Generating a FESA-3 PLC class based on your SILECS configuration

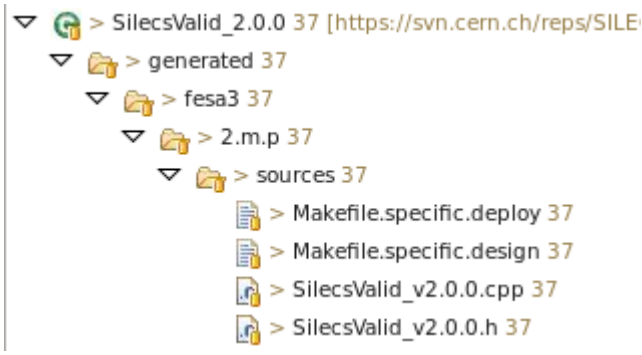
User Manual

(July 2015, SILECS team – BE/CO-FE)

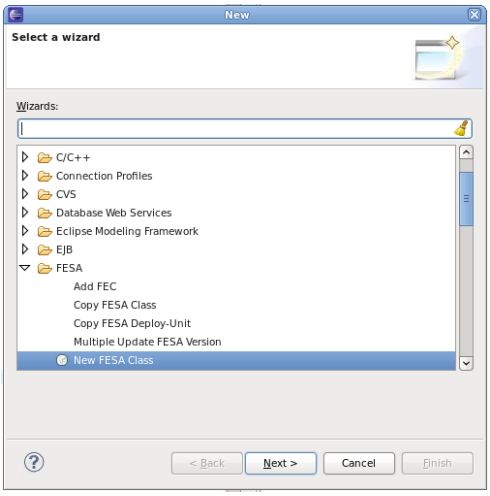
Table of Contents

FESA Eclipse plugin: Importing your design document

Considering our Tutorial SILECS Design, SilecsValid/2.0.0, in the SILECS workspace you will find the following files:

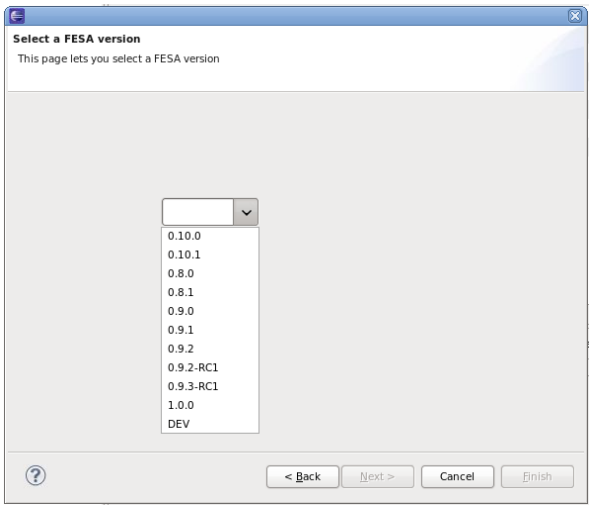


To import the FESA class start Eclipse and go to the menu **File>>New>>Other...** , look for **FESA** and select **New FESA Class**.



⚠ When entering the name for the FESA class, use the **SAME** name of your SILECS class otherwise the compilation will fail!

In the next window, add a name for the class. **You must use the same name of your SILECS class (case sensitive)**. Click **Next** and select the FESA version for the class. Always select the latest FESA release of the major version you selected in SILECS (for example, if in SILECS tool you chose to generate for *FESA Release 0*, then in this window you should select *0.10.1*).



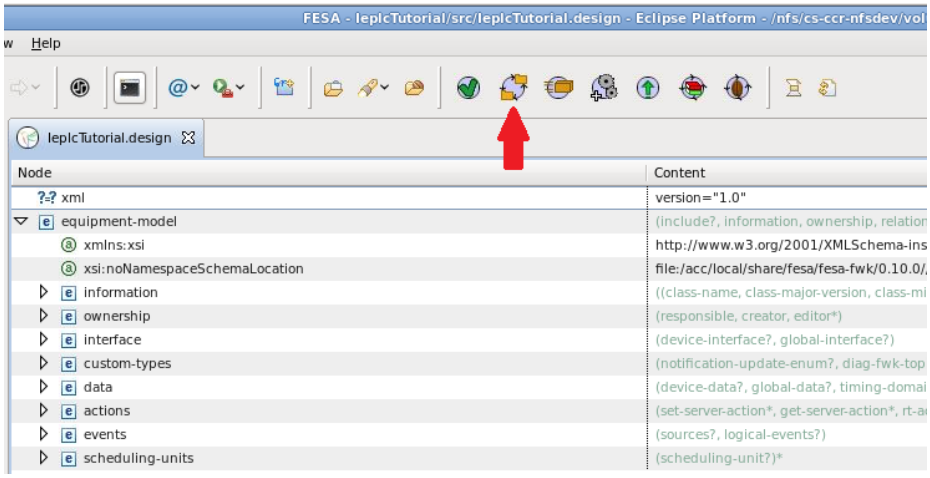
Click **Next** to display the window from which you will be able to import your generated class. Select the radio button **Custom** and a new window will appear allowing you to select the generated XML class design to import. Just go to the location where you chose to retrieve the generated sources and select the design XML file of the class you wish to import in FESA: <myclass>.design.xml

