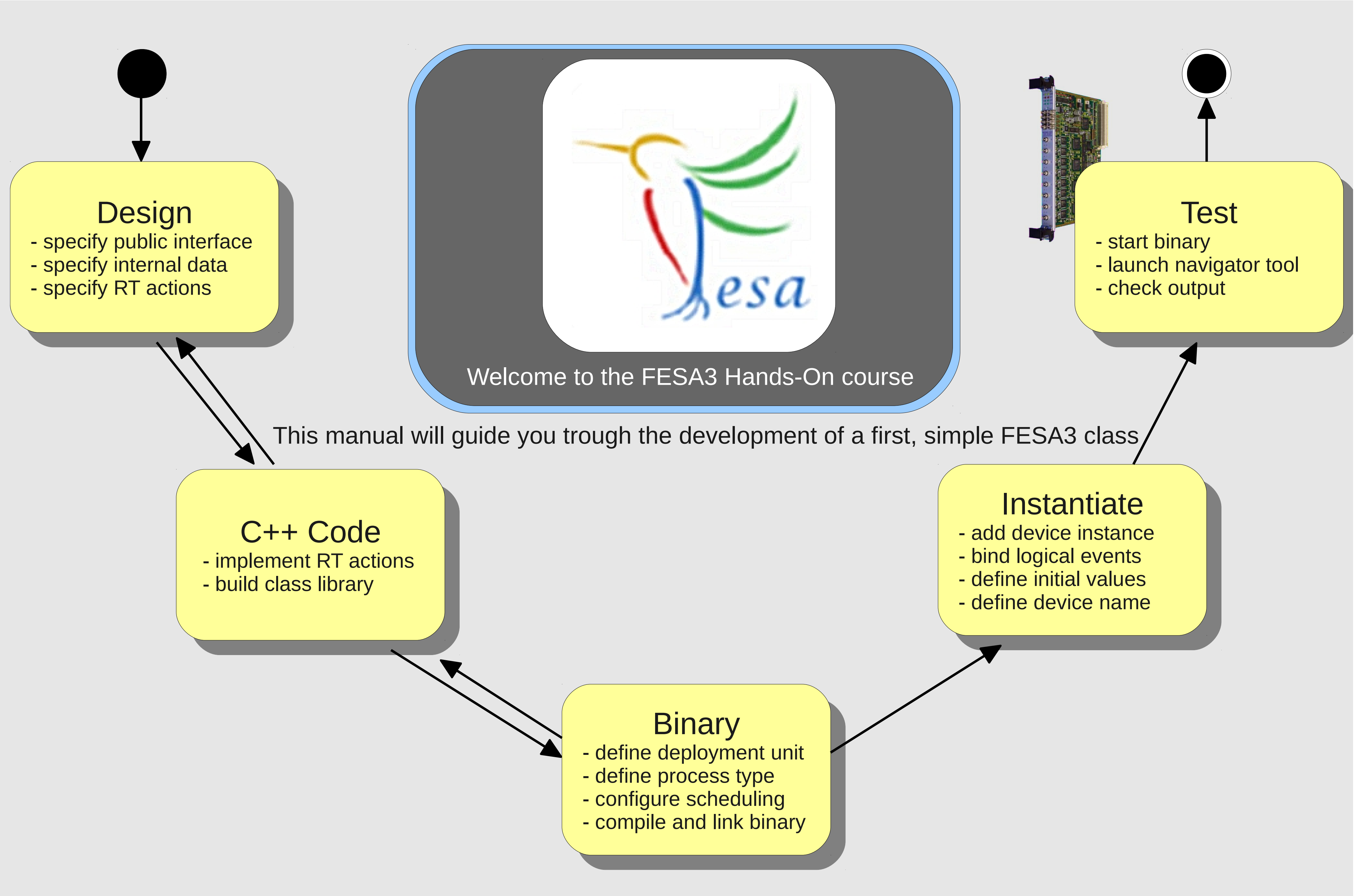
