

This manual will guide you through the development of a first, simple FESA3 class

# The Mission

Implement a random number generator using FESA

Requirements:

- Generate one random number per second
- Allow clients to subscribe to the generated random numbers
- Allow clients to specify max-limit for the random numbers (greatest number which can be generated)

On any problems: [fesa-support@gsi.de](mailto:fesa-support@gsi.de)

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** “

## Design

- specify public interface
- specify internal data
- specify RT actions

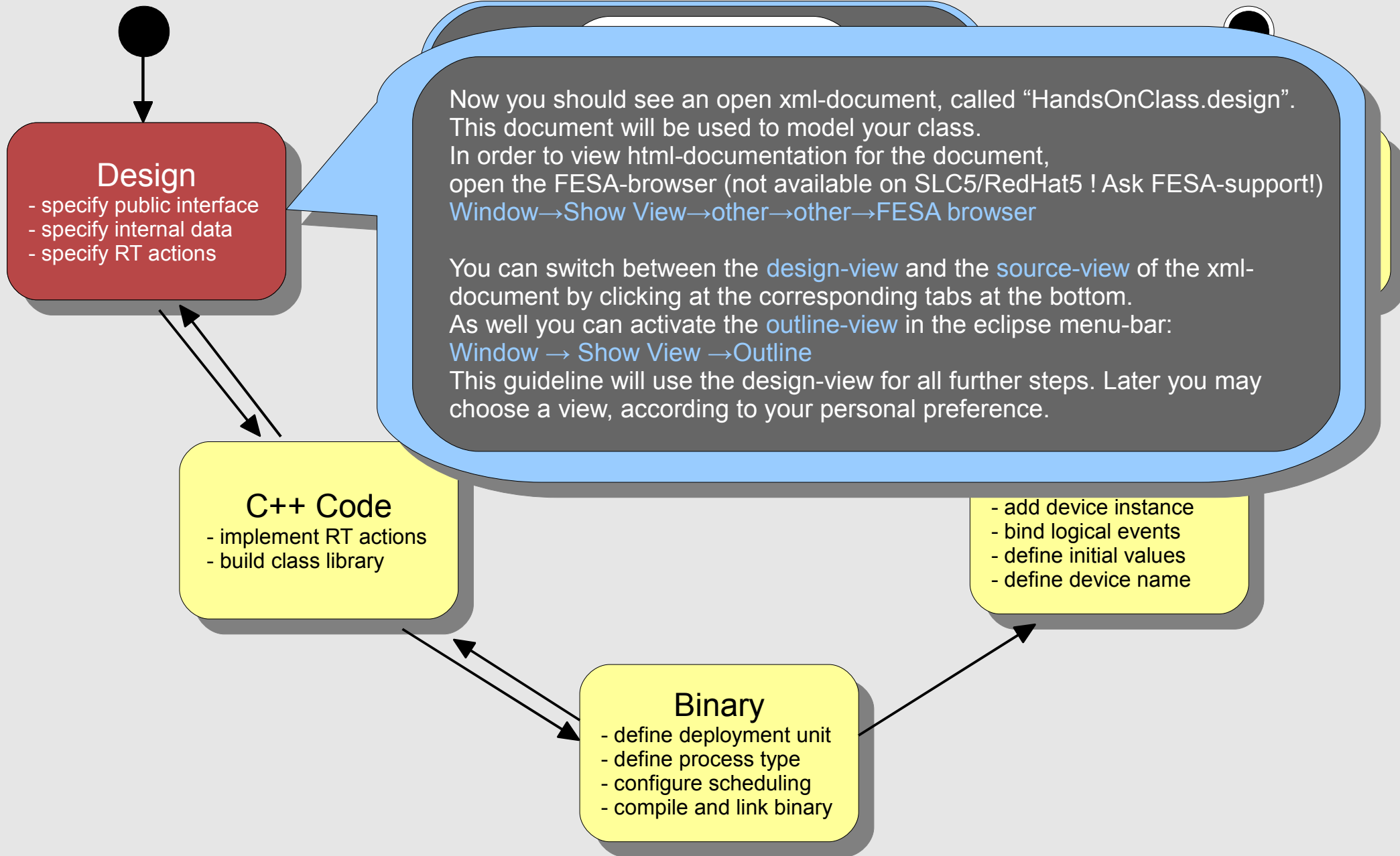
## C++ Code

- implement RT actions
- build class library

## Binary

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

- add device instance
- bind logical events
- define initial values
- define device name

