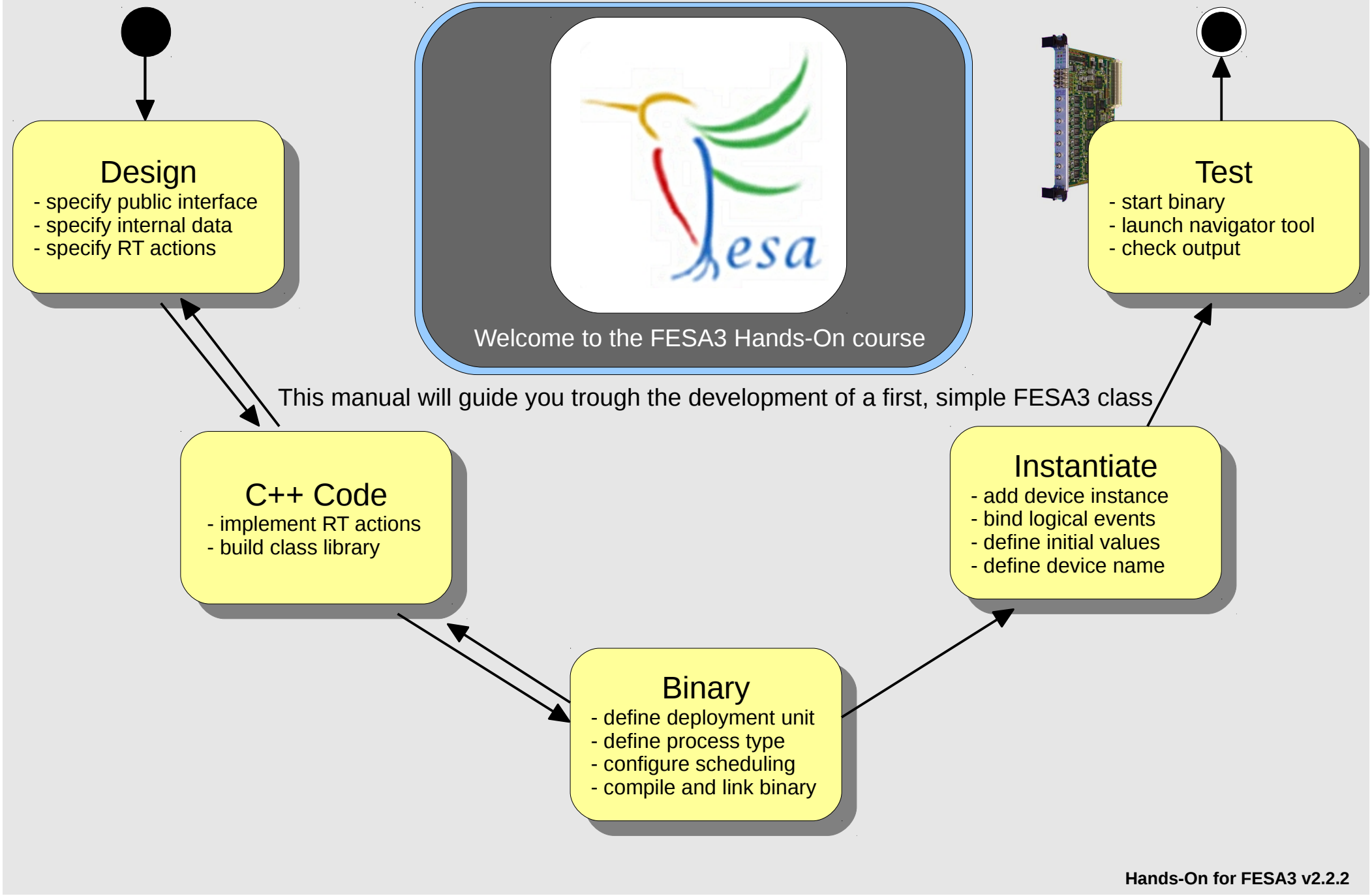
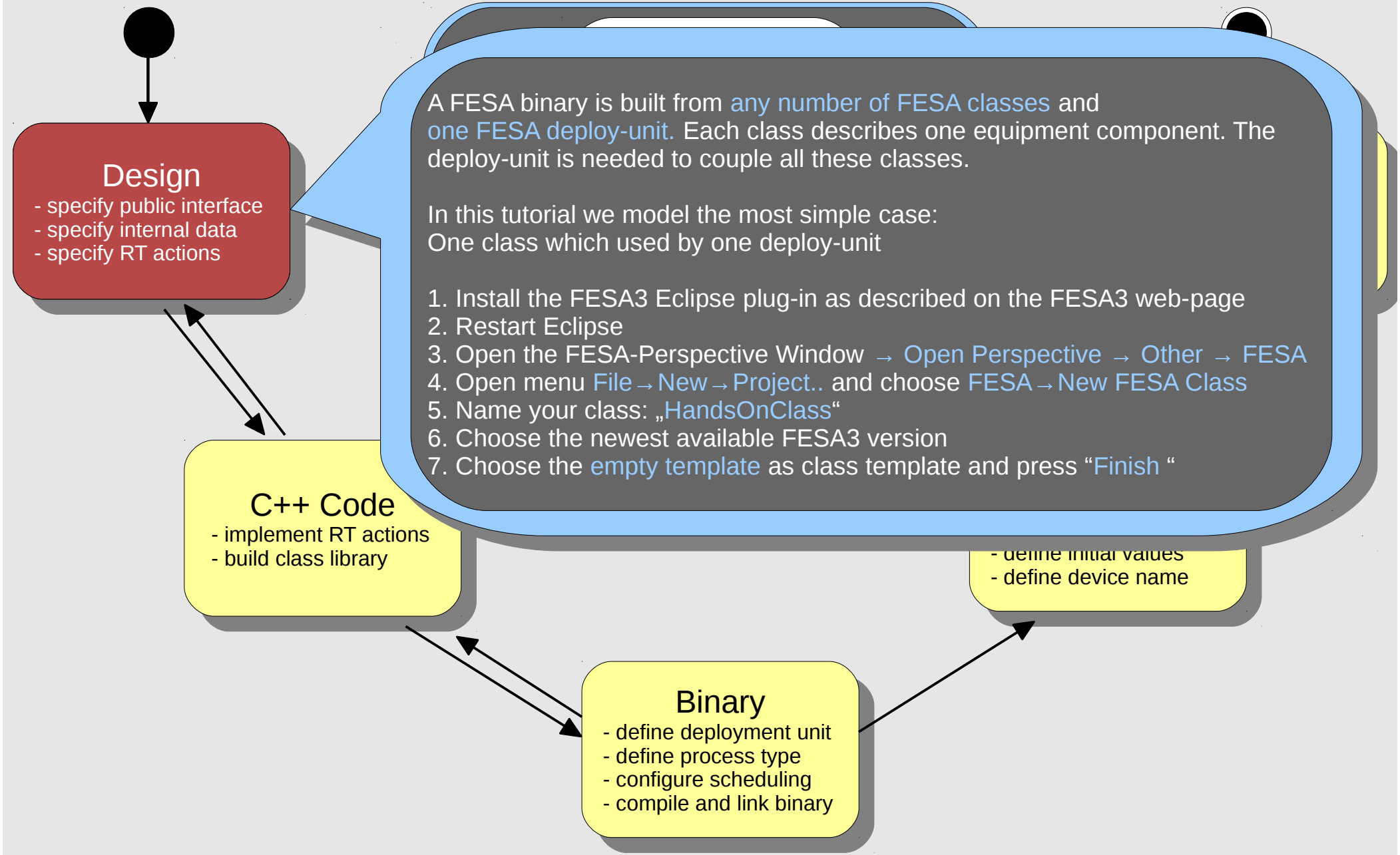
