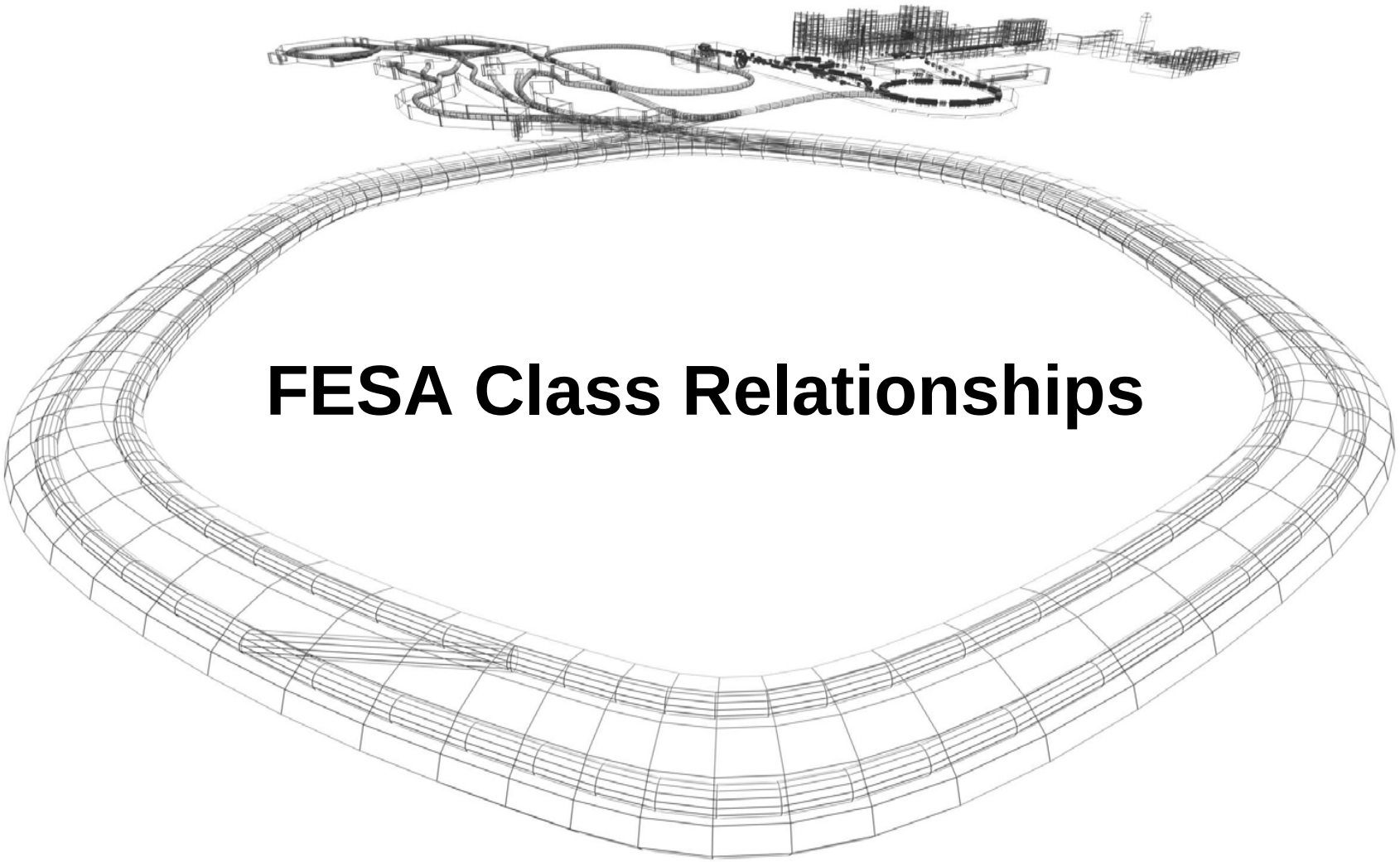