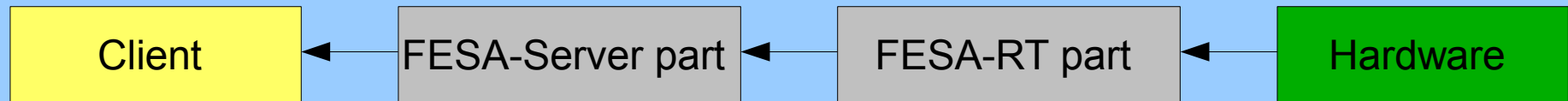


# Theoretic background

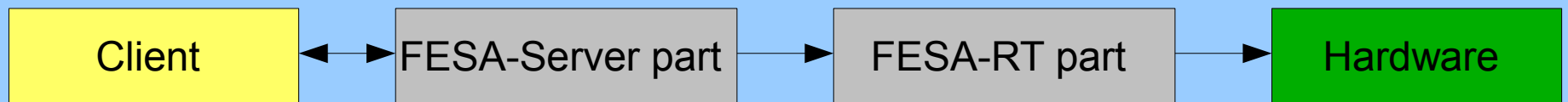
▼ [e] data
▼ [e] device-data
▶ [e] configuration
▶ [e] setting
▶ [e] acquisition
▼ [e] global-data
[e] configuration
[e] setting
[e] acquisition

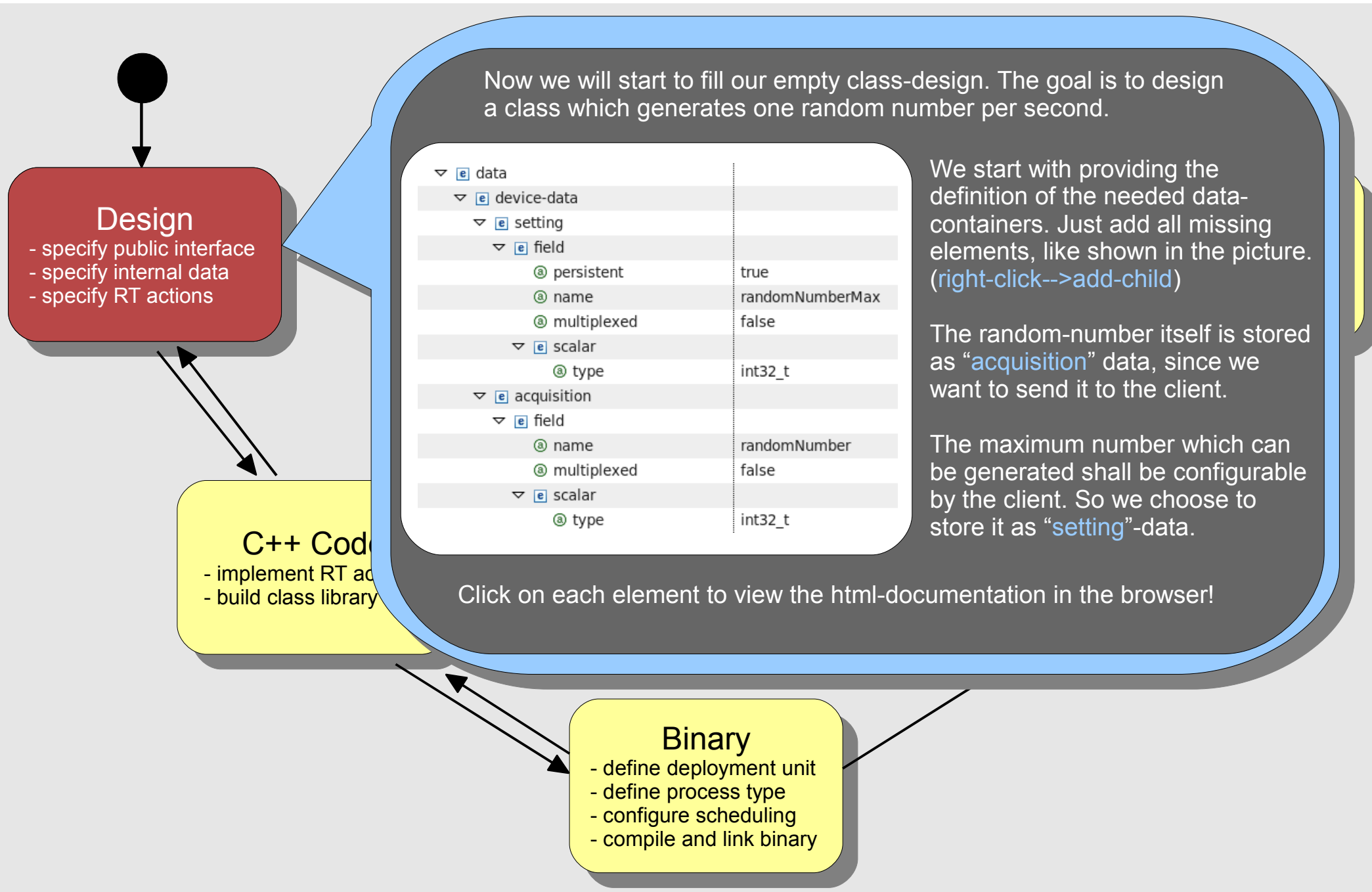
▼ [e] global-data	(configu
[e] configuration	((field?
▼ [e] setting	((cycle
▼ [e] field	(descrip
@a persistent	true
@a name	
@a multiplexed	true
▼ [e] scalar	
@a type	bool