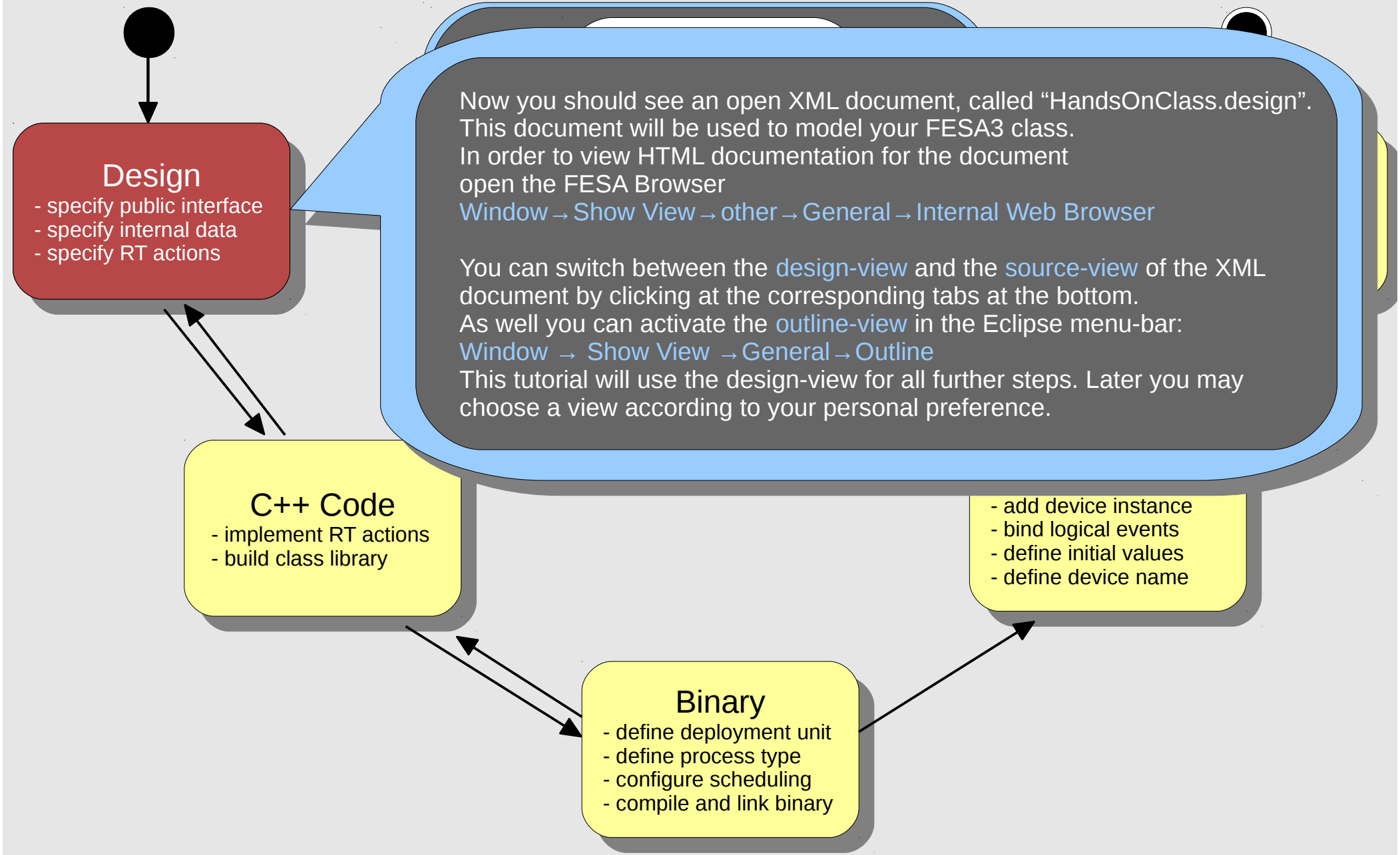