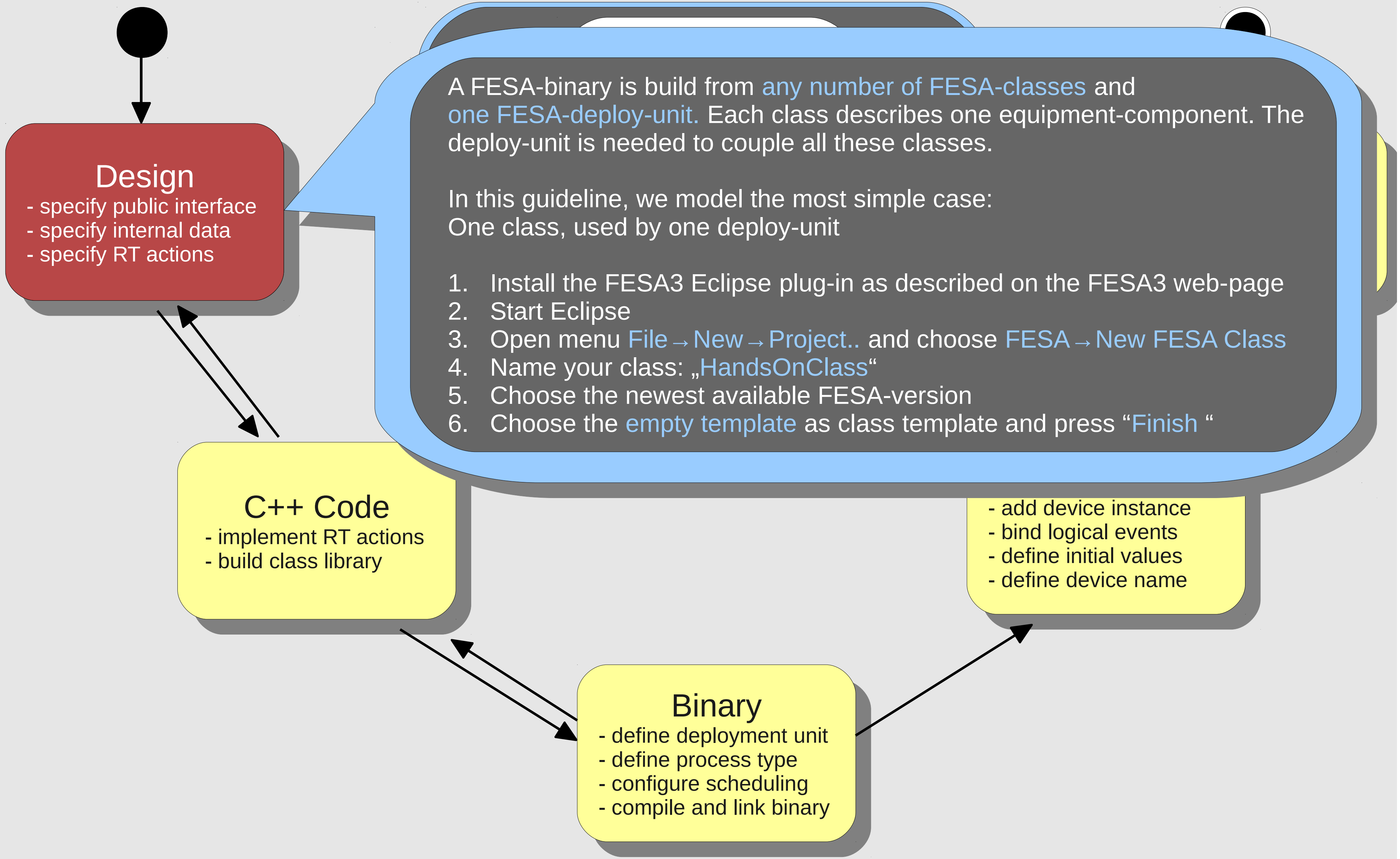