## Data-flow – Acquisition



## Data-flow – Setting





**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

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 providing 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!

# Theoretic background

- spe  
- spe  
- spe

▼ [e] interface	((de
▼ [e] device-interface	(se
▶ [e] setting	((cc
▶ [e] acquisition	((ac
▼ [e] global-interface	(se
▶ [e] setting	((cc
▶ [e] acquisition	((ac

▼ [e] setting-property	((description*), (f
@ [e] visibility	operational
@ [e] name	MyProperty
@ [e] multiplexed	false
▼ [e] value-item	((description*, (sc
@ [e] name	myValueItem
@ [e] direction	INOUT
▼ [e] scalar	
@ [e] type	bool
▼ [e] data-field-ref	
@ [e] field-name-ref	mySettingField
▶ [e] update-flag-item	(description*, cus
▶ [e] cycle-name-item	(description*, arra
▼ [e] set-action	(server-action-ref
▼ [e] server-action-ref	
@ [e] server-action-name-ref	MySetAction
▶ [e] get-action	(server-action-ref

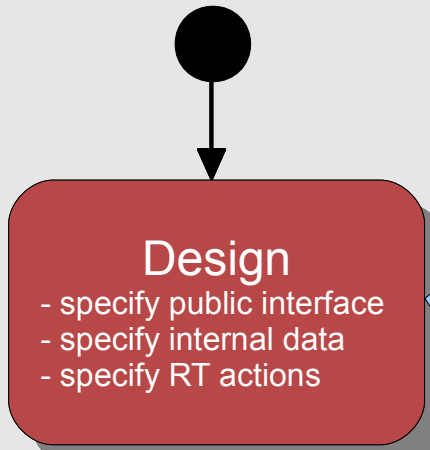
## Client-code-examples :

```

device = rda → getDeviceHandle( devicename, servername );
data =    device → get ( propertyname, cyclename, context );
          device → set ( propertyName, cyclename, settingdata, context );
Request = device → monitorOn ( propertyName, cyclename, replyhandler, context );
          device → monitorOff ( request );
  
```

- actions
  - get/set-server-action (1..n)
    - @implementation
    - @name
    - ...

