Simulated Railroad Framework, http://simulrr.sourceforge.net
Synopsis: 000_Synopsis

This file valid for step 0033.10
Issue Date: 2017-03-17

Extensibility, The Core Prototypes

1 Synopsis

During first implementation of the support for tracks, turnouts and vehicles in the SRR Controller and Module Coordinator, it turned out that the SRR Controller and the Module Coordinator became too big. It became impossible to handle them.

As a solution, the SRR Controller and the Module Coordinator were split into modules,
- a base module and
- an extension module (which was called "Train Manager").

This happened, by the way, in autumn 2009.

Now, during re-design in step 0033 this solution is improved and re-thought more thoroughly.

2 Purpose

The SRR/SMUOS Framework consists – roughly – from three kinds of X3D nodes:
(1) the Simple Scene Controller (former SRR Controller),
(2) the Module Coordinator and
(3) the MIDAS Objects (former SRR Objects).

(1) is the – let's say – lowest layer. It provides central facilities and it is used by (2) and (3). It is used by the frame author, too.
(2) is a medium layer, which uses (1) and which is used by (3). The module authors use it, too.
(3) is the highest layer, which is used only by the model authors. It uses all other parts of the SRR/SMUOS Framework ((1) and (2)).

Now the SRR/SMUOS Framework is extensible in following ways.

First, you can develop your own MIDAS Objects. This first way provides a pretty good kind of extensibility. In particular this is true, because the SRR/SMUOS Framework contains some basic functions that are needed by nearly any MIDAS Object (see the description of the MIDAS Base (MIB) in paper 301_MidasObjects).

This first way may be acceptable for simple extensions of the SRR/SMUOS Framework.

However, when developing your own MIDAS Objects you are dependent on the teams of the simulrr/smuos projects to adapt the module coordinator and the Simple Scene Controller to the specific needs of your MIDAS Objects (if necessary).

Second, the SMUOS Framework provides the possibility to develop extensions of the Simple Scene Controller and/or of the Module Coordinator. MIDAS Objects may be developed that depend on your extensions of the SSC and/or of the Module Coordinator.

This second way is more general and more powerful. This is the way that is described in the present paper.

3 External View – How To Use Existing SMUOS Extensions

Usually, an extension of the SMUOS Framework consists of one or both of
- an SSC Extension and
- a Module Coordinator Extension

and it is accompanied by a set of MIDAS Objects, the so-called
- Extension MIDAS Objects.

Note: The current version of the SMUOS Framework is accompanied by three example extensions (where the SMUOS Framework itself and the first two extensions are provided by the project http://smuos.sourceforge.net and where the SRR Framework consists of all this and of the third extension – http://simulrr.sourceforge.net):

- Beamer Manager..extends the SSC; does not maintain a global state; accompanied by 2 MIDAS Objects (Beamer and BeamerDestination)
- Key Manager.....extends the SSC; maintains a global state; accompanied by 3 MIDAS Objects (KeyContainer, LockA and LockB)
- Train Manager*).extends the SSC and the Module Coordinator; maintains global states; supports a class of unbound models (rail vehicles); accompanied by quite a lot of MIDAS Objects

*) not yet finished

An SSC Extension must be registered at the SSC Base, before it can be used.

A Module Coordinator Extension must be registered at the Module Coordinator Base, before it can be used in that module.

3.1 How to Register a Simple Scene Controller Extension

Best to have a look at the example in the official demo layout. The main file of the official demo layout is written in X3D syntax (not in VRML syntax), the part that instantiates the SSC Base and all of its extensions, follows:

```
<!--  ************************************************************  -->
<!--  *******  Simple Scene Controller                          *******  -->
<!--  *******  The Simple Scene Controller is the central part of  *******  -->
<!--  *******  the SMUOS Framework, which controls the basics of   *******  -->
<!--  *******  the simulation                                     *******  -->
<!--  ************************************************************  -->
<ProtoInstance DEF='SscBase' name='SscBase'>
    <fieldValue name='useConnection' value='Conn'/>
    <fieldValue name="mandatorySscBaseExtensions">
        <ProtoInstance DEF="TrainManager" name="SrrControlTm"/>
        <ProtoInstance DEF="KeyManager" name="SscKeyManager"/>
        <ProtoInstance DEF="BeamerManager" name="SscBeamerManager"/>
    </fieldValue>
</ProtoInstance>
```

This example shows, how the frame author registers the SSC Extensions at the SSC Base.

