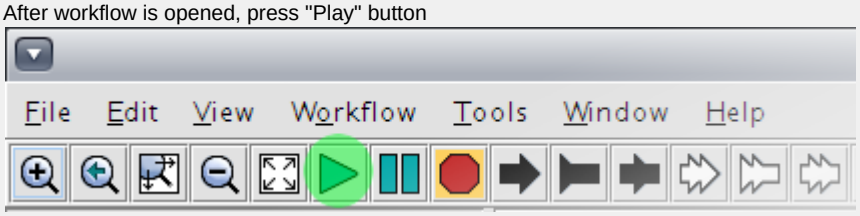
1. Start Kepler application by issuing:

```
kepler
```

2. Open "If-else" workflow by issuing: File -> Open and navigate to:

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/if_else_simple.xml
```

3. After workflow is opened, press "Play" button



Workflow should generate output within Display actor

3.7.2 Building "if-else" from the scratch

After this exercise you will:

- know how to use different paths for data flow
- know how to split workflow execution path
- know how to use **Boolean Switch** actor

Exercise no. 10 (approx. 20 minutes)

Film available: http://www.youtube.com/watch?v=3M7IFyzSTAY

In this exercise you will build "if-else" workflow.

You should complete previous examples before starting this one

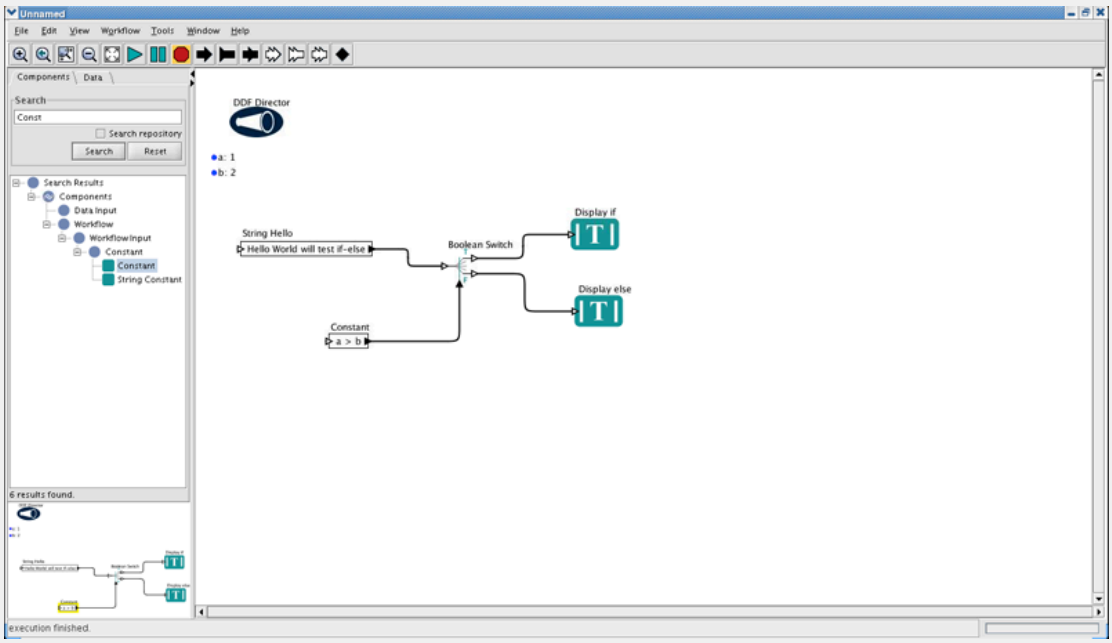
In this example it is assumed that you already know how to use actor/director browser (left panel) and how to put actors into workflow (right panel)

1. Start Kepler application by issuing:

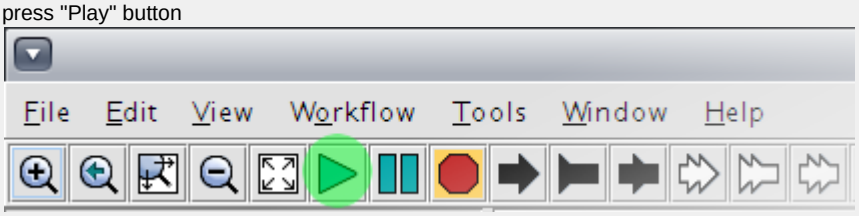
```
kepler
```

2. Drag and Drop "DDF Director" into workflow
3. Drag and Drop "String Constant" actor into workflow
4. Change it's name to "String Hello" (Right-click -> Custimize name)
5. Change it's value to "Hello world will test if-else" (Right-click -> Configure Actor -> value)
6. Change it's firingCountLimit to "1" (Right-click -> Configure Actor -> firingCountLimit)
7. Drag and Drop "Parameter" actor into workflow
8. Change it's name to "a" (Right-click -> Customize name)
9. Change it's value to "1" (Double click -> value)
10. Drag and Drop "Parameter" actor into workflow
11. Change it's name to "b" (Right-click -> Customize name)
12. Change it's value to "2" (Double click -> value)
13. Drag and Drop "Boolean Switch" actor into workflow
14. Drag and Drop "Display" actor into workflow next to "Boolean Switch" actor
15. Change it's name to "Display if"
16. Drag and Drop "Display" actor into workflow below "Display if" actor
17. Change it's name to "Display else"
18. Drag and Drop "Constant" actor into workflow below "Boolean Switch" actor
19. Change it's value to "a < b" (Right-click -> Configure Actor -> value)
20. Change it's firingCountLimit to "1" (Right-click -> Configure Actor -> firingCountLimit)
21. After all actors are at the workflow's area, you have to connect them
22. Connect **Boolean Switch::trueOutput** with **Display if::input**
23. Connect **Boolean Switch::falseOutput** with **Display else::input**
24. Connect **Boolean Switch::input** with **String Hello::output**
25. Connect **Boolean Switch::control** with **Constant::output**
    At this point your workflow should be similar to the one below

Intermediate results



26. press "Play" button



    Workflow should generate output within "Display else" actor
27. Change value of **Constant** to "a > b" and execute workflow once again
28. Save the workflow (e.g. as ~/my_workflow.xml) - we will need it in next excersise

3.7.3 Building "if-else-expression" from the scratch

After this exercise you will:

- know how to use different paths for data flow
- know how to split workflow execution path
- know how to use **Boolean Switch** actor
- know how to use **Expression**
- know how to use data flowing within workflow

Exercise no. 11 (approx. 20 minutes)

Film available: http://www.youtube.com/watch?v=qC6eVPXW4Fs

In this exercise you will build "if-else-expression" workflow.

You should complete previous examples before starting this one

In this example it is assumed that you already know how to use actor/director browser (left panel) and how to put actors into workflow (right panel)

1. Start Kepler application by issuing:

```
kepler
```

2. Load workflow that you have previously saved (~/my_workflow.xml) or open workflow at following location:

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/if_else_simple.xml
```

3. Remove **link** between **Display if** and **Boolean Switch** (select link and press "Delete" or choose **Edit -> Delete**)
4. Remove **link** between **Display else** and **Boolean Switch**
5. Add **Expression** between **Display if** and **Boolean Switch**
6. Set **Expression** Display name to "Expression if"
7. Add input port **in** into **Expression if**
8. Set **Expression if** value to

```
in + " - this was added by Expression if"
```

remember to copy " as well!

9. Connect **Expression if::in** with **Boolean Switch::trueOutput**
10. Connect **Expression if::output** with **Display if::input**
11. Add **Expression** between **Display else** and **Boolean Sitch**
12. Set **Expression** Display name to "Expression else"
13. Add input port **in** into **Expression else**
14. Set **Expression else** value to

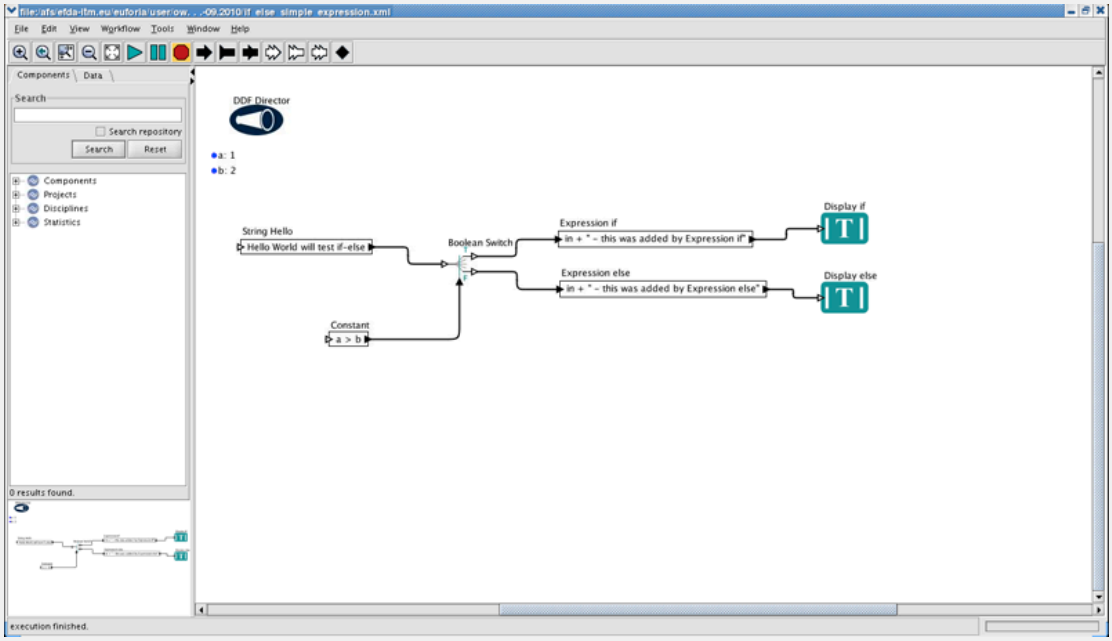
```
in + " - this was added by Expression else"
```

remember to copy " as well!

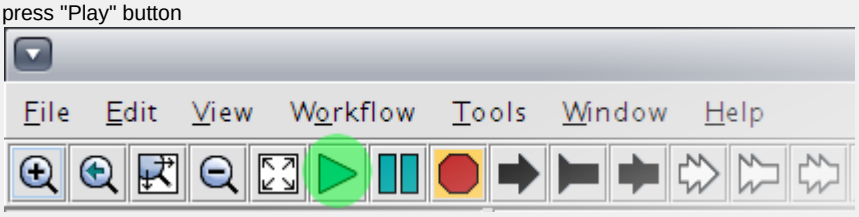
15. Connect **Expression else::in** with **Boolean Switch::falseOutput**

16. Connect **Expression else::output** with **Display else::input**
    At this point your workflow should be similar to the one below

Intermediate results



17. press "Play" button



    workflow should generate output within "Display if" actor
18. Change value of **Constant** to "a < b" and execute workflow once again

Animating workflows

In Kepler it is possible to animate workflows during execution. In order to animate workflow you have to turn on animations. You can do this by choosing: ***Tools -> Animate at Runtime...***
Demo movie for this feature can be found at following location: animation

2. Tutorial - Introduction to Kepler - Loops (Garching 09.2011)

Introduction to Kepler - Loops

Table of Contents

1. Introduction

This tutorial is designed to introduce the concept of building simple loop workflows within Kepler-1.0/Kepler-2.0. These workflows are assumed to repeat processing until some final conditions are met.

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. Requirements for the tutorial

Backing up Kepler home directory

Before you proceed with installation of the Kepler application be sure to make a backup of your Kepler home directory

```
mv ~/.kepler ~/.kepler_09_2011
mv ~/kepler ~/kepler_09_2011
mv ~/serpens ~/serpens_09_2011
```

2.1 Using ITM Kepler installation at Gateway

In order to make Kepler installation for the tutorial faster we will use preinstalled version of the Kepler that is available for Gateway users.

In order to install Kepler and ITM example workflow you have to follow instructions at following page:

Kepler installation

1. Kepler installation at Gateway (Garching 09.2011)

After you follow all the installation steps, you should see Kepler loading.

Starting Kepler

No matter which way have you used to install Kepler, make sure to export some variables before you start Kepler again.

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

3 Loops within Kepler

In this section of tutorial we will go through basic concepts of looping within Kepler. We will execute simple loop, build it from the scratch and, at the end, we will go through more complex examples of loops.

3.1 Executing simple loop example (classic)

After this exercise you will:

- know how to build simple loops
- know how to use SampleDelay actor
- know how to create loop condition checks
- know difference between SDF and DDF Directors

Exercise no. 1 (approx. 15 minutes)

Film available: http://www.youtube.com/watch?v=fJlV7Jd30cQ

In this exercise you will execute simple loop example. In order to this follow the instructions:

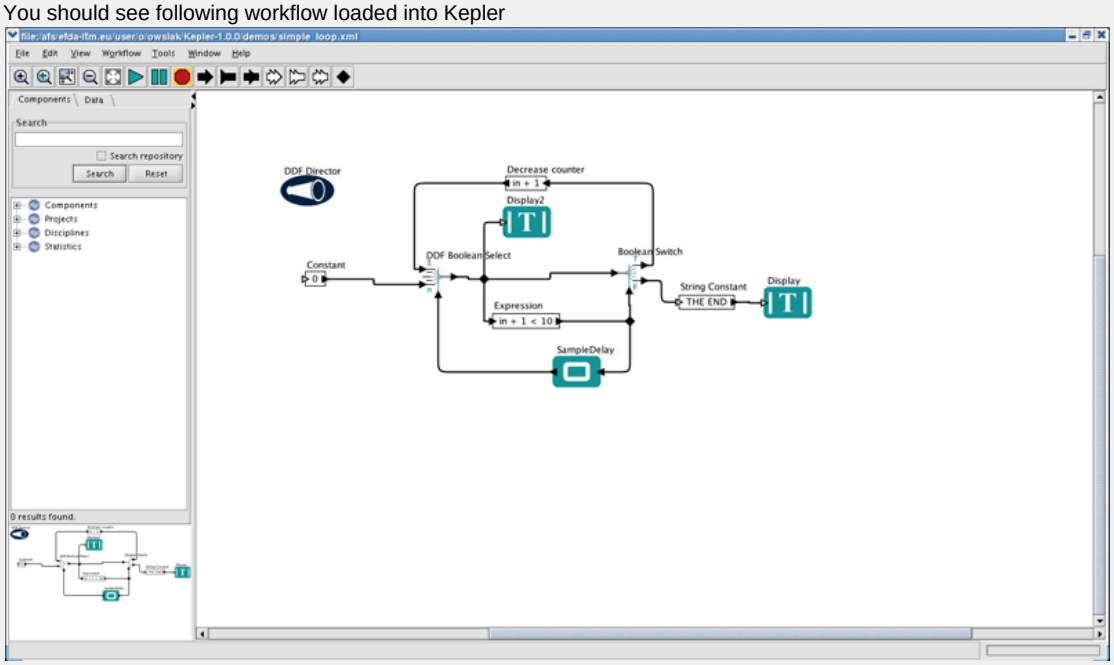
1. If Kepler is not already running start it by issuing:

```
kepler
```

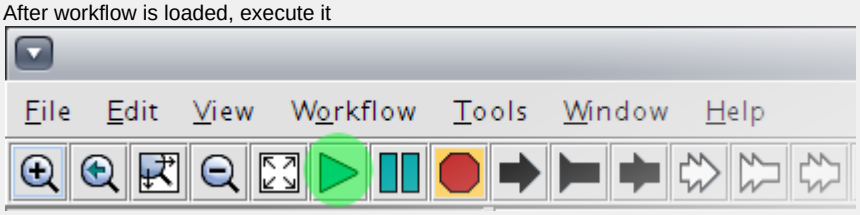
2. Open workflow

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/loops/simple_loop.xml
```

You should see following workflow loaded into Kepler



3. After workflow is loaded, execute it



workflow should generate output within Display actor

4. You can change **Constant** values and see what happens after you start workflow again

3.2 Building simple loop from the scratch

After this exercise you will:

- know how to build simple loops
- know how to use SampleDelay actor
- know how to create loop condition checks
- know difference between DDF and SDF directors

Exercise no. 2 (approx. 30 minutes)

Film available: http://www.youtube.com/watch?v=oYdOYnK7WI4

In this exercise you will build simple loop. In order to this follow the instructions:

1. Start Kepler application by issuing:

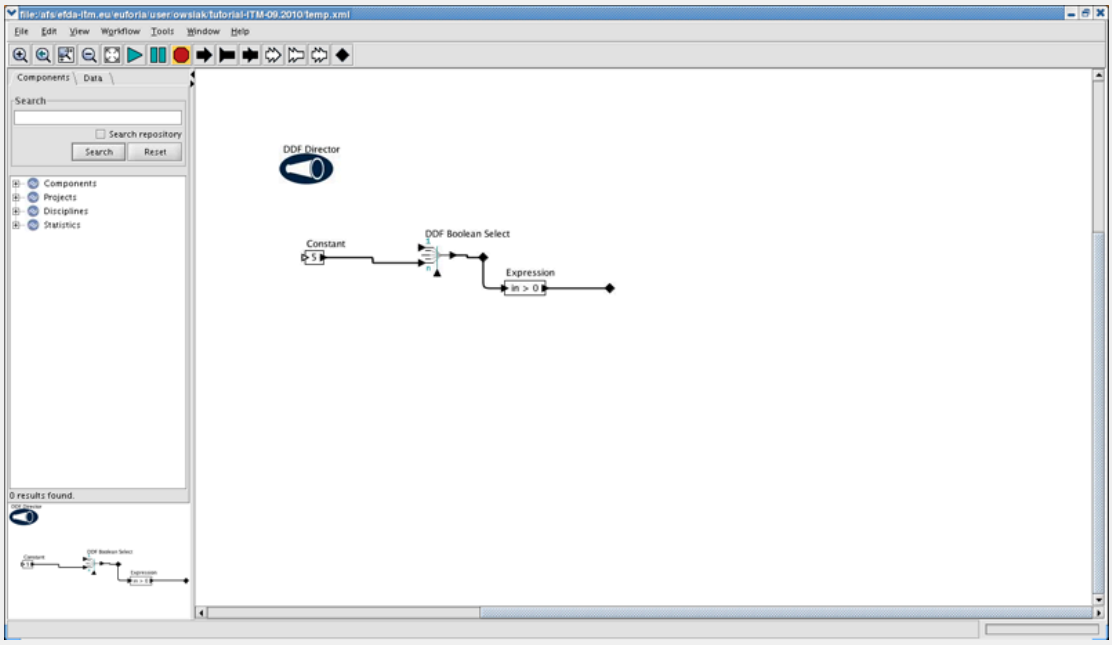
```
kepler
```

2. Add **DDF director** into workflow
3. Add **Constant** into workflow, set it's value to **5**
4. Add **DDF Boolean Select** actor to the workflow
5. Connect **Constant::output** with **DDF Boolean select::falseInput**
6. Add **Relation** next to **DDF Boolean Select**
7. Connect **Relation** (we will call it **Relation A**) with **DDF Boolean Select::output**
8. Add **Expression** actor to the workflow (next to **Relation A**)
9. Add input port **in** into **Expression**
10. Connect **Relation A** with **Expression::in**
11. Set **Expression** value to

```
in > 0
```

12. Add **Relation** next to **Expression** (we will call it **Relation B**)
13. Connect **Expression::output** with **Relation B**

Intermediate result



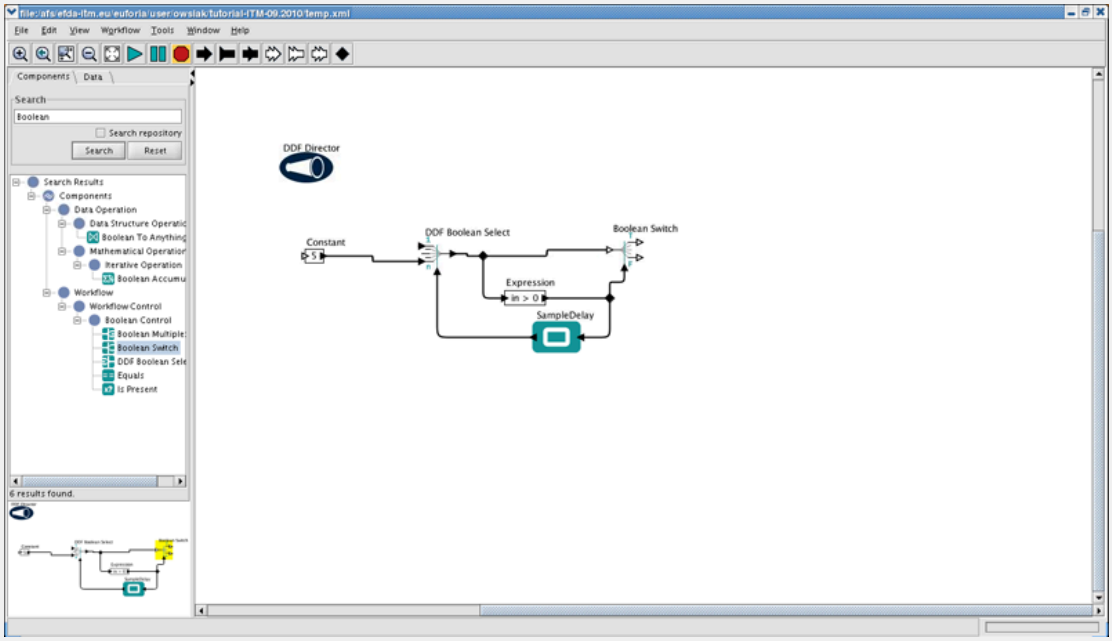
14. Add **SampleDelay** actor to workflow
15. Change **SampleDelay::input** port direction to **EAST** (Right click -> Configure Ports -> Direction)
16. Change **SampleDelay::output** port direction to **WEST** (Right click -> Configure Ports -> Direction)
17. Connect **SampleDelay::output** with **DDF Boolean Select::control**
18. Connect **SampleDelay::input** with **Relation B**
19. Set **SampleDelay** value to

```
{false}
```

20. Add **Boolean Switch** next to **Relation B**
21. Connect **Boolean Switch::control** with **Relation B**

22. Connect **Boolean Switch::input** with **Relation A**

Intermediate result



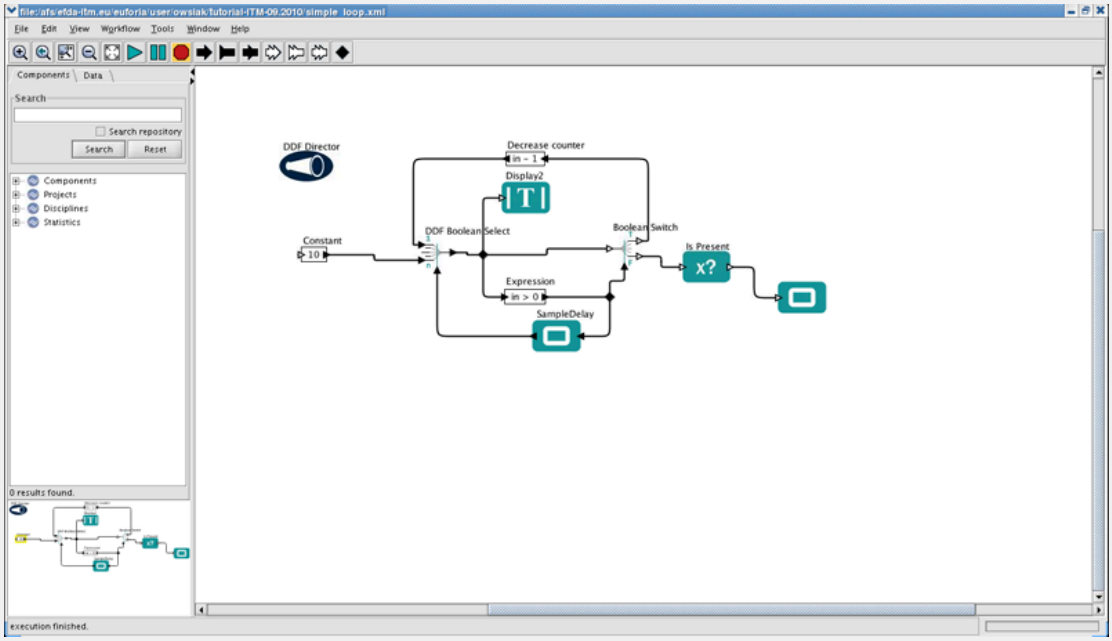
23. Add **Expression** into workflow and set it's name to **Decrease counter**
24. Add input port **in** into **Decrease counter** and set it's Direction to **EAST**
25. Set output port Direction to **WEST**
26. Set **Expression** value to