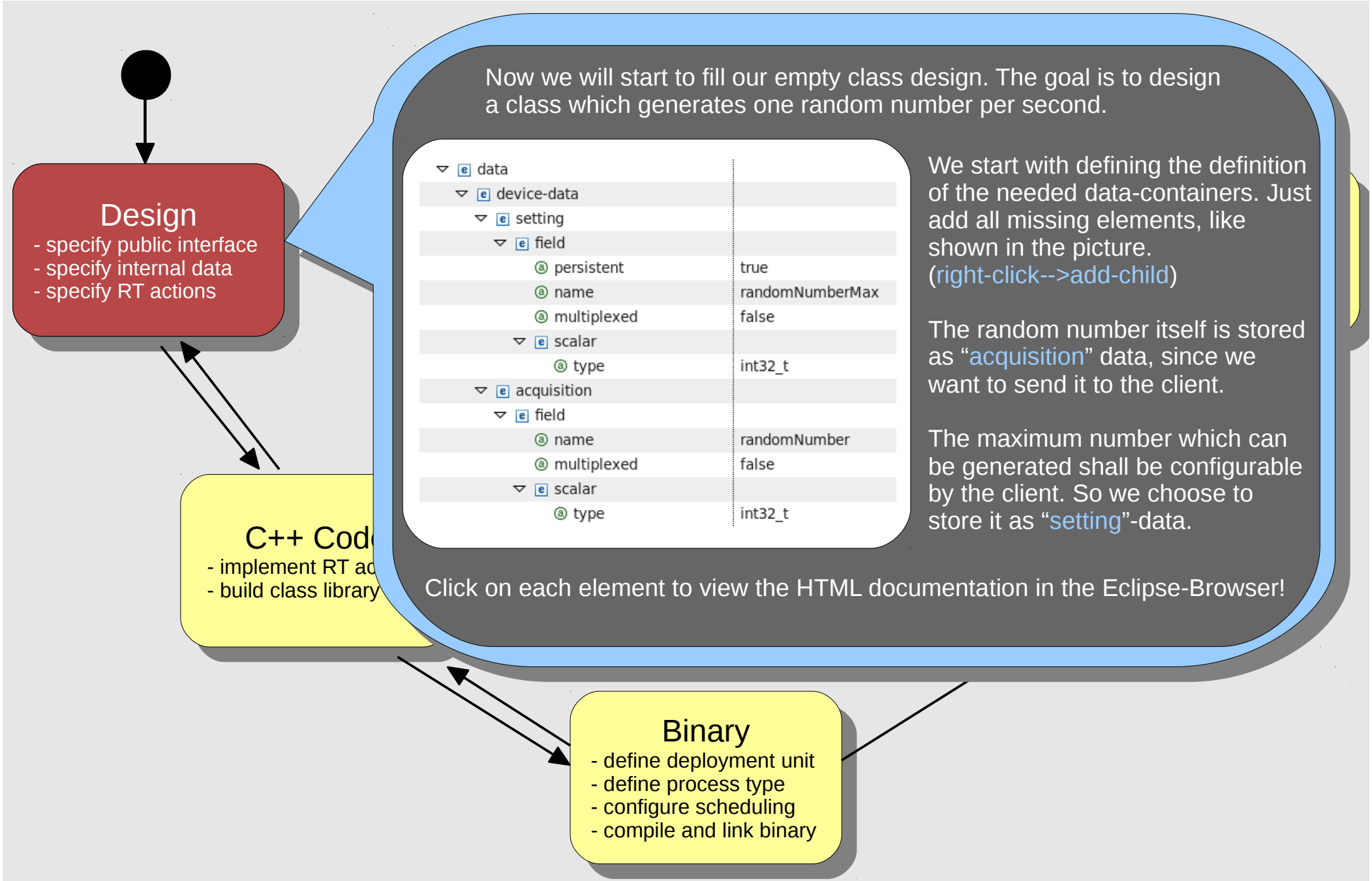
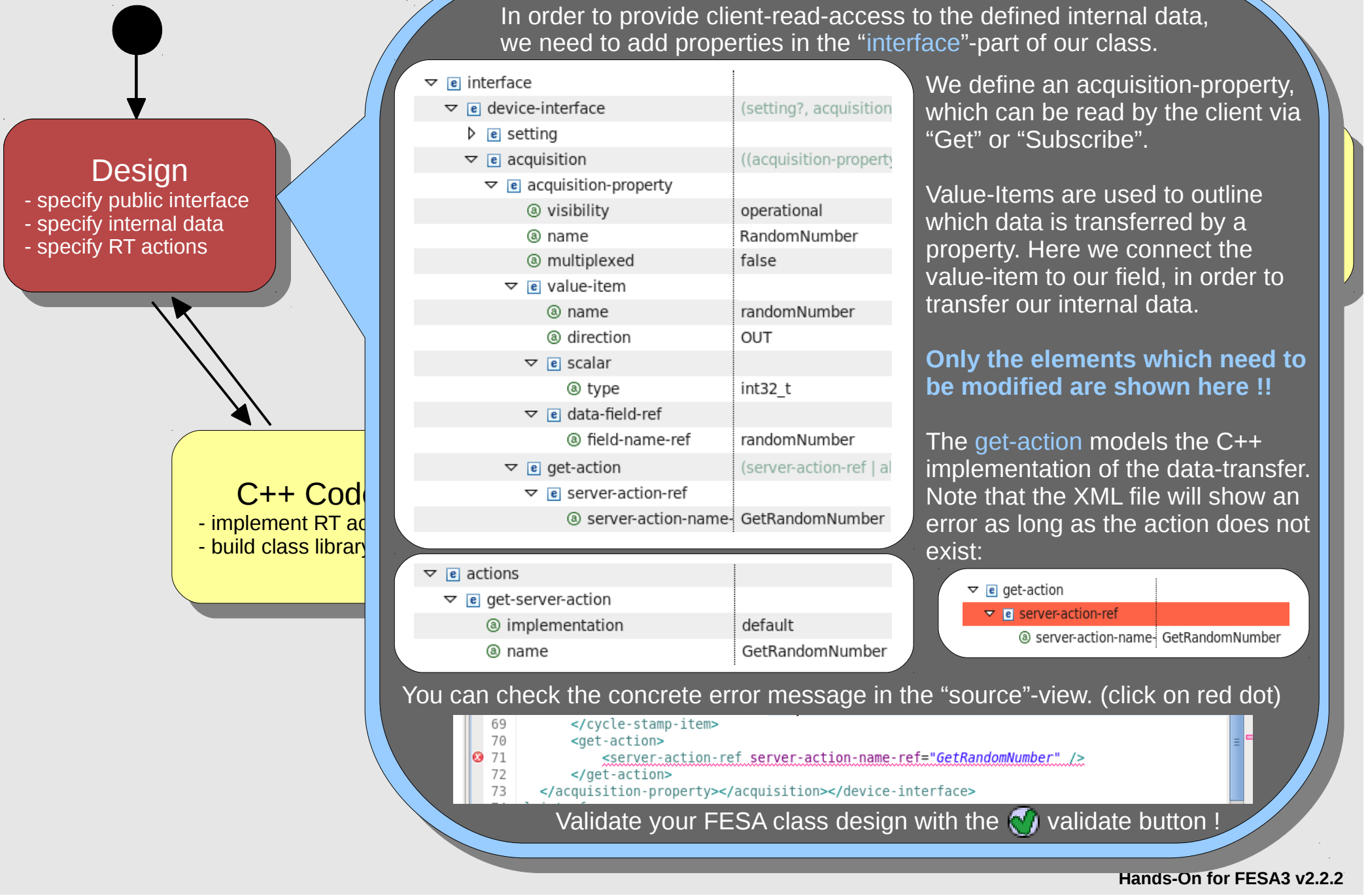