The frame author needs not care about the initialization of the SSC Extensions, this is handled by the SSC Base and by the SSC Core (described later in this document).

Notes:
- The field "useConnection" is needed to provide a network connection to the network sensors of the SRR/SMUOS Framework
- The field "mandatorySscBaseExtensions" is of type "MFNode"
- The used prototypes are external prototypes, the declarations are not shown

3.2 How To Register a Module Coordinator Extension

Best to have a look at the example in the official demo layout. The two modules "City" and "Mountains" use the "Train Manager" extension of the Module Coordinator.

The following code shows, how the module author registers the Module Coordinator Extension at the Module Coordinator:

```
<!--  ****************************************************************  -->
<!--  *******  Module Coordinator                                 *******  -->
<!--  *******  The Module Coordinator coordinates the SRR/SMUOS   *******  -->
<!--  *******  Framework within one module                        *******  -->
<!--  *******  It cares for module activity and for the module     *******  -->
<!--  *******  parameters (modParam), hence for the MIDAS Objects  *******  -->
<!--  ****************************************************************  -->
<!--  *** The SMS Module Coordinator   -->
<ProtoInstance DEF='SmsModCoord' name='McBase'>
    <fieldValue name="mandatoryModCoordExtensions">
        <ProtoInstance DEF='TrainManager' name="SrrModCoordTm"/>
    </fieldValue>
    <IS>
        <connect nodeField="moduleWrapper" protoField="moduleWrapper"/>
        <connect nodeField="initialized" protoField="initialized"/>
        <connect nodeField="disable" protoField="disable"/>
    </IS>
</ProtoInstance>
```

The module author needs not care about the initialization of the Module Coordinator Extensions, this is handled by the Module Coordinator Base (McBase) and by the MC Core (described later in this document).

Notes:
- The field "mandatoryModCoordExtensions" is of type "MFNode"
- The used prototypes are external prototypes, declarations are not shown
- The <IS> clause is used to connect fields "moduleWrapper", "initialized" and "disable" of the module coordinator with the external interface of the module (miModule), the module itself is a prototype, too.

3.3 How To Use Extension MIDAS Objects

Extension MIDAS Objects are based on the same principles as Basic MIDAS Objects are.

However, they need a specific extension of the SSC and/or of the Module Coordinator to function properly.

Thus following statements are true:

1) If an Extension MIDAS Object needs the specific SSC Extension X, then the MIDAS Object can only be used in SMSs, whose frame instantiates SSC Extension X.

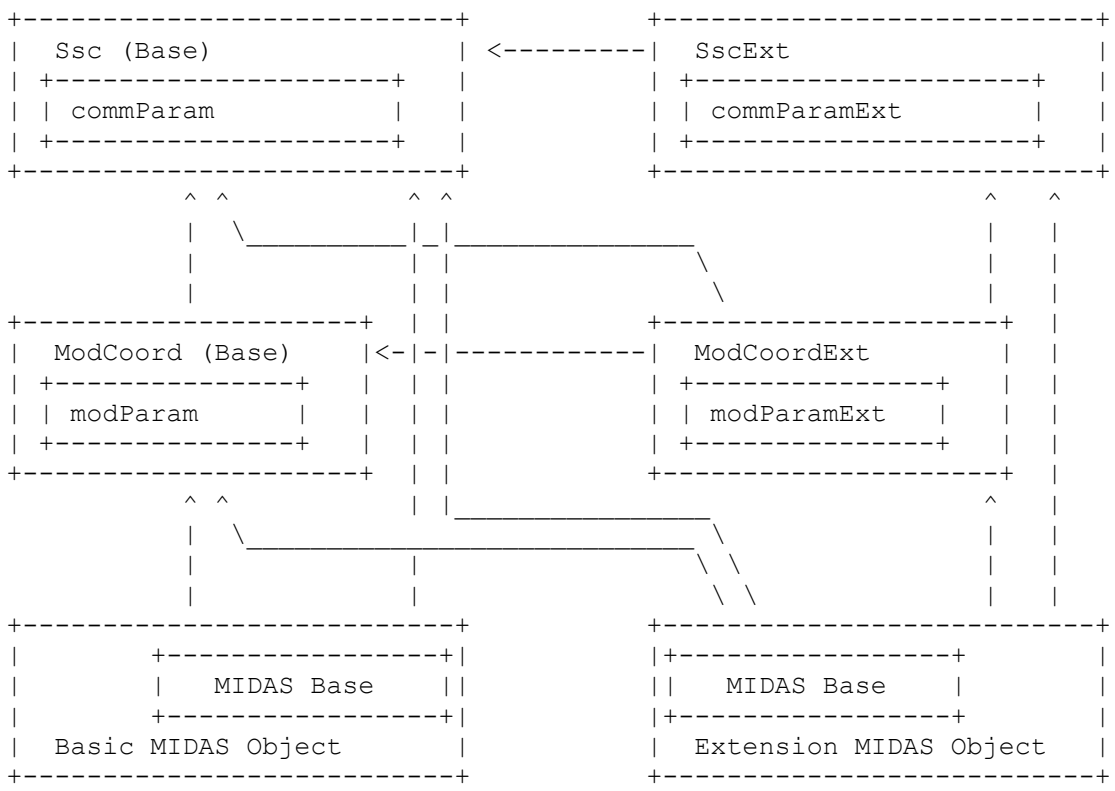
2) If an Extension MIDAS Object needs the specific Module Coordinator Extension Y, then the MIDAS Object can only be used in modules that instantiate Module Coordinator Extension Y.