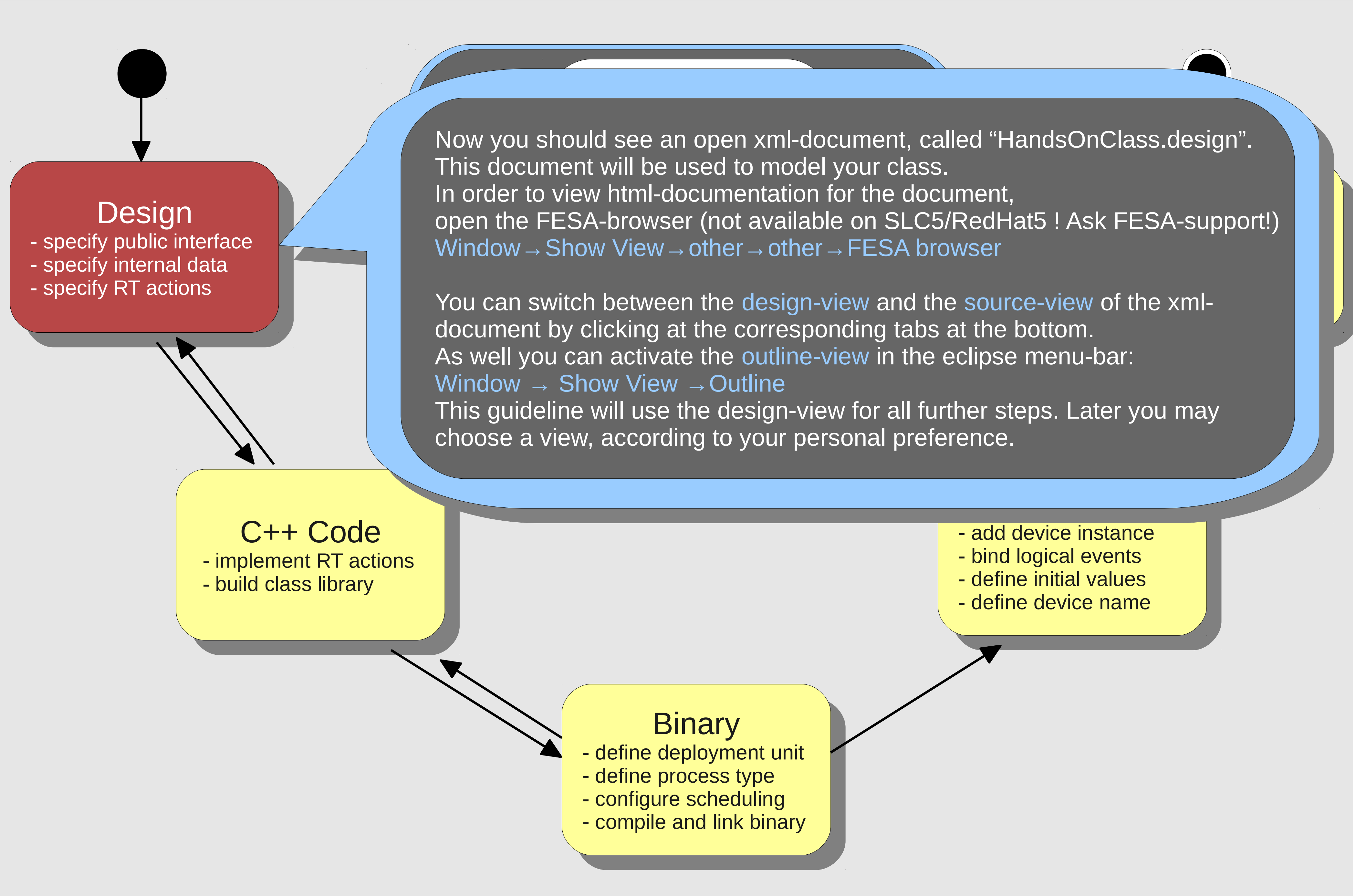