```
in - 1
```

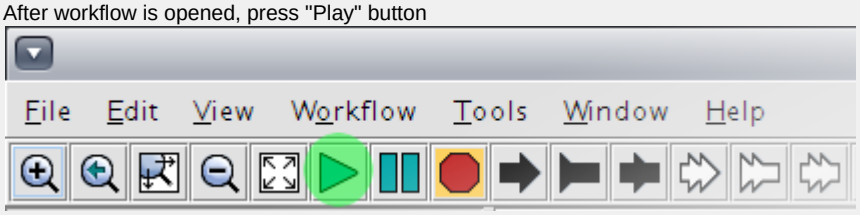
27. Connect **Decrease counter::output** with **DDF Boolean Select::trueInput**
28. Connect **Decrease counter::input** with **Boolean Switch::trueOutput**
29. Add **Is Present** next to **Boolean Switch**
30. Connect **Is Present::input** with **Boolean Switch::falseOutput**
31. Add **Stop** next to **Is Present**
32. Connect **Stop::input** with **Is Present::output**
33. Add **Display** next to **Relation A**

34. Connect **Display::input** with **Relation A**

Intermediate result



35. After workflow is opened, press "Play" button



36. You can change **Constant** values and see what happens next time you start workflow

3.3 Executing simple loop example (using relation instead of SampleDelay)

After this exercise you will:

- know how to build simple loops
- know how to create loop condition checks

Exercise no. 3 (approx. 15 minutes)

In this exercise you will execute simple loop example. In order to this follow the instructions:

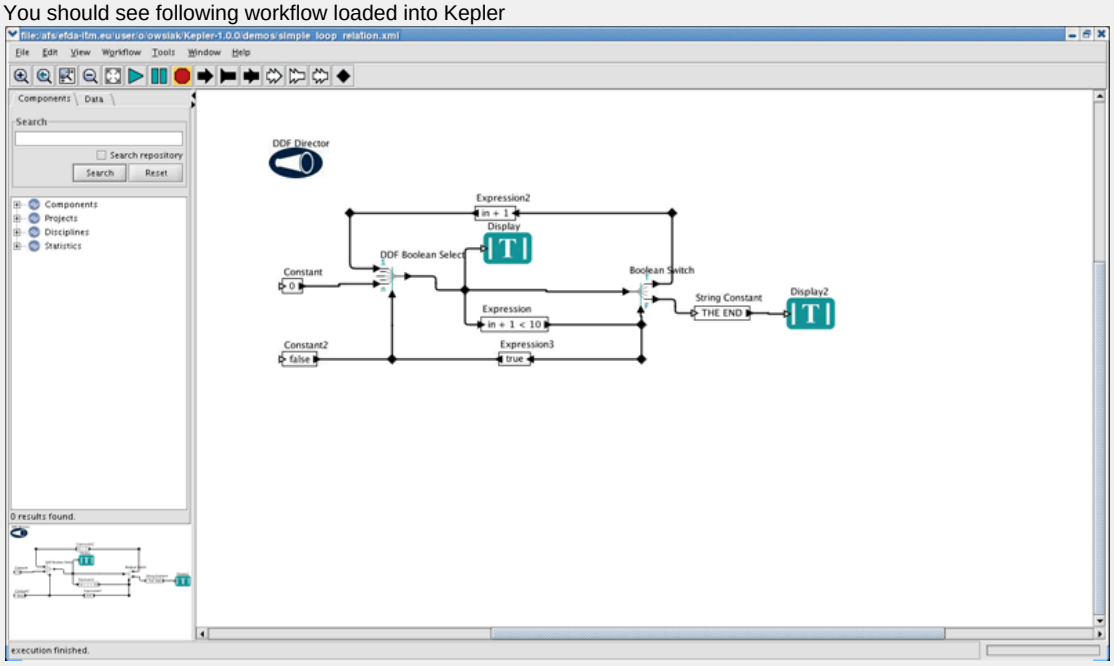
1. If Kepler is not already running start it by issuing:

```
kepler
```

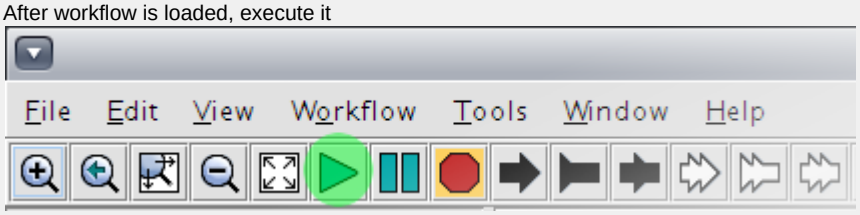
#Open workflow

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/loops/simple_loop_relation.xml
```

You should see following workflow loaded into Kepler



2. After workflow is loaded, execute it



workflow should generate output within Display actor

3. You can change **Constant** values and see what happens after you start workflow again

3.4 Executing simple loop example (using parameters and Variable Setter)

After this exercise you will:

- know how to build simple loops
- know how to use Variable Setter actor
- know how to create loop condition checks

Exercise no. 4 (approx. 15 minutes)

In this exercise you will execute simple loop example. In order to this follow the instructions:

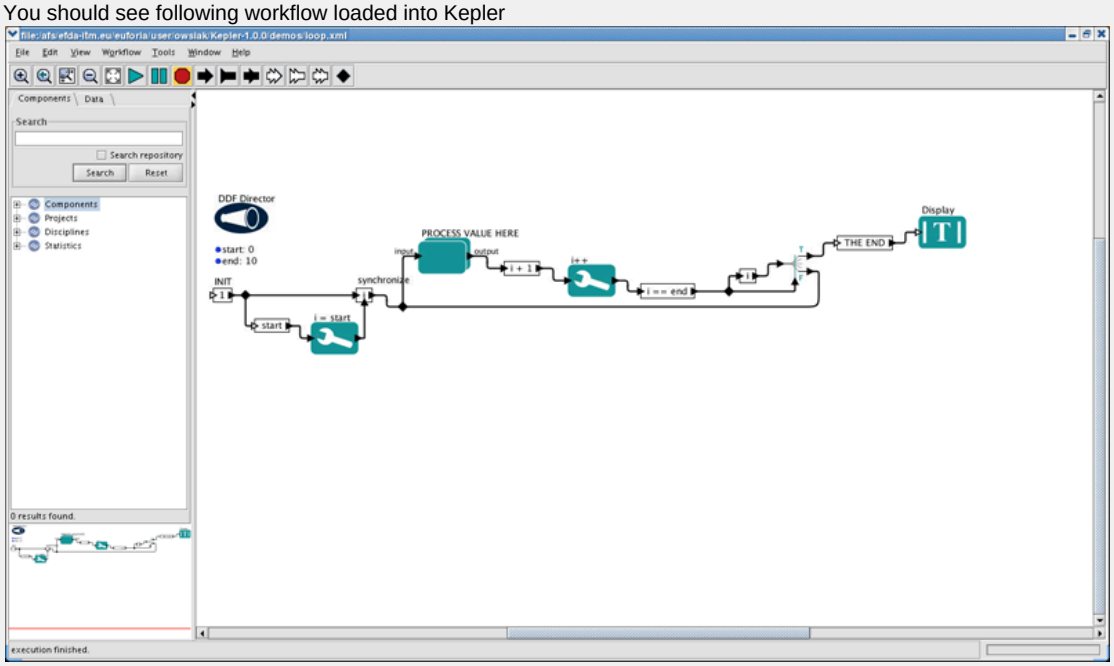
1. If Kepler is not already running start it by issuing:

```
kepler
```

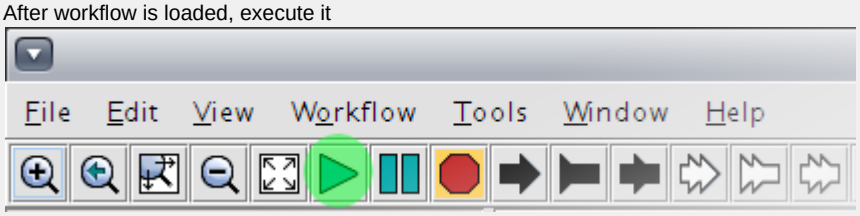
2. Open workflow

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/loops/loop-variables.xml
```

You should see following workflow loaded into Kepler



3. After workflow is loaded, execute it



workflow should generate output within Display actor

4. You can change **Constant** values and see what happens after you start workflow again

3.5 Executing simple loop example (without DDF Boolean Select actor)

After this exercise you will:

- know how to build simple loops
- know how to create loop condition checks

Exercise no. 5 (approx. 15 minutes)

In this exercise you will execute simple loop example. In order to this follow the instructions:

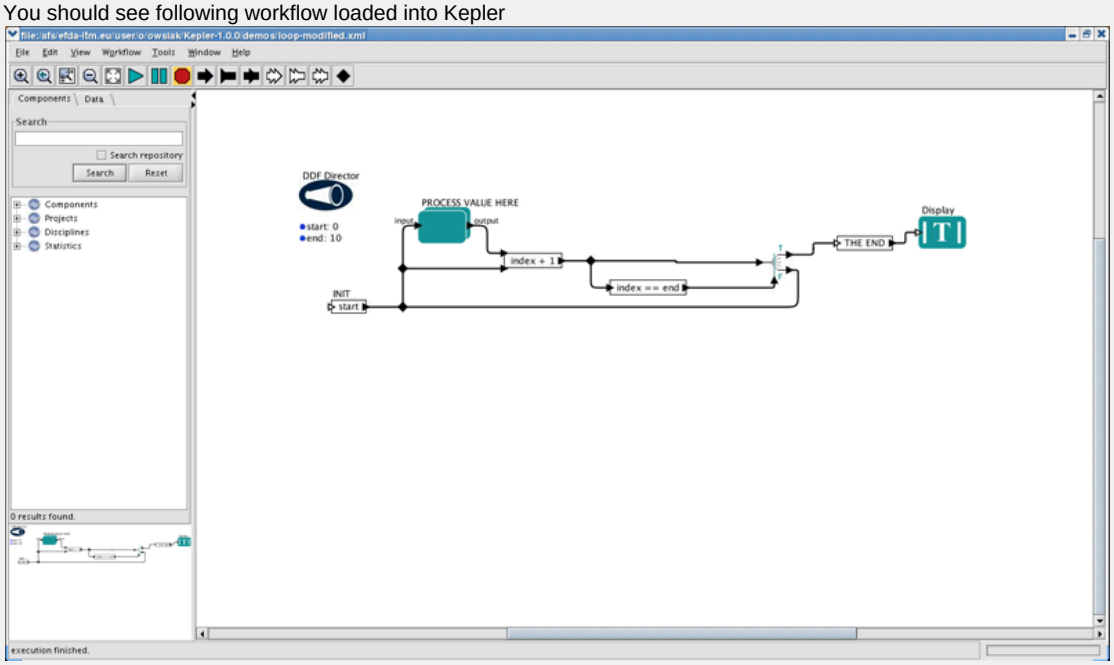
1. If Kepler is not already running start it by issuing:

```
kepler
```

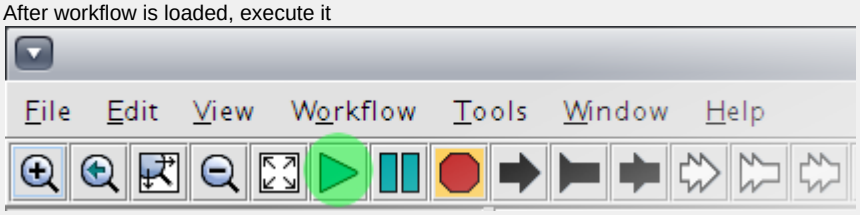
2. Open workflow

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/loops/loop-no-ddf-b-s.xml
```

You should see following workflow loaded into Kepler



3. After workflow is loaded, execute it



workflow should generate output within Display actor

4. You can change **Constant** values and see what happens after you start workflow again

3.6 Executing advanced loop workflow (Composite loop + Repeat)

After this exercise you will:

- know how to utilize loop concept
- know how to build advanced loop workflows
- know how to use **Repeat** actor
- know how to set value of the parameter

Exercise no. 6 (approx. 15 minutes)

Film available: http://www.youtube.com/watch?v=muhBH7jM5dU

In this exercise you will execute advanced loop workflow. In order to this follow the instructions:

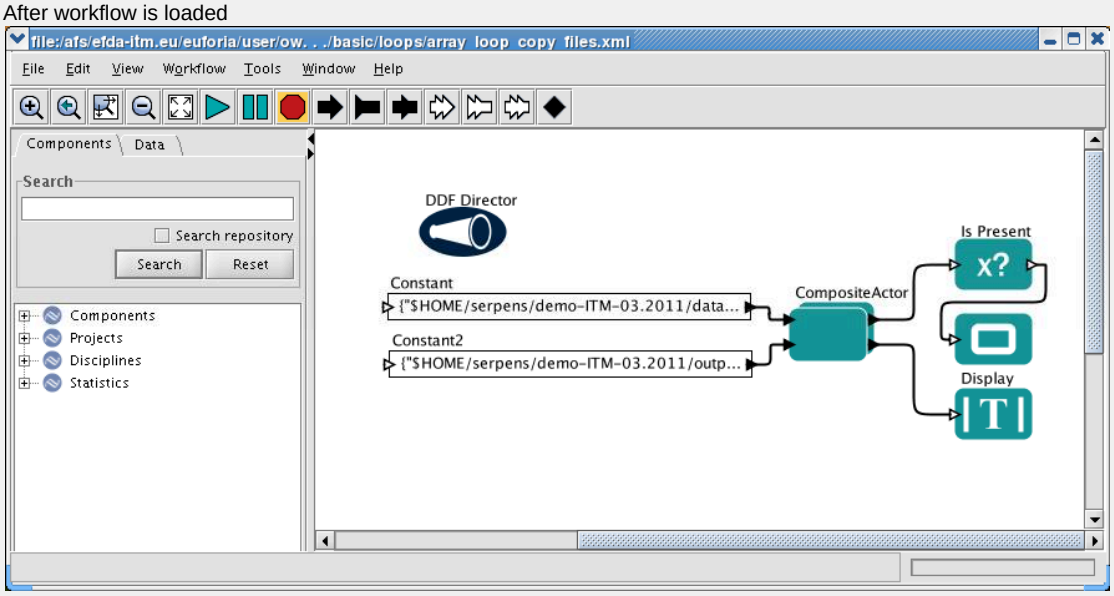
1. Start Kepler application by issuing:

```
kepler
```

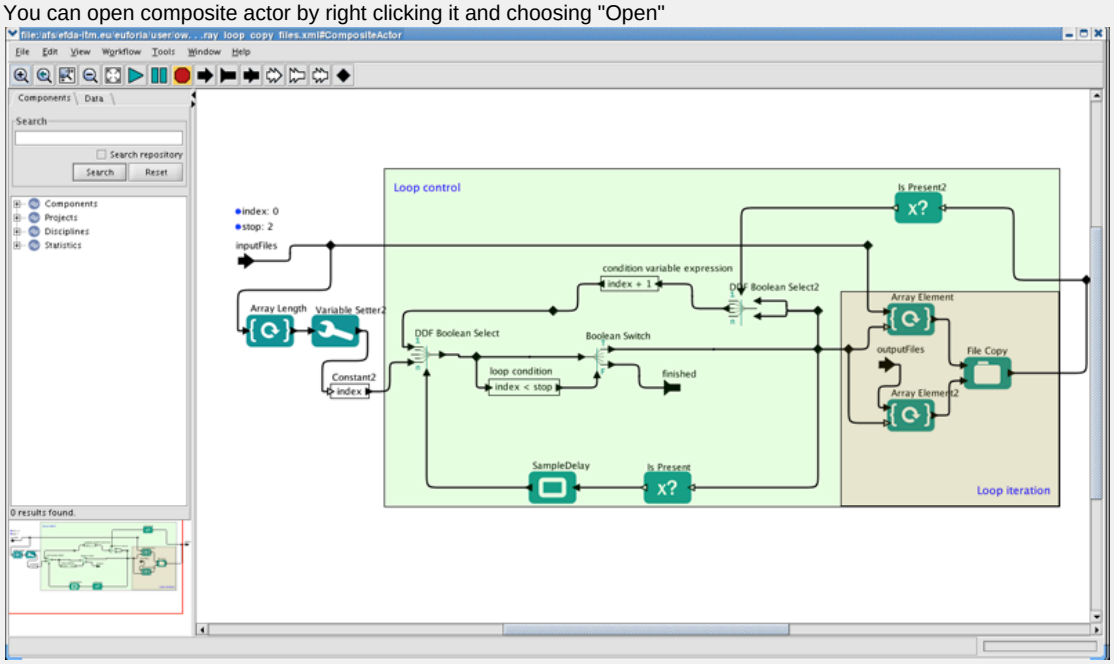
2. Open workflow

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/loops/array_loop_copy_files.xml
```

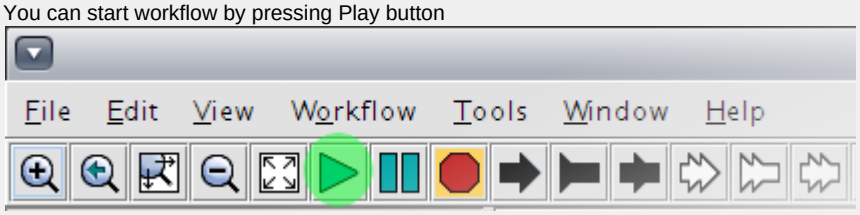
3. After workflow is loaded



You can open composite actor by right clicking it and choosing "Open"



4. You can start workflow by pressing Play button



workflow should copy input files from/to

```
$HOME/serpens/demo-ITM-09.2011/data -> $HOME/serpens/demo-ITM-09.2011/output
```

You can open terminal and verify it's execution results

```
ls -la ~/serpens/demo-ITM-09.2011/data
ls -la ~/serpens/demo-ITM-09.2011/output
```

3.7 Executing advanced loop workflow (Composite loop + feedback)

After this exercise you will:

- know how to utilize loop concept
- know how to build advanced loop workflows
- know how to use **Repeat** actor
- know how to set value of the parameter

Exercise no. 7 (approx. 15 minutes)

Film available: http://www.youtube.com/watch?v=4xjLcI776vg

In this exercise you will execute advanced loop workflow with feedback. In order to this follow the instructions:

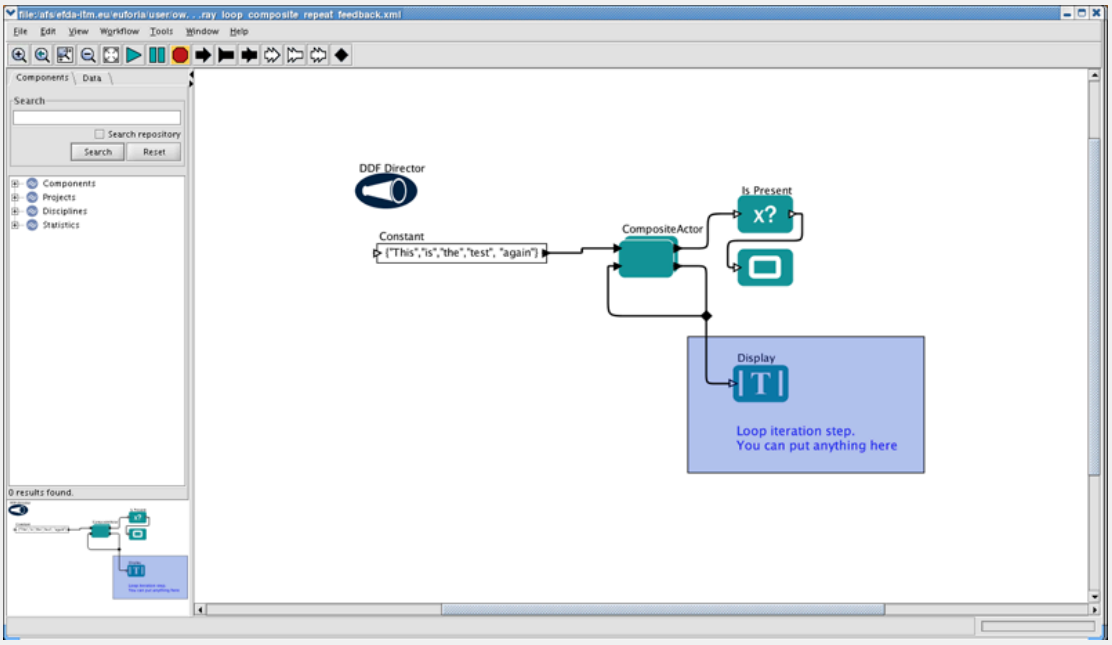
1. Start Kepler application by issuing:

```
kepler
```

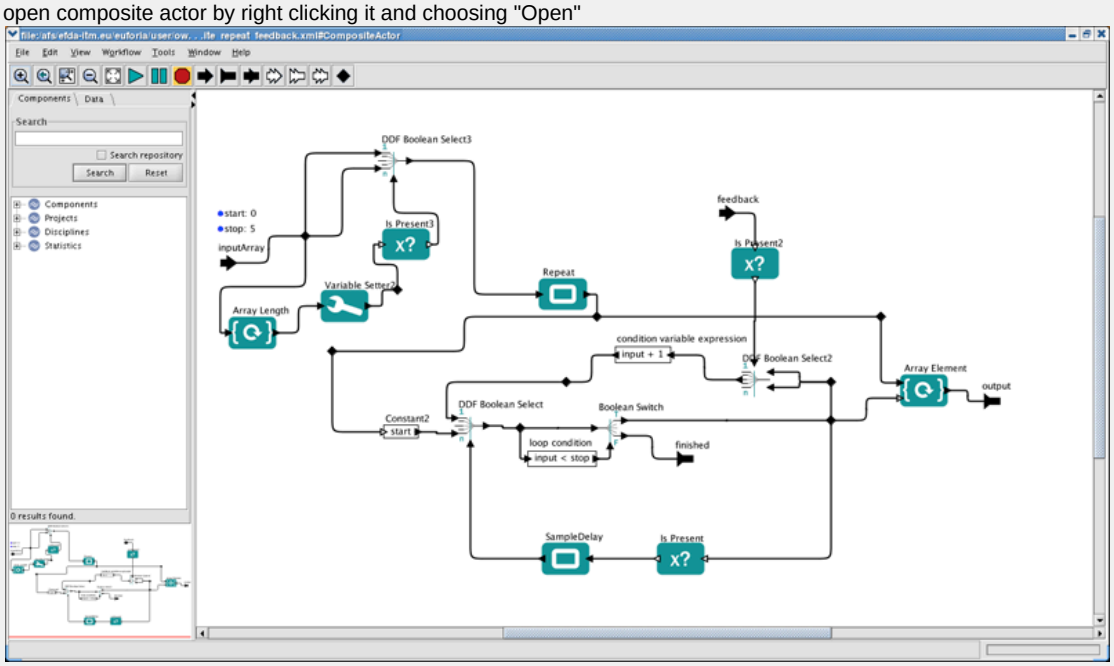
2. Open workflow

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/loops/array_loop_composite_repeat_fe
```

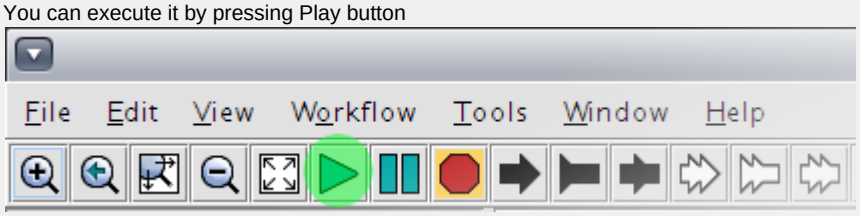
3. After workflow is loaded



open composite actor by right clicking it and choosing "Open"



4. You can execute it by pressing Play button



workflow should generate output within Display actor

Animating workflows

In Kepler it is possible to animate workflows during execution. In order to animate workflow you have to turn on animations. You can do this by choosing: ***Tools -> Animate at Runtime...***
Demo movie for this feature can be found at following location: animation

3.8 Creating a loop using PythonScript actor

In some cases, looping can be very conveniently incorporated in Kepler workflow inside a script written in Python programming language. This topic will be covered in the next part of the tutorial, where Python script embedding is described in details. Please refer to it later: 3. Tutorial - Introduction to Kepler - Python (Garching 09.2011)

3.9 Creating a time-loop with series plotting

After this exercise you will:

- know how to provide data from the loop to the plotting actor

Exercise no. 8 (approx. 15 minutes)

In this exercise you will create a simple loop containing some potentially time-consuming operations which will be plotted live. In order to do this follow the instructions:

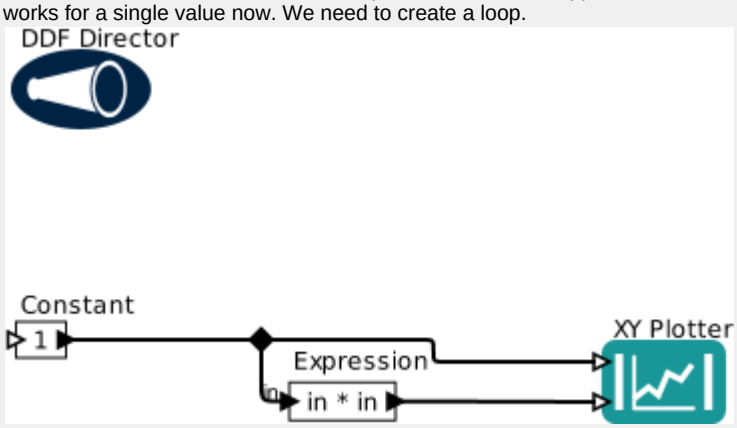
1. Start Kepler application by issuing:

```
kepler
```

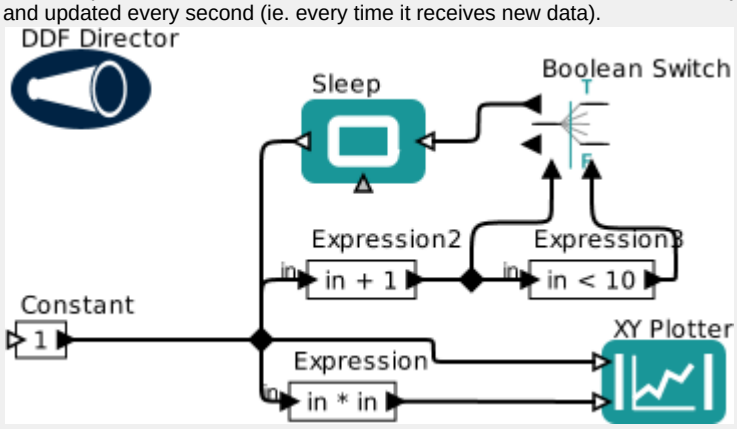
You can find this workflow at following location:

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/python/xyplotter.xml
```

2. Put DDF Director.
3. Put Constant actor and set its *firingCountLimit* to **1**. This will be the starting source for the example workflow.
4. Put a relation symbol (Ctrl+click) next to the Constant and link them.
5. Put an Expression actor and add an input port named *in*. Let's set the expression to **in * in** which indicates that we will plot the square function. Then connect the relation with *in* port.
6. Put an XYPlotter actor and connect its *inputX* with the relation symbol and *inputY* with Expression actor's output port.
7. Your workflow should look like the one presented below. It applies some function to input data and plots it. However, it works for a single value now. We need to create a loop.



8. Put another Expression actor. Add an input port named *in*, set expression to **in + 1** and connect the input port with relation. This is responsible for the step of loop.
9. Create another relation symbol next to this Expression actor and connect them.
10. Put next Expression actor. Again add an input port named *in*, set expression to **in < 10** and connect the input port with the just created relation. This is responsible for loop termination when it reaches specific point.
11. Put Boolean Switch actor, connect its *input* port to the **in + 1** expression and its *control* port to the **in < 10** expression.
12. Now we want to simulate the time-consuming behaviour, so we are going to add an artificial sleep time. To do this, you need to choose from menu *Tools -> Instantiate Component* and set as *Class name* a value *ptolemy.actor.lib.Sleep*. A new Sleep actor will appear. It's purpose is to grab some input, wait for the specified amount of time and then send the data. For this workflow, please set its *sleepTime* to 1000 (the unit here is milliseconds, so we will simulate one second of time-consuming operations).
13. Connect Boolean Switch *trueOutput* port with Sleep's *input*, and Sleep's *output* to the relation symbol at the beginning of the loop. Your workflow should look like the one below. You can run it and you will see that the output is plotted live and updated every second (ie. every time it receives new data).



3. Tutorial - Introduction to Kepler - Python (Garching 09.2011)

Introduction to Kepler - Python

Table of Contents

1. Introduction

This tutorial is designed to introduce the concept of building simple loop workflows within Kepler-1.0/Kepler-2.0. These workflows are assumed to repeat processing until some final conditions are met.

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. Requirements for the tutorial

Backing up Kepler home directory

Before you proceed with installation of the Kepler application be sure to make a backup of your Kepler home directory

```
mv ~/.kepler ~/.kepler_09_2011
mv ~/kepler ~/kepler_09_2011
mv ~/serpens ~/serpens_09_2011
```

2.1 Using ITM Kepler installation at Gateway

In order to make Kepler installation for the tutorial faster we will use preinstalled version of the Kepler that is available for Gateway users.

In order to install Kepler and ITM example workflow you have to follow instructions at following page:

Kepler installation

1. Kepler installation at Gateway (Garching 09.2011)

After you follow all the installation steps, you should see Kepler loading.

Starting Kepler

No matter which way have you used to install Kepler, make sure to export some variables before you start Kepler again.

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

3. Using Python code within Kepler actor

After this exercise you will:
• know how to use Python script inside a Kepler actor

Exercise no. 1 (approx. 20 minutes)

In this exercise you will create a simple actor that will use Python script. You can find this workflow at following location:

```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/python/python_script.xml
```

In order to this follow the instructions:

1. Start Kepler application by issuing:

```
kepler
```

2. Instantiate a PythonScript actor by choosing menu *Tools -> Instantiate Component* and setting as *Class name* a value *ptolemy.actor.lib.python.PythonScript*.
3. This actor starts with zero ports. They need to be added manually. Please right-click on PythonScript and choose *Configure Ports*.
4. Add an input port named *in* and output port named *out*.

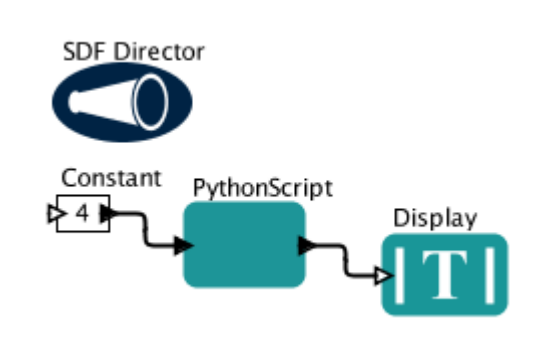
> Kepler's automatic type resolver may not correctly infer types of PythonScript ports due to dynamic features of Python programming language. This may lead to errors and unexpected behaviour. Thus you need to specify these types explicitly. For this tutorial, please set type of *in* to **int** and type of *out* to **arrayType(int)**.

5. By default Kepler initialises the *script* parameter of this actor to be of type *Line*. To develop a script in Python, it needs to be changed. Please right-click on PythonScript and choose *Configure Actor*. Go to *Preferences* and select *expert mode*.
6. Close the window with actor's preferences and once again start with right-clicking and choosing *Configure Actor*. Again choose *Preferences* and change type of *script* parameter to *Text*.
7. Now you can see a Python code displayed in several lines. Some remarks here:

- Python is a dynamic language, so no typecasting takes place,
- Do not declare any constructor.
- You only need to fill the fire() method.
- You can assume that the configured ports are already instantiated (ie. you can use names *in* and *out* to work with actor's ports)

```
import ptolemy.data

class Main:
    def fire(self):
        # read value of input token
        val = self.in.get(0).intValue()
        self.out.send(0, ptolemy.data.IntToken(val))
        return
```

1. You can now instantiate Constant actor. Set its *firingCountLimit* to 1 and *value* to 4. Connect it with *in* port of PythonScript.
2. Instantiate also a Display actor and connect PythonScript's *out* with it.
3. Finally add an SDF actor and execute the workflow.



4. You will see 4 in the Display window. Now you can change the input value 4 to some other one. Or you can change the actor source code to execute a different task.
5. Please save this workflow as `$HOME/python_script.xml`. The next exercises will depend on it.

4. Creating a loop using PythonScript actor

After this exercise you will:

- know how to create a generic loop using PythonScript actor

Exercise no. 2 (approx. 10 minutes)

In this exercise you will create a simple loop in Python and put it inside a special Kepler actor. You can find this workflow at following location:

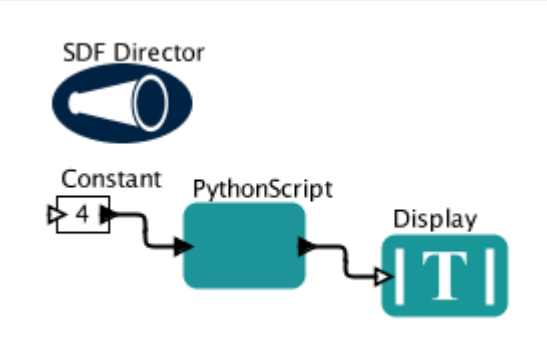
```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/python/python_loop_script.xml
```

In order to this follow the instructions:

1. Assuming you have finished the previous exercise, please load the workflow with basic Python actor.
2. In this exercise you will learn how to loop and create arrays using Python programming language. Let's assume a following problem to solve. The actor receives a number which will be interpreted as iteration count. In each **i-th** iteration, the actor will output $\mathbf{i}^2$. Example: *in* = 4, *out* = {0, 1, 4, 9}.
3. In Python, the following script will do this:

```
import ptolemy.data

class Main:
    def fire(self):
        # read value of input token
        val = self.in.get(0).intValue()
        arr = []
        for i in range(val):
            # create a new IntToken with each value
            arr.append(ptolemy.data.IntToken(i**2))
        # send an ArrayToken with array of values
        self.out.send(0, ptolemy.data.ArrayToken(arr))
        return
```

4. Execute the workflow.



5. You will see {0, 1, 4, 9}. Now you can change the input value 4 to some other one. Or you can change the actor source code to execute a different task.

5. An advanced loop using PythonScript actor

After this exercise you will:

- know how to create advanced Kepler actors by incorporating simple yet powerful Python code inside it

Exercise no. 3 (approx. 10 minutes)

In this exercise you will create an advanced loop in Python which will parse an array token. You can find this workflow at following location:

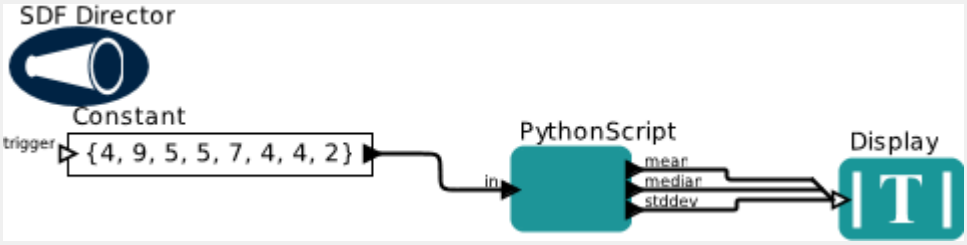
```
$HOME/serpens/demo-ITM-09.2011/workflow/basic/python/python_advanced.xml
```

In order to this follow the instructions:

1. Assuming you have finished the first exercise, please load the workflow with basic Python actor.
2. In this exercise you will learn how to process an existing array. Let's assume you want to calculate some statistics from numerical data: mean, median and standard deviation values.
3. You have to modify your existing PythonScript actor ports (right click on actor -> Configure ports):
    a. Delete *out* port
    b. Add *mean*, *median* and *stddev* input ports
    c. Set their **Type** field to *double*
4. The following script will calculate desired values:

```
import ptolemy.data.DoubleToken

class Main:
    def fire(self):
        # parse input array token
        token = self.in.get(0)
        array = list()
        for i in range(token.length()):
            value = token.getElement(i)
            array.append(value.doubleValue())

        # calculate mean
        size = len(array)
        mean = sum(array)/size

        # get median
        array.sort()
        if size % 2 == 0:
            median = (array[size/2 - 1] + array[size/2])/2.0
        else:
            median = array[size/2]

        # calculate standard deviation
        stddev = 0
        for value in array:
            stddev = (value - mean)*(value - mean)
        stddev /= size

        # send results to output ports
        self.mean.send(0, ptolemy.data.DoubleToken(mean))
        self.median.send(0, ptolemy.data.DoubleToken(median))
        self.stddev.send(0, ptolemy.data.DoubleToken(stddev))
```

5. In the workflow, please set the initial Constant actor value to `{4, 9, 5, 5, 7, 4, 4, 2}` or any other numerical array.
6. Connect all three output ports to the Display actor.
7. Execute the workflow.



8. You will see three lines with our calculated values. If you followed the tutorial steps exactly, and put the same initial value to the Constant actor, then you should see:

```
5.0
4.5
2.0
```

3. Tutorial - Using FC2K with Fortran, C++ (Garching 09.2011)

Using FC2K with Fortran, C++ codes

Table of Contents

Please update your installation

Please execute following commands within your terminal window

cp ~owsiak/public/itmdb/itm_trees/test/4.09a/mdsplus/0/* ~/public/itmdb/itm_trees/test/4.09a/mdsplus/0
cp -r ~owsiak/public/garching-09.2011/ISE ~/public/garching-09.2011/

1. Introduction

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

FC2K is a tool for wrapping a Fortran or C++ source code into a Kepler actor. Before using it, your physics code should be ITM-compliant (i.e. use CPOs as input/output). After running the ITMv1 script (to properly set up the environment variables), FC2K can be run simply by typing fc2k in the Linux command line. FC2K was developed by ISIP in Java/Python. You can find more regarding FC2K at following location.

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.

NX connection to the Gateway

This tutorial assumes that Gateway accounts will be used for starting up Kepler application.
If you are not familiar with NX setup for the Gateway, take a look at following location NX setup

2. Requirements for the tutorial

Backing up Kepler home directory

Before you proceed with installation of the Kepler application be sure to make a backup of your Kepler home directory

```
mv ~/.kepler ~/.kepler_12_09_2011
mv ~/kepler ~/kepler_12_09_2011
mv ~/serpens ~/serpens_12_09_2011
```

2.1 Using ITM Kepler installation at Gateway

In order to make Kepler installation for the tutorial faster we will use preinstalled version of the Kepler that is available for Gateway users.

In order to install Kepler and ITM example workflow you have to follow instructions at following page:

Kepler installation

Kepler installation steps

After you follow all the installation steps, you should see Kepler loading.

Starting Kepler

No matter which way have you used to install Kepler, make sure to export some variables before you start Kepler again.

```
setenv JAVA_HOME /usr/java/latest
setenv KEPLER ~/kepler
kepler
```

3. Incorporating simple Fortran/C++ codes into Kepler using FC2K

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

1. Simple Fortran code that will be incorporated into Kepler via FC2K tool - multiplying input value by two
2. Simple C++ code that will be incorporated into Kepler via FC2K tool - adding one to input value

3.1 Fortran code within Kepler

After this exercise you will:

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

Exercise no. 1 (approx. 30 min)

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

1. Get familiar with codes that will be incorporated into Kepler

Go to Code Camp related materials within your home directory

```
cd ~/public/garching-09.2011/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. Build the code by issuing

```
make clean
make
```

Codes are ready to be used within FC2K

3. Prepare environment for FC2K

Make sure that all required system settings are correctly set

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
```

4. Start FC2K application

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

```
fc2k
```

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

Default settings

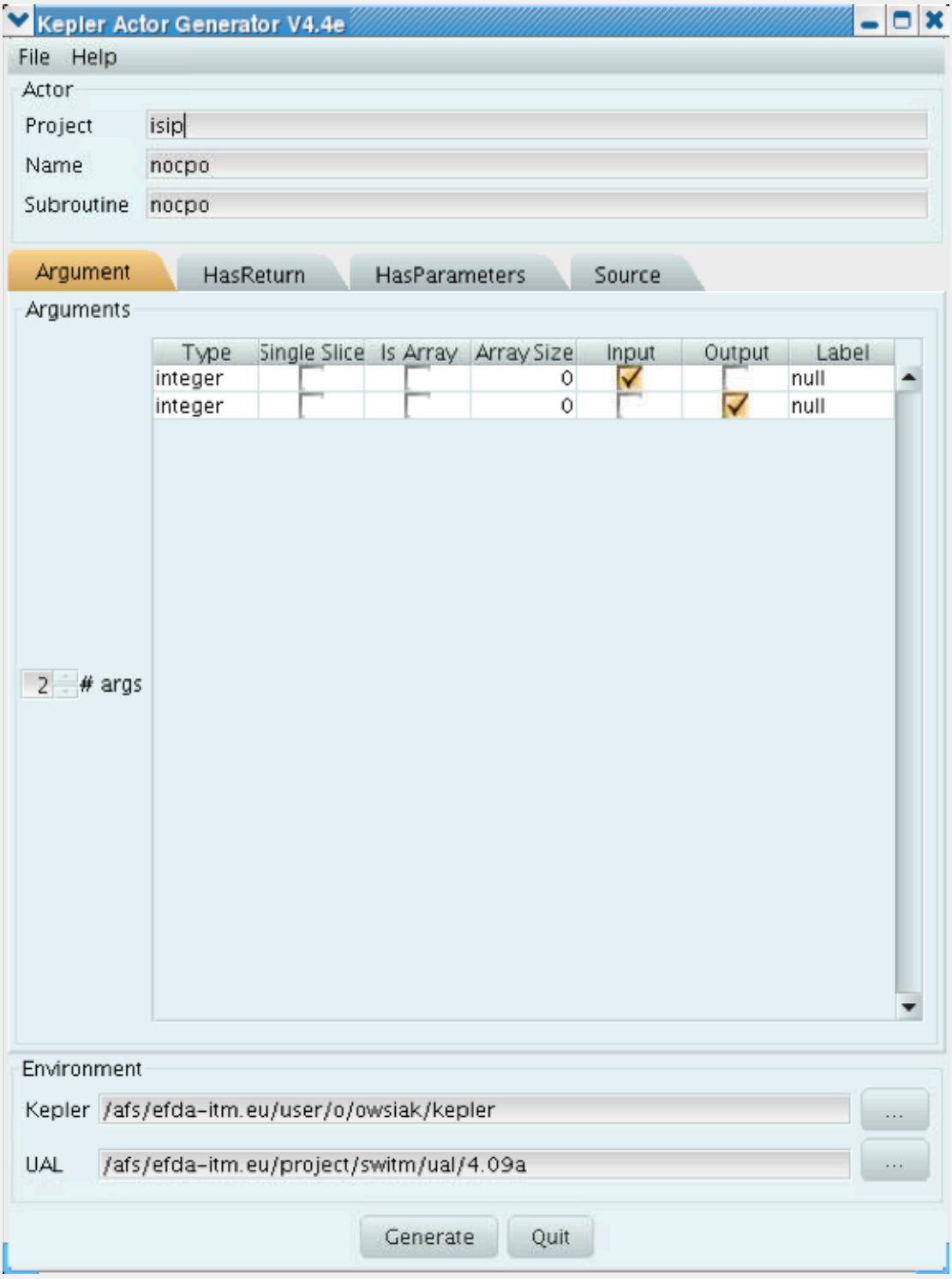
Note, that your settings will be slightly different. Your **Kepler** location should point to a valid path for your account.

5. Open existing parameters settings

Choose **File -> Open** and navigate to **~/public/garching-09.2011/FC2K/nocpo_example_1**. Open file **nocpo_fc2k.xml**. You should see new parameter settings loaded into FC2K.

6. Make sure that Kepler location is correct

After loading parameters you can notice that parameters point to locations within my home directory (**~owsiak**).

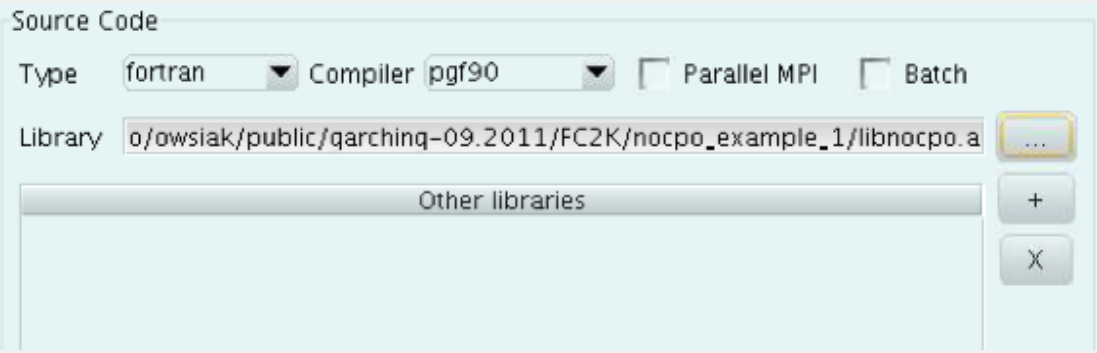


You should modify these setting, so they point to locations within you home directory. They will typically be as follows:

```
$HOME/kepler
```

7. Make sure that **library** location is correct

After loading parameters you can notice that library location points to location within my home directory (**~owsiak**).



You should modify this location, so it points to location of the library within your public directory. It should point to:

```
~/public/garching-09.2011/FC2K/nocpo_example_1/libnocpo.a
```

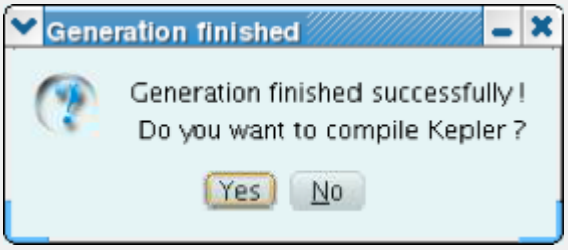
8. After all the settings are correct, you can generate actor

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

> **Generating an actor for the second time**
>
> This tutorial assumes that Gateway accounts will be used for starting up Kepler application.
> If you are not familiar with NX setup for the Gateway, take a look at following location NX setup

9. Confirm Kepler compilation

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



10. You can now start Kepler and use generated actor

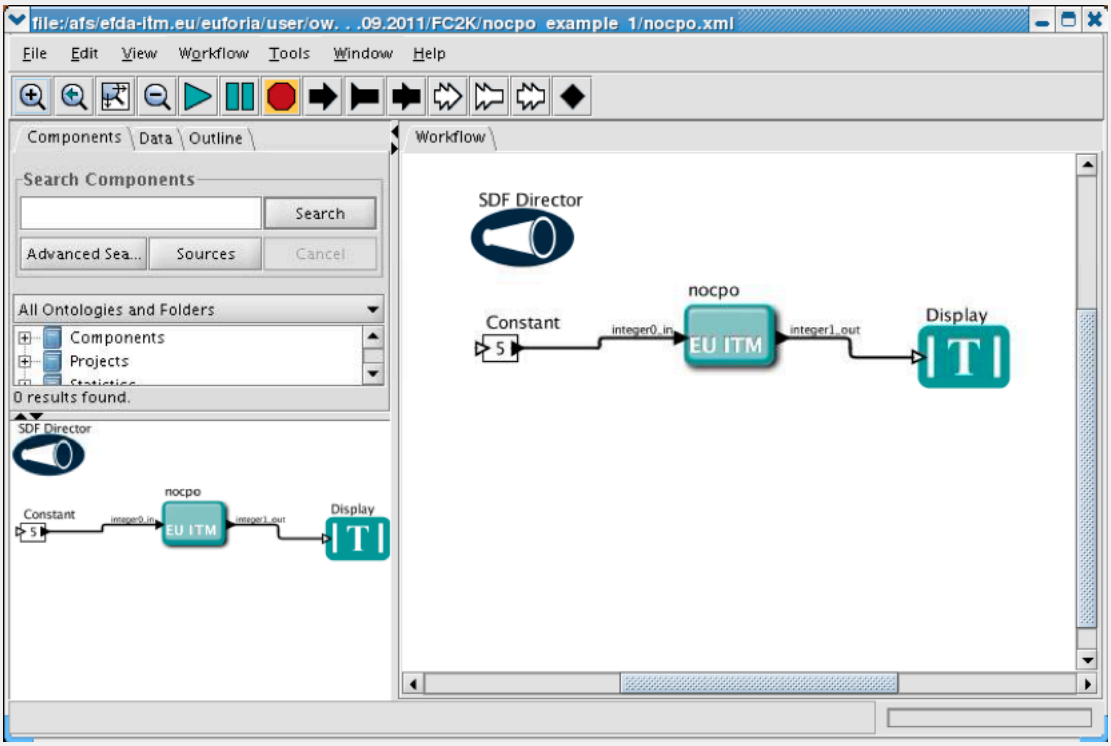
Open new terminal window and make sure that all environment settings are correctly set and execute Kepler.

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

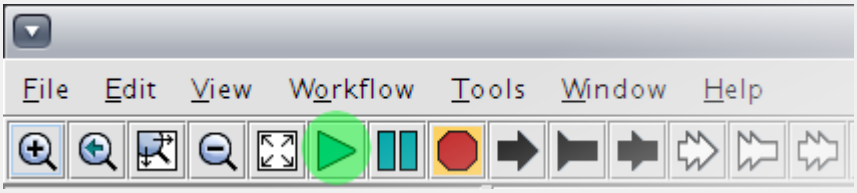
After Kepler is started, open example workflow from the following location

```
~/public/garching-09.2011/FC2K/nocpo_example_1/nocpo.xml
```

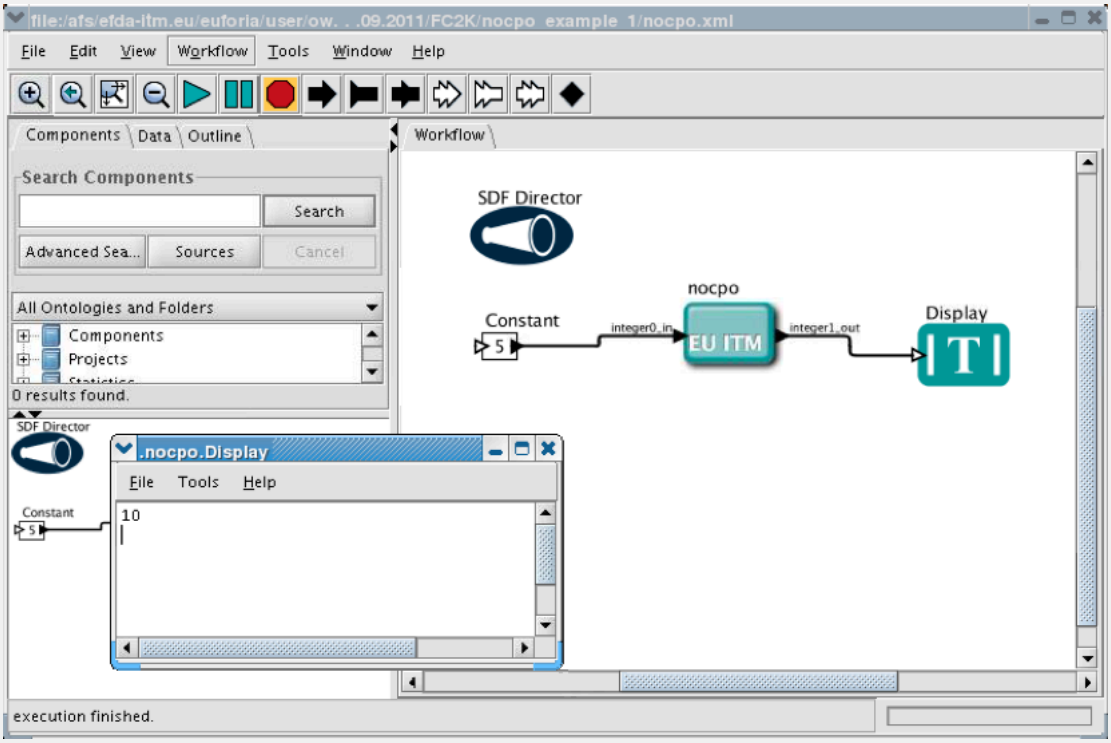
You should see similar workflow on screen.



You can start it, by pressing "Play" button



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



11. Exercise no. 1 finishes here.

3.2 C++ code within Kepler

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

Exercise no. 2 (approx. 30 min)

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

1. Get familiar with codes that will be incorporated into Kepler

Go to Code Camp related materials within your home directory

```
cd ~/public/garching-09.2011/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. Build the code by issuing

```
make clean
make
```

Codes are ready to be used within FC2K

3. Prepare environment for FC2K

Make sure that all required system settings are correctly set

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
```

4. Start FC2K application

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