▼ [e] get-server-action	((description*),
@ [e] implementation	default
@ [e] name	MyGetAction
▼ [e] set-server-action	((description*),
@ [e] implementation	custom
@ [e] name	MySetAction



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   acquisition-property?)
server-action-ref	
server-action-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 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:

actions	
get-server-action	
implementation	default
name	GetRandomNumber

get-action	
server-action-ref	
server-action-name	GetRandomNumber

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

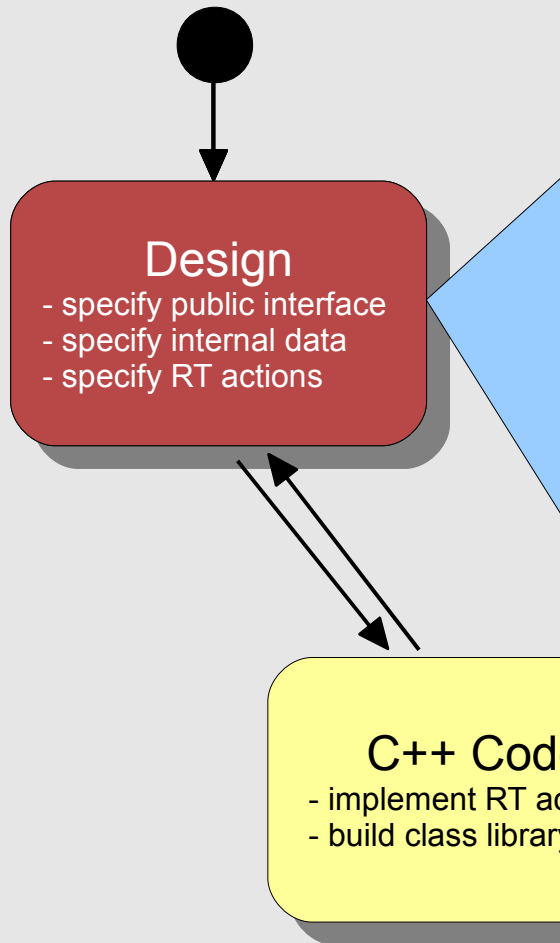
```

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 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..

```

    ▾ [e] set-server-action | (((description*), (disabling-
      @ implementation | default
      @ name | SetRandomNumberLimits
    ▾ [e] get-server-action | (((description*), (disabling-
      @ implementation | default
      @ name | 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.

```

    ▾ [e] interface | (device-interface?, global-in
      ▾ [e] device-interface | (setting?, acquisition?)
        ▾ [e] setting | ((command-property*, settir
          ▾ [e] setting-property | ((description*), (filter-item*,
            @ multiplexed | false
            @ name | RandomNumberLimits
            @ visibility | operational
          ▾ [e] value-item | ((description*, (scalar | array
            @ direction | INOUT
            @ name | randomNumberMax
          ▾ [e] scalar |
            @ type | int32_t
          ▾ [e] data-field-ref |
            @ field-name-ref | randomNumberMax
          ▾ [e] set-action | (server-action-ref | abstract-
            ▾ [e] server-action-ref |
              @ server-action-name-ref | SetRandomNumberLimits
          ▾ [e] get-action | (server-action-ref | abstract-
            ▾ [e] server-action-ref |
  