3) If a Module Coordinator Extension Y needs an SSC Extension Z, then a module that instantiates Module Coordinator Extension Y, can only be used in SMSs, whose frame instantiates SSC Extension Z.

4 Internal View - How To Implement Your Own SMUOS Extension

4.1 Overview

The potential "uses"-relations that result from the extensibility of the Simple Scene Controller (Ssc) and of the Module Coordinator (ModCoord) are depicted in following figure:



Where the element at the shaft uses the element at the arrowhead.

The Simple Scene Controller contains a node, whose fields store the "Common Parameters" commParam (please refer to chapter "The Common Parameters" in 121_SimpleSceneController). One of those fields is the field "extensions" (MFNode) that stores pointers to all "Common Parameter Extensions" (commParamExt). This field is set during initialization of the SSC and it provides for initial access to each SscExt.

A similar field "extensions" (MFNode) exists in the "Module Parameters" modParam (please refer to chapter "The Module Parameters" in 201_ModuleCoordinator), which provides for initial access to each ModCoordExt.

4.1.1 MIDAS Base (MIB)

We said, extending the SRR/SMUOS Framework by programming your own MIDAS Objects is a quite basic kind of extension of the SMUOS Framework.

On the other hand, implementing your own MIDAS Objects is quite simple. This holds true, the more the SMUOS Framework provides some basic functions that are needed in nearly any MIDAS Object.

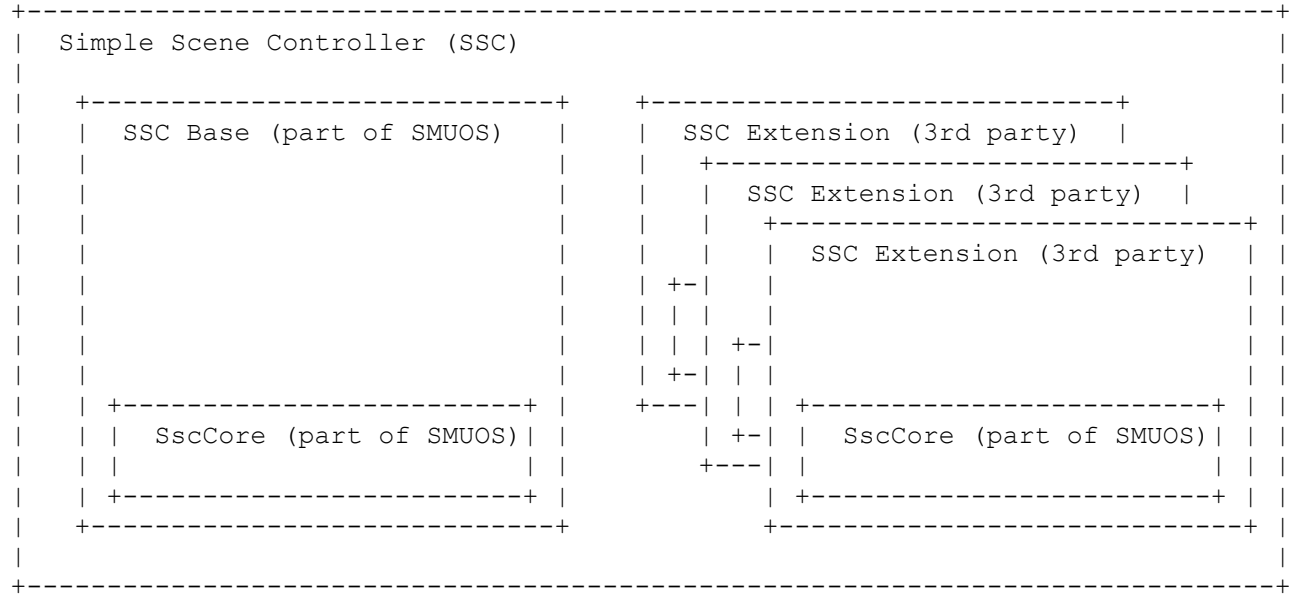
These basic functions are called the "MIDAS Base (MIB)". There is some general information about models and objects - 013_ModelsAndObjects - and there is a more specific introduction to MIDAS Objects - 301_MidasObjects.