```
fc2k
```

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

Default settings

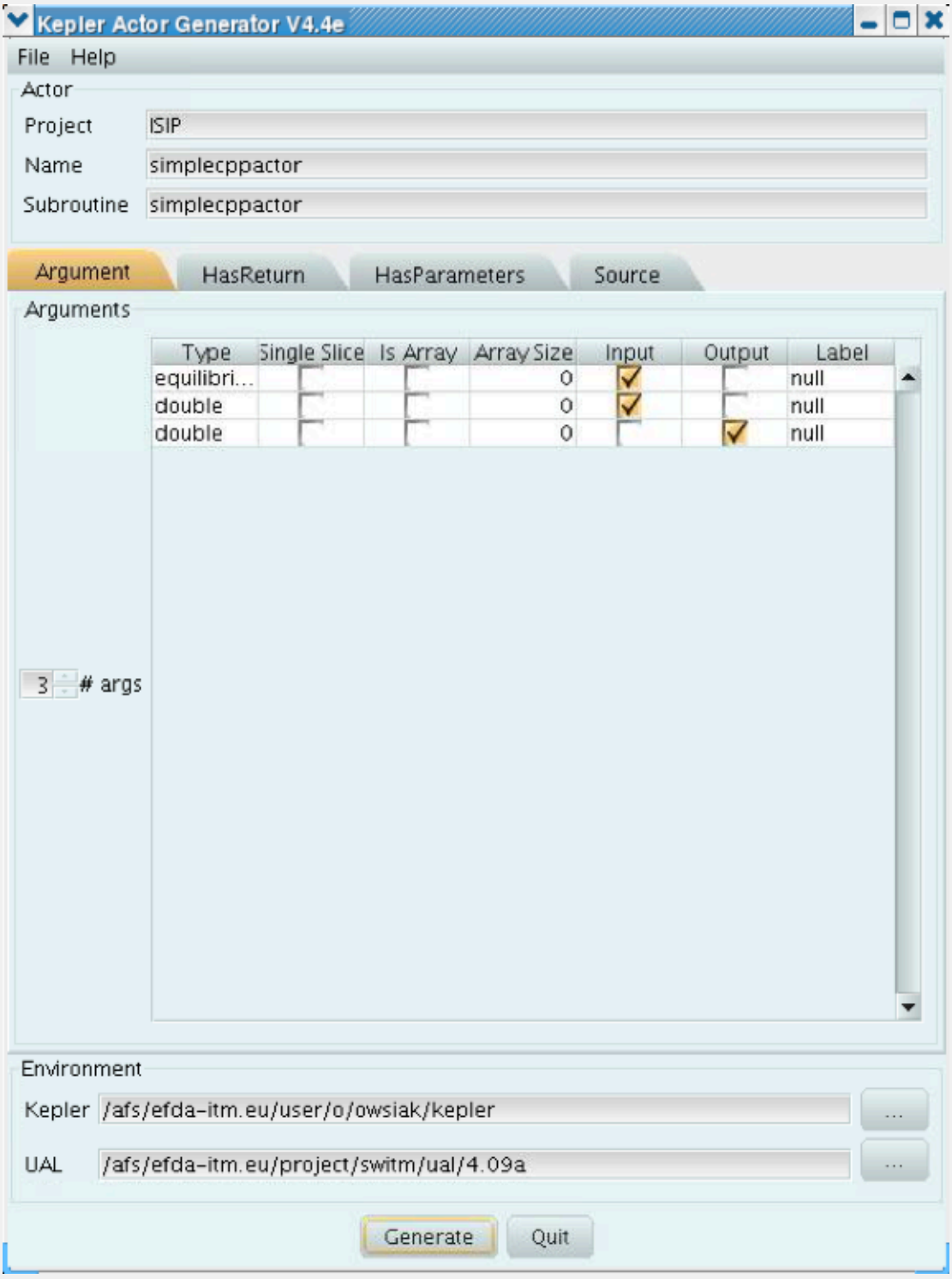
Note, that your settings will be slightly different. Your **Kepler** location should point to a valid path for your account.

5. Open existing parameters settings

Choose **File -> Open** and navigate to **~/public/garching-09.2011/FC2K/simplecppactor**. Open file **simplecppactor_fc2k.xml**. You should see new parameter settings loaded into FC2K.

6. Make sure that Kepler location is correct

After loading parameters you can notice that parameters point to locations within my home directory (**~owsiak**).

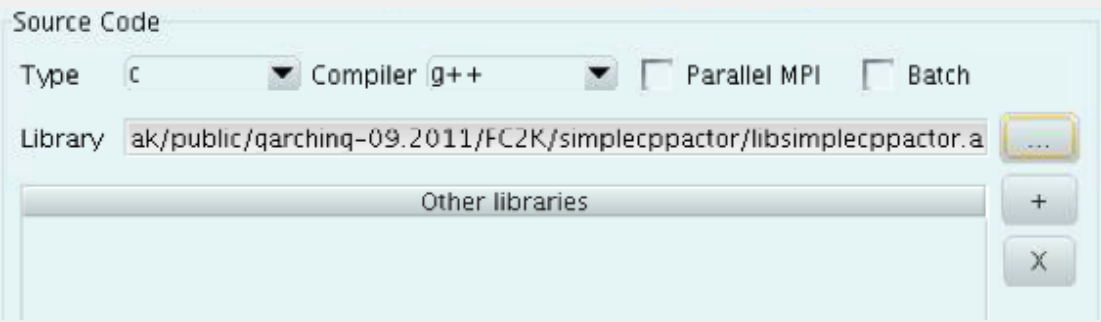


You should modify these setting, so they point to locations within you home directory. They will typically be as follows:

```
$HOME/kepler
```

7. Make sure that **library** location is correct

After loading parameters you can notice that library location points to location within my home directory (**~owsiak**).



You should modify this location, so it points to location of the library within your public directory. It should point to:

```
~/public/garching_09.2011/FC2K/simplecppactor/libsimplecppactor.a
```

8. After all the settings are correct, you can generate actor

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

> **Generating an actor for the second time**
>
> This tutorial assumes that Gateway accounts will be used for starting up Kepler application.
> If you are not familiar with NX setup for the Gateway, take a look at following location NX setup

9. Confirm Kepler compilation

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

10. You can now start Kepler and use generated actor

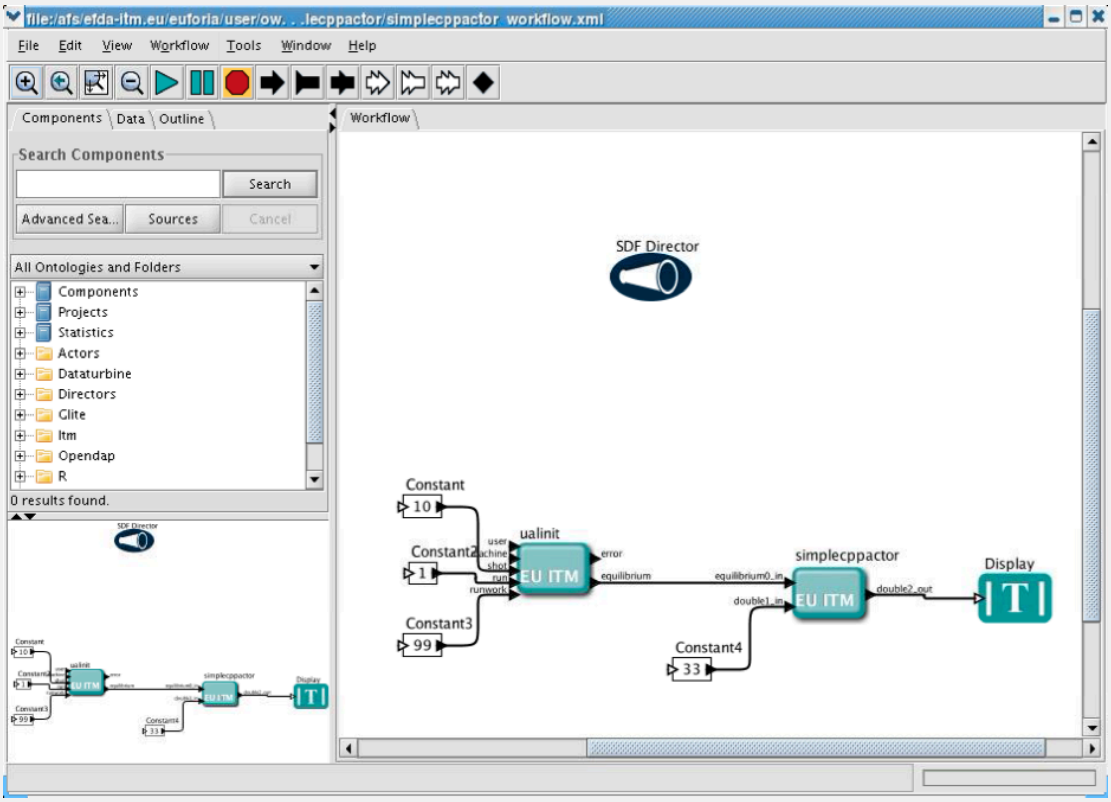
Open new terminal window and make sure that all environment settings are correctly set and execute Kepler.

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

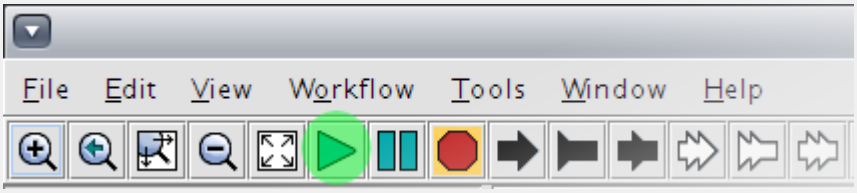
After Kepler is started, open example workflow from the following location

```
~/public/garching-09.2011/FC2K/simplecppactor/simplecppactor_workflow.xml
```

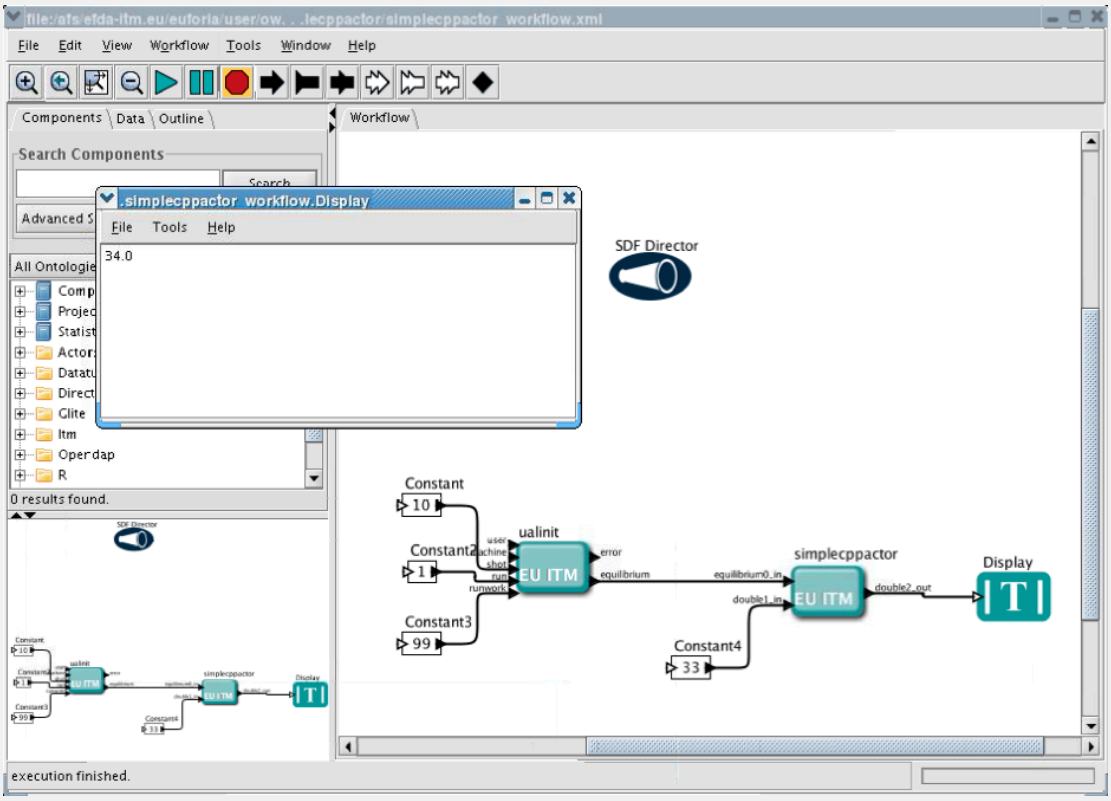
You should see similar workflow on screen.



You can start it, by pressing "Play" button



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



11. Exercise no. 2 finishes here.

4. Fortran UAL example

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

Exercise no. 3 (approx. 30 min)

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:

1. Get familiar with codes that will be incorporated into Kepler

Go to Code Camp related materials within your home directory

```
cd ~/public/garching-09.2011/FC2K/coreprof2mhd
```

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

- coreprof2mhd.f90 - Fortran source code that will be executed from Kepler - 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. Build the code by issuing

```
make clean
make
```

Codes are ready to be used within FC2K

3. Prepare environment for FC2K

Make sure that all required system settings are correctly set

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
```

4. Start FC2K application

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

```
fc2k
```

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

Default settings

Note, that your settings will be slightly different. Your **Kepler** location should point to a valid path for your account.

5. Open existing parameters settings

Choose **File -> Open** and navigate to **~/public/garching-09.2011/FC2K/coreprof2mhd**. Open file **cposlice2cposlicef_fc2k.xml**. You should see new parameter settings loaded into FC2K.

6. Make sure that Kepler location is correct

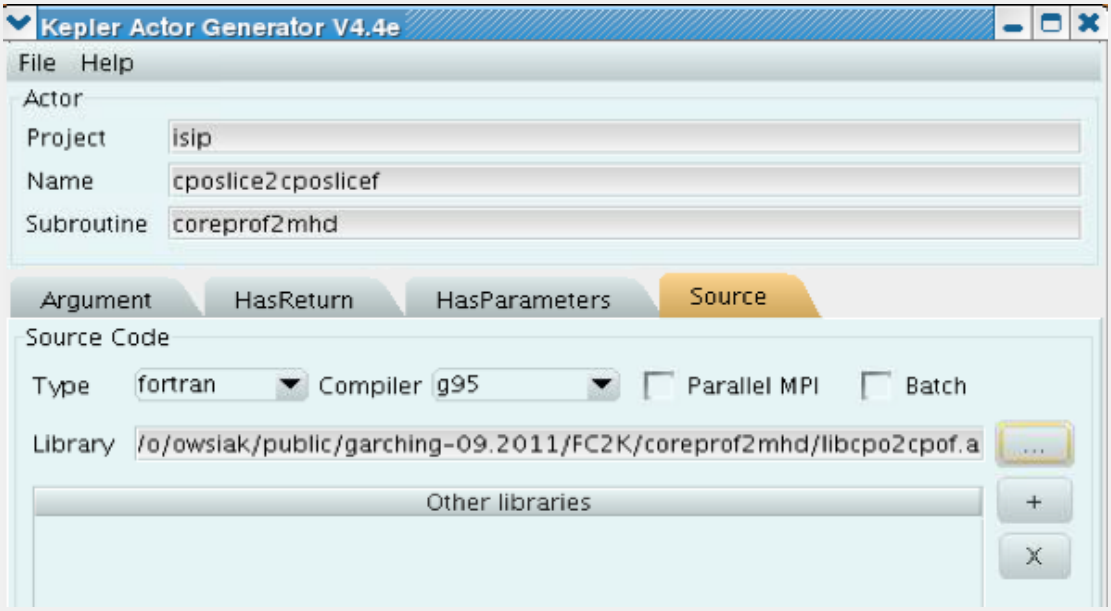
After loading parameters you can notice that parameters point to locations within my home directory (**~owsiak**).

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

```
$HOME/kepler
```

7. Make sure that **library** location is correct

After loading parameters you can notice that library location points to location within my home directory (**~owsiak**).



You should modify this location, so it points to location of the library within your public directory. It should point to:

```
~/public/garching-09.2011/FC2K/coreprof2mhd/libcpo2cpof.a
```

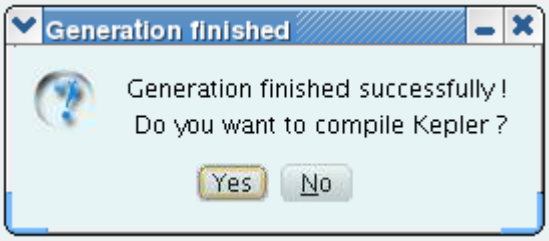
8. After all the settings are correct, you can generate actor

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

> **Generating an actor for the second time**
>
> This tutorial assumes that Gateway accounts will be used for starting up Kepler application.
> If you are not familiar with NX setup for the Gateway, take a look at following location NX setup

9. Confirm Kepler compilation

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



10. You can now start Kepler and use generated actor

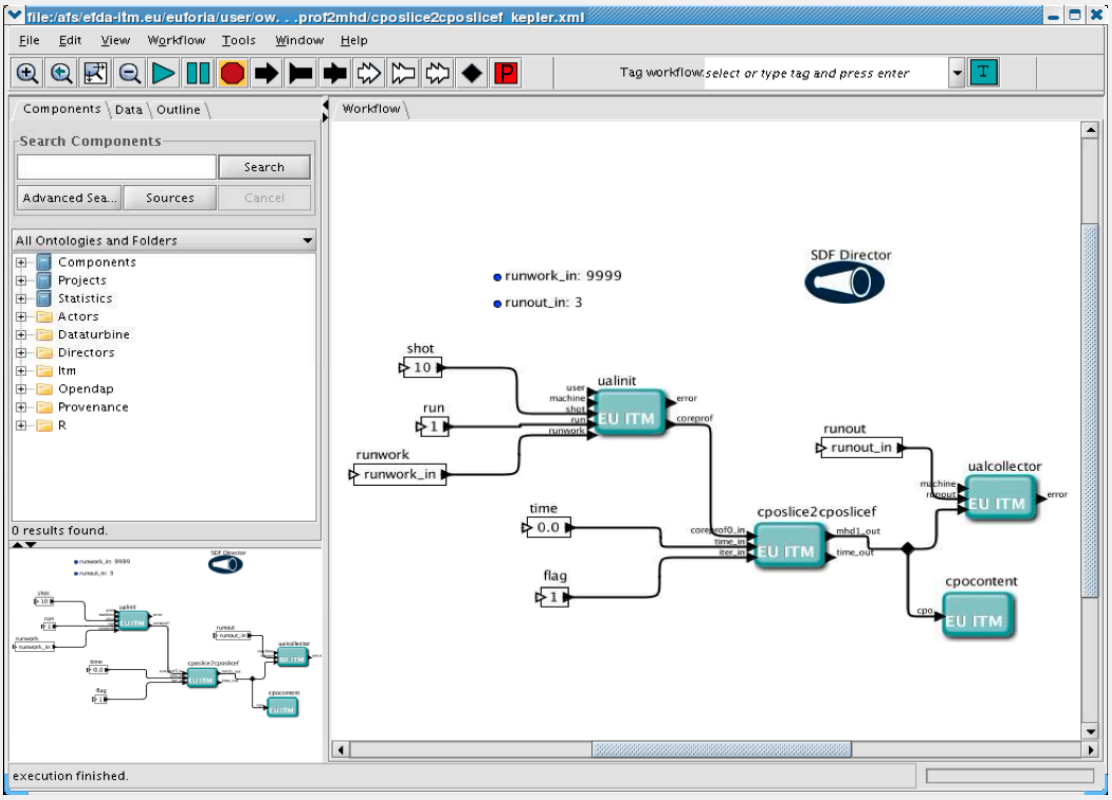
Open new terminal window and make sure that all environment settings are correctly set and execute Kepler.

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

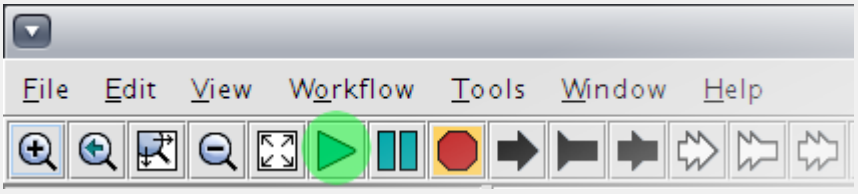
After Kepler is started, open example workflow from the following location

```
~/public/garching-09.2011/FC2K/coreprof2mhd/cposlice2cposlicef_kepler.xml
```

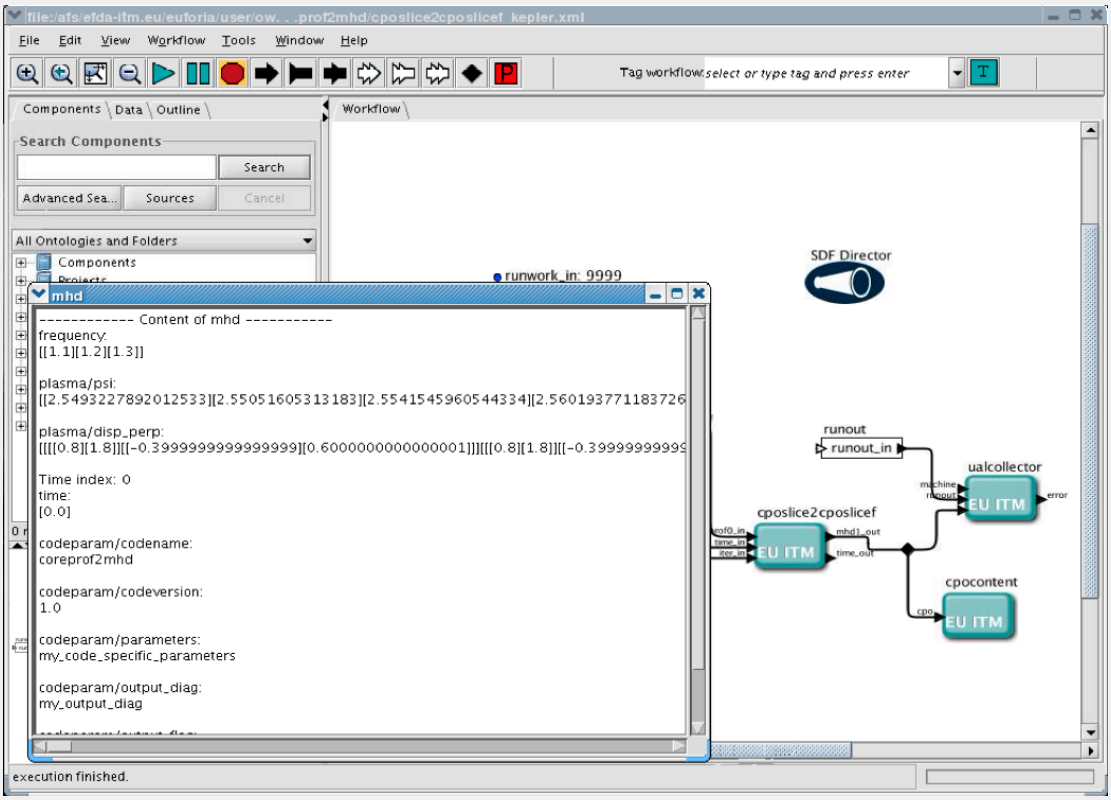
You should see similar workflow on screen.



You can start it, by pressing "Play" button



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



11. Exercise no. 3 finishes here.

5. Data visualization within Kepler using demux actor

After this exercise you will:

- know how to use demux actor
- know how to visualize data using Kepler actors

Exercise no. 4 (approx. 30 min)

In this exercise you will execute simple Kepler workflow that uses demux actor.