```

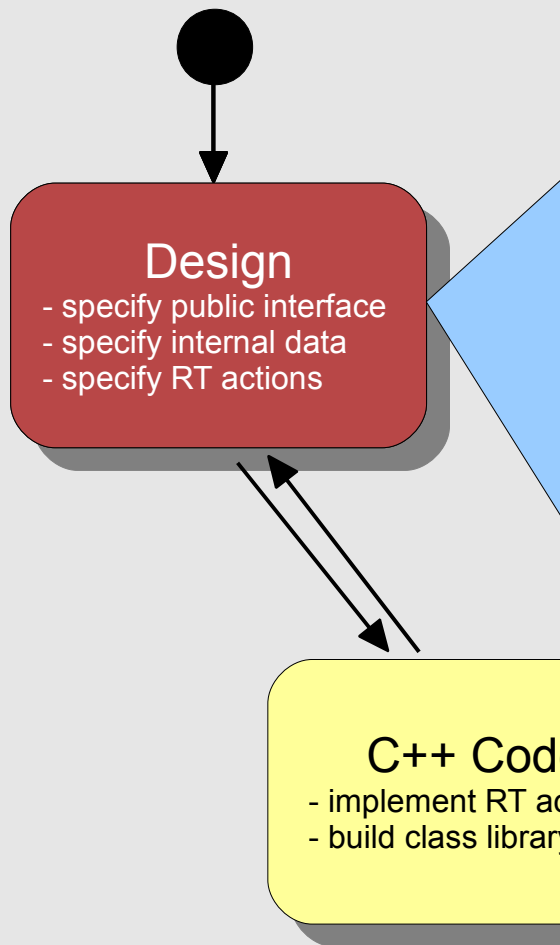
A setting-property can be written by a client with “Set” or re-read via “Get”.

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

# Theoretic background

- events
  - sources
    - timer
    - timing
    - on-demand
    - on-subscription
    - custom
  - logical-events (1 .. n)
    - @use
    - @name
    - @type
- actions
  - rt-action (1..n)
    - @name
      - notified-property (1..n)
        - @property-name-ref
        - @automatic
- scheduling-units
  - scheduling-unit (1..n)
    - @name
    - rt-action-ref
    - logical-event-ref

▷	<a href="#">e</a>	actions
▷	<a href="#">e</a>	events
▷	<a href="#">e</a>	scheduling-units



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

▼ [e] events	
▼ [e] sources	
▶ [e] timing-event-source	
▶ [e] timer-event-source	
▼ [e] logical-events	
▼ [e] 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**".


▼ [e] actions	
▼ [e] rt-action	
ⓐ name	GenerateRandomNumber
▼ [e] 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**".

▼ [e] scheduling-units	
▼ [e] scheduling-unit	
ⓐ name	TimerSchedulingUnit
▼ [e] rt-action-ref	
ⓐ rt-action-name-ref	GenerateRandomNumber
▼ [e] 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.

# Theoretic background

- **C++ coding**

- Use the Wiki !!!

- “FESA3 C++ Code Snippets”
    - “Quick overview on how to develop, run and test a simple FESA3 binary”
    - All slides of the fesa-courses are available in the Wiki !

- First compile the class to enable the eclipse-auto-completion.
  - Than start coding

- Disable the Eclipse-Code Analysis

- Window → Preferences → C++ → Code Analysis
  - Uncheck all boxes !

- spe  
- spe  
- spe

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 „**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 );

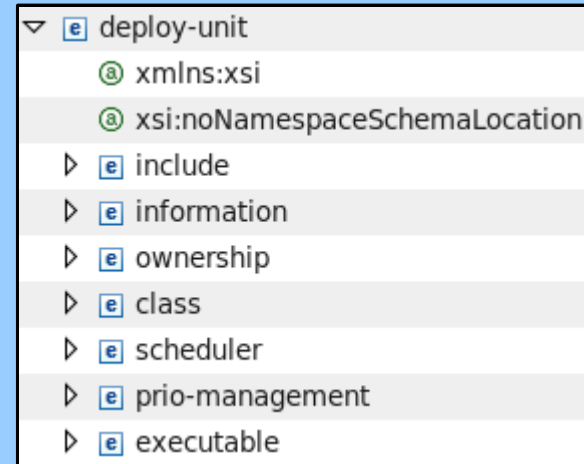
        // produce some output
        std::ostringstream message;
        message << " Produced random number: " << rand_number << " for device: " << (*device)->getName();
        LOG_TRACE_IF(logger, message.str());

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

You may want to copy + paste this source code!