In the next window choose whether you want to commit automatically your class to the FESA SVN and click **Finish**. You should now have your FESA class showing in the list of projects on the left panel of Eclipse. You just need to modify the **Ownership** node of your class design, adding your group in the **responsible** node (i.e. BE/RF, TE/ABT).

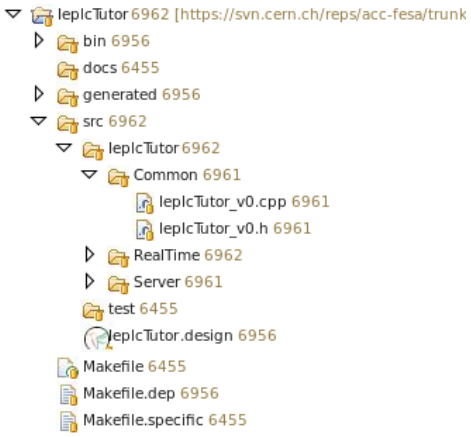
FESA Eclipse plugin: Importing your C++ generated code

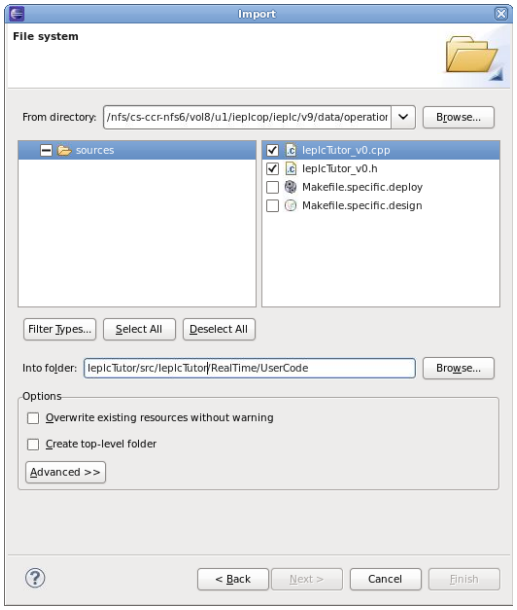
SILECS tool also generates other files, to give you an easier way to interface PLCs and FESA. These files are contained in the **sources** folder of the class and consist of a C++ wrapper to provide access to the FESA fields, the SILECS registers and the PLC controller, and two makefiles, one for FESA design and the other for deploy units. To use them just follow these instructions:

1. Open the FESA Perspective from the Eclipse menu, **Window>>Open Perspective>>Other...>>FESA**
2. Open the FESA class design (contained in the FESA project in the **src** folder)
3. On the menu on the top of the editor click **Generate Source Code**



4. From the project folder on the left panel of Eclipse go to the **Common** folder, right click on it and select **Import...**. Look **for General>>File System**, double-click it, go to the location where you chose to retrieve the generated sources and select the .cpp and .h files contained in the **sources** folder





5. Import the **Makefile.specific.design** file into the root of the FESA class and rename it as **Makefile.specific**.
6. If you are developing a PXI-based control system, open the so-renamed Makefile.specific and enable the WITH_CNV variable (be careful to not insert useless blank after the 'YES').
7. To build the sources, go to the right panel of Eclipse. There you should see the **Make Target** view. If not, just go to **Window>>Show View>>Make Target** to make it visible. The view will show a list of build tasks for your FESA class. Double click on the one you need and wait for the build to be completed.

What to know about the generated FESA design document

The design document is fully generated and does not require additional editing. It consists of:

- One FESA class per SILECS class configuration
- Generated C++ code relies on the SILECS class name. It's possible to use a different name for the FESA class itself but in this case, #include paths lines and namespace must be changed in the <.cpp/.h> generated files every time the FESA class is (re-)generated (not recommended)
- One FESA field is provided for each SILECS register
- One setting-property is provided for each WRITE-ONLY SILECS block (with custom set-action and default get-action)
- One setting-property is provided for each READ-WRITE SILECS block (with custom set-action and custom get-action)
- One acquisition-property is provided for each READ-ONLY SILECS block (with custom get-action only)
- One rt-action, logical-event timer and scheduling-unit for each READ-ONLY SILECS block acquisition (polling mode)
- One server-action for each WRITE-ONLY and READ-WRITE SILECS block setting (on-demand mode)
- One server-action for each READ-WRITE SILECS block acquisition (on-demand mode)
- Naming convention for the FESA objects:

property (setting/acquisition)	<SILECS Block-name>Property
field (setting/acquisition):	<SILECS Register-name>
default get-server-action	Get<SILECS Block-name>
custom server-action	Send/Recv<SILECS Block-name>
custom rt-action	Recv<SILECS Block-name>
scheduling-unit	Recv<SILECS Block-name>Unit
timer logical-event	Recv<SILECS Block-name>Event