Welcome to the FESA3 Hands-On course
This manual will guide you trough the development of a first, simple FESA3 class
Design
- specify public interface
- specify internal data
- specify RT actions
C++ Code
- implement RT actions
- build class library
Binary
- define deployment unit
- define process type
- configure scheduling
- compile and link binary
Instantiate
- add device instance
- bind logical events
- define initial values
- define device name
Test
- start binary
- launch navigator tool
- check output



A FESA-binary is build from any number of FESA-classes and one FESA-deploy-unit. Each class describes one equipment-component. The deploy-unit is needed to couple all these classes.

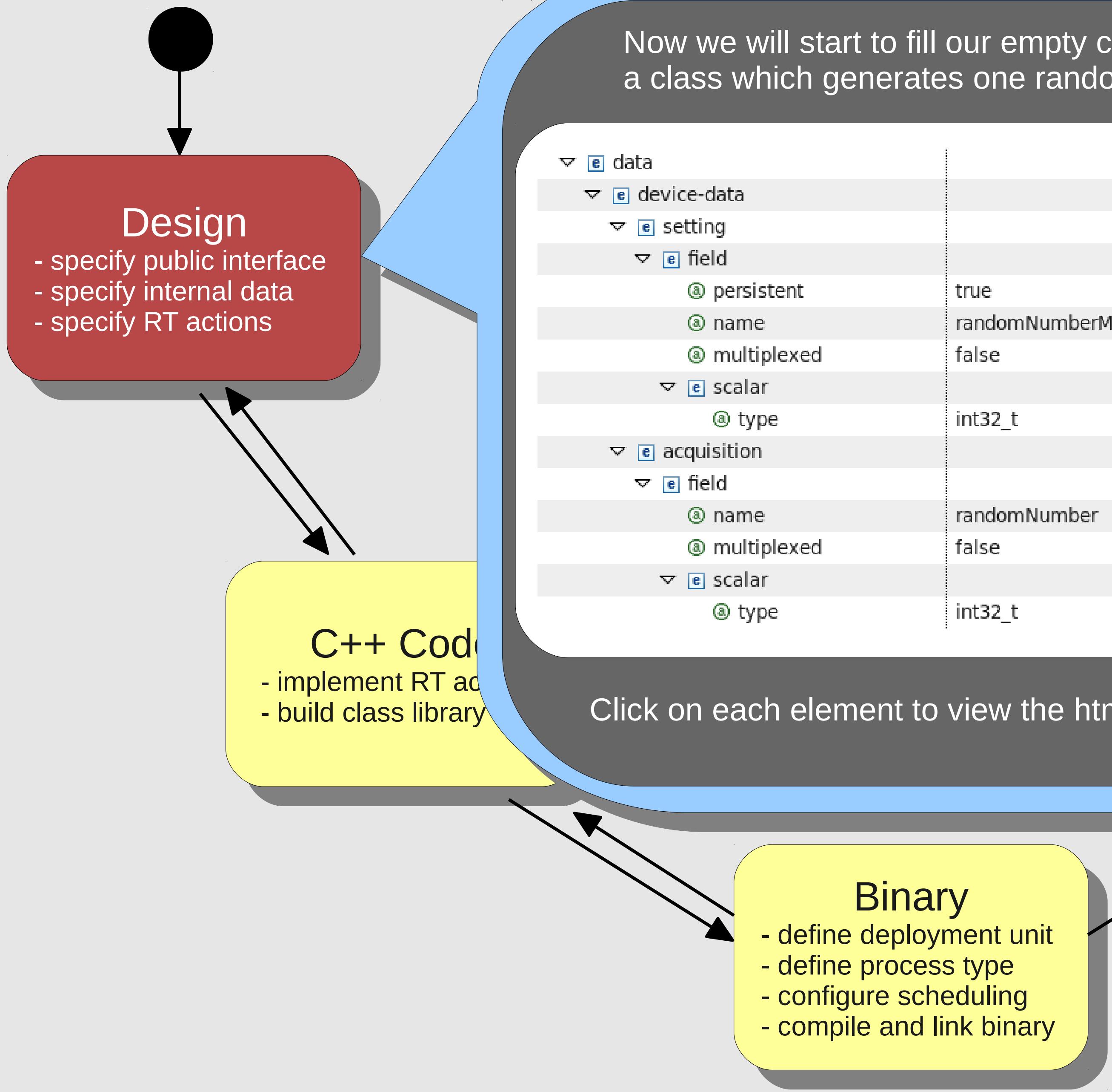
In this guideline, we model the most simple case:
One class, used by one deploy-unit

1. Install the FESA3 Eclipse plug-in as described on the FESA3 web-page
2. Start Eclipse
3. Open menu File → New → Project.. and choose FESA → New FESA Class
4. Name your class: „HandsOnClass“
5. Choose the newest available FESA-version
6. Choose the empty template as class template and press “Finish “



Now you should see an open xml-document, called “HandsOnClass.design”.
This document will be used to model your class.
In order to view html-documentation for the document,
open the FESA-browser (not available on SLC5/RedHat5 ! Ask FESA-support!)
Window → Show View → other → other → FESA browser

You can switch between the design-view and the source-view of the xml-document by clicking at the corresponding tabs at the bottom.
As well you can activate the outline-view in the eclipse menu-bar:
Window → Show View → Outline
This guideline will use the design-view for all further steps. Later you may choose a view, according to your personal preference.