1. Prepare input data

In order to use the actor, you have to create data set. This can be done using **put_cpo.py** script. Make sure that ITM script was executed and start **put_cpo.py**

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
cd ~/public/garching-09.2011/visualization
python put_cpo.py
```

2. Start Kepler application

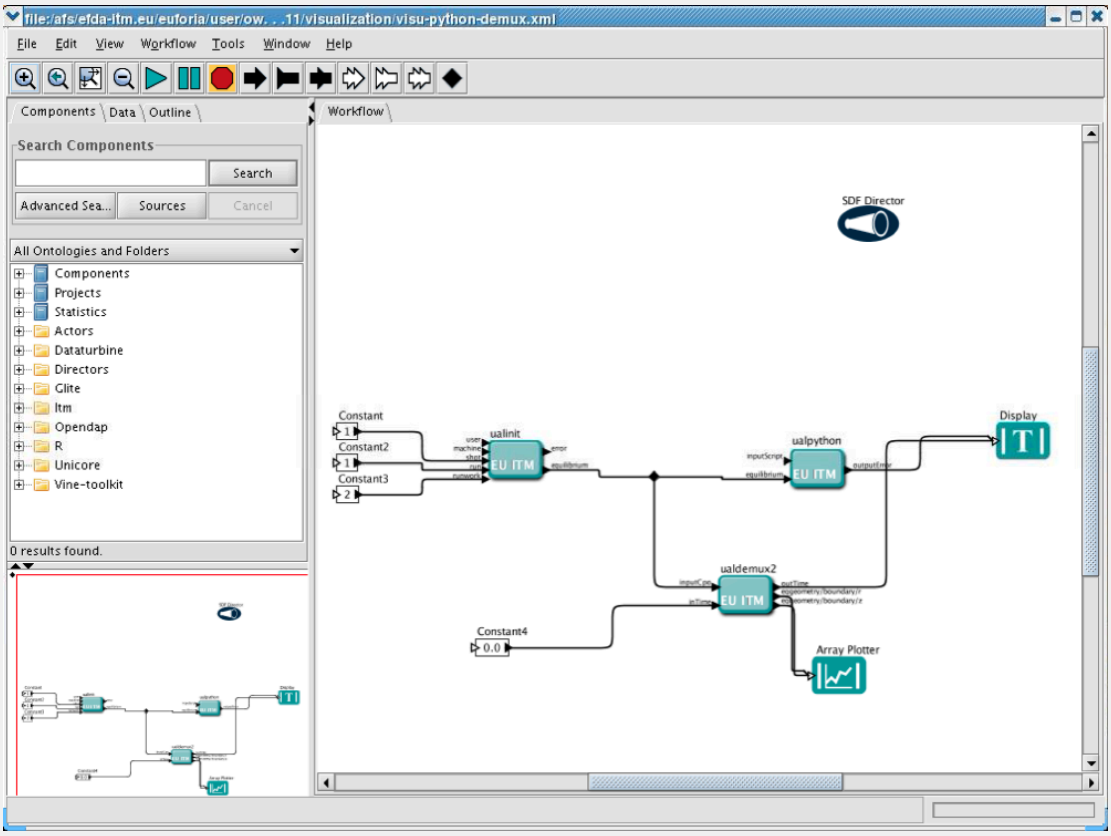
```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

3. Open example workflow

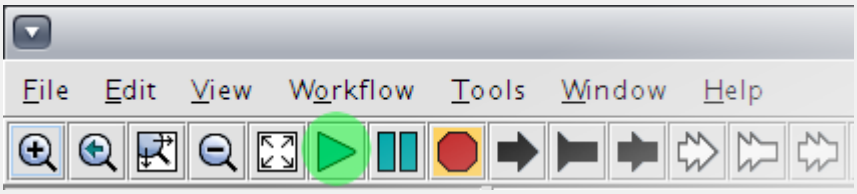
Choose **File -> Open File** and open following file:

```
~/public/garching-09.2011/visualization/visu-python-demux.xml
```

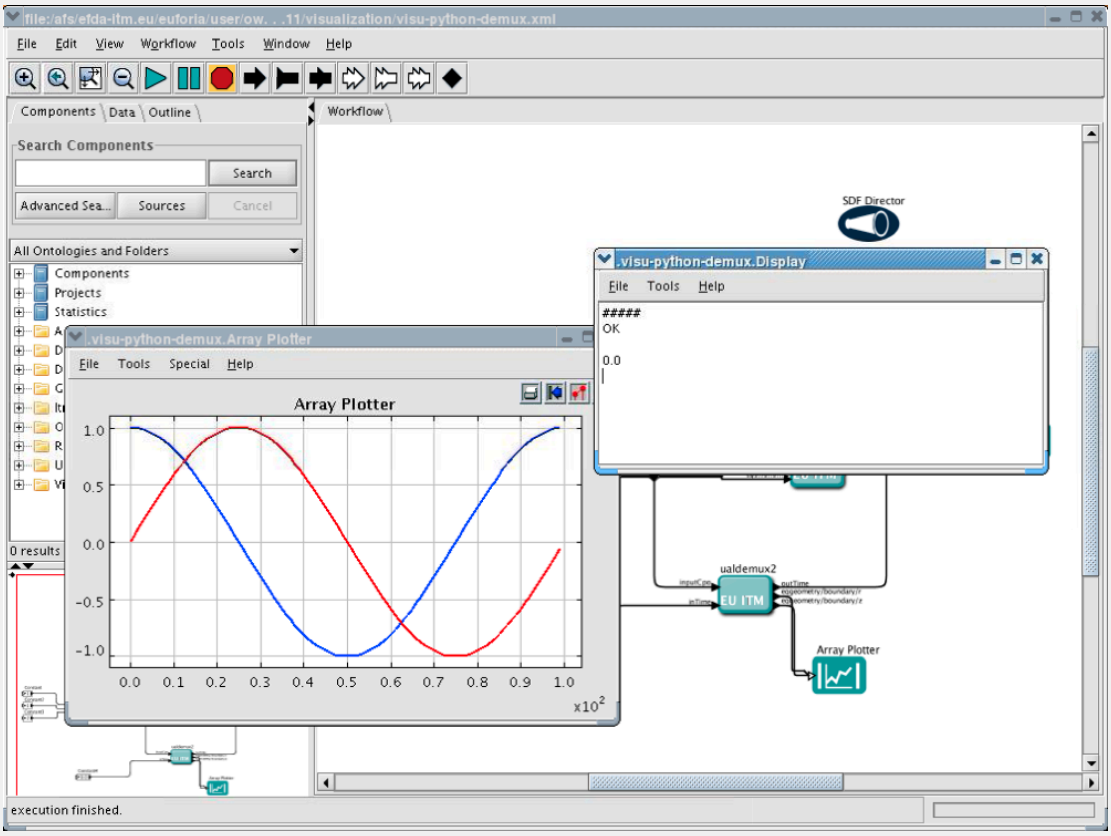
You should see workflow similar to one below:



Execute workflow by pressing "Play" button:



After workflow is finished, you should see image similar to one below:



4. Exercise no. 4 finishes here.

cp ~owsiak/public/itmdb/itm_trees/test/4.09a/mdsplus/0/* ~/public/itmdb/itm_trees/test/4.09a/mdsplus/0

4.1 Tutorial - ISE - visualizing data (Garching 09.2011)

Table of contents

Video

There is a Video material related to this section: movie, movie (for Safari browser)

Integrated Simulation Editor

Integrated Simulation Editor is available to Gateway users via ***ise*** command. It allows to:

- Visualize and edit the values of a simulation in the current database
- Associate the dataset with a Kepler worflow
- Run Kepler within ISE
- Follow the evolution of some parameters during the execution of the workflow
- Display the results with Matlab or Scilab

However, it has few restrictions:

- Visualize only 1D and 2D data
- ISE not useful for huge simulations

(source: Introduction to ISE by J. Signoret and P. Huynh)

In this tutorial section you will get familiar with basic features of ISE.

1. Starting editor

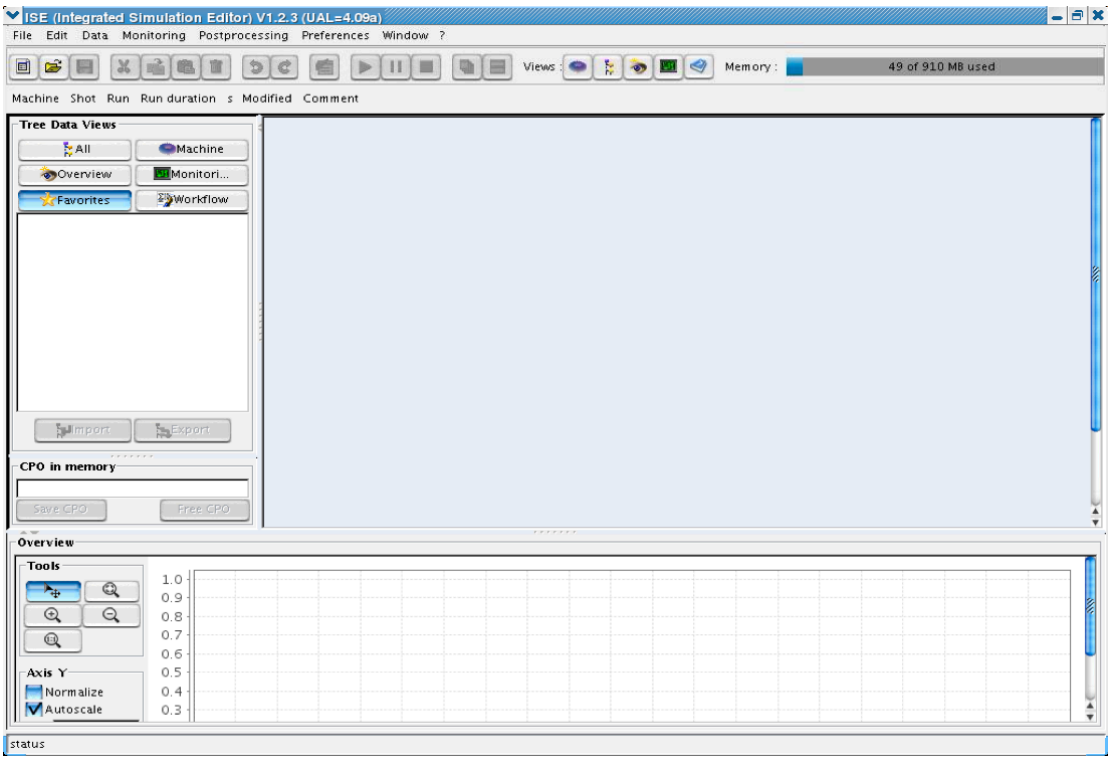
In order to start ISE you have to make sure that:

- database structure was created for your account
- you have executed ITMv1 script

If you want to start ISE, follow the instruction below:

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null

echo "Creating database structure is required only in case you haven't done it before"
/afs/efda-itm.eu/project/switm/scripts/create_user_itm_dir test 4.09a

ise
```

After a while, you should see ISE main window.

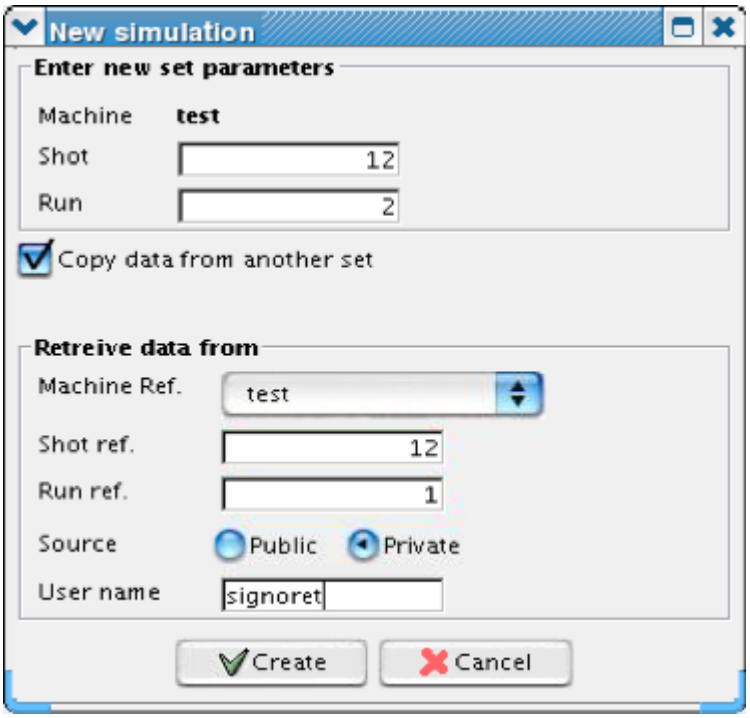


2. Adding example data

In this tutorial we will base on private data from 4.09a database. We will use following import settings:

```
Shot: 12
Run: 2
[x] Copy data from another set
Shot: 12
Run: 1
User: signoret
Source: private
```

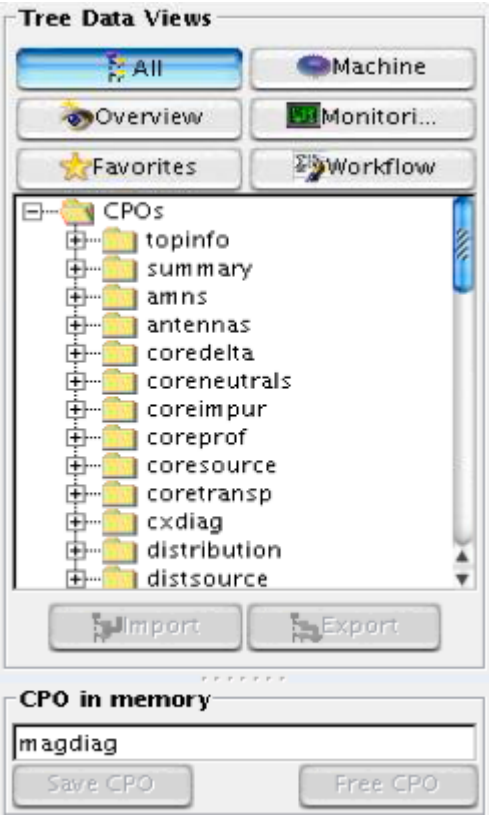
In order to import data, you have to choose: File -> New. You should see window similar to this one:



You should see the same window as shown at the picture and press ***Create***

> **Machine Ref.**
>
> If you don't see "test" at the list of machine names simply click the Combo Box and type it in for yourself: movie, movie (for Safari browser)

After a while, data will be imported. You can see data tree on the left panel, by pressing ***All***

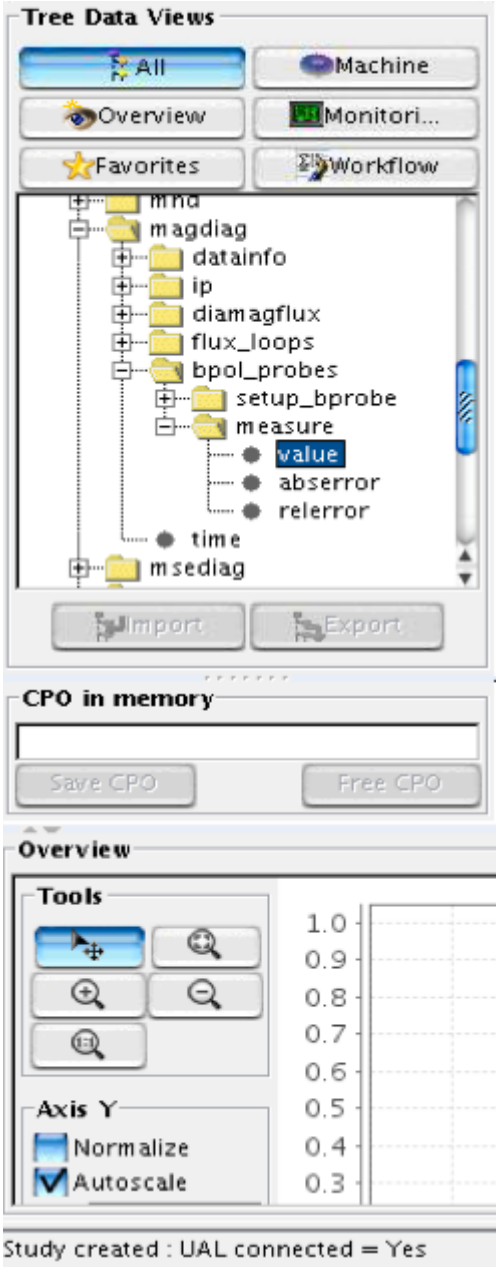


3. Browsing the data

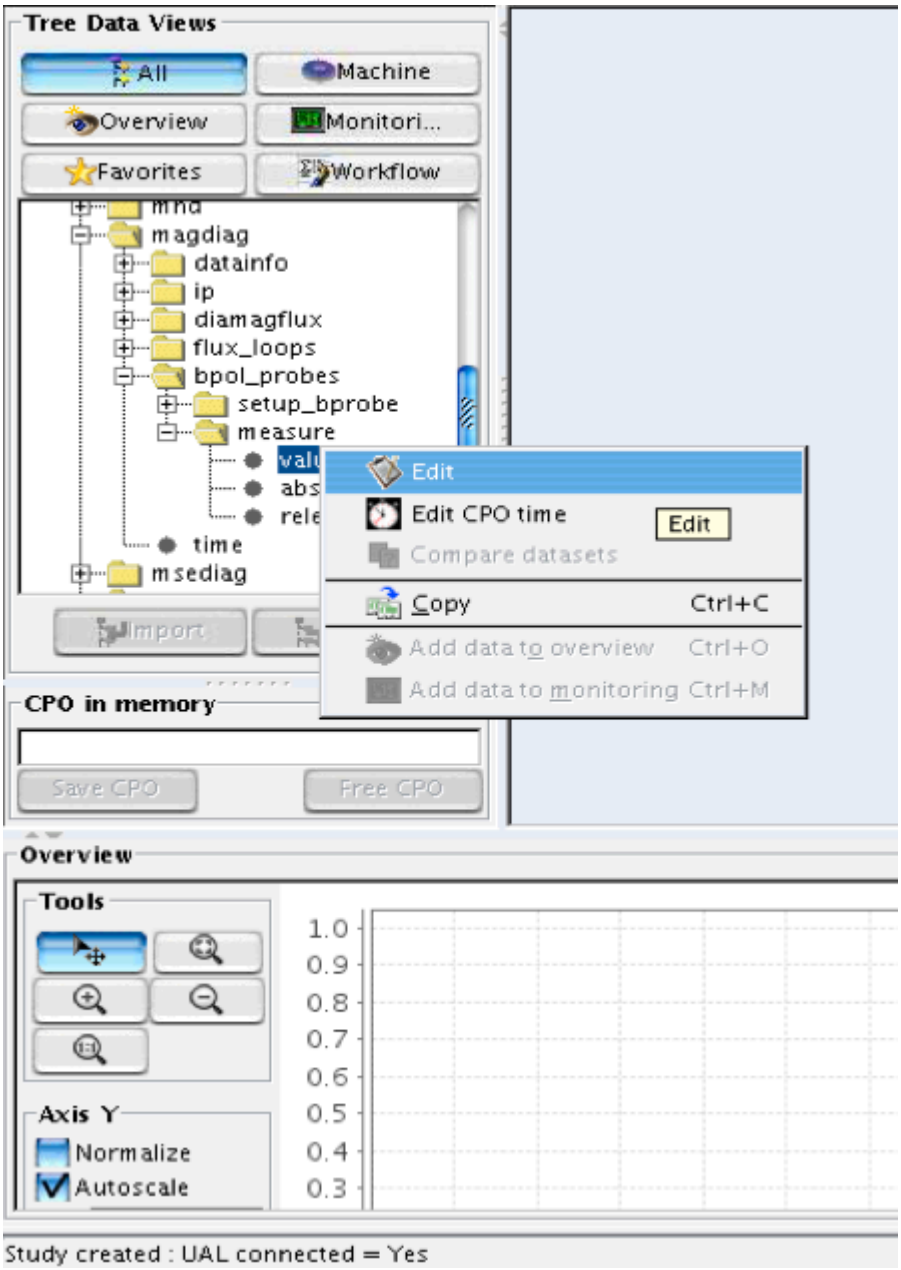
With ISE, you can visual data nodes by choosing particular node and entering ***Edit*** mode. In this example we will visualize node:

Node name
magdiag/bpol_probes/measure/value

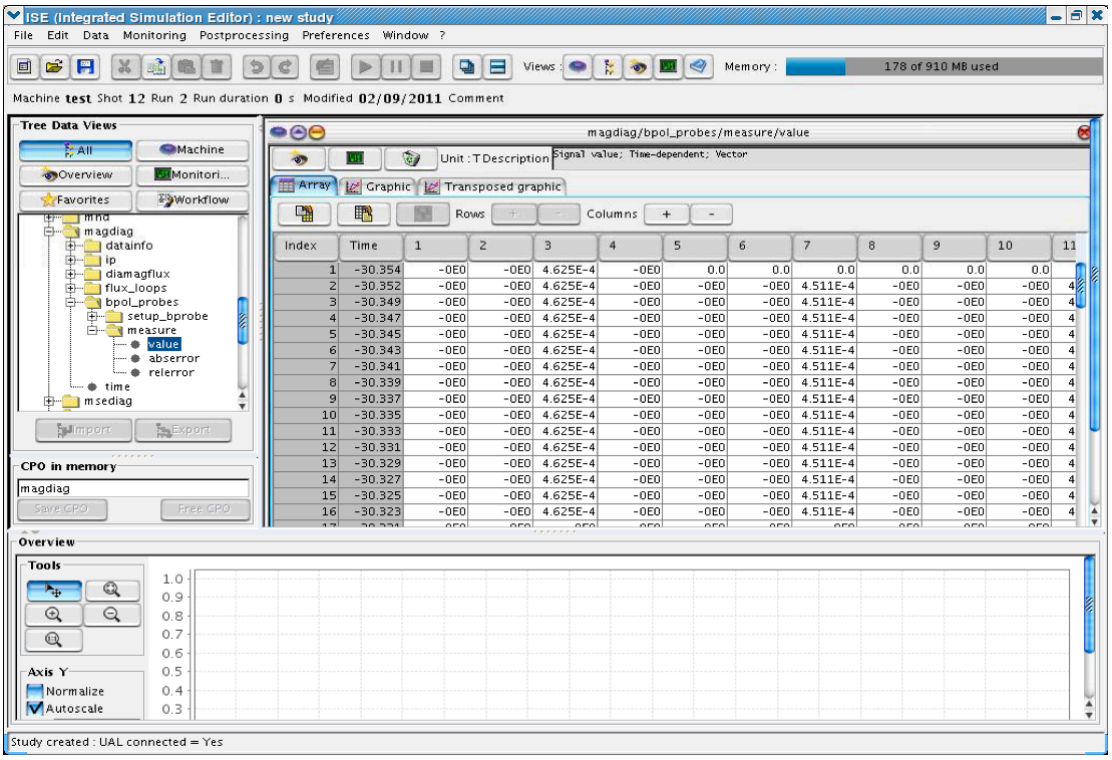
Select this node, by choosing: Tree Data Views and navigate to node: ***magdiag/bpol_probes/measure/value***. You should end up with situation like this:



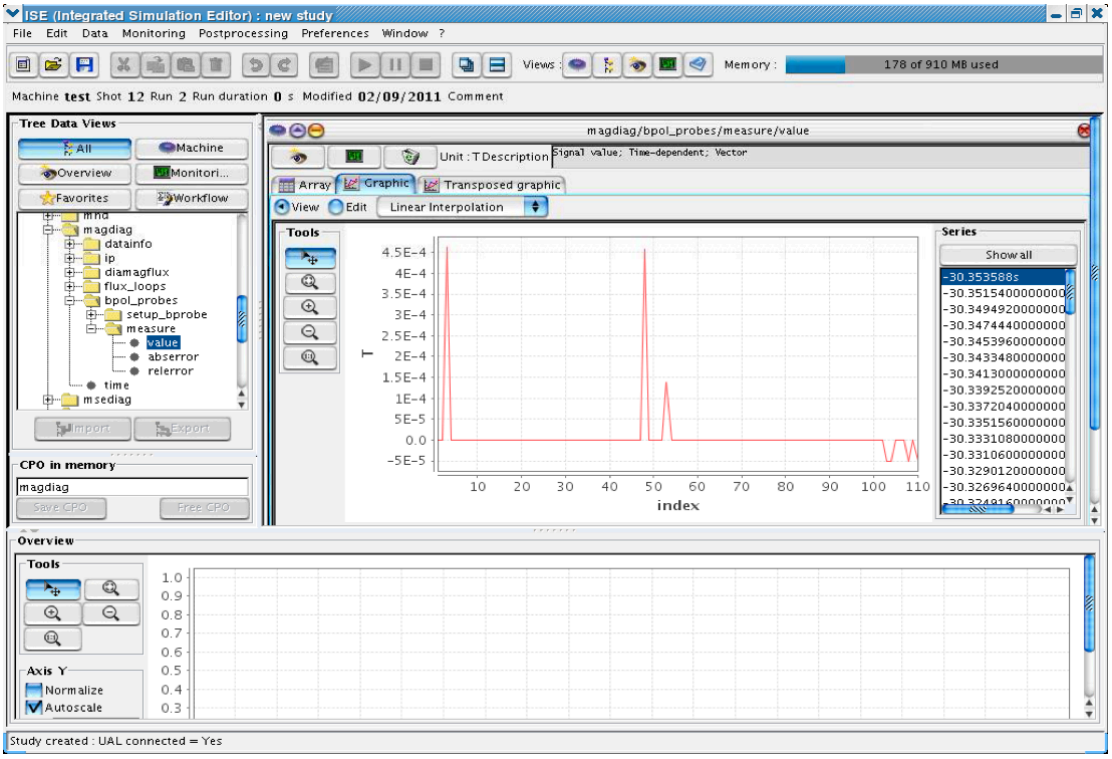
Right-click node "value" and choose "Edit" item from the context menu.



This particular node will be visualized. You can either view data in numeric form:



Or visualize them.



4.2 Tutorial - ISE - executing Kepler workflows (Garching 09.2011)

Table of contents

Integrated Simulation Editor - executing Kepler workflows

Integrated Simulation Editor is available to Gateway users via ***ise*** command. It allows to:

- Run Kepler within ISE
- Follow the evolution of some parameters during the execution of the workflow
- Display the results with Matlab or Scilab

(source: Introduction to ISE by J. Signoret and P. Huynh)

In this tutorial section you will get familiar with execution of Kepler workflows from ISE.

1. Starting editor

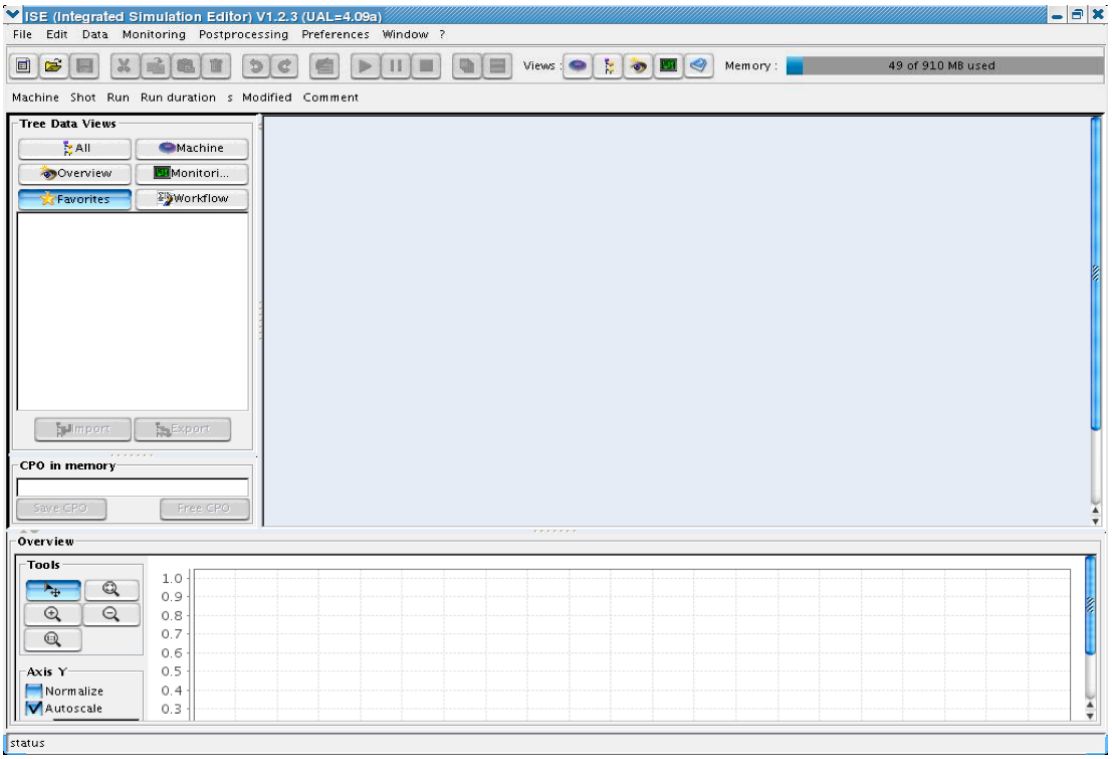
In order to start ISE you have to make sure that:

- database structure was created for your account
- you have executed ITMv1 script
- you have imported actors used during tutorial

If you want to start ISE, follow the instruction below:

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null

echo "Creating database structure is required only in case you haven't done it before"
/afs/efda-itm.eu/project/switm/scripts/create_user_itm_dir test 4.09a

cd ~/public/garching-09.2011/ISE/actors

import_actor checktearing
import_actor equil2toroidfieldf
import_actor ntmDeff
import_actor ntmmodule

ise
```

After a while, you should see ISE main window.

