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

This file valid for step 0033.11
Issue Date: tbd.

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 as a general guideline for the SMS Concept.

2 Purpose

The SRR/SMUOS Framework consists – roughly – from three kinds of X3D nodes:
(1) Nodes for the Simple Scene Controller (former SRR Controller),
(2) Nodes for the Module Coordinator and
(3) Nodes for 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 (please refer to the introduction of the MIDAS Base (MIB) in the paper 013_ModelsAndObjects and its closer description at 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

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="mwParam" protoField="mwParam"/>
        <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 the fields "mwParam", "initialized" and "disable" of the module coordinator with the external interface of the module (uiModule), 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 in addition to usual principles of MOBs:

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.

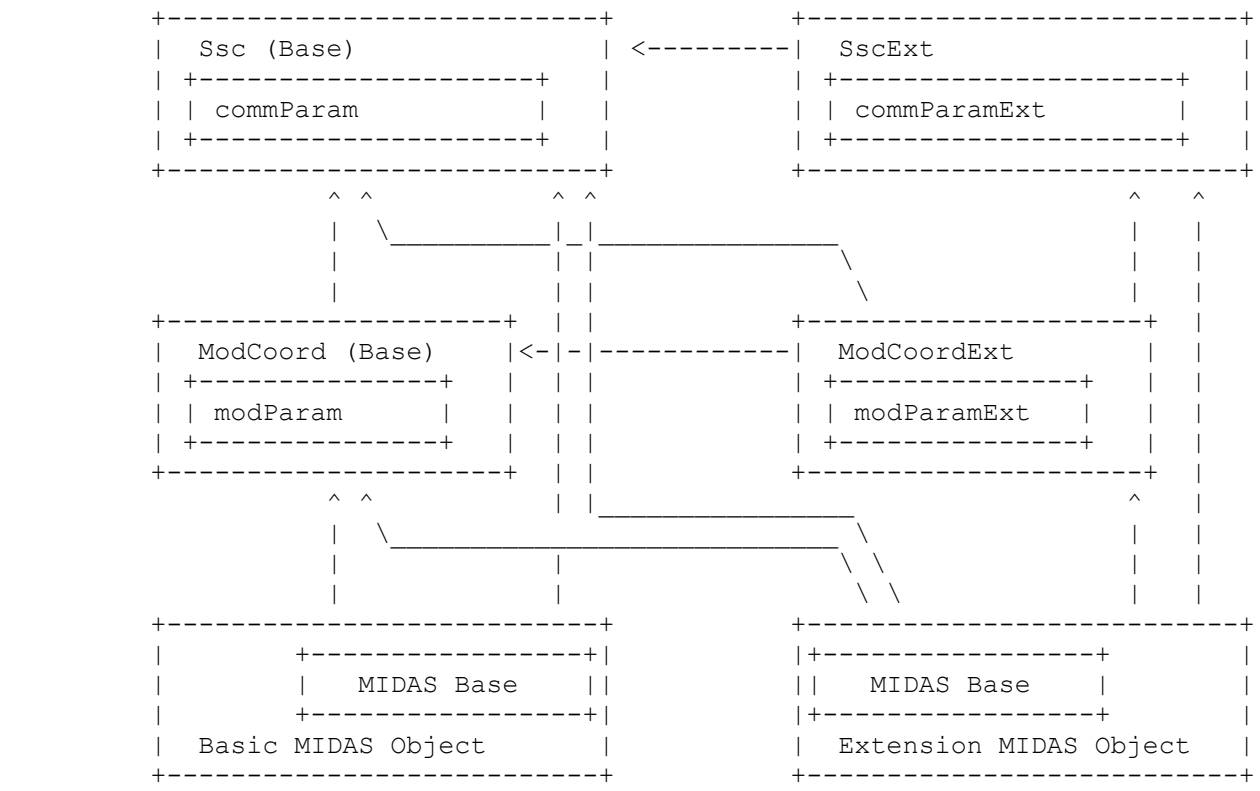
2) If an Extension MIDAS Object needs the specific Module Coordinator Extension Y, then the MIDAS Object can only be used in modules that instantiate Module Coordinator Extension Y.

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

4 Internal View - How To Extend the SMUOS Framework

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.

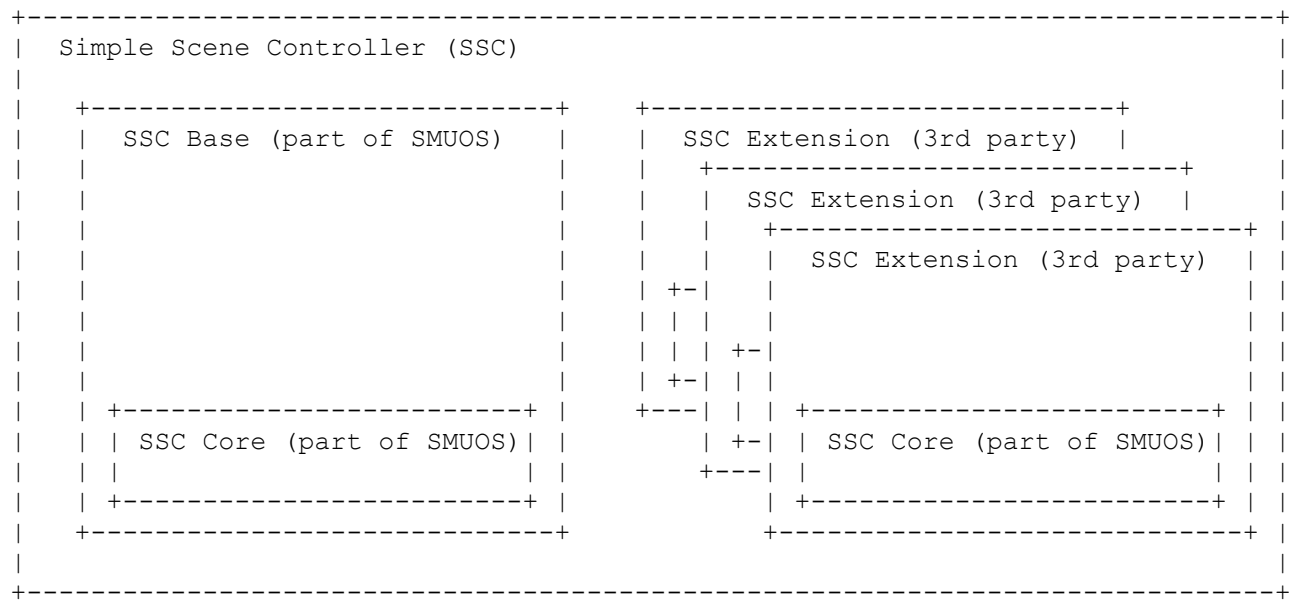
These basic functions are called the "MIDAS Base (MIB)". There is some general information about models and objects - 013_ModelsAndObjects - and there is a more specific introduction to MIDAS Objects - 301_MidasObjects.

4.1.2 SSC Core and MC Core, 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".