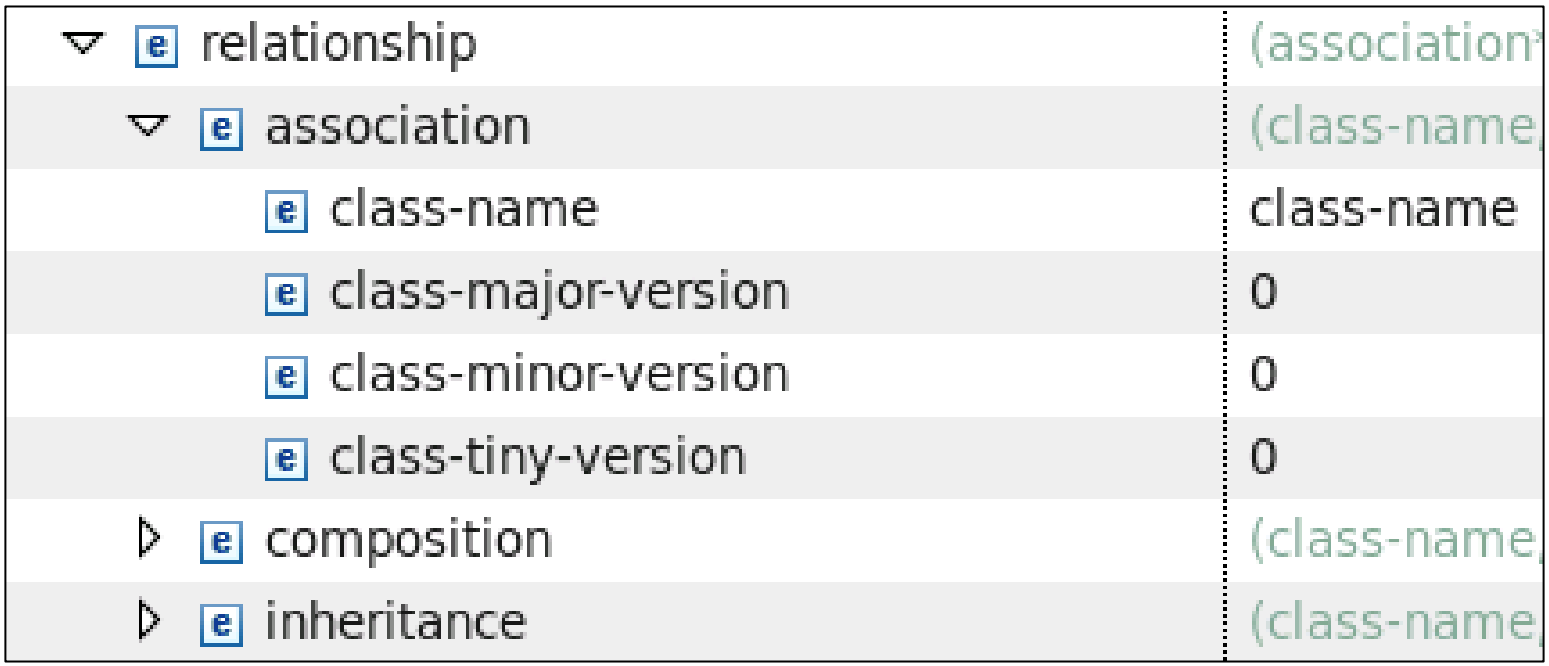