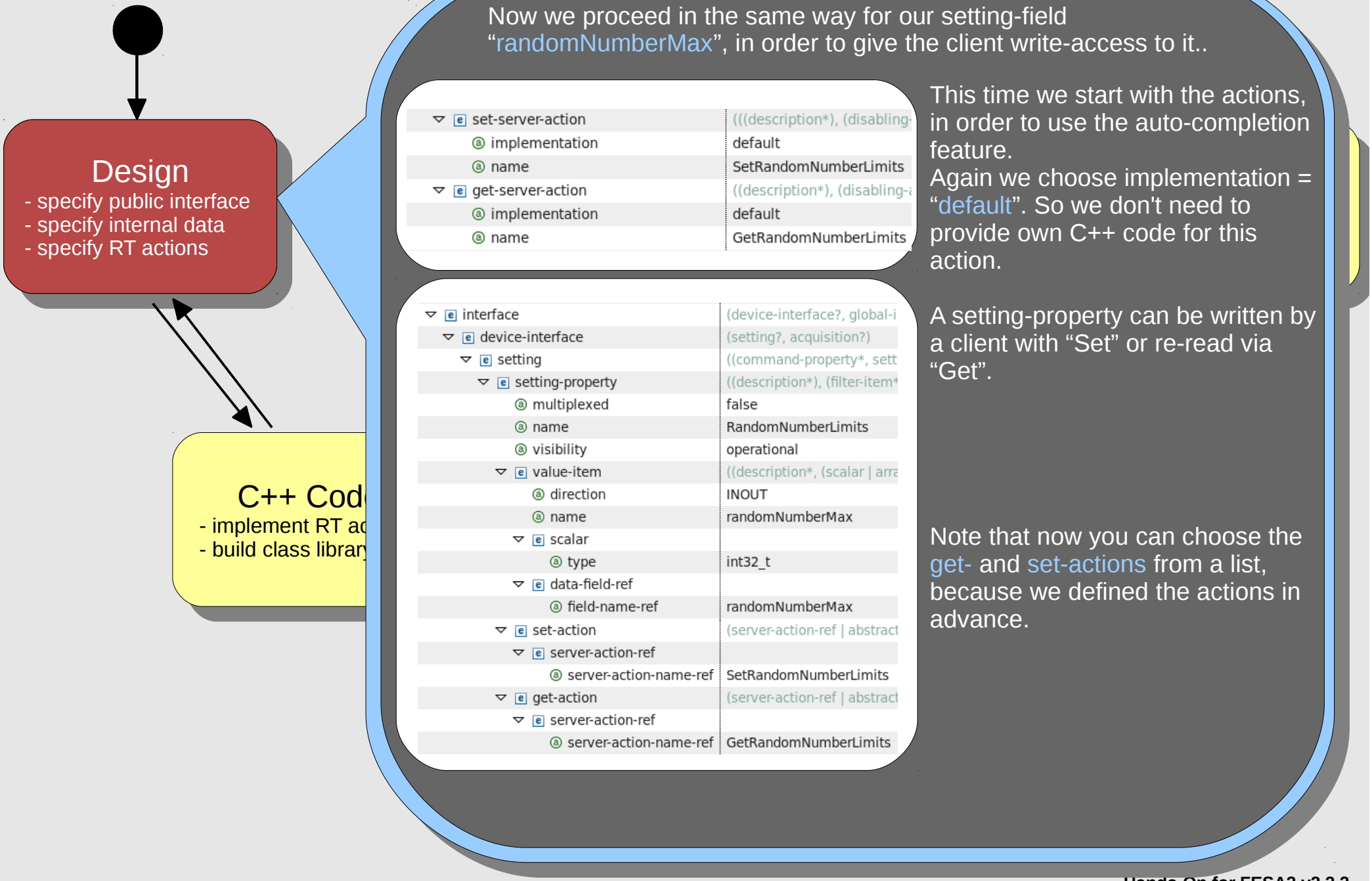