What you need to do to finalise the FESA design

- Generate the source code and save the design document
- Implement the SILECS communication (see details in section 'USER code implementation' of this document)
- Compile and generate the FESA class objects

What to know about the FESA deploy document

Now you need to deploy your new PLC class and instantiate the related FESA devices in your deploy-unit. You can use an existing deploy-unit or create a new one if needed.

What you need to do for the FESA deploy document

- Add a class node for each PLC class you want to schedule in the same Server (in a separate concurrency-layer if required)
- Add a scheduler node if missing and edit the name of the concurrency-layer (e.g.: 'mainLayer')
- Add a scheduling-unit node for each acquisition unit of your class (drop-down list or automatic with recent FESA version)
- Import the **Makefile.specific.deploy** file into the root of the FESA deploy unit and rename it as **Makefile.specific**
- If you are developing a PXI-based control system, open the so-renamed Makefile.specific and enable the WITH_CNV variable (be careful to not insert useless blank after the 'YES').
- Generate the source code and save the deploy-unit document
- Compile and generate the FESA executable

What to know about the FESA device-instances document

Number of devices, global and instances parameters depends on each application. The definition of the FESA instances relies also on the SILECS configuration. Thus complete the document by respecting the following conventions:

- One FESA instance per SILECS device (as defined in the SILECS mapping document)
- SILECS devices of the same equipment class can be deployed on (controlled from) different PLCs
- FESA/SILECS device one-to-one relationship is done thanks to predefined fields:
  - A global-instance configuration field to store the name/version of the SILECS equipment class.
  - A device-instance configuration field to store the related SILECS device name (label) and the hostname of its controller (PLC, PXI, etc.)

What you need to do for the FESA device-instances document

- Create a new instance document for the current deploy-unit
- Edit the name of the global-instance (e.g.: <SILECS class-name>-global)
- Edit the global **plcClassName** field (name of the SILECS class)
- Edit the global **plcClassVersion** field (version of the SILECS class)
- Add a new event-configuration for each **Recv** event node of the events-mapping (recommended naming: **Recv<SILECS Block-name>Config**)
- Adjust the timer period of this event
- Add one device-instance per SILECS device, then:
  - edit its FESA name
  - in the configuration node adjust the **plcHostname** and the **plcDeviceLabel** of the related SILECS device
  - in the events-mapping node select the appropriate event-configuration-ref you have just added before

For massive edition it's recommended to duplicate and edit device-instances from the text document directly (Source tab)

USER code implementation

C++ generated code provide a high-level wrapper on top of the SILECS C++ library to free the user from implementing iterative and systematic actions. To implement the PLC communication he simply has to call atomic methods in the appropriate real time and server actions. In the following example FESA class and SILECS class name is **SilecsValid** (version number of the SILECS class is **2.0.0**):

Open the **RealTime/RTDeviceClass.cpp** file (and/or the **Server/ServerDeviceClass.cpp** – see "Where to use the setup() method" section of this document) and insert include reference of the generated <.h> file:

```
#include <SilecsValid/Common/SilecsValid_v2.0.0.h>
```

- In the **specificInit()** method initiate the SILECS communication by calling the **setup()** method:

```
void RTDeviceClass::specificInit()
{
    SilecsValid::setup(SilecsValidServiceLocator_);
}
```

- For each generated **rt-action** file, insert include reference to the generated <.h> file:

```
#include <SilecsValid/Common/SilecsValid_v2.0.0.h>
...
void RecvArrayAQN::execute(fesa::RTEvent* pEvt)
{
    // get the multiplexing context
    MultiplexingContext* pContext = pEvt->getMultiplexingContext();
```

and call the appropriate PLC communication method to update the FESA fields with the related PLC registers:

```
// to get all the devices of the complete cluster (all PLCs which are connected from the setup)
SilecsValid::ArrayAQN.getAllDevices(SilecsValidServiceLocator_, true, pContext);

// to get a specific collection of devices based on the selection-criterion
SilecsValid::ArrayAQN.getSomeDevices(deviceCol_, true, pContext);

// to get all the devices of one particular PLC
Silecs::PLC* pPLC = …
SilecsValid::ArrayAQN.getPLCDevices(pPLC, SilecsValidServiceLocator_, true, pContext);

// to get one particular device
Device* pDevice = …
SilecsValid::ArrayAQN.getOneDevice(pDevice, true, pContext);
```

There is one communication object per SILECS block definition. They must be used in the appropriate action (read strings).