FESA Class Relationships

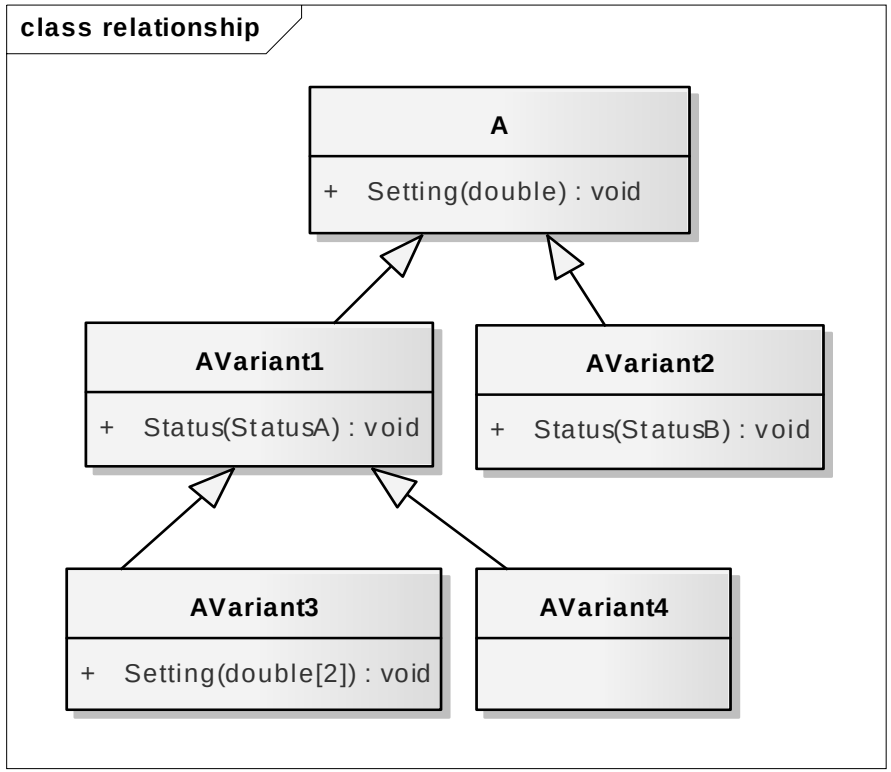

relationship
(association
association
(class-name,
class-name
class-name
class-major-version
0
class-minor-version
0
class-tiny-version
0
composition
(class-name,
inheritance
(class-name,

Topics

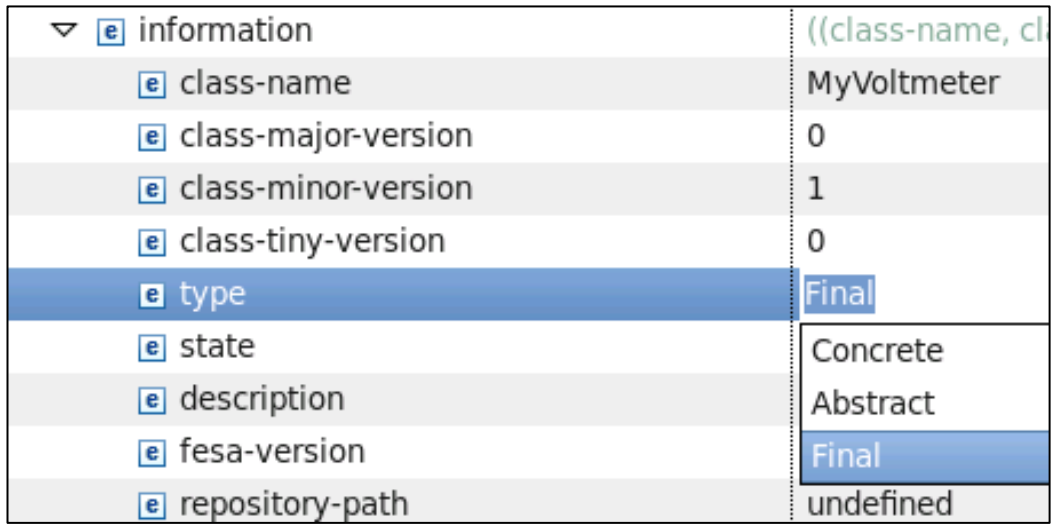
- **Inheritance**
- **Composition**
- **Association**

Inheritance

- FESA Inheritance definition:
  - Properties/RTActions defined by a base-class are available for any sub-class
  - Properties/RTAction defined in a base-class can be overridden (explicitly)
  - Fields of the base-class can be used in any sub-class
  - Custom-types of the base can be used in any sub-class
- Example:
  - PowerSupplyBase
  - MyPowerSupply