# Theoretic background

- The deployment-unit
  - class (1..n)
  - executable
    - mixed
    - rt
    - server
  - scheduler
    - concurrency-layer (1..n)
      - @name
      - @prio
      - @event-queue-size
    - scheduling-unit (1..n)
      - @per-device-group
      - @scheduling-unit-name-ref
  - prio-management





A screenshot of an XML schema tree for the 'deploy-unit' element. The tree is displayed in a light gray box with a black border. The root element is 'deploy-unit', which is expanded to show its children. The children are: 'xmlns:xsi' (a green circle with an 'a' inside), 'xsi:noNamespaceSchemaLocation' (a green circle with an 'a' inside), 'include' (a blue square with an 'e' inside), 'information' (a blue square with an 'e' inside), 'ownership' (a blue square with an 'e' inside), 'class' (a blue square with an 'e' inside), 'scheduler' (a blue square with an 'e' inside), 'prio-management' (a blue square with an 'e' inside), and 'executable' (a blue square with an 'e' inside).

▼ [e] deploy-unit
@a xmlns:xsi
@a xsi:noNamespaceSchemaLocation
▶ [e] include
▶ [e] information
▶ [e] ownership
▶ [e] class
▶ [e] scheduler
▶ [e] prio-management
▶ [e] executable

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

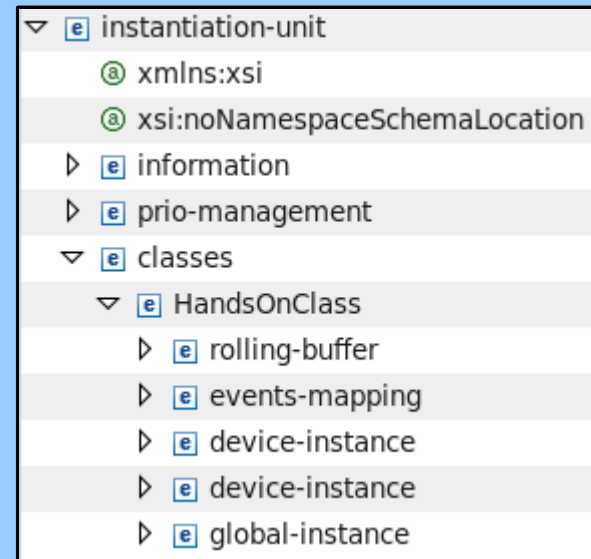
Note: After adding the class-name, **save the document!** The plugin will automatically add the elements “path” and “include”. Now you will be able to pick the right scheduling-unit from a list.

### Binary


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

# Theoretic background


- classes
  - rolling-buffer
  - events-mapping
  - logicalEventName (1..n)
    - eventConfiguration (1..n)
      - @name
      - concrete-event (source-specific design)
- device-instance (1..n)
  - @name
  - configuration
  - setting
    - myField
      - value
    - events-mapping
      - LogicalEventName (1..n)
  - global instance (1)
    - ...
- prio-management






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.

▼ <b>e</b> classes	
▼ <b>e</b> HandsOnClass	
▶ <b>e</b> multiplexing	
▼ <b>e</b> events-mapping	
▼ <b>e</b> timerEvent	
▼ <b>e</b> event-configuration	
ⓐ name	OncePerSecond
▼ <b>e</b> Timer	
▼ <b>e</b> timer-event	
ⓐ period	1000
▶ <b>e</b> 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.

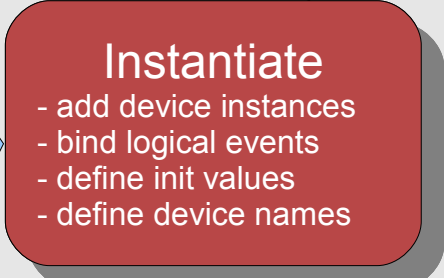
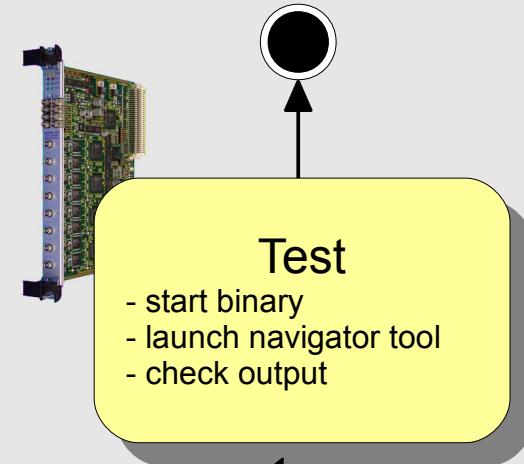
Note that we use the event-configuration “OncePerSecond” which we defined at our own in the section “events-mapping”.