4.1.2 SscCore and McCore, the "Core Prototypes"

If you want to implement your own SMUOS Extension, then the SMUOS Framework has got some prototypes that will help you, we call them the "core prototypes".

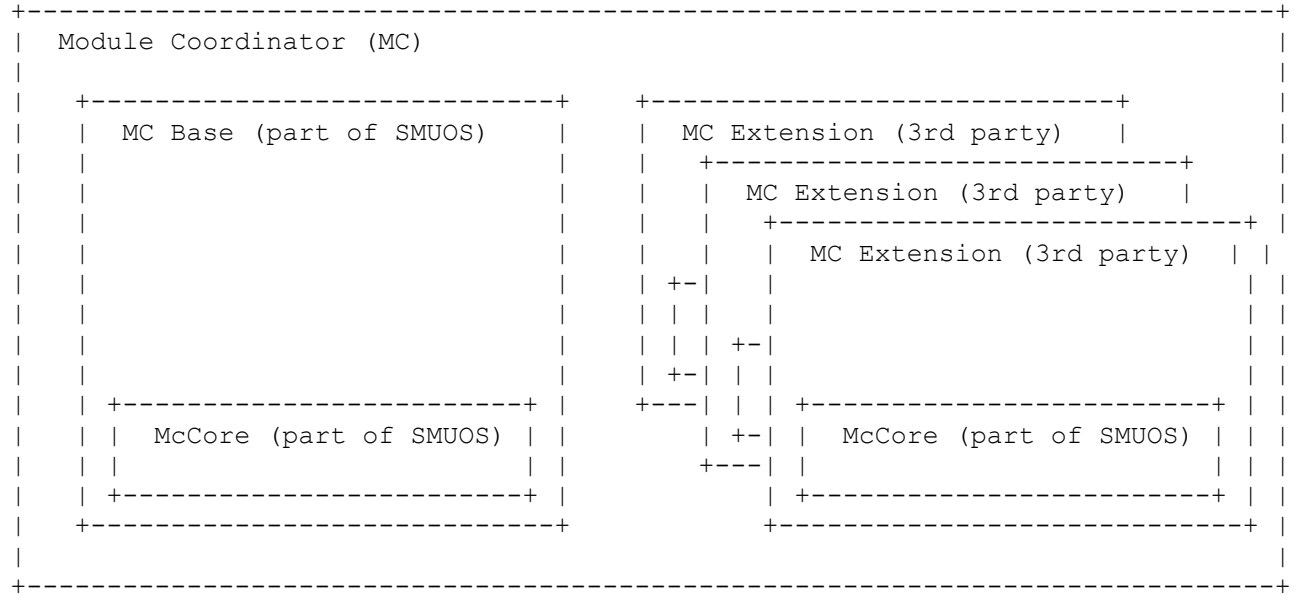
SscCore cares about some basic topics, which are relevant for all parts of the SSC, i.e. for the SSC Base and for any SSC Extension.

- initialization of all SSC Extensions (incl. UOC Related SSC Dispatchers and Network Sensors)
- starting the activity of all SSC Extensions



McCore cares about some basic topics, which are relevant for all parts of the Module Coordinator, i.e. for the MC Base and for any MC Extension.