Inheritance



Abstract
- cannot be instantiated (no devices)
- used for base-classes

Concrete
- no restriction, can be instantiated and/or extended

Final
- Can only be instantiated, cannot be extended

Inheritance

```
using namespace fesa;
namespace InheritanceTestChild
{
class Device :  public InheritanceTestBase::Device
{
    public:
        Device();

        SettingFieldScalar<int32_t> myChildField;

    private:
};
}
#endif
```

generated code

actions	(set-server-action*, get-serve
rt-action	((description*), (triggered-ev
name	MyRTAction
notified-property	
property-name-ref	InheritanceTestBase::Status
automatic	false

Inheritance

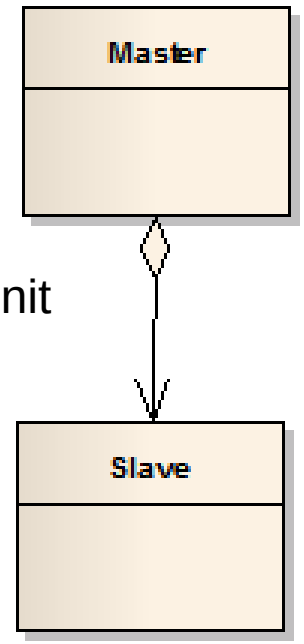
Node	Content
?=? xml	version="1.0" encoding="UTF-8"
e deploy-unit	(include?, information, ownership, class+, sched
@ xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
@ xsi:noNamespaceSchemaLocation	file:/common/home/bel/schwinn/lnx/tmp/opt/fesa
e include	(class-scheduling-view+)
e information	(deploy-unit-name, deploy-unit-major-version, d
e ownership	(responsible, creator, editor*)
e class	((class-name, class-major-version, class-minor-v
e class	((class-name, class-major-version, class-minor-v
e scheduler	(concurrency-layer)+
e executable	(rt?, server?, mixed?)

Topics

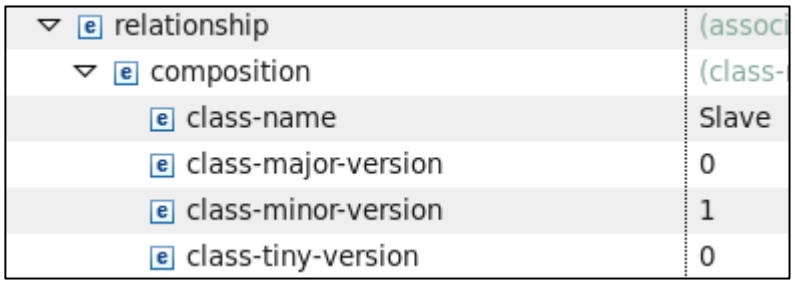
- Inheritance
- **Composition**
- Association

Composition

- Strong coupling (“Master” can access fields of “Slave”)
- Deployed on a single computer, by the use of one deployment-unit
  - Priority management
  - Reusability of compound-classes
- Example:
  - InterfaceModuleMaster + ChannelCards