Validate your instantiation document by pressing .

▼ <b>e</b> HandsOnClass	
▼ <b>e</b> device-instance	
ⓐ name	TestDevice1
▶ <b>e</b> configuration	
▼ <b>e</b> events-mapping	
▼ <b>e</b> timerEvent	
▼ <b>e</b> event-configuration-ref	
ⓐ name	OncePerSecond
▼ <b>e</b> global-instance	
ⓐ name	HandsOnGlobalInstance

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

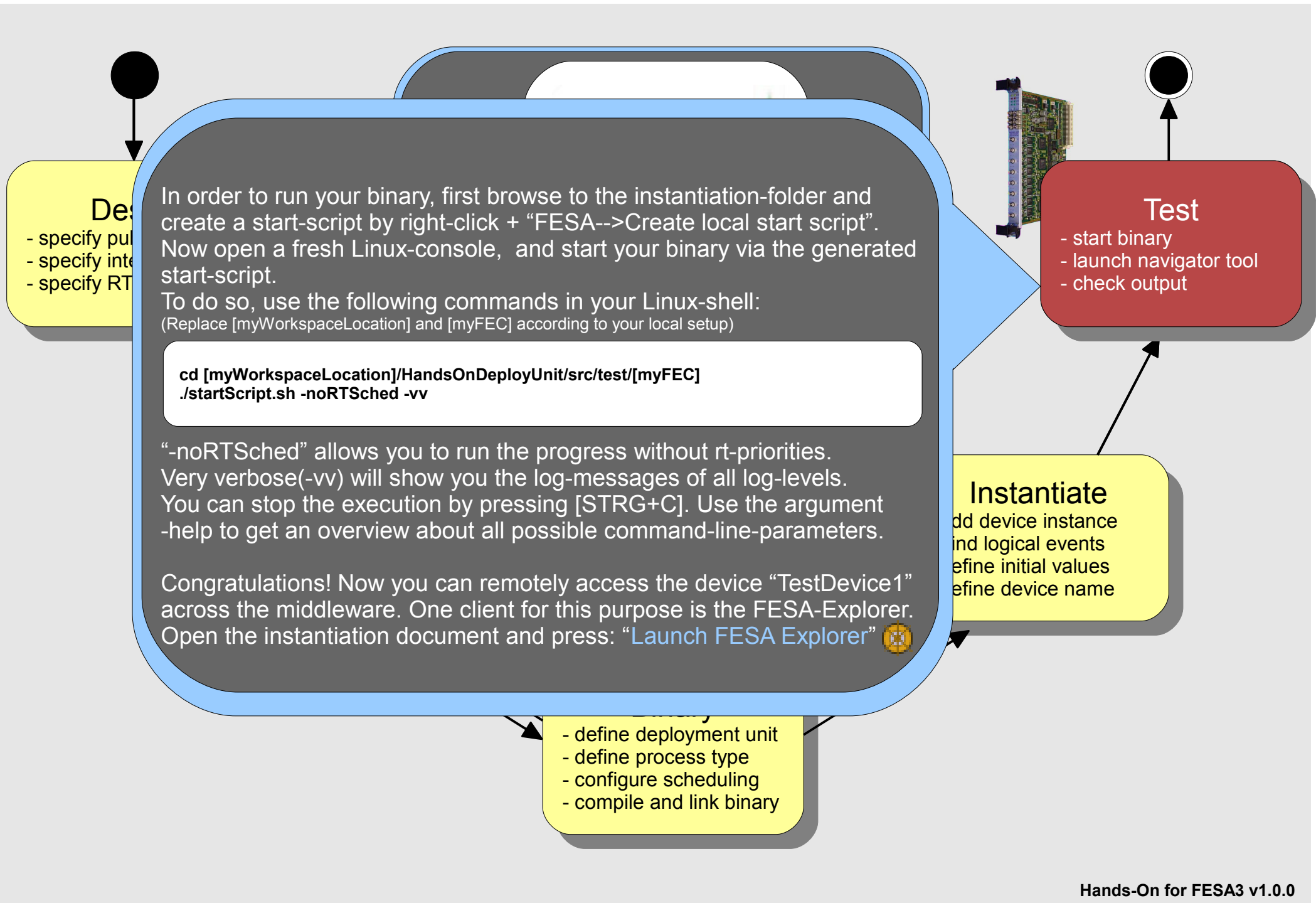
- compile and link binary



# Theoretic background

- startscript
  - Will be generated by plugin for deployment on FEC
  - Since we dont want to deploy, we use our own script
  - `cp ~schwinn/Inx/misc/startScriptHandsOn.sh ~/workspace/HandsOnDeployUnit/src/test/myFEC`
  - All possible application-arguments: `./startScriptHandsOn.sh -help`
- FESA – Explorer
  - What is it used for ?
  - Thanks to the BeamDiagnostics-Department !

- spe  
- spe  
- spe

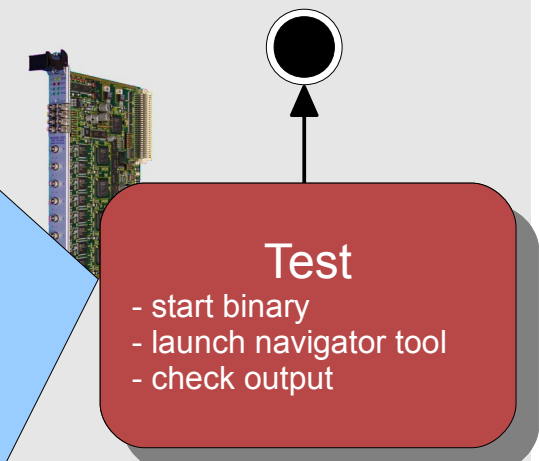
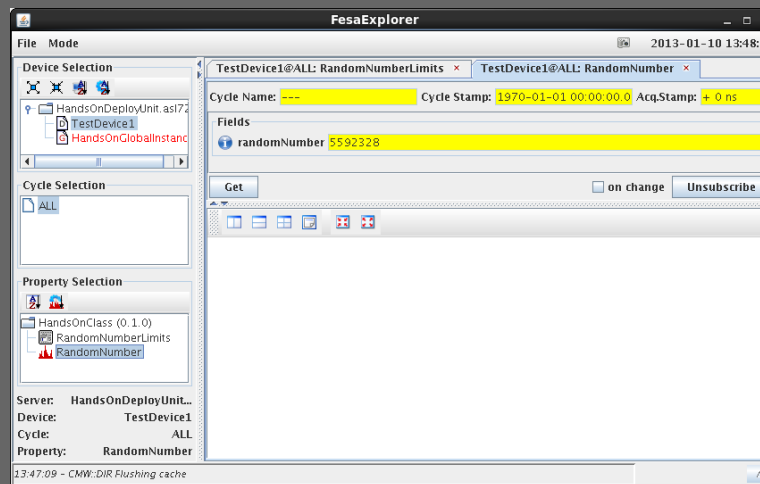


Once the FESA-Explorer opened, select the “TestDevice1” and double-click on the property “RandomNumberLimits”. Put some value into the field “randomNumber\_max” and press “Set”, in order to send the data via the middleware to your class.

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!



- start binary
- launch navigator tool
- check output

- instantiate device instance
- define logical events
- define initial values
- define device name

- specify p
- specify i
- specify F

# The Mission (II)

Implement a random number generator using FESA

## Requirements (I):

- Generate one random number per second
- Allow clients to subscribe to the generated random numbers
- Allow clients to specify max-limit for the random numbers (greatest number which can be generated)

## Requirements (II):

- Allow clients to specify min-limit for the random numbers (smallest number which can be generated)
- Generate some console-output for the class, whenever a client got data. (custom-server-action)

No guideline-support here! On any problems, feel free to ask!

**On any problems: [fesa-support@gsi.de](mailto:fesa-support@gsi.de)**