- initialization of all MC Extensions
- performing an attachment of all MC Extensions



SSC Base and SSC Extensions are used within the frame, MC Base and MC Extensions are used in modules. Please find more information about frame and modules at 012_Modules.

4.2 Services

The figure in chapter 4.1 depicts possible "uses" relations. Now it is interesting to know, which services are available via these relations.

The author of a frame may rely on following services:
- Services of SSC Base: (see 121_SimpleSceneController for details)
  - all basic services that are provided via the uiControl interface
    in particular: the registration of SSC Extensions (chapter 3.1)
- Services of SscExt: (see 121_SimpleSceneController and 131_SrrControllerTm)
  - extension services via uiControl (as defined by the SSC Extension)

The author of a module may rely on following services:
- Services of MC Base: (see 201_ModuleCoordinator for details)
  - all basic services that are provided via the uiMod interface
    in particular the registration of MC extensions (chapter 3.2)
- Services of ModCoordExt: (201_ModuleCoordinator / 221_ModuleCoordinatorTm)
  - extension services via uiMod (as defined by the MC extension)

The programmer of an SscExt may rely on following services:
- Services of SSC Base: (see 121_SimpleSceneController), e.g.
  - services and information available via commParam (e.g. tracer)
- Services of SscCore:
  - being triggered for initialization by SscCore (see chapter 4.3)
  - being triggered for "Start Activity" by SscCore (see chapter 4.3)
  - being informed about "iAmController" by SscCore (see chapter 4.7)

The programmer of a ModCoordExt may rely on following services:
- Services of SSC Base: (see 121_SimpleSceneController), e.g.
  - services and information available via commParam (e.g. tracer)
- Services of MC Base: (see 201_ModuleCoordinator), e.g.
  - services and information available via modParam
- Services of McCore:
  - being triggered for initialization by McCore (see chapter 4.5)
  - being triggered for attachment by McCore (see chapter 4.5)
  - address resolution service to detect SscExt (see chapter 4.5)
- Services of SscExt: (see 121_SimpleSceneController and 131_SrrControllerTm)
  - any extension services as defined by the programmer of the extension

The programmer of an Extension MIDAS Object may rely on following services:
- Services of SSC Base: (see 121_SimpleSceneController)
  - all services and information available via commParam (e.g. tracer)
- Services of MC Base: (see 201_ModuleCoordinator)
  - all services and information available via modParam
- Services of MidasBase:
  (see 311_MidasBase/312_MidasBaseNoState/313_MidasBaseAnim)
  - all (basic) services, e.g.
    - address resolution service to detect SscExt (chapter 4.4)
    - address resolution service to detect ModCoordExt (chapter 4.6)
- Services of SscExt: (see 121_SimpleSceneController / 131_SrrControllerTm)
  - any extension services as defined by the programmer of the extension
- Services of ModCoordExt: (see 221_ModuleCoordinatorTm)
  - any extension services as defined by the programmer of the extension

4.3 Services of SscCore, How To Implement an SSC Extension

4.3.1 The Minimum Interface miControl

Each SSC Extension must provide at least following fields at the external interface miControl (plus any fields for the user interface uiControl):

```
<field accessType='initializeOnly' name='instanceIdClient' type='SFString'
value="MyClientInstanceId"/>
<field accessType='initializeOnly' name='instanceIdServer' type='SFString'
value="MyServerInstanceId"/>
<field accessType='inputOnly' name='commParam' type='SFNode'/>
<field accessType='inputOnly' name='disable' type='SFTime'/>
<field accessType='inputOutput' name='enabledOut' type='SFBool' value="true"/>
<field accessType='inputOnly' name='startActivity' type='SFBool'/>
<field accessType='inputOnly' name='initializeStateAndStartActivity'
type='SFBool'/>
<field accessType="inputOutput" name="traceLevelControl" type="SFInt32"/>
<field accessType="inputOutput" name="traceLevelComm" type="SFInt32"/>
<field accessType='inputOutput' name='basicallyInitialized' type='SFBool'
value='false'/>
<field accessType='outputOnly' name='initialized' type='SFNode'/>
<field accessType='outputOnly' name='activityStarted' type='SFNode'/>
```