In this example, **getAllDevices** method is responsible to acquire the **ArrayAQN** blocks from the hardware (transmitNow = true) and to update all the FESA fields with the corresponding SILECS registers. Using false with transmitNow parameter would disable the hardware access and simply update the FESA fields with current registers values (not usual in common application).

- For each generated **'Recv' server-action** file, insert include reference to the generated <.h> file and call the appropriate PLC communication method to update the FESA fields with the related PLC registers, e.g.:

```
#include <SilecsValid/Common/SilecsValid_v2.0.0.h>
...
void RecvScalarCFG::execute(fesa::RequestEvent* pEvt, Device* pDev, ScalarCFGProperty_DataType&
data)
{
    // get the multiplexing context
    MultiplexingContext* pContext = pEvt->getMultiplexingContext();

    // on-demand get to the related PLC device
    SilecsValid::ScalarCFG.getOneDevice(pDevice, true, pContext);
   ...
 }
```

In this example, **getOneDevice** method is responsible to acquire the **ScalarCFG** block from the hardware (transmitNow = true) and to update all the FESA fields of that particular device with the corresponding SILECS registers.

- For each generated **'Send' server-action** file, insert include reference to the generated <.h> file:

```
#include <SilecsValid/Common/SilecsValid_v2.0.0.h>
...
void SendScalarCFG::execute(fesa::RequestEvent* pEvt, Device* pDev, ScalarCFGProperty_DataType&
data)
{
    // get the multiplexing context
     MultiplexingContext* pContext = pEvt->getMultiplexingContext();
```

and call the appropriate PLC communication method to update the SILECS registers with the related FESA fields:

```
  // set registers of all devices of the complete cluster (all connected PLCs)
   SilecsValid::ScalarCFG.setAllDevices(SilecsValidServiceLocator_, true, pContext);

   // set registers of a specific collection of devices based on the selection-criterion
   SilecsValid::ScalarCFG.setSomeDevices(deviceCol, true, pContext);

   // set registers of all the devices of one particular PLC
   Silecs::PLC* pPLC = …
   SilecsValid::ScalarCFG.setPLCDevices(pPLC, SilecsValidServiceLocator, true, pContext);

   // set registers of the related PLC device
   Device* pDevice = …
   SilecsValid::ScalarCFG.setOneDevice(pDevice, data, true, pContext);
 }
```

setAllDevices, setSomeDevices and **setPLCDevices** methods can used to update all the SILECS registers of a device collection from the corresponding FESA fields and send the block (ScalarCFG in that example) to the hardware (transmitNow = true). Just update the SILECS registers without sending the block if 'transmitNow' is false.

setOneDevice method provides an additional API to update the SILECS registers from the RDA data object. This particular API method must be used to send data on-demand from the server-action.

Where to use the setup() method

Depending on the executable you want to use, you need to call the setup() in the right **specificInit()**. The following table shows where to call it from, depending on the synchronisation mode of the SILECS registers and on the type of FESA executable you want to use.

SILECS register synchro	FESA-3 executable			
	Mixed	**Server**	**RT**	**Server + RT**
Read-Write	RT or Server	Server	RT	Server + RT
Read-only	RT or Server	Server	RT	Server + RT
Write-only	RT or Server	Server	RT	Server + RT

Exception handling

The generated code contains the code necessary for setting up the SILECS service using the setup() method. In case of connection problems with the controller or in the synchronisation between the SILECS registers and the FESA fields, the method throws an exception.

Handling exceptions in the custom code (real time and server actions) is left to the user. For example, in a real time action the user could use the following code:

```
void RecvArrayAQN::execute(fesa::RTEvent* pEvt)
{
    // get the multiplexing context
    MultiplexingContext* pContext = pEvt->getMultiplexingContext();
    for (std::vector<Device*>::iterator itr = deviceCol_.begin(); itr != deviceCol_.end(); ++itr)
    {
       Device* pDev = (*itr);
       try
       {
           SilecsValid::ArrayAQN.getOneDevice(pDev, true, pContext);
       }
       catch(const Silecs::Exception& ex)
       {
           throw fesa::FesaException(__FILE__, __LINE__, ex.getMessage());
       }
    }
}
```

For further information, please contact: silecs-support@cern.ch

or have a look to the SILECS wikis: https://wikis.cern.ch/display/SIL/SILECs+Home