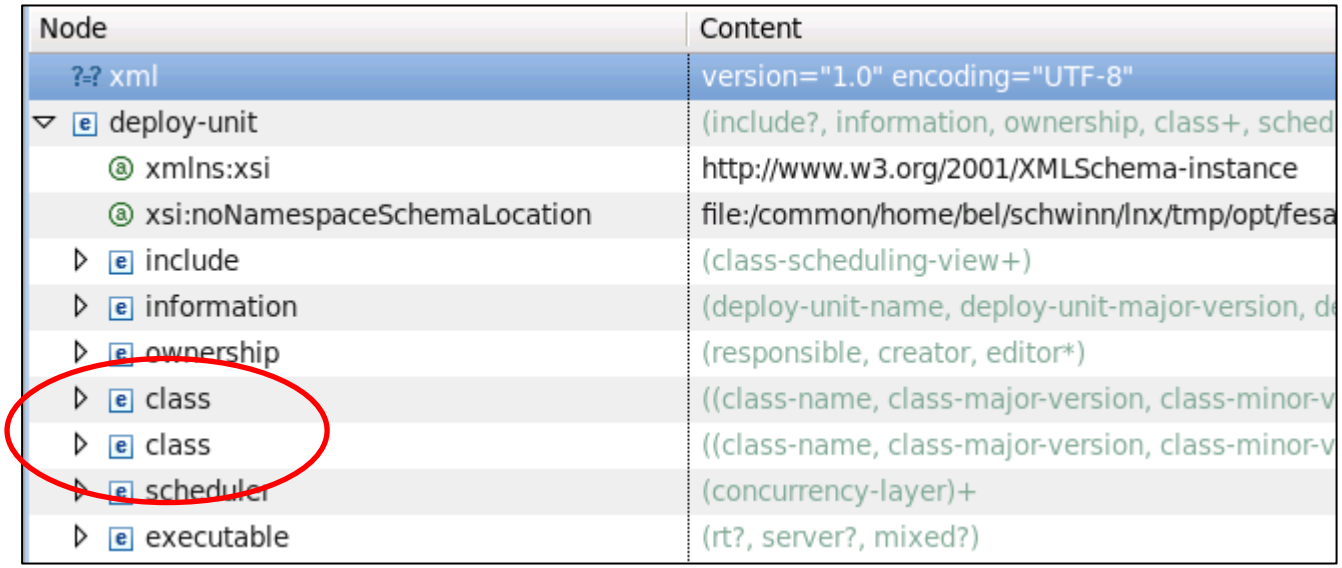


Composition




```
namespace Master
{
void MyAction::execute(fesa::RTEvent* pEvt)
{
    Slave::Device* slave = this->SlaveServiceLocator_->getDevice("myDevice");
    bool myValue = slave->myField.get(pEvt->getMultiplexingContext());
    this->SlaveServiceLocator_->getDeviceCollection();
    this->SlaveServiceLocator_->getGlobalDevice();
}
```