Each SSC Extension must contain a <Script> node which is called the "Common Parameters Extension Script". That <Script> node provides for central data services of the SSC Extension and it must contain at least following fields:

```
<!--  ....EXPECTED BY SSC CORE  -->
<field accessType='inputOutput' name='commParam' type='SFNode' value='NULL'/>
<field accessType='inputOutput' name='enabled' type='SFBool' value='true' />
<field accessType='inputOutput' name='initialized' type='SFBool' value='false'/>
<field accessType='inputOutput' name='iAmActive' type='SFBool' value='false' />
<field accessType='inputOutput' name='iAmController' type='SFBool' value='false'
/>
<field accessType='inputOutput' name='sscBaseExt' type='SFNode' value='NULL' />
<field accessType='inputOutput' name='extensions' type='MFNode'></field>
<!--  ....EXPECTED BY SSC BASE  -->
<field accessType='inputOutput' name='wellKnownId' type='SFString'
value='MyWKI'/>
```

Fields in the Responsibility of the Programmer of the SSC Extension

The CommParamExt field

- "sscBaseExt" must be set during basic initialization. Please find info about Basic Initialization at 101_SmsBase
- "wellKnownId" must be set to a unique WKI, which must be coordinated with all possible SSC Extensions

The external fields

- "startActivity", "initializeStateAndStartActivity", "traceLevelControl", "traceLevelComm", "basicallyInitialized", "initialized" and "activityStarted" must be handled by the SSC Extension

Fields in the Responsibility of SscCore

The CommParamExt fields

- "commParam", "enabled", "initialized", "iAmActive" and "iAmController" are set by SscCore (see next chapter)

The external fields

- "instanceIdClient", "instanceIdServer", "commParam", "disable" and "enabledOut" are handled by SscCore, given they are <connect>ed

4.3.2 The External Interface of the SscCore Prototype

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.3.3 Initialization of an SSC Extension

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.3.4 Activation of an SSC Extension

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.4 SSC Extensions Required by an Object

4.4.1 Requiring an SSC Extension by a MIDAS Object (MIDAS Base)

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.4.2 Requiring an SSC Extension by a Bound Model (Model Base)

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.5 Services of McCore, How To Implement an MC Extension

4.5.1 The Minimum Interface miMod

Each MC Extension must provide at least following fields at the external interface miMod (plus any fields for the user interface uiMod):

```
<field accessType='initializeOnly' name='instanceIdMod' type='SFString'
value="MyModuleInstanceId"/>
<field accessType='inputOnly' name='modParam' type='SFNode'/>
<field accessType='inputOnly' name='disable' type='SFTime'/>
<field accessType='inputOutput' name='enabledOut' type='SFBool' value="true"/>
<field accessType='inputOnly' name='startAttachment' type='SFInt32'/>
<field accessType="inputOutput" name="localTraceLevel" type="SFInt32"
value="1"/>
<field accessType='inputOutput' name='basicallyInitialized' type='SFBool'
value='false'/>
<field accessType='outputOnly' name='initialized' type='SFNode'/>
<field accessType='outputOnly' name='attachmentFinished' type='SFNode'/>
```

Each MC Extension must contain a <Script> node which is called the "Module Parameters Extension Script".

That <Script> node provides for central data services of the MC Extension and it must contain at least following fields:

```
<!--  ....EXPECTED BY MC CORE  -->
<field accessType='inputOutput' name='modParam' type='SFNode' value='NULL'/>
<field accessType='inputOutput' name='enabled' type='SFBool' value='true' />
<field accessType='inputOutput' name='initialized' type='SFBool'
value='false' />
<field accessType='inputOutput' name='moduleIx' type='SFInt32' value='-1' />
<field accessType='inputOutput' name='smsModCoordExt' type='SFNode'
value='NULL'/>
<field accessType='inputOutput' name='extensions' type='MFNode'></field>
<!--  ....EXPECTED BY MC BASE  -->
<field accessType='inputOutput' name='wellKnownId' type='SFString'
value='MyWKI'/>
```

Fields in the Responsibility of the Programmer of the MC Extension

The ModParamExt field

- "smsModCoordExt" must be set during basic initialization. Please find info about Basic Initialization at 101_SmsBase
- "wellKnownId" must be set to a unique WKI, which must be coordinated with all possible MC Extensions

The external fields

- "startAttachment", "localTraceLevel", "basicallyInitialized", "initialized" and "attachmentFinished" must be handled by the MC Extension

Fields in the Responsibility of McCore

The ModParamExt fields

- "modParam", "enabled", "initialized" and "moduleIx" are set by McCore (see next chapter)

The external fields

- "instanceIdMod", "modParam", "disable" and "enabledOut" are handled by McCore, given they are <connect>ed

4.5.2 The External Interface of the McCore Prototype

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.5.3 Initialization of an MC Extension

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.5.4 Attachment of an MC Extension

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.5.5 Requiring an SSC Extension by an MC Extension

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.6 MC Extensions Required by an Object

4.6.1 Requiring an MC Extension by a MIDAS Object (MIDAS Base)

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.6.2 Requiring an MC Extension by a Bound Model (Model Base)

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.7 The Controller Role of an SSC Extension

This chapter will be written later.