Now we will start to fill our empty class-design. The goal is to design a class which generates one random number per second.

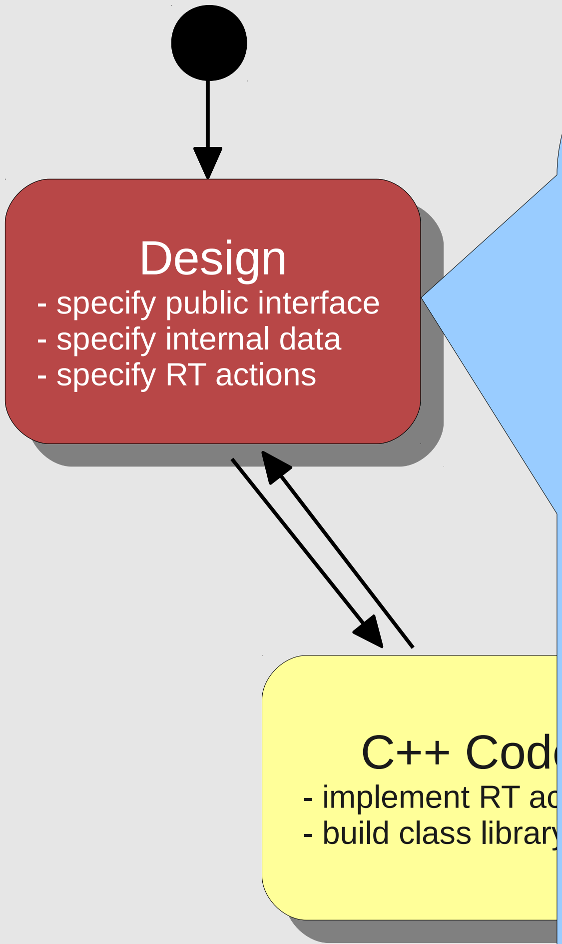
data	
device-data	
setting	
field	
persistent	true
name	randomNumberMax
multiplexed	false
scalar	
type	int32_t
acquisition	
field	
name	randomNumber
multiplexed	false
scalar	
type	int32_t

We start with defining the definition of the needed data-containers. Just add all missing elements, like shown in the picture. (right-click-->add-child)

The random-number itself is stored as “acquisition” data, since we want to send it to the client.

The maximum number which can be generated shall be configurable by the client. So we choose to store it as “setting”-data.

Click on each element to view the html-documentation in the browser!



In order to provide client-read-access to the defined internal data, we need to add properties in the “interface”-part of our class.

▽ e interface	
▽ e device-interface	(setting?, acquisition
▷ e setting	
▽ e acquisition	((acquisition-property
▽ e acquisition-property	
ⓐ visibility	operational
ⓐ name	RandomNumber
ⓐ multiplexed	false
▽ e value-item	
ⓐ name	randomNumber
ⓐ direction	OUT
▽ e scalar	
ⓐ type	int32_t
▽ e data-field-ref	
ⓐ field-name-ref	randomNumber
▽ e get-action	(server-action-ref | al
▽ e server-action-ref	
ⓐ server-action-name-	GetRandomNumber

▽ e actions	
▽ e get-server-action	
ⓐ implementation	default
ⓐ name	GetRandomNumber

We define an acquisition-property, which can be read by the client via “Get” or “Subscribe”.

Value-Items are used to show which data is transfered by a property. Here we connect the value-item to our field, in order to transfer our internal data.

Only the elements which need to be modified are shown here!