Welcome to the FESA3 Hands-On course
Design
- specify public interface
- specify internal data
- specify RT actions
C++ Code
- implement RT actions
- build class library
Binary
- define deployment unit
- define process type
- configure scheduling
- compile and link binary
Instantiate
- add device instance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

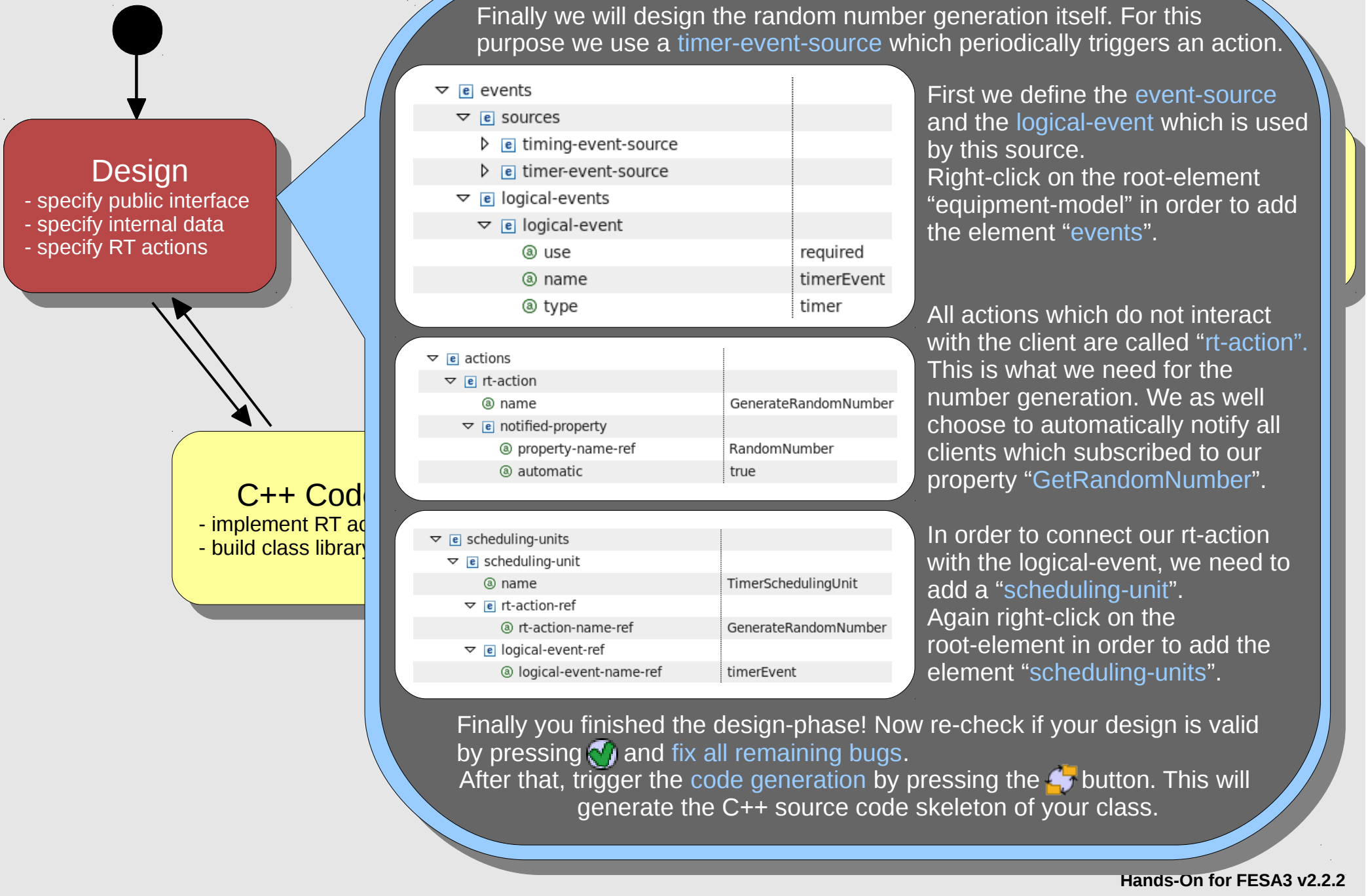

Design
- specify public interface
- specify internal data
- specify RT actions
A FESA binary is built from any number of FESA classes and one FESA deploy-unit. Each class describes one equipment component. The deploy-unit is needed to couple all these classes.
In this tutorial we model the most simple case:
One class which used by one deploy-unit
1. Install the FESA3 Eclipse plug-in as described on the FESA3 web-page
2. Restart Eclipse
3. Open the FESA-Perspective Window → Open Perspective → Other → FESA
4. Open menu File→New→Project.. and choose FESA→New FESA Class
5. Name your class: „HandsOnClass“
6. Choose the newest available FESA3 version
7. Choose the empty template as class template and press “Finish “
C++ Code
- implement RT actions
- build class library
- define device name
Binary
- define deployment unit
- define process type
- configure scheduling
- compile and link binary


Design
- specify public interface
- specify internal data
- specify RT actions
Now you should see an open XML document, called “HandsOnClass.design”.
This document will be used to model your FESA3 class.
In order to view HTML documentation for the document
open the FESA Browser
Window → Show View → other → General → Internal Web 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 → General → Outline
This tutorial will use the design-view for all further steps. Later you may choose a view according to your personal preference.
C++ Code
- implement RT actions
- build class library
- add device instance
- bind logical events
- define initial values
- define device name
Binary
- define deployment unit
- define process type
- configure scheduling
- compile and link binary


Design
- specify public interface
- specify internal data
- specify RT actions
C++ Cod
- implement RT ac
- build class library
Binary
- define deployment unit
- define process type
- configure scheduling
- compile and link binary
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 Eclipse-Browser!


Design
- specify public interface
- specify internal data
- specify RT actions
C++ Cod
- implement RT ac
- build class library
In order to provide client-read-access to the defined internal data, we need to add properties in the “interface”-part of our class.
interface
device-interface (setting?, acquisition
setting
acquisition ((acquisition-property
acquisition-property
visibility operational
name RandomNumber
multiplexed false
value-item
name randomNumber
direction OUT