Currently, please analyse the examples in the directories sms/ and tmm/.

4.8 Programming Support

You can use the "Key Manager" extension and the "Beamer Manager" extension of the SSC, as well as the "Train Manager" extension of the SSC and of the Module Coordinator as templates to develop your own extensions.

4.9 Hints for Using the SMS Tracer

Following topics should be considered, when using the SMS Tracer for your SMUOS extension.

- extensibility of trace levels
- extensibility of subsystems and tracer instances

4.9.1 Extensibility of Trace Levels

SSC extensions and Module Coordinator extensions can just inherit the current values of the trace levels from the SSC Base and from the MC Base (which is done for the "KeyManager" and "BeamerManager" extensions), or they can provide their own - maybe sophisticated – ways to set various trace levels (which is planned for the "TrainManager" extension).

4.9.2 Extensibility of Tracer Subsystems and Instances

The SSC Base, the MC Base, the basic MIDAS Objects and all extensions that are delivered together with the SRR/SMUOS Framework, use "Subsystem"/"Instance" values as follow:

For a good automatic evaluability of traces, it is recommended to follow these examples with your own extensions.

```
SIMPLE SCENE CONTROLLER
   (1) "SimpleSceneController"/"CommControl"......SSC Base (server)
   (3) "SimpleSceneController"/"KeyControl".......SSC Key Manager (server)
   (5) "SimpleSceneController"/"TrainControl".....SSC Train Manager (server)

   (1) "SimpleSceneController"/"SscBase"..........SSC Base (client)
   (1) "SimpleSceneController"/
                  "<moduleName>.Client".........module related SSC dispatcher
   (1) "SimpleSceneController"/
                  "<moduleName>.Server"......... ------------ " ------------
   (1) "SimpleSceneController"/
                  "<uocName>.Client"............UOC related SSC dispatcher
   (1) "SimpleSceneController"/
                  "<uocName>.Server"............  ---------- " ----------
   (1) "SimpleSceneController"/
                  "<extObjId>.Server"...........SSC Dispatcher Stub
   (3) "SimpleSceneController"/"SscKeyManager"....SSC Key Manager (client)
   (4) "SimpleSceneController"/"SscBeamerManager".SSC Beamer Manager (client)
   (5) "SimpleSceneController"/"SscTrainManager"..SSC Train Manager (client)
MODULE COORDINATOR
   (1) "ModuleCoordinator"/
          "<moduleName>-McBase.Coord"............Module Coordinator Base
   (5) "ModuleCoordinator"/
          "<moduleName>-McTrainManager.Coord"....Module Coordinator Train Man.
MIDAS BASE
   (1) "MidasBase" + "<extObjId>.Control".........MIDAS Base (client)
   (1) "MidasBase" + "<extObjId>.ObCo............MIDAS Base (server)
MIDAS OBJECTS
   (2) "BasicMobs.<objectType>" + "<extObjId>.Control".......(client)
   (2) "BasicMobs.<objectType>" + "<extObjId>.ObCo..........(server)
   (3) "KeyMobs.<objectType>" + "<extObjId>.Control.........(client)
   (3) "KeyMobs.<objectType>" + "<extObjId>.ObCo............(server)
   (4) "BeamerMobs.<objectType>" + "<extObjId>.Control......(client)
   (4) "BeamerMobs.<objectType>" + "<extObjId>.ObCo.........(server)
   (5) "TrainMobs.<objectType>" + "<extObjId>.Control.......(client)
   (5) "TrainMobs.<objectType>" + "<extObjId>.ObCo..........(server)

(1) = part of SMUOS Framework; (2) = Example Basic MIDAS Objects;
(3) = "Key Manager" extension; (4) = "Beamer Manager" extension;
(5) = "Train Manager" extension
```

5 Additional Info

None