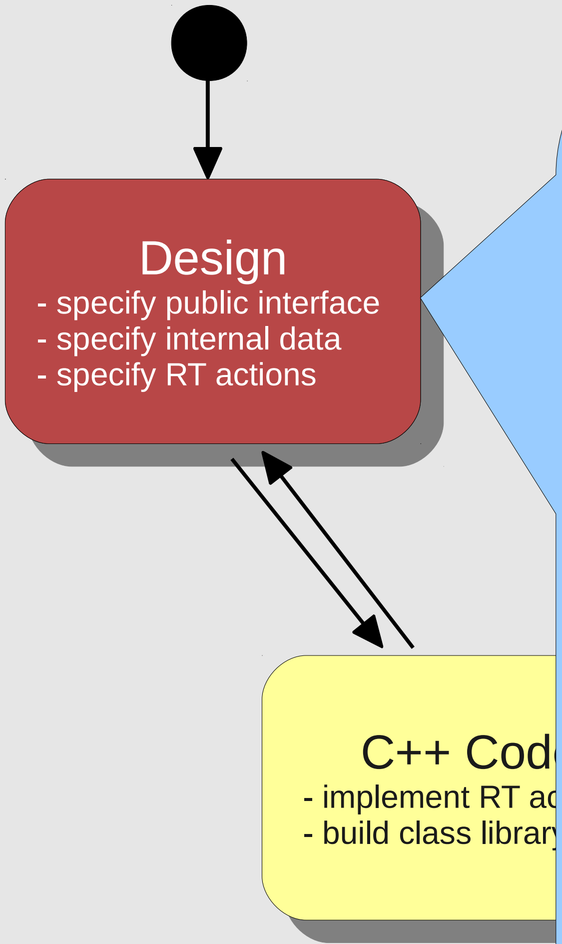
The get-action models the C++ implementation of the data-transfere. Note that the XML-File will show an error as long as the action does not exist:

▽ e get-action	
▽ e server-action-ref	
ⓐ server-action-name-	GetRandomNumber

You can check the concrete error-message in the “source”-view. (click on red dot)

```
        </cycle-stamp-item>
        <get-action>
            <server-action-ref server-action-name-ref="GetRandomNumber" />
        </get-action>
    </acquisition-property></acquisition></device-interface>
```

Validate your class-design with the ✓ validate button !



Now we proceed in the same way for our setting-field "randomNumberMax", in order to give the client write-access to it..

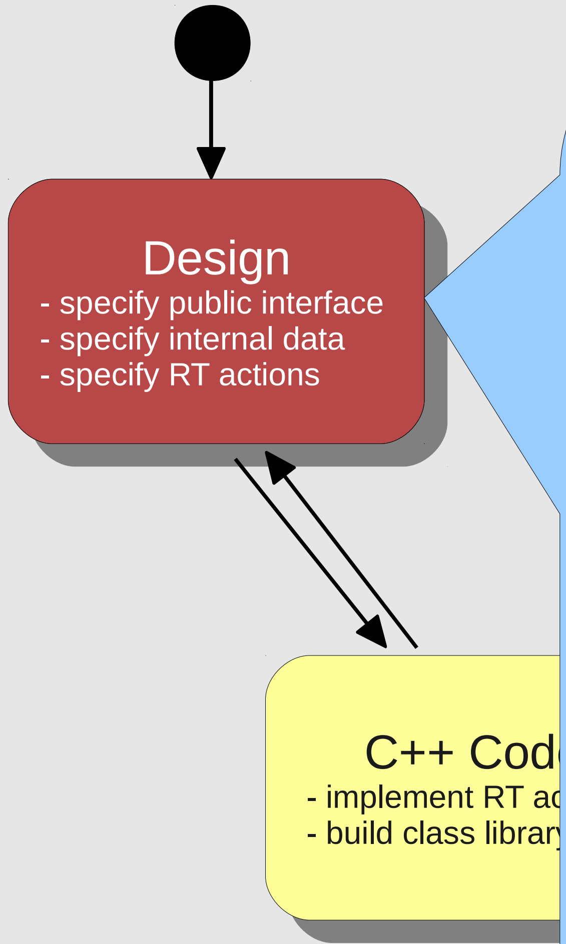
actions	
set-server-action	
implementation	default
name	SetRandomNumberMax
get-server-action	
implementation	default
name	GetRandomNumberMax

This time we start with the actions, in order to use the auto-completion feature.
Again we choose implementation = "default". So we don't need to provide own C++ code for this action.

interface	
device-interface	
setting	
setting-property	
visibility	operational
name	RandomNumberLimits
multiplexed	false
value-item	
name	randomNumberMax
direction	INOUT
scalar	
type	int32_t
data-field-ref	
field-name-ref	randomNumberMax
set-action	
server-action-ref	
server-action-name-ref	SetRandomNumberMax
get-action	
server-action-ref	
server-action-name-ref	GetRandomNumberMax

A setting-property can be written by a client with "Set" or re-read via "Get".