scalar
type int32_t
data-field-ref
field-name-ref randomNumber
get-action (server-action-ref | al
server-action-ref
server-action-name- GetRandomNumber
actions
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 outline which data is transferred 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-transfer. Note that the XML file will show an error as long as the action does not exist:
get-action
server-action-ref
server-action-name- GetRandomNumber
You can check the concrete error message in the “source”-view. (click on red dot)
69 </cycle-stamp-item>
70 <get-action>
71 <server-action-ref server-action-name-ref="GetRandomNumber" />
72 </get-action>
73 </acquisition-property></acquisition></device-interface>
Validate your FESA class design with the validate button !

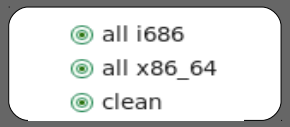

Design
- specify public interface
- specify internal data
- specify RT actions
C++ Cod
- implement RT ac
- build class library
Now we proceed in the same way for our setting-field "randomNumberMax", in order to give the client write-access to it..
set-server-action (((description*), (disabling-
implementation default
name SetRandomNumberLimits
get-server-action ((description*), (disabling-
implementation default
name GetRandomNumberLimits
interface (device-interface?, global-i
device-interface (setting?, acquisition?)
setting ((command-property*, sett
setting-property ((description*), (filter-item*
multiplexed false
name RandomNumberLimits
visibility operational
value-item ((description*, (scalar | arra
direction INOUT
name randomNumberMax
scalar
type int32_t
data-field-ref
field-name-ref randomNumberMax
set-action (server-action-ref | abstract
server-action-ref
server-action-name-ref SetRandomNumberLimits
get-action (server-action-ref | abstract
server-action-ref
server-action-name-ref GetRandomNumberLimits
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.
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.


Design
- specify public interface
- specify internal data
- specify RT actions
C++ Cod
- implement RT ac
- build class library
Finally we will design the random 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-click 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”. This is 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.

After you finished the implementation you can compile your FESA class library. Go to the project folder and execute the make target „all x86_64“. This can be done in Eclipse using the „Make Targets“ view in the FESA or C++ perspective.

By executing the target „clean“ you can remove all object files and libraries from previous builds.



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 built 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 generate the executable FESA binary execute the make target „all x86_64“ as well for the deploy-unit.