Composition

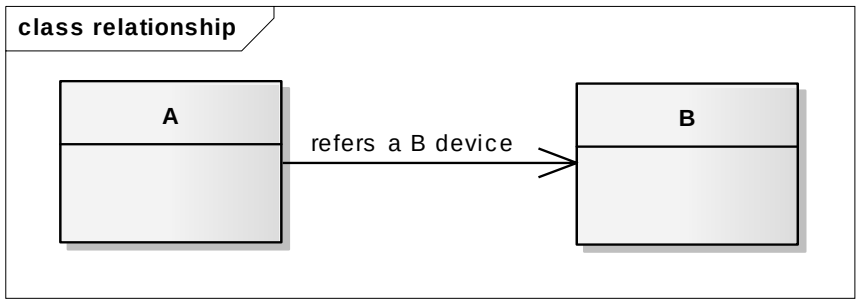


Topics

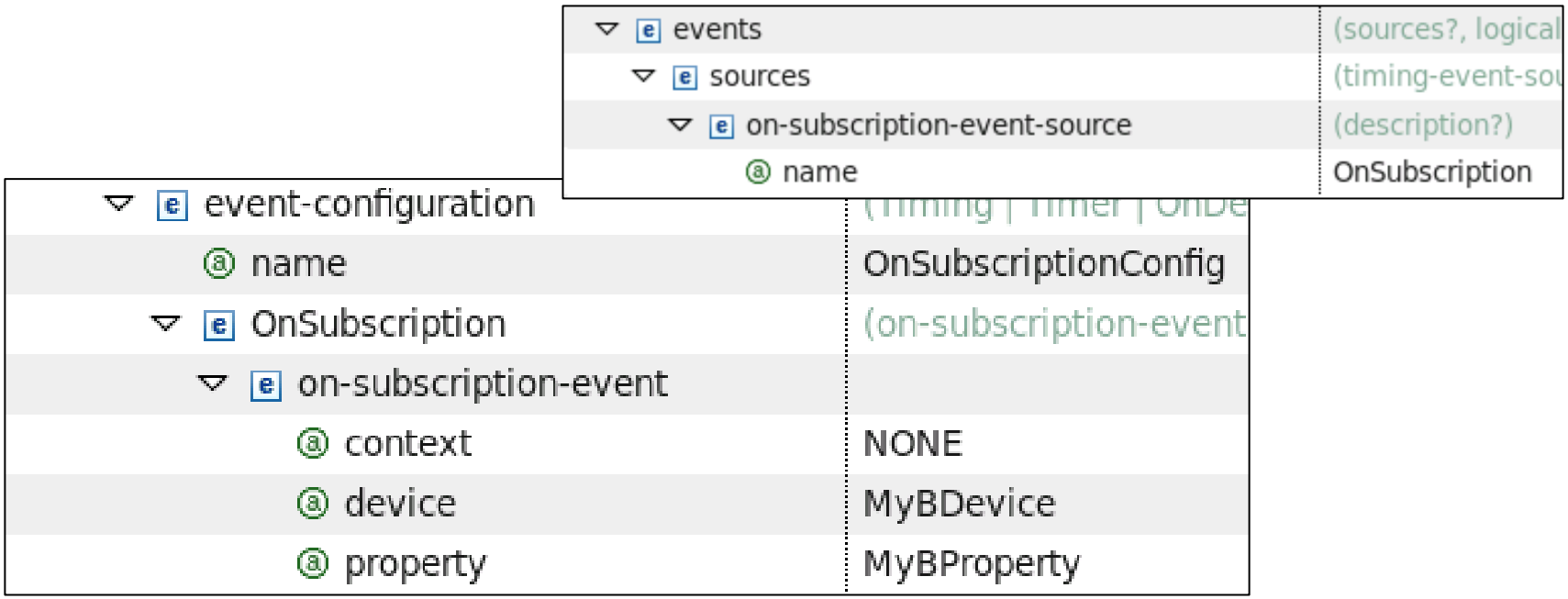
- Inheritance
- Composition
- **Association**

Association

- Light coupling through the Middleware (properties).
- Stand-alone FESA classes running independently.
- Independent lifetime: “A” can shutdown while “B” is still running.
- The classes can be deployed on different computers.
- Example: MyPowerSupply + DataAggregator



Association




```
void RTOnSubscription::execute(fesa::RTEvent* pEvt)
{
    PayloadOnSubscription_DataType payloadData;

    const OnSubscriptionRTEventPayload* payload =
            dynamic_cast<const OnSubscriptionRTEventPayload*> (pEvt->getPayload().get());

    rdaData data = payload->getRDAData();
    payloadData.setData(data,false,false);
}
```

Mission

- PowerSupplyBase
  - Create an abstract base-class “PowerSupplyBase” by using the GSIClassTemplate
  - Set information/type to “abstract”
  - Generate code + compile
- MyPowerSupply
  - Create a child-class “MyPowerSupply” which inherits from “PowerSupplyBase” (add relationship/inheritance)
  - Define a RTAction which notifies the Property “Status” from the baseclass (manual notification)
  - Inside the RTAction, set some value to the field “control” of the base-class and print the value to the screen
  - Trigger the RT action periodically, once a second
- FESA-Explorer
  - Try to subscribe to the property (support for Status-prop?)

On any problem: fesa-support@gsi.de

Mission

```
for (std::vector<Device*>::iterator itr = deviceCol_.begin(); itr != deviceCol_.end(); ++itr)
{
    try
    {
        // You can find all custom types in the file PowerSupplyBase/generated/cpp/PowerSupplyBa
        DEVICE_CONTROL::DEVICE_CONTROL controlState;

        // We just toggle the value of the controlState with each execution of this RTAction
        if(toggle_)
        {
            controlState = DEVICE_CONTROL::LOCAL;
            std::cout << "New value of control is 'local'." << std::endl;
            toggle_ = false;
        }
        else
        {
            controlState = DEVICE_CONTROL::REMOTE;
            std::cout << "New value of control is 'remote'." << std::endl;
            toggle_ = true;
        }

        // We set the field of the base-class
        (*itr)->control.set(controlState,pEvt->getMultiplexingContext());


    }
    catch (...)
    {
        std::cout << "Some error happened in the user-code !!!" << std::endl;
        throw;
    }
```