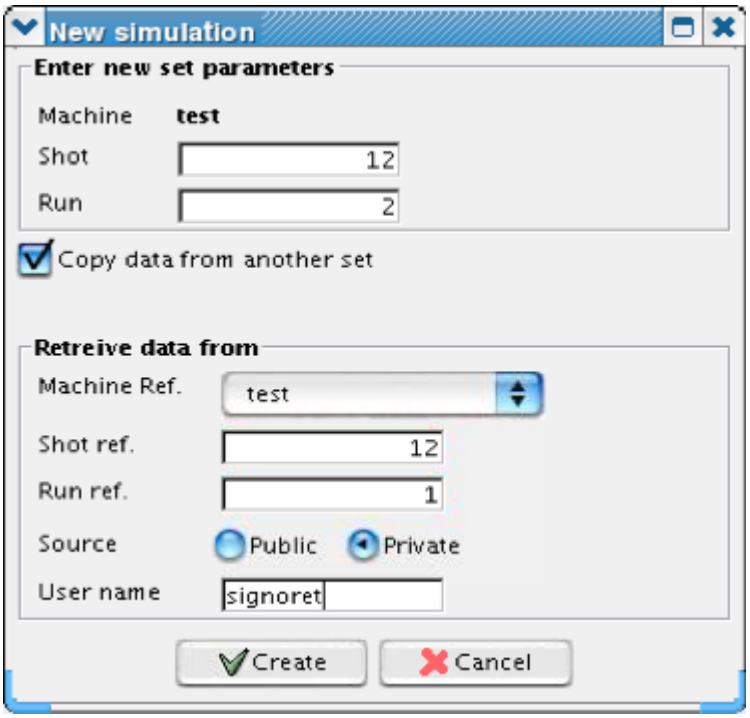


2. Adding example data

In this tutorial we will base on private data from 4.09a database. We will use following import settings:

```
Shot: 12
Run: 2
[x] Copy data from another set
Shot: 12
Run: 1
User: signoret
Source: private
```

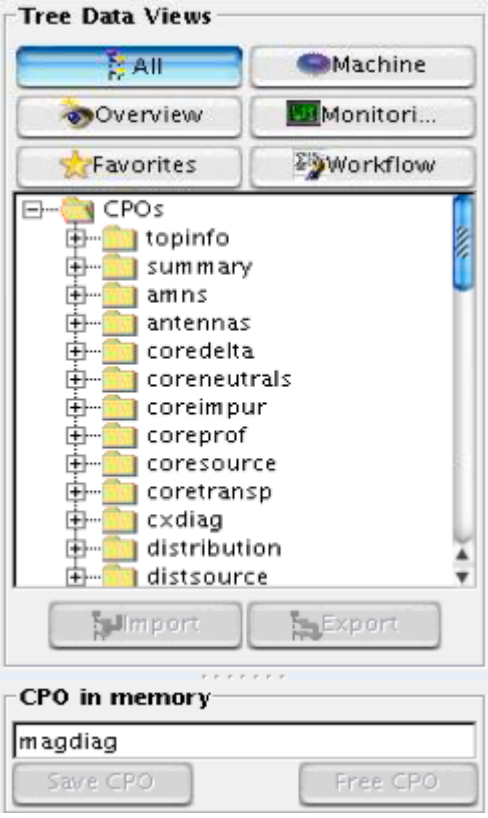
In order to import data, you have to choose: File -> New. You should see window similar to this one:



You should see the same window as shown at the picture and press ***Create***

> **Machine Ref.**
>
> If you don't see "test" at the list of machine names simply click the Combo Box and type it in for yourself: movie, movie (for Safari browser)

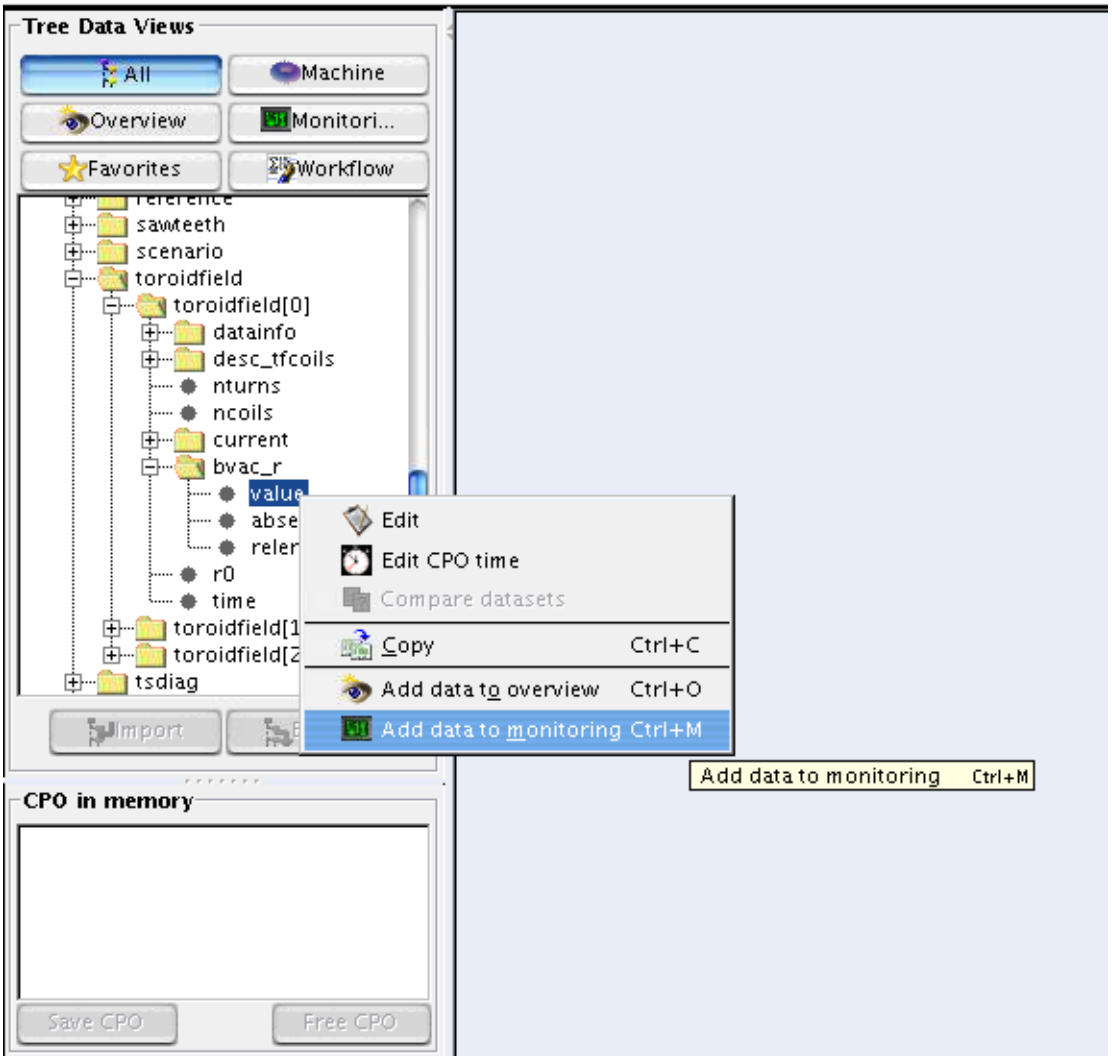
After a while, data will be imported. You can see data tree on the left panel, by pressing ***All***



3. Monitoring values

Values for the loaded data can be monitored during workflow execution. In case of this, demo, workflow we will monitor value of: ***toroidfield/torofield[0]/bvac_r/value*** node.

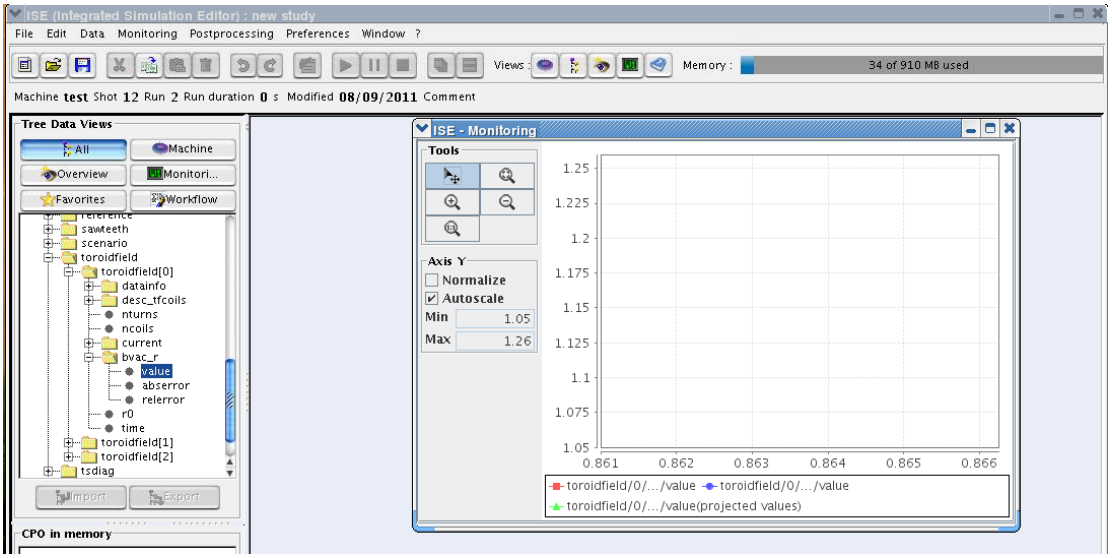
In order to add this node into Monitoring choose the node, right click it and select "Add data to monitoring"



4. Enabling Monitoring Dialog

Monitoring Dialog can be easily enabled via option at menu bar: ***"Show/Hide Monitoring"***

After you choose this option, you will notice Monitoring Dialog floating above the ISE's window

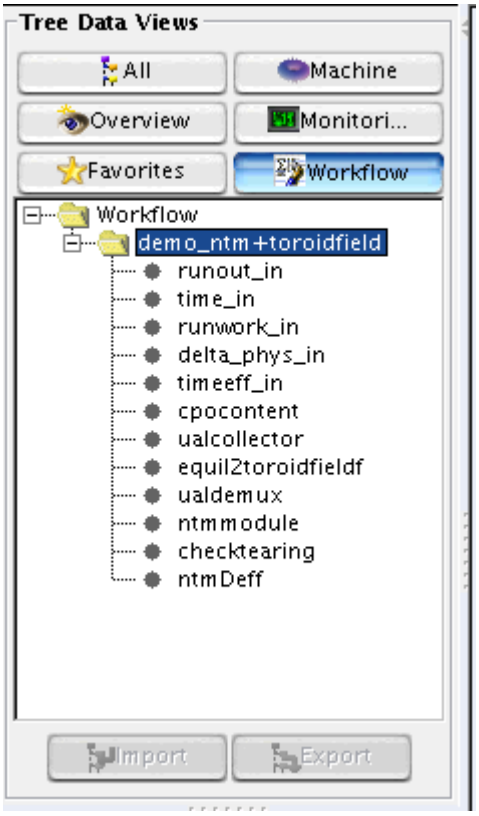


5. Loading workflow

With ISE, you can bind the workflow to a study. In order to do so, choose: **Data->Select Kepler Workflow** and navigate to:

```
~/public/garching-09.2011/ISE/demo_ntm+toroidfield.xml
```

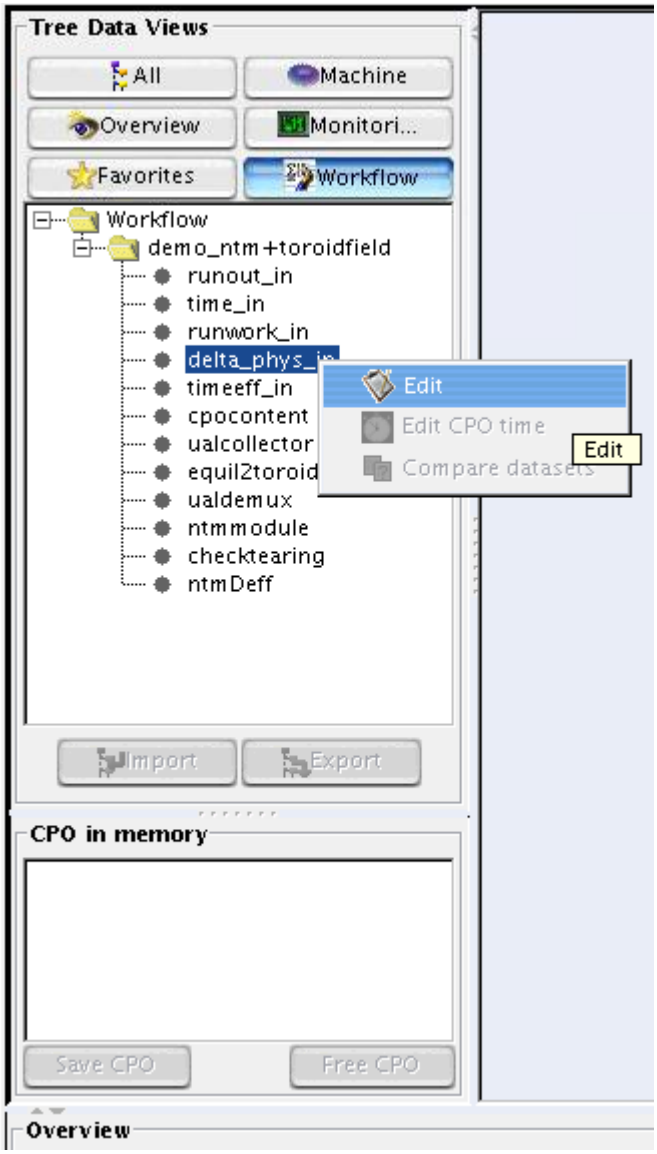
Select the file and load it. After file is loaded you should be able to see the workflow within **Workflow** tab.



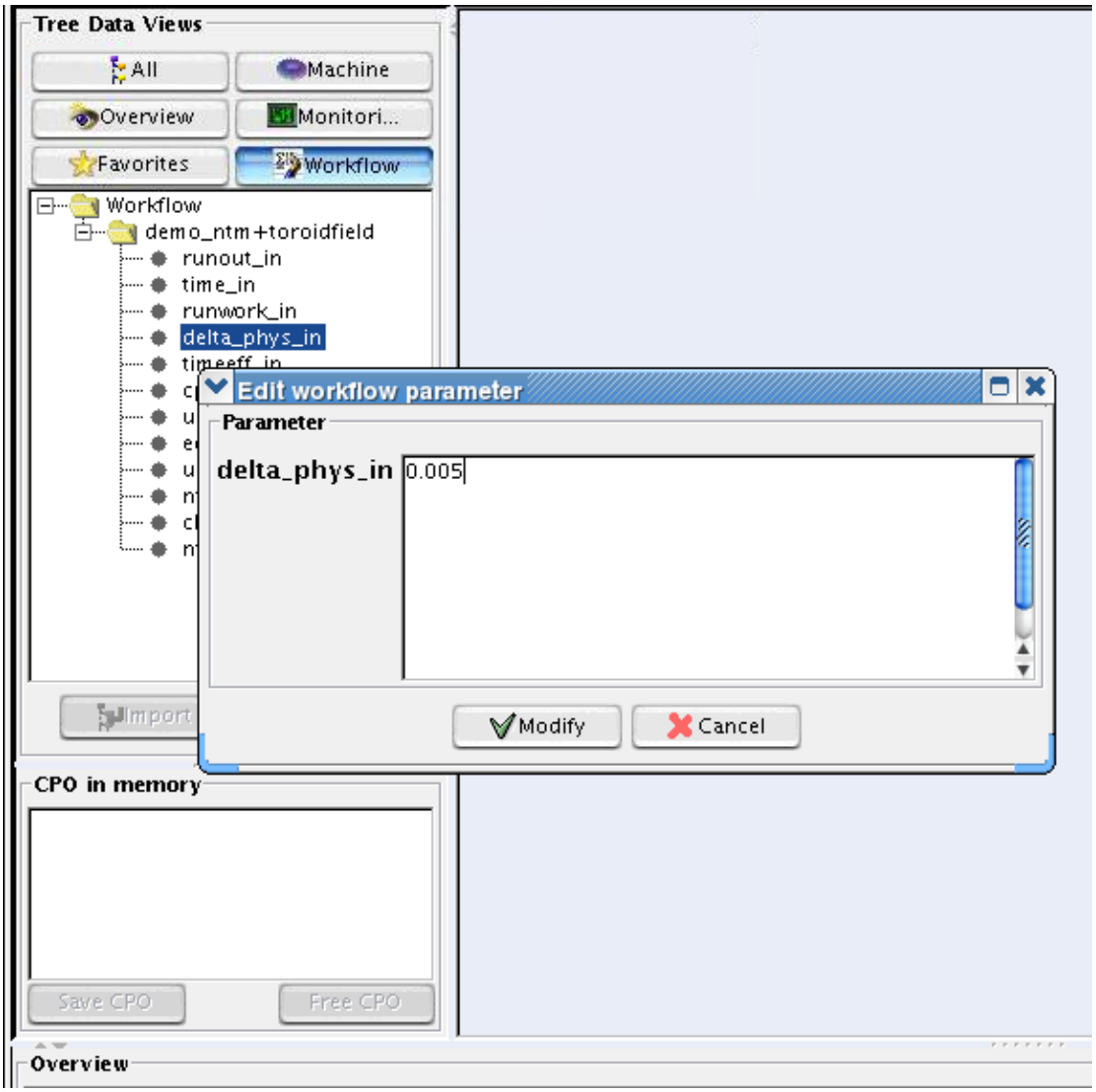
> **Editing parameters**
>
> ISE allows to change workflow's parameters (***blue dots***). However, you have to stick to a naming convention. Parameters must comply to following naming schema:
>
> **parameterprefix_*in***

6. Modification of parameters

In case of this study, we will modify value of ***delta_phys_in*** parameter - it is mandatory to change it's value to "0.005". Select the parameter and choose: ***Edit***

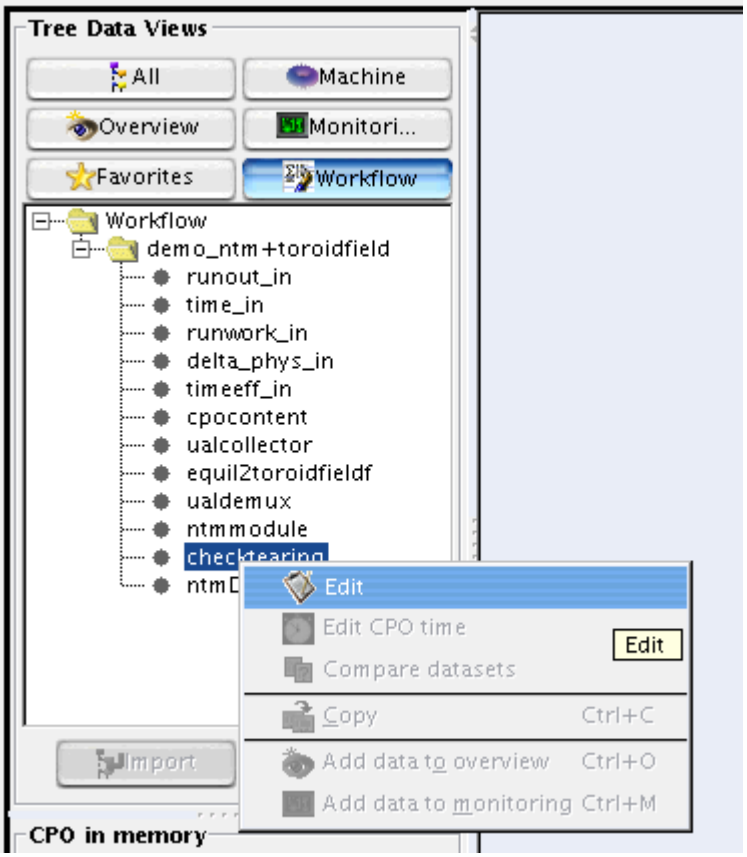


After Editor window opens, set the value of parameter to "0.005"

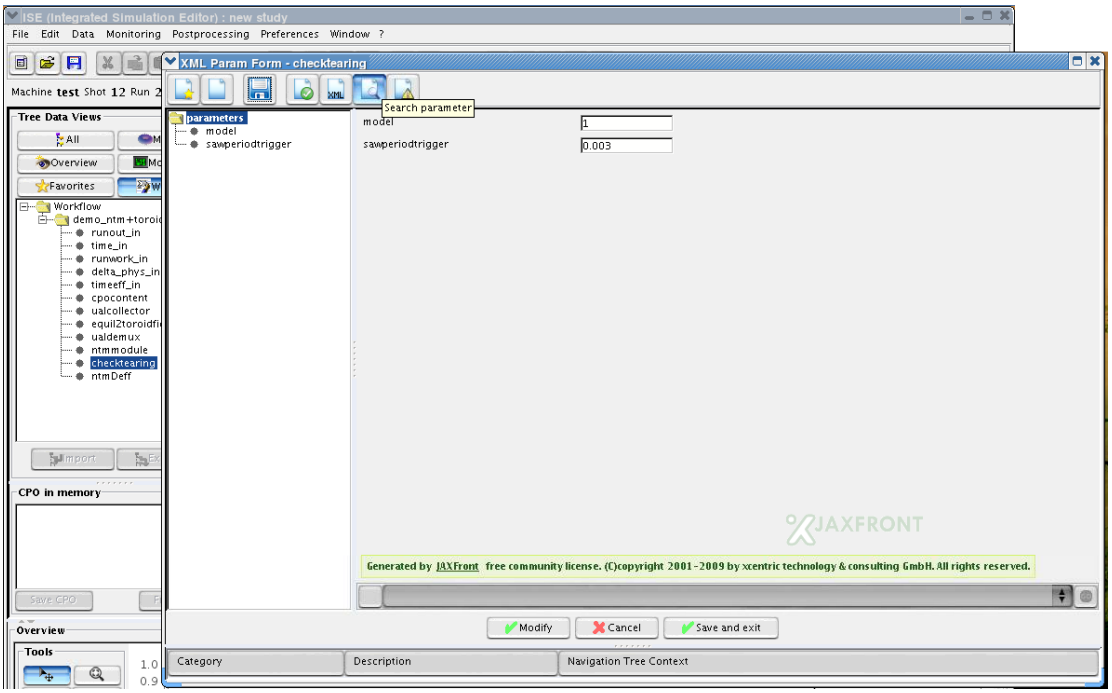


7. Modyfing actor's parameters

ISE allows you to modify actor's parameter as well. You can do this by selecting an actor and choosing ***Edit*** from the context menu. In this example, we will examine the values of ***checktearing*** actor. Select the actor, right-click on it, and choose "***Edit***"

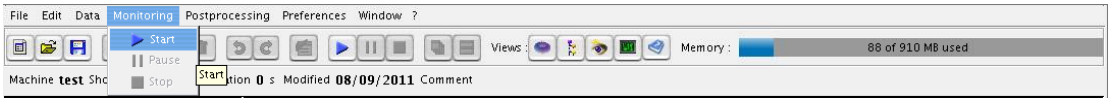


After the moment, dialog window should appear. This dialog allows you to modify actor's parameters directly from ISE.