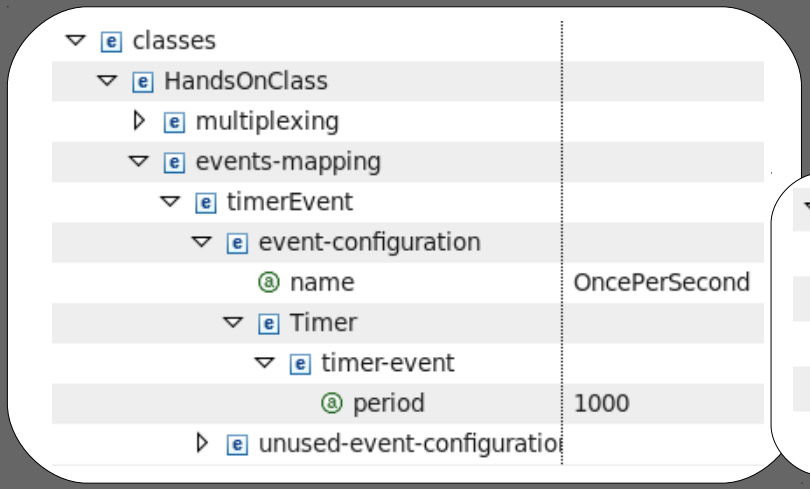
scheduler	(concurrency-layer)+
concurrency-layer	(scheduling-unit)+
name	TimerLayer
prio	19
scheduling-unit	
per-device-group	no
scheduling-unit-name-ref	HandsOnClass::TimerSchedulingUnit

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

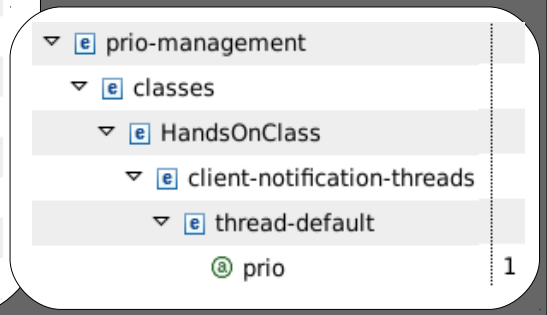
Binary

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

For the next step you need to configure on which front-end (FEC) your binary should run. To do so, open the deploy-unit document and push the „Add FEC“ button. Put in the hostname of the front-end on which you currently work. Select the timing network 'NONE'.

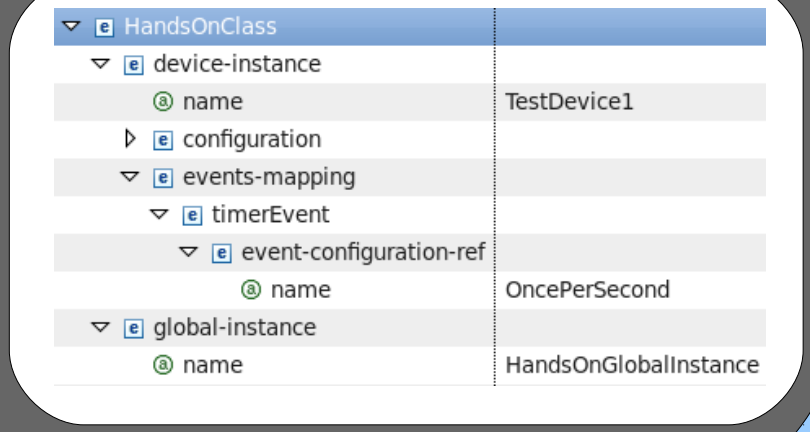


Configure the devices of your class as shown on the screenshots.



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

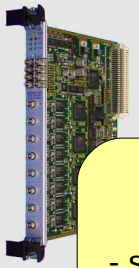
Validate your instantiation document by pressing .