Note that now you can choose the get- and set-actions from a list, because we defined the actions in advance.



Finally we will design the number-generation itself. For this purpose we use a Timer-event-source which periodically triggers an action.

events	
sources	
timing-event-source	
timer-event-source	
logical-events	
logical-event	
use	required
name	timerEvent
type	timer

First we define the event-source and the logical-event which is used by this source.
Right-cllick on the root-element “equipment-model” in order to add the element “events”.

actions	
rt-action	
name	GenerateRandomNumber
notified-property	
property-name-ref	RandomNumber
automatic	true

All actions which do not interact with the client are called “rt-action” that's what we need for the number-generation. We as well choose to automatically notify all clients which subscribed to our property “GetRandomNumber”.

scheduling-units	
scheduling-unit	
name	TimerSchedulingUnit
rt-action-ref	
rt-action-name-ref	GenerateRandomNumber
logical-event-ref	
logical-event-name-ref	timerEvent

In order to connect our rt-action with the logical-event, we need to add a “scheduling-unit”.
Again right-click on the root-element in order to add the element “scheduling-units”.

Finally you finished the design-phase! Now re-check if your design is valid by pressing ✓ and fix all remaining bugs.
After that, trigger the code generation by pressing the button. This will generate the C++ source code skeleton of your class.

As next step we will add some C++ code in order to generate the random-numbers itself. To do so, open the file “HandsOnClass/src/HandsOnClass/RealTime/GenerateRandomNumber.cpp” from the Eclipse-Project-Explorer and modify it, according to the source-code below.

- spe
- spe
- spe

After you finished the implementation, you can compile your FESA-class library. Go to the project-folder and execute „make all“. This can be done in Eclipse using the „Make Targets“ view in the C++ perspective.

By executing „make clean“ you can remove all older libraries and object files.

- all
- clean

C++ Code

- implement RT actions
- build class library

```
void GenerateRandomNumber::execute(fesa::RTEvent* pEvt)
{
        std::vector<Device*>::iterator device;
        for(device=deviceCol_.begin();device!=deviceCol_.end();++device)
        {
                // get upper limit for random-numbers from internal field
                int32_t rand_max = (*device)->randomNumberMax.get(pEvt->getMultiplexingContext());

                // generate random-number between 0 and rand_max
                int32_t rand_number = rand() % ( rand_max - 1 );

                // produce some output
                std::ostringstream message;
                message << " Produced random number: " << rand_number << " for device: " << (*device)->getName();
                LOG_TRACE_IF(logger, message.str());

                // save produced random-number in internal field
                (*device)->randomNumber.set(rand_number,pEvt->getMultiplexingContext());
        }
}
```

You may want to copy + paste this source code!

A FESA-binary is build from any number of FESA-classes and one FESA-deploy-unit. Each class describes one equipment component. The deploy-unit is needed to couple all these classes.
To create a deploy-unit-project, choose: File → New → Project.. → FESA → New FESA Deploy Unit.
According to the class, we name it “HandsOnDeployUnit”.

class	
class-name	HandsOnClass
class-major-version	0
class-minor-version	1
class-tiny-version	0
device-instance	required

executable	
mixed	
extension	_M

Only the items that you need to add or change are listed here. When you finished editing the deployment document, validate it and generate the C++ source code.

To obtain the executable binary-file, trigger „make all“ as well for the deploy-unit.

scheduler	
concurrency-layer	
name	TimerLayer
event-queue-size	7
prio	7
scheduling-unit	
per-device-group	no
scheduling-unit-name-ref	HandsOnClass::TimerSchedulingUnit

Note: After adding the class-name, save the document! The plugin will automatically add the elements “path” and “include”. Now you will be able to pick the right scheduling-unit from a list.

Binary

- define deployment unit
- define process type
- configure scheduling
- compile and link binary

For the next step you need to configure on which front-end your binary should run. To do so, open the deploy-unit document and push the „Add FEC“ button. Put in the name of the front-end on which you currently work.

classes	
HandsOnClass	
multiplexing	
events-mapping	
timerEvent	
event-configuration	
name	OncePerSecond
Timer	
timer-event	
period	1000
unused-event-configuratio	

Press to create a new instance of your class for this front-end.