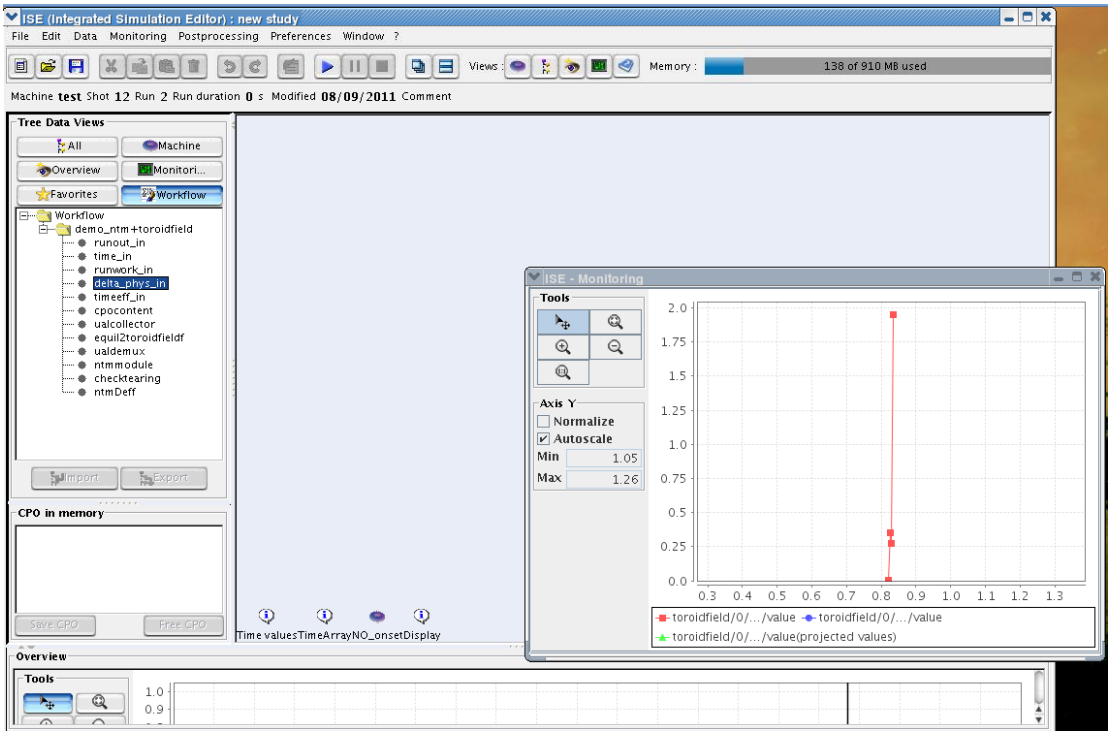
8. Starting the workflow

At this point, all the parameters are correctly set, data are loaded, workflow is bound to the study. We can start it by pressing "Play" button at the top of the screen



> **Save the study**
>
> Before starting the workflow, ISE will ask whether study should be saved. It is good a habit to save the study before starting the workflow. This way, you can easily go back to once configured settings.

After some time, you should see the result of execution.



5. Tutorial - HPC2K (Garching 09.2011)

HPC2K

Table of Contents

1. Introduction

This tutorial is designed to introduce the concept of building ITM tools based workflows within Kepler.

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.

NX connection to the Gateway

This tutorial assumes that Gateway accounts will be used for starting up Kepler application.
If you are not familiar with NX setup for the Gateway, take a look at following location NX setup

2. Requirements for the tutorial

Backing up Kepler home directory

Before you proceed with installation of the Kepler application be sure to make a backup of your Kepler home directory

```
mv ~/.kepler ~/.kepler_09_2011
mv ~/kepler ~/kepler_09_2011
mv ~/serpens ~/serpens_09_2011
```

2.1 Using ITM Kepler installation at Gateway

In order to make Kepler installation for the tutorial faster we will use preinstalled version of the Kepler that is available for Gateway users.

In order to install Kepler and ITM example workflow you have to follow instructions at following page:

Kepler installation

1. Kepler installation at Gateway (Garching 09.2011)

After you follow all the installation steps, you should see Kepler loading.

Starting Kepler

No matter which way have you used to install Kepler, make sure to export some variables before you start Kepler again.

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

3. VOMS proxy

Your certificate and key

Certificates and keys are preinstalled in `$HOME/serpens/core/cert/`. During this tutorial, you will be given an id.
Please execute:

```
cp ~/serpens/core/cert/POZNAN##-cert.pem ~/usercert.pem
cp ~/serpens/core/cert/POZNAN##-key.pem ~/userkey.pem
```

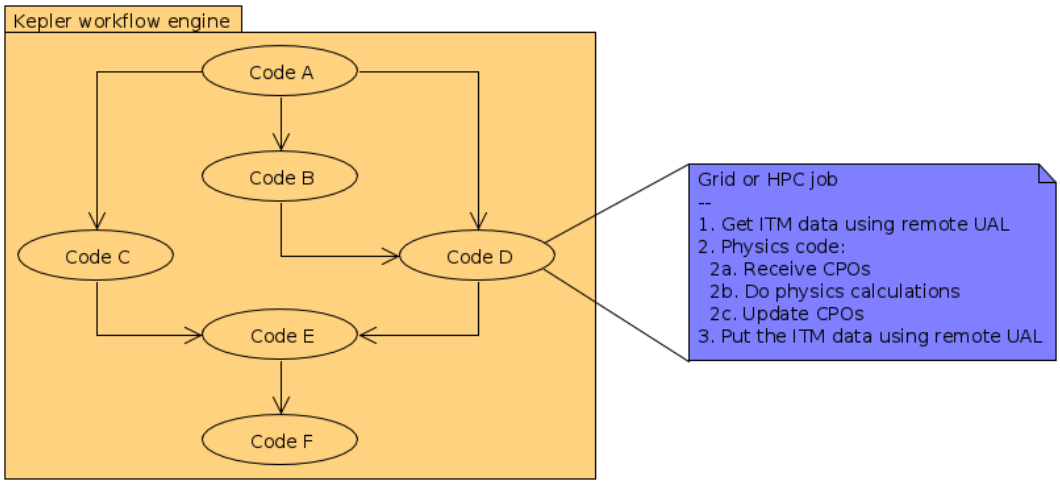
4. HPC2K

HPC2K is a tool designed to create a Kepler actor which:

- uploads the input files,
- submits the job,
- gets its status,
- downloads the output files.

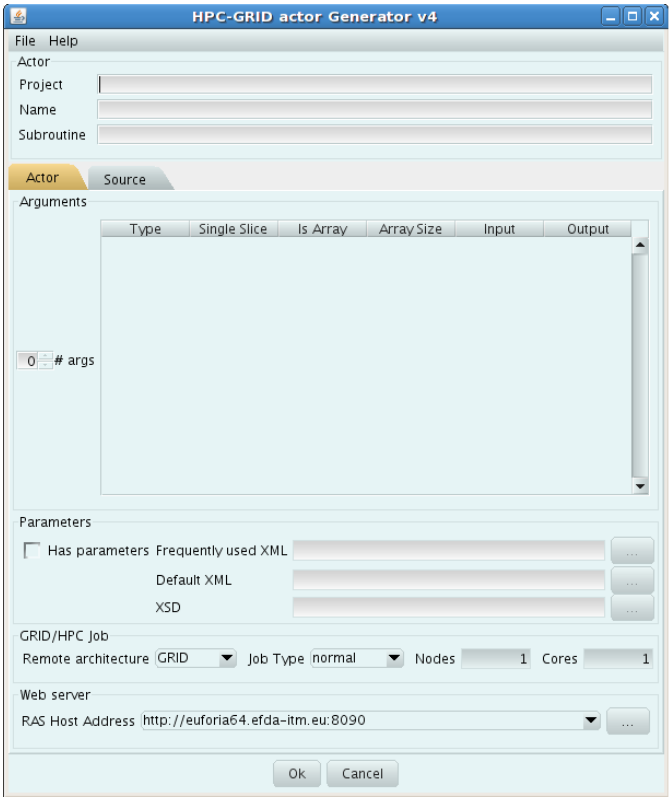
The job can be either run on an High Performance Computer (HPC) or in a grid (distributed computing environment). The job is based on user code (Fortran or C++) which accesses the CPO data through the UAL. This way, once your code is CPO-compatible, you can easily generate Kepler actors which will run this code on HPC or grid computing environments.

This "single actor for single code" solution allows you to design complex workflows with sophisticated dependencies between components. This concept is shown on the following diagram:



Furthermore, in the workflow you have to use only the CPO name, shot/run number, username and tokamak name. The CPOs themselves are not copied and spread through the workflow structure. Only the last components - the computing nodes - access the CPOs addresses earlier.

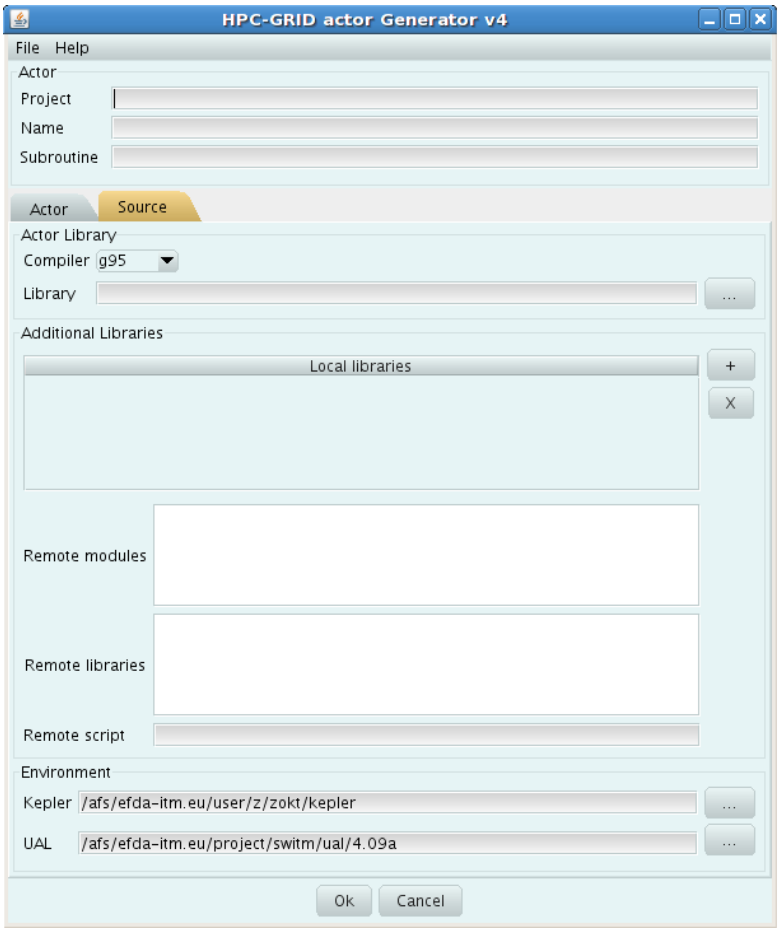
The HPC2K user interface is divided into two tabs:



Project, actor name, subroutine

Arguments

Grid or HPC?



Your compiled code

Additional libraries

Kepler and UAL

4.1 Submitting an HPC job

For the purpose of tutorial, we will run a simple code reading a CPO and sending a single integer to the output.

Exercise 1

1. Get the example code from the prepared location:

Contents of the example

$HOME/cpo2ipHPC/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

  write(*,*) ' cpo2ip: in the computation routine '
  write(*,*) 'time deb',equi_in(:)%time,size(equi_in)
  call flush(6)
  ip=23
  return

end subroutine cpo2ip
```

$HOME/cpo2ipHPC/Makefile

```
# ----------------------------------
# MAKEFILE FORTRAN 90 FOR cpo2ip
# ----------------------------------
F90=ifort
LIBSTDCPP=`g++ -print-file-name=libstdc++.so`
UAL= /lustre/jhome8/fsnaplm/fsaplm02/ual
OPTS = -g -O0 -assume no2underscore -fPIC -shared-intel
INCLUDES = -I$(UAL)/include/amd64_ifort
LIBS = -L$(UAL)/lib -lpthread $(LIBSTDCPP) -lrt
-lUALFORTRANInterface_ifort

all: cpo2ip.o libcpo2ip.a

OBJ_INTERPOL= cpo2ip.o

# ------------
# COMPILATION
# ------------
cpo2ip.o: cpo2ip.f90
        $(F90) $(OPTS) -c -o $@ $^ ${INCLUDES} $(LIBS)

libcpo2ip.a: $(OBJ_INTERPOL)
        ar -rv libcpo2ip.a cpo2ip.o
```

The code is already precompiled on HPC-FF and in the directory *cpo2ipHPC* you can find **libcpo2ip.a**. Please remember that in order to submit an HPC job basing on HPC2K tool, you will have to compile the codes on target machine and copy the generated library back to your local computer (i.e. the one running Kepler).

2. Run HPC2K.

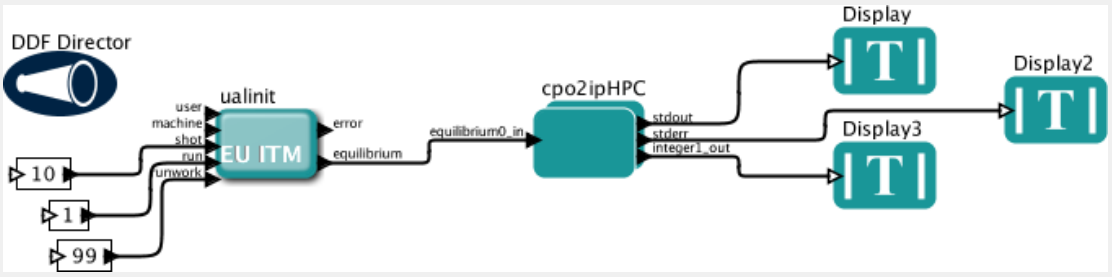
```
cd $HOME
hpc2k-hpc/actors/install $HOME/kepler
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a
>/dev/null
setenv HPC2K $HOME/hpc2k-hpc
hpc2k-hpc/hpc2k
```

3. Load the predefined configuration from `~/cpo2ipHPC/cpo2ipHPC.xml` via *File->Open* menu.

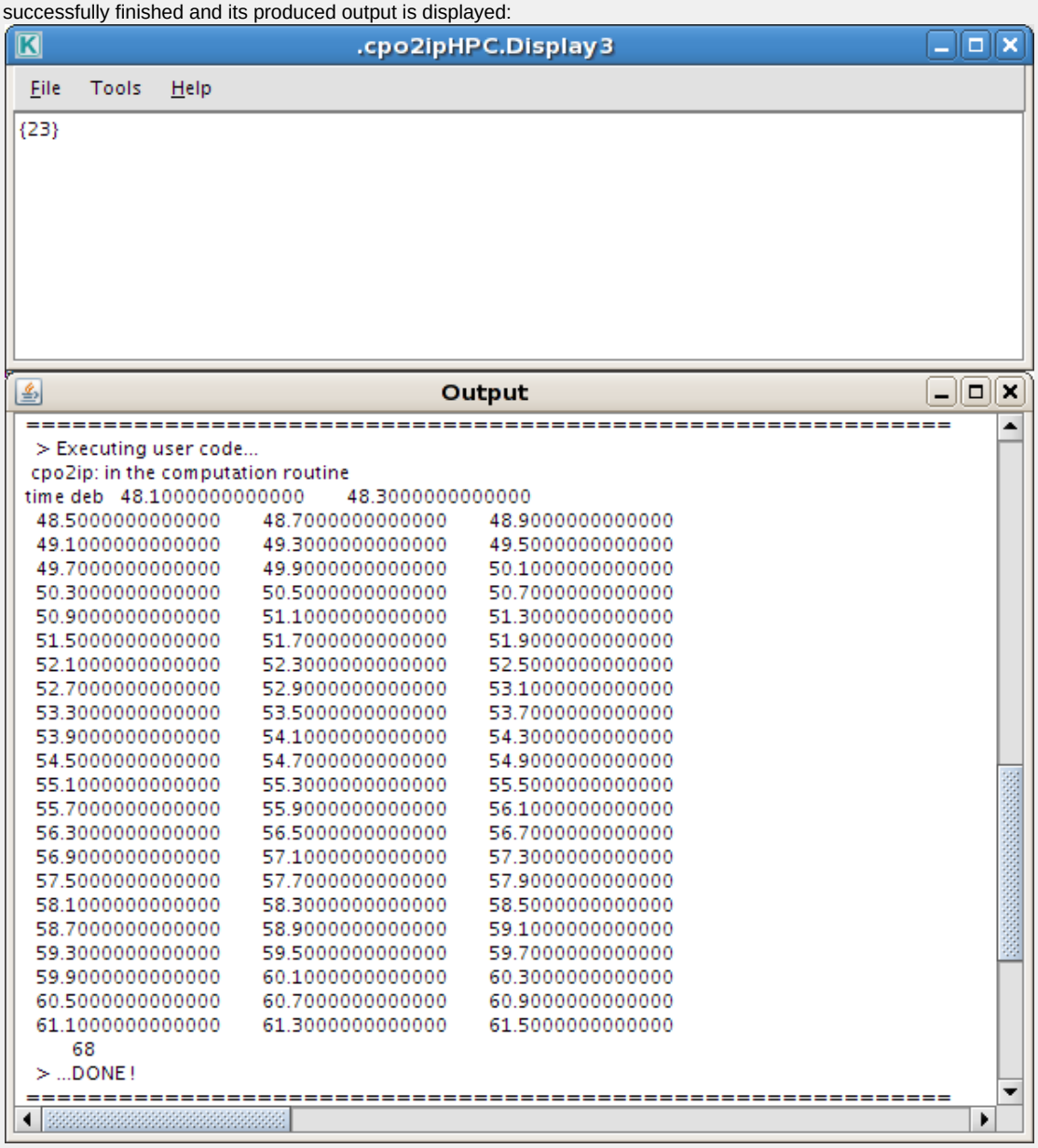
4. You have to change the following fields, so that they point to your directories:
   - **Library** in *Actor Library* section.
   - **Kepler** in *Environment* section.
5. Click on *Ok* button. Upon successful generation agree to compile Kepler.
6. Close HPC2K and open Kepler in the same terminal by executing:

```
kepler
```

7. Instantiate your newly created actor via *Tools->Instantiate Component* menu. The classname is **eu.itm.GRID-HPC.cpo2ipHPC.cpo2ipHPC**
8. Put **DDF director**, three **Constant** actors, **ualinit** and three **Display** actors.
9. Add an output port named *equilibrium* to **ualinit**.
10. Fill **Constants** and connect all actors as shown on the screenshot below. Also, make sure that the **Constants** have *fireCountLimit* set to 1.

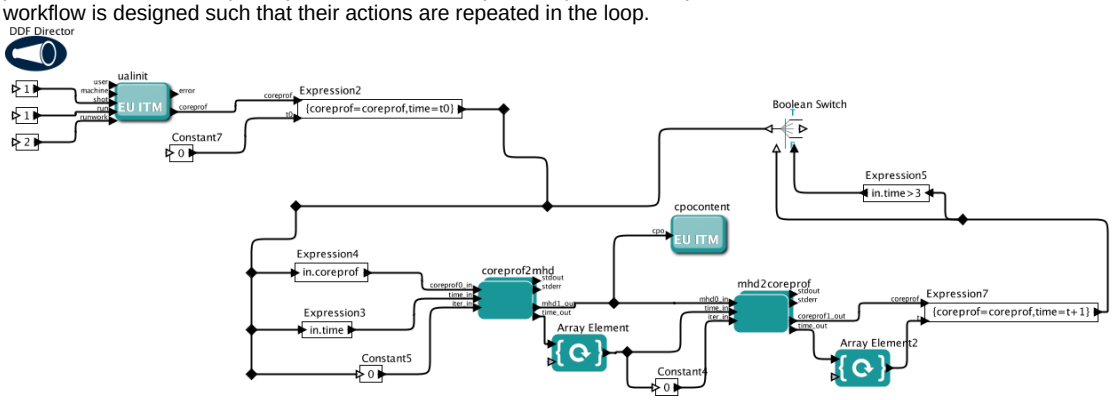


11. Execute the workflow. You will be informed about following stages of execution up to the moment when job is successfully finished and its produced output is displayed:



4.2 Interactive HPC use case

HPC workflows generated by HPC2K represent interactive jobs. After re-entering the composite actor, the job is not submitted again, so once you have waited in the queue you can use the computation power directly. To demonstrate this, two codes have been prepared and the workflow is designed such that their actions are repeated in the loop.



Exercise 2

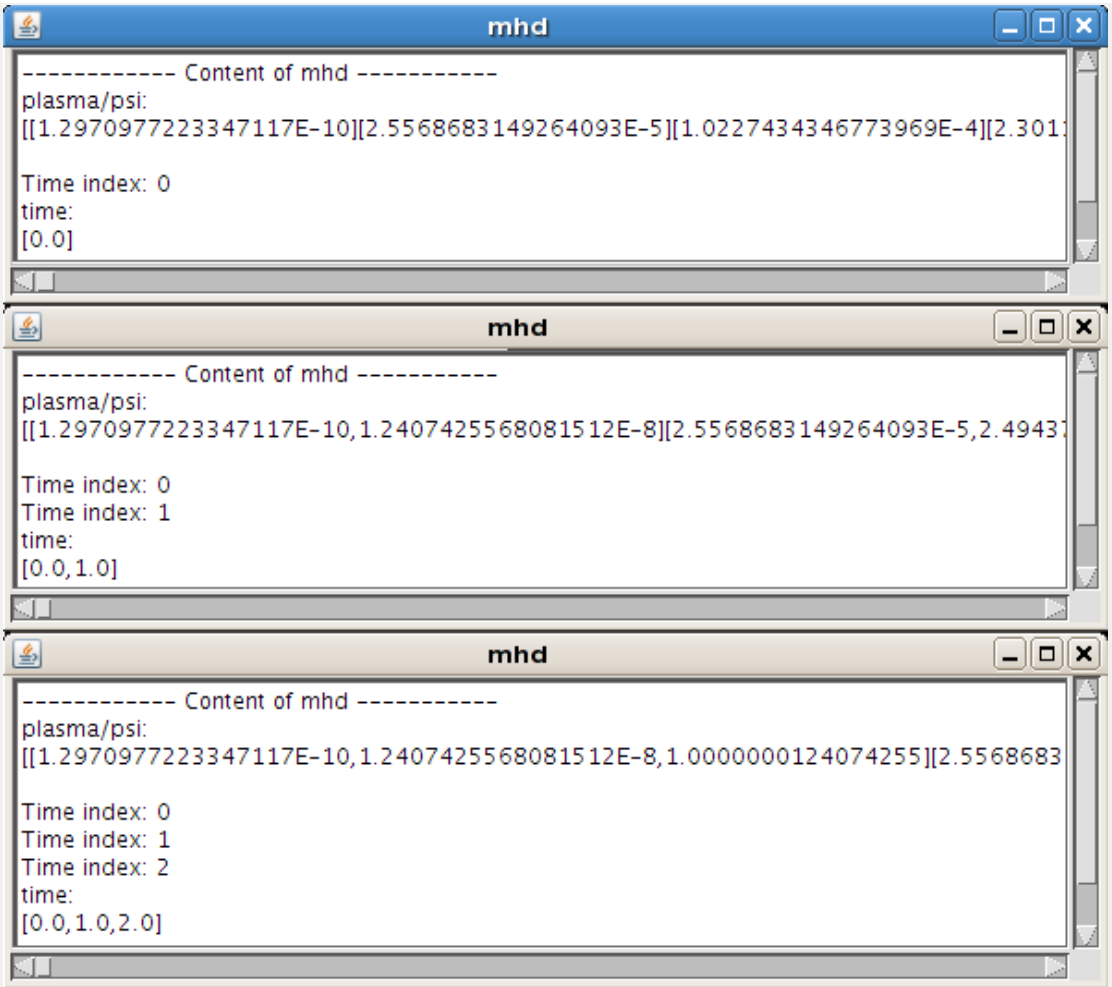
1. Run HPC2K.

```
cd $HOME
hpc2k-hpc/actors/install $HOME/kepler
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a
>/dev/null
setenv HPC2K $HOME/hpc2k-hpc
hpc2k-hpc/hpc2k
```

2. Load the predefined configuration from `~/coreprof2mhd/coreprof2mhd.xml` via *File->Open* menu.
3. You have to change the following fields, so that they point to your directories:
   - **Library** in *Actor Library* section.
   - **Kepler** in *Environment* section.
4. Click on *Ok* button. Upon successful generation agree to compile Kepler.
5. Repeat steps 3-5 with the predefined configuration `~/coreprof2mhd/mhd2coreprof.xml`.
6. Close HPC2K and open Kepler in the same terminal by executing:

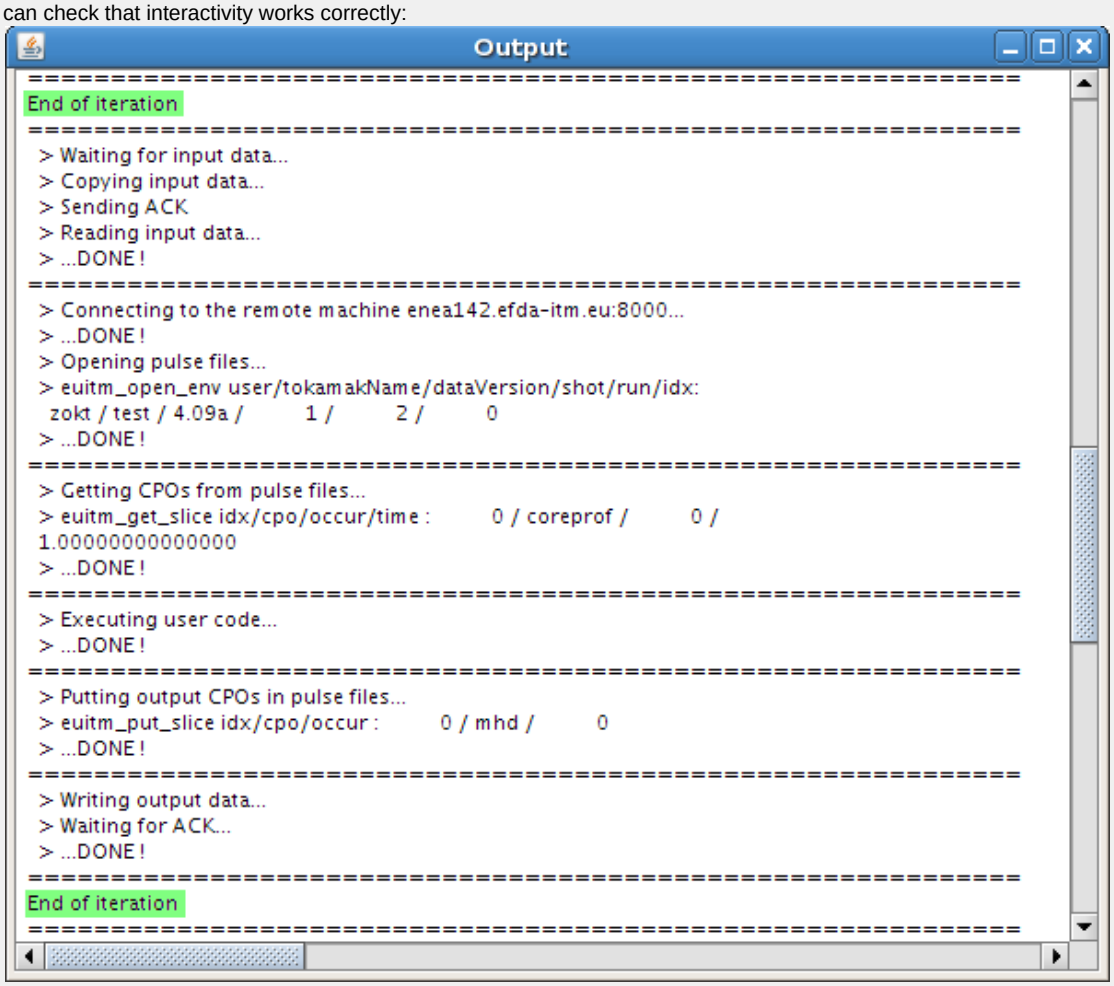
```
kepler
```

7. Load the workflow from `~/coreprof2mhd/HPCRepeat.xml` via *File->Open* menu.
8. Execute the workflow. You will see the content of immediate MHD CPO data and can observe the changes done remotely on HPC machine as shown on this screenshot:



After the initial waiting time in the queue, two interactive jobs directly process the data sent to them from Kepler

workflow. When the workflow is finished you will be presented with standard output and error from both jobs where you can check that interactivity works correctly:



4.3 Submitting a grid job

We will run the same code as before, but this time on grid infrastructure.

Exercise 3

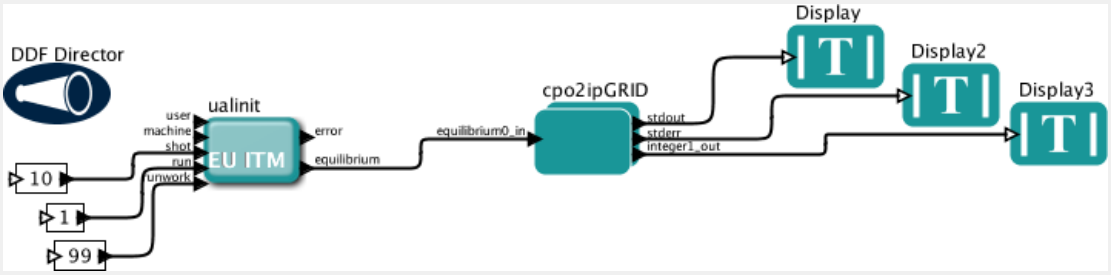
1. Run HPC2K.