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

rse

- compile and link binary

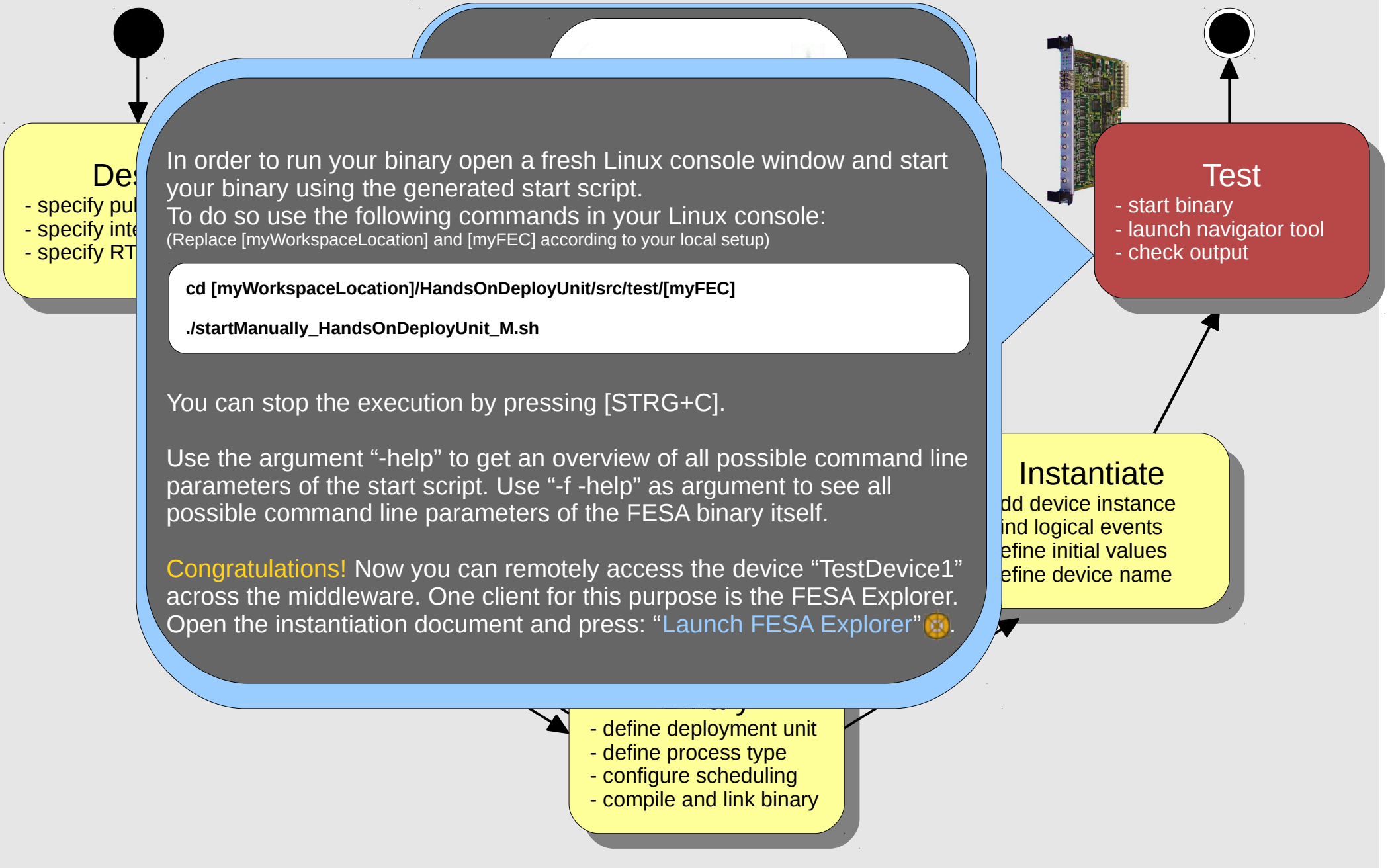


Test

- start binary
- launch navigator tool
- check output

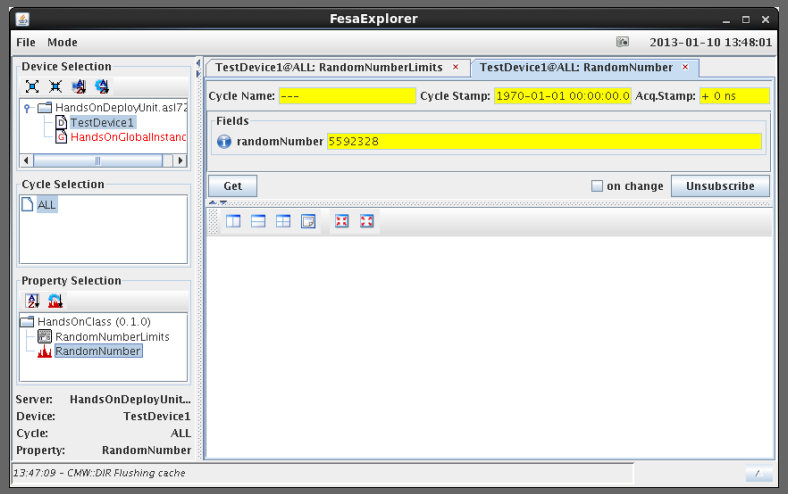
Instantiate

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


Des
- specify pul
- specify inte
- specify RT
In order to run your binary open a fresh Linux console window and start your binary using the generated start script.
To do so use the following commands in your Linux console:
(Replace [myWorkspaceLocation] and [myFEC] according to your local setup)
cd [myWorkspaceLocation]/HandsOnDeployUnit/src/test/[myFEC]
./startManually_HandsOnDeployUnit_M.sh
You can stop the execution by pressing [STRG+C].
Use the argument “-help” to get an overview of all possible command line parameters of the start script. Use “-f -help” as argument to see all possible command line parameters of the FESA binary itself.
Congratulations! Now you can remotely access the device “TestDevice1” across the middleware. One client for this purpose is the FESA Explorer. Open the instantiation document and press: “Launch FESA Explorer”.
Test
- start binary
- launch navigator tool
- check output
Instantiate
dd device instance
ind logical events
efine initial values
efine device name
- define deployment unit
- define process type
- configure scheduling
- compile and link binary

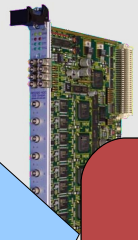
Once the FESA Explorer is open select the “TestDevice1” and double-click on the property “RandomNumberLimits”.
Put a value into the field “randomNumber_max” and press “Set” in order to send the data via the middleware to your class.

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 arrived here you finished the FESA3 HandsOn tutorial. On any problems please do not 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 Browser if you face any unknown FESA XML elements.



Test
- start binary
- launch navigator tool
- check output

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

D
- specify p
- specify i
- specify F