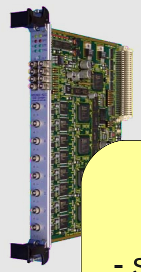
Configure the devices of your class, as shown on the screen-shots.

Note that we use the event-configuration “OncePerSecond” which we defined at our own in the section “events-mapping”.

Validate your instantiation document by pressing .

HandsOnClass	
device-instance	
name	TestDevice1
configuration	
events-mapping	
timerEvent	
event-configuration-ref	
name	OncePerSecond
global-instance	
name	HandsOnGlobalInstance

Later you can find this file in: HandsOnDeployUnit/src/test/[FEC]



Test
- start binary
- launch navigator tool
- check output

Instantiate
- add device instances
- bind logical events
- define init values
- define device names

rse

- compile and link binary

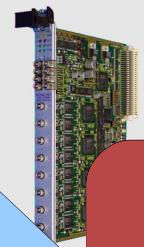
Des
- specify pu
- specify int
- specify RT

In order to run your binary, open a fresh Linux-console, browse to the instantiation-folder and start your binary via the generated start-script.
To do so, use the following commands in your Linux-shell:
(Replace [myWorkspaceLocation] and [myFEC] according to your local setup)

```
cd [myWorkspaceLocation]/HandsOnDeployUnit/src/test/[myFEC]
./startScript.sh -noRTSched -vv
```

“-noRTSched” allows you to run the progress without rt-priorities.
Very verbose(-vv) will show you the log-messages of all log-levels.
You can stop the execution by pressing [STRG+C]. Use the argument -help to get an overview about all possible command-line-parameters.

Congratulations! Now you can remotely access the device “TestDevice1” across the middleware. One client for this purpose is the Navigator.
Open the instantiation document and press: “Launch FESA Navigator” .



Test
- start binary
- launch navigator tool
- check output

Instantiate
dd device instance
ind logical events
efine initial values
define device name

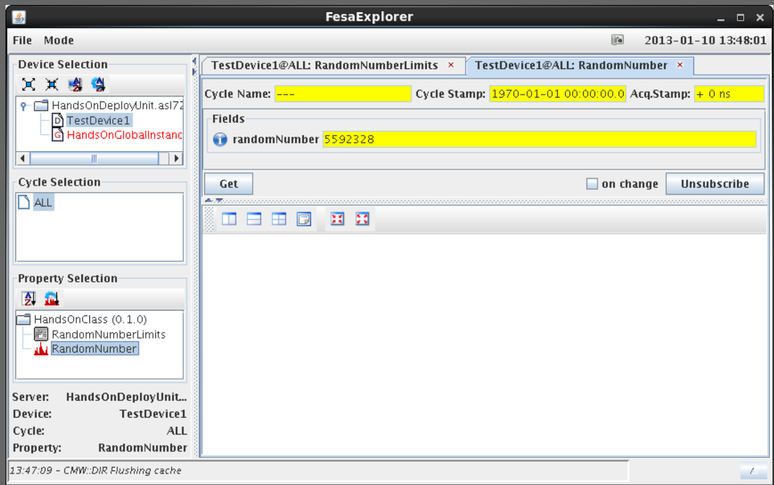
Binary
- define deployment unit
- define process type
- configure scheduling
- compile and link binary

Warning: This page probably is already outdated on release. Contact the FESA-Support on any problems

Once the Navigator opened, select the “TestDevice1” and double-click on the property “RandomNumberLimits”.
Put some value into the field “randomNumber_max” and press “Set”, in order to send the data via the middleware to your class.

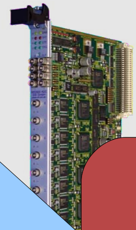
Now double-click on the property “RandomNumber” and press “Subscribe”. If you implemented everything in the right way, you should receive one random-number per second.

Congratulations!
If you arrive here, you finished the FESA3 HandsOn course. On any problems, dont hesitate to check the FESA-Wiki or to contact the FESA-support-team.



For further training, you may want to add a field “randomNumber_min” to your class and write a custom-server-action which produces additional output. Feel free to extend your class to whatever you want! As well check the html-documentation in the FESA-Browser if you face any unknown xml-elements!

De
- specify p
- specify i
- specify F



Test
- start binary
- launch navigator tool
- check output

Instantiate
d device instance
d logical events
ine initial values
ine device name