```
cd $HOME
hpc2k-grid/actors/install $HOME/kepler
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a
>/dev/null
setenv HPC2K $HOME/hpc2k-grid
hpc2k-grid/hpc2k
```

2. Load the predefined configuration from `~/cpo2ipGRID/cpo2ipGRID.xml` via *File->Open* menu.
3. You have to change the following fields, so that they point to your directories:
   - **Library** in *Actor Library* section.
   - **Kepler** in *Environment* section.
4. Click on *Ok* button. Upon successful generation agree to compile Kepler.
5. Close HPC2K and open Kepler in the same terminal by executing:

```
kepler
```

6. Before running the grid job, you have to create a proxy certificate. Please follow the instructions in this guide. Set **vo** parameter to value *gilda* and please store the proxy in your `$HOME`:
7. Instantiate your newly created actor via *Tools->Instantiate Component* menu. The classname is **eu.itm.GRID-HPC.cpo2ipGRID.cpo2ipGRID**
8. Put **DDF director**, three **Constant** actors, **ualinit** and three **Display** actors.
9. Add an output port named *equilibrium* to **ualinit**.
10. Fill **Constants** and connect all actors as shown on the screenshot below. Also, make sure that the **Constants** have *fireCountLimit* set to 1.



11. Execute the workflow. You will be informed about following stages of execution up to the moment when job is successfully finished and its produced output is displayed.

6. Tutorial - Parametric grid job submission (Garching 09.2011)

Parametric grid job submission

Table of Contents

1. Introduction

This tutorial is designed to introduce the concept of building ITM tools based workflows within Kepler.

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.

NX connection to the Gateway

This tutorial assumes that Gateway accounts will be used for starting up Kepler application.
If you are not familiar with NX setup for the Gateway, take a look at following location NX setup

2. Requirements for the tutorial

Backing up Kepler home directory

Before you proceed with installation of the Kepler application be sure to make a backup of your Kepler home directory

```
mv ~/.kepler ~/.kepler_09_2011
mv ~/kepler ~/kepler_09_2011
mv ~/serpens ~/serpens_09_2011
```

2.1 Using ITM Kepler installation at Gateway

In order to make Kepler installation for the tutorial faster we will use preinstalled version of the Kepler that is available for Gateway users.

In order to install Kepler and ITM example workflow you have to follow instructions at following page:

Kepler installation

1. Kepler installation at Gateway (Garching 09.2011)

After you follow all the installation steps, you should see Kepler loading.

Starting Kepler

No matter which way have you used to install Kepler, make sure to export some variables before you start Kepler again.

```
source /afs/efda-itm.eu/project/switm/scripts/ITMv1 kepler test 4.09a >/dev/null
kepler
```

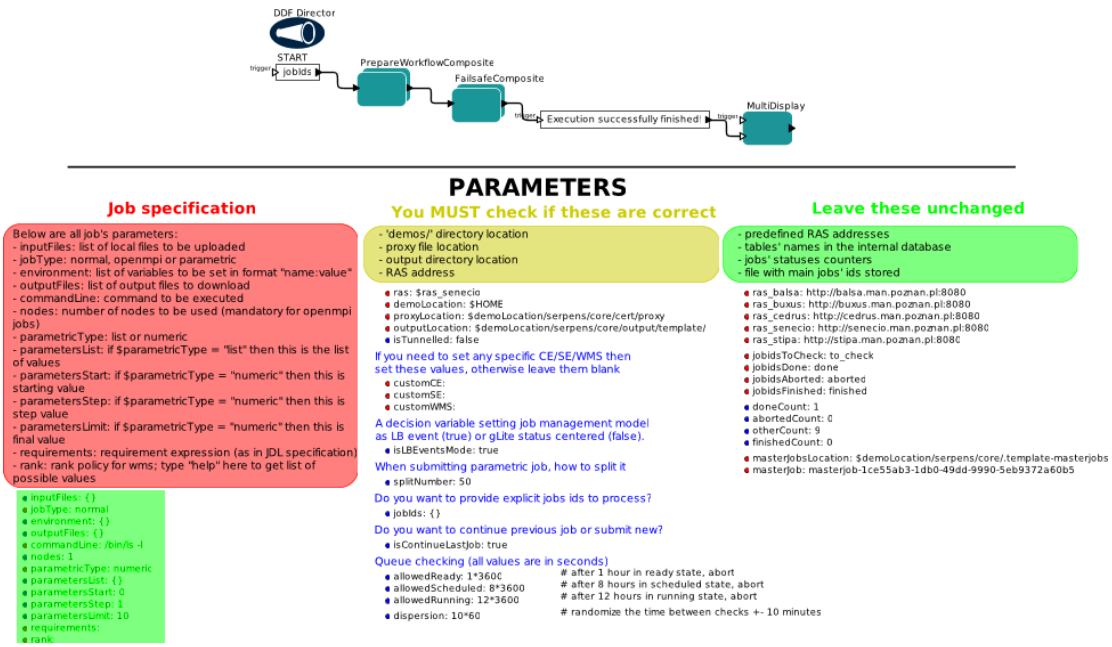
3. VOMS proxy

The first step is to create a proxy certificate. Please follow the instructions in this guide. Set **vo** parameter to value *gilda*.

4. The template workflow

In order to address various requirements of grid users from different fields of science, a single template workflow was created. This workflow contains a set of parameters which define the job to be submitted and mechanisms of its management.

4.1 Job specification



The first section of parameters defines the job to be submitted. The most important are:

- **jobType**: either normal (ie. single job) or parametric (ie. multiple subjobs managed as a single entity),
- **inputFiles**: an array of paths to local files which will be uploaded into job's working directory,
- **outputFiles**: an array of names of expected output files,
- **commandLine**: a full command with arguments that will be run on a worker node.

> ⚠ **Double check the *outputFiles* parameter**
>
> The template workflow tries to avoid failures or fix them if they occur. If the job is finished, but its output files are unavailable, the job will be resubmitted. Please double check your **outputFiles** array not to put there any misspelled filename!

If your job cannot be defined by a single command, you will have to write all commands in a shell script, and set **commandLine** parameter to the script name eg.:

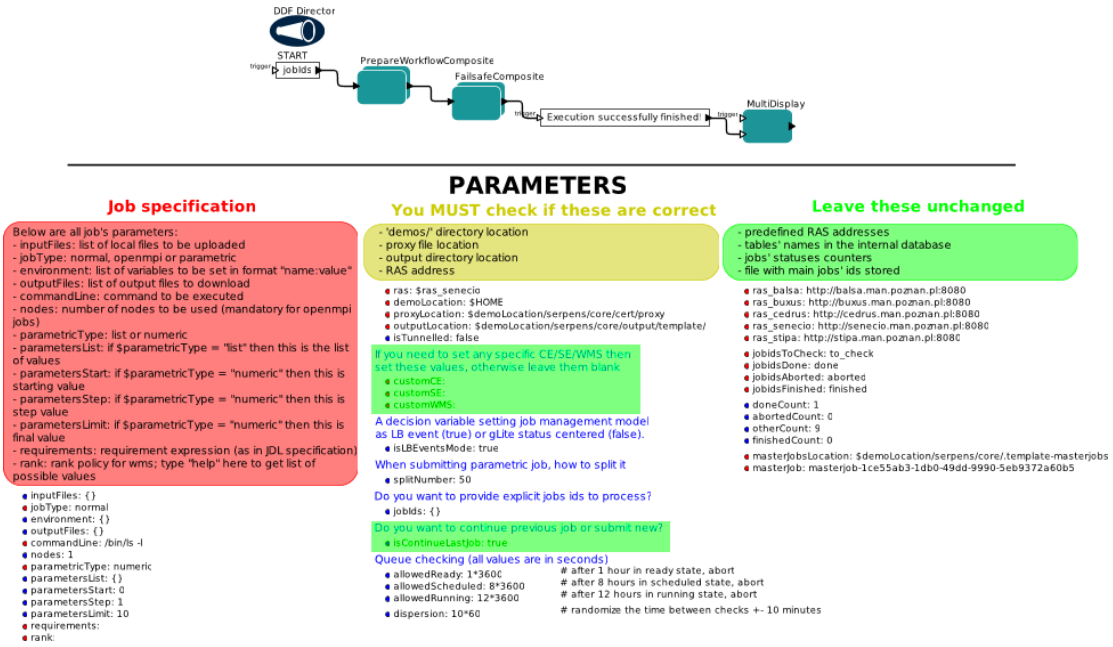
$HOME/script.sh

```
#! /bin/sh
ls -l
ps -e
```

Template workflow settings

```
commandLine: "/bin/sh script.sh"
inputFiles: {"$HOME/script.sh"}
```

4.2 Other parameters



The template workflow contains also a set of parameters which define its behaviour as a job manager:

- **isContinueLastJob**: the template workflow stores the state of job execution in an internal database, so that you can choose to continue your previous job even if Kepler stopped working,
- **customCE**, **customSE**, **customWMS**: you can force the usage of specified Computing Element (CE), Storage Element (SE) or Workload Management System (WMS).

During the exercises we will submit several jobs. To avoid unnecessary confusion it is recommended that you set **isContinueLastJob** to false. Then, each workflow run will always imply a new job submission which is what we need for educational purposes.

The custom resources choice is a very useful tool, but needs to be used responsibly. If you submit hundreds or thousands of jobs and you force all of them to use a single Workload Management System or even worse a single Computing Element, then the workload will be extremely unbalanced.

5. Single job submission

After this exercise you will:
• know how to run a single grid job

Exercise 1

1. Run Kepler.
2. Make sure you have a valid VOMS proxy.
3. Load template workflow from this location:

```
$HOME/serpens/core/workflow/grid/glite/template.xml
```

4. Set parameters according to the table below:

Parameter	Value
jobType	`normal`
inputFiles	`{"$HOME/serpens/core/data/input.txt"}`
outputFiles	`{"output.txt"}`
commandLine	`echo $(whoami) | cat input.txt - >output.txt`

5. This will prove that:
   - the input.txt was successfully uploaded and available to the job,
   - it was read and a new line was added to it in the end,
   - the concatenated result was stored in a new file,
   - you can use nested execution `$(...)`, pipes `|` and stream redirection > in your command definition.
6. Execute the workflow. When the job is finished you will be informed about the location of downloaded output. Please verify yourself if the command worked as expected.

6. Parametric job submission

Parametric jobs can be defined in two ways:

1. By giving a list of parameters.
2. By generating parameter values numerically.

In the first case, parameters can have any value and you have to specify them explicitly eg. {"1", "abc", "test"}. In the second case, you are obliged to provide three generator parameters: start, step and limit eg. for given `start = 1`, `step = 2`, `limit = 9` the parameters will be set to {"1", "3", "5", "7"}.

When you submit a parametric job, the Workload Management System launches multiple subjobs at the same time. Each subjob is separated from others, has its own copy of input files and has its own parameter value. To access this parameter value, you must use a special reserved keyword **`_PARAM_`**. You can use it in your command specification. For example, a command `echo _PARAM_` with parameter list set to {"1", "abc", "test"} will run three subjobs. Each of them will create a standard output file with its parameter value.

For the next exercise, let's choose a problem which is suitable for parametric job. We will submit several subjobs, where each of them will produce its own output depending on the parameter value. We will choose the numeric generation of parameters and factorial calculation as a grid job.

After this exercise you will:

- know how to run a parametric grid job

Exercise 2

1. Run Kepler.
2. Make sure you have a valid VOMS proxy.
3. Load template workflow from this location:

```
$HOME/serpens/core/workflow/grid/glite/template.xml
```

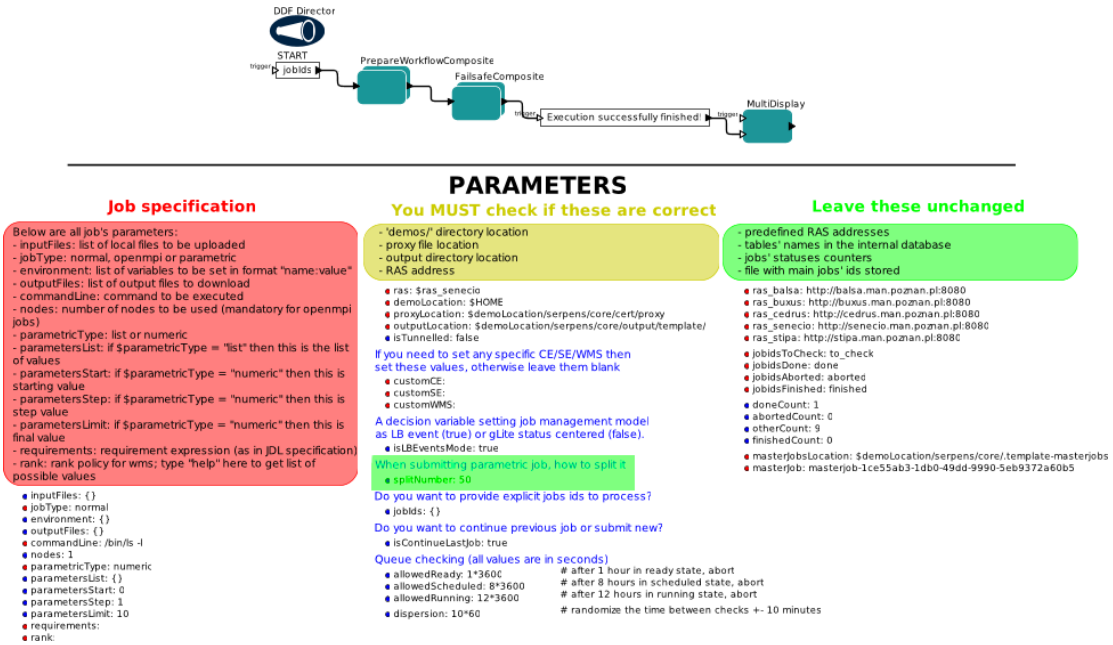
4. Set parameters according to the table below:

Parameter	Value
jobType	`parametric`
parametricType	`numeric`
parametersStart	`1`
parametersStep	`1`
parametersLimit	`10`
inputFiles	`{"$HOME/serpens/core/data/factorial.sh"}`
outputFiles	`{}`
commandLine	`/bin/sh factorial.sh _PARAM_`

5. This will do the following:
   - submit nine subjobs,
   - each subjob will calculate a factorial for its parameter value,
   - each subjob will be managed separately - if any fails, it will be resubmitted, if any finishes, it will be checked and its output will be downloaded.
6. Execute the workflow. When any of the subjobs is finished you will be informed about the location of downloaded output. Each subjob will have a separate output directory named after its parameter value eg. for parameter 4, the output should be 24.

6.1 Advanced control of parametric jobs

6.1.1 Jobs splitting



When you submit a parametric job, its ID holds information about all its subjobs. There is no formal constraint on the number of subjobs ie. you can set **parametersStart**, **parametersStep** and **parametersLimit** to arbitrarily high values yielding hundreds or thousands of job. However, there is a technical limitation of servers, network bandwidth, etc.

To bypass this problem, the template workflow seamlessly splits your parametric jobs. From the user point of view, this is indistinguishable from normal parametric job submission. Everything takes place in the background of workflow execution. There is a parameter **splitNumber** which is responsible for the splitting mechanism. The default value of 50 means that if you submit for example 140 jobs, they will be split in the background into 50, 50, 40 groups. You can change the **splitNumber** to obtain different grouping.

Exercise 3

1. Run Kepler.
2. Make sure you have a valid VOMS proxy.
3. Load template workflow from this location:

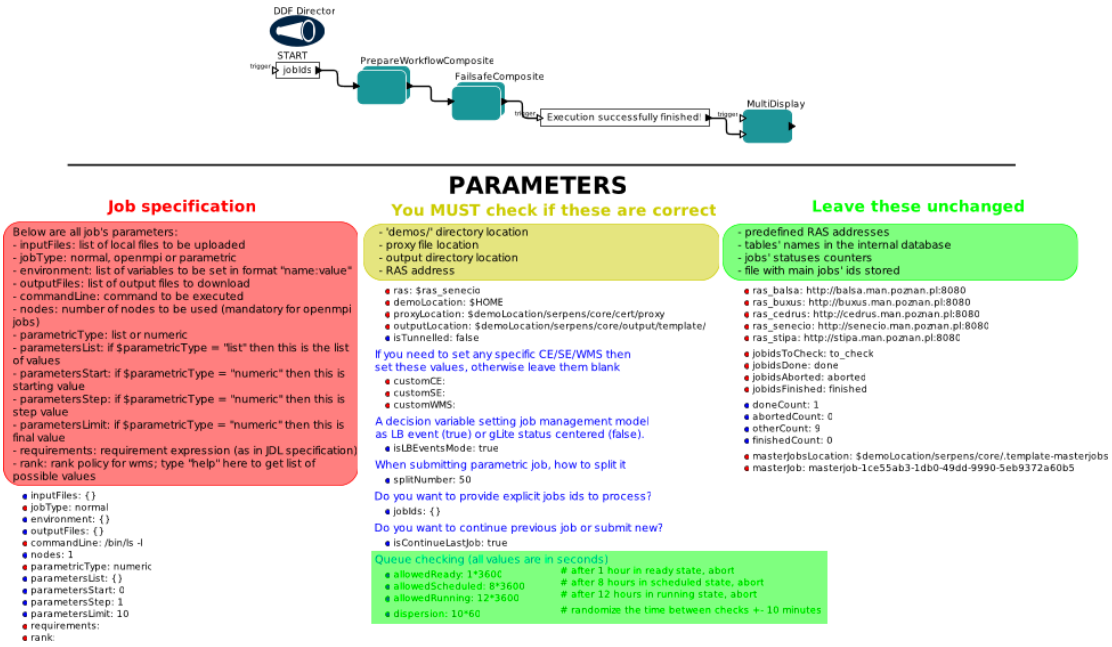
```
$HOME/serpens/core/workflow/grid/glite/template.xml
```

4. Set parameters according to the table below:

Parameter	Value
jobType	`parametric`
parametricType	`numeric`
parametersStart	`1`
parametersStep	`1`
parametersLimit	`10`
inputFiles	{}
outputFiles	{}
commandLine	`echo _PARAM_`
splitNumber	4

5. This will do the following:
   - submit nine subjobs,
   - workflow will show them as a single entity (9 separate jobs),
   - however in the background there will be three parametric jobs submitted grouped like this: 4, 4, 1
6. Execute the workflow.

6.1.2 Additional queue checking

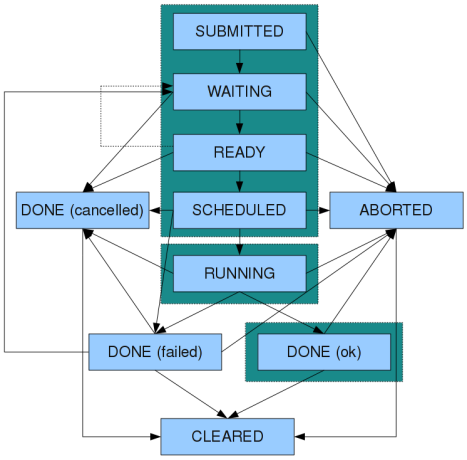


From time to time it may happen that the Workload Management System or Logging and Bookkeeping service are malfunctioning. They will

accept the job, but fail to update its status, enqueue/dequeue it properly or communicate with Computing Element. Or it could happen that the information services are inaccurate and your job ends in a days-long queue , despite other computing nodes being unused.

One aid for this problem is the aforementioned custom resource choice via **customCE**, **customSE** or **customWMS** parameters. For large parametric jobs, this can on the contrary overload a single resource very quickly leading to even worse situation.

Thus an additional queue checking mechanism have been introduced. Each of the managed subjobs is having its state stored in a local database. Its state consists of its job status and the time it first came up. This way, the template workflow can periodically check for jobs queue dynamics and intervene if abnormal situation occurs.



The following diagram shows job lifetime. The problems described in the the first paragraph manifest themselves in the following:

1. Job stays in a READY state forever if WMS/LB is broken.
2. Job stays in a SCHEDULED state for a very long time (days, weeks) if the information services failed or CE administrator has a hidden queue policy.
3. Job stays in a RUNNING state for abnormal long time if WMS/LB is broken. This one however is not so straightforward, because it could be that the application is stuck in an infinite loop, so please double check your application before assuming the infrastructure has failed.

The three statuses mentioned above are the three vulnerable points if there is a problem with the infrastructure. Thus, the template workflow periodically checks for these indicators of problems. There are three main parameters defining this mechanism behaviour:

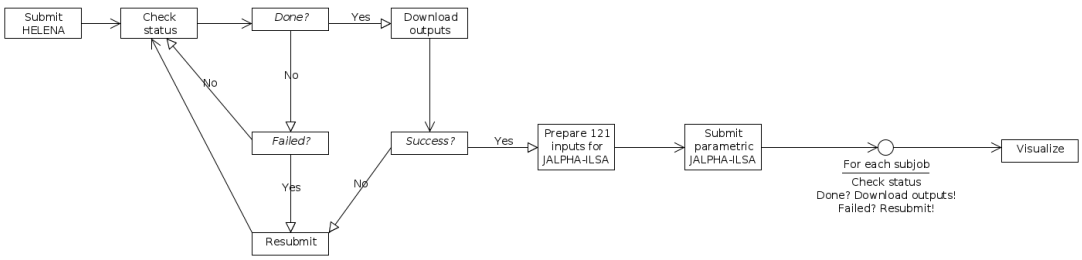
- **allowedReady**: time that the job is allowed to be in READY state, by default set to one hour,
- **allowedScheduled**: time that the job is allowed to be in SCHEDULED state, by default set to eight hours,
- **allowedRunning**: time that the job is allowed to be in RUNNING state, by default set to twelve hours.

> While the job being in READY state for long is a clear indicator of problem (thus the low limit of one hour), the situation with SCHEDULED and RUNNING is not so simple. If your job requirements are very specific and there are only few computing nodes that can fulfill them, it is very likely that your job will stay in a queue (SCHEDULED state) for a very long time, in which case it is not a problem of infrastructure. Also if your job is very time-consuming, the twelve hour threshold can also be insufficient and can lead to false alarms. Please be very considerate when tweaking these parameters!

The additional queue checking can be time- and resource-expensive. To avoid resource overuse in short time, these checks are not carried out on all of the subjobs at once. There is a **dispersion** parameter, by default set to ten minutes, which will randomize jobs checks.

7. Real use case example

Template workflow despite being heavily configurable allows to interchange its components to produce really complex solutions. With Christian Konz, we have developed together a HELENA-JALPHA-ILSA solution which can be schematically described as in this diagram:



In fact, most of the tasks here are already done by the template workflow. The first loop checking the status, resubmission upon failure and output downloading are already incorporated inside the workflow. What we added specifically for HELENA-JALPHA-ILSA use case was the input generator and another job submission in the end of the cycle. This way once HELENA is successfully finished, then the inputs are

prepared and a parameter scan is initiated for JALPHA-ILSA. Later, the same scheme is used for each subjob - it is checked, resubmitted upon failure and its outputs are downloaded after successful finish. In the end, the results are visualized.

Technically speaking, preparation of this real and complex use case consisted of:

- joining together two template workflows (one for HELENA, one for JALPHA-ILSA),
- editing some of the workflows components to conform to the general use case.

You can watch a movie from this use case run here: http://scilla.man.poznan.pl/euforia/movies/helena-jalpha-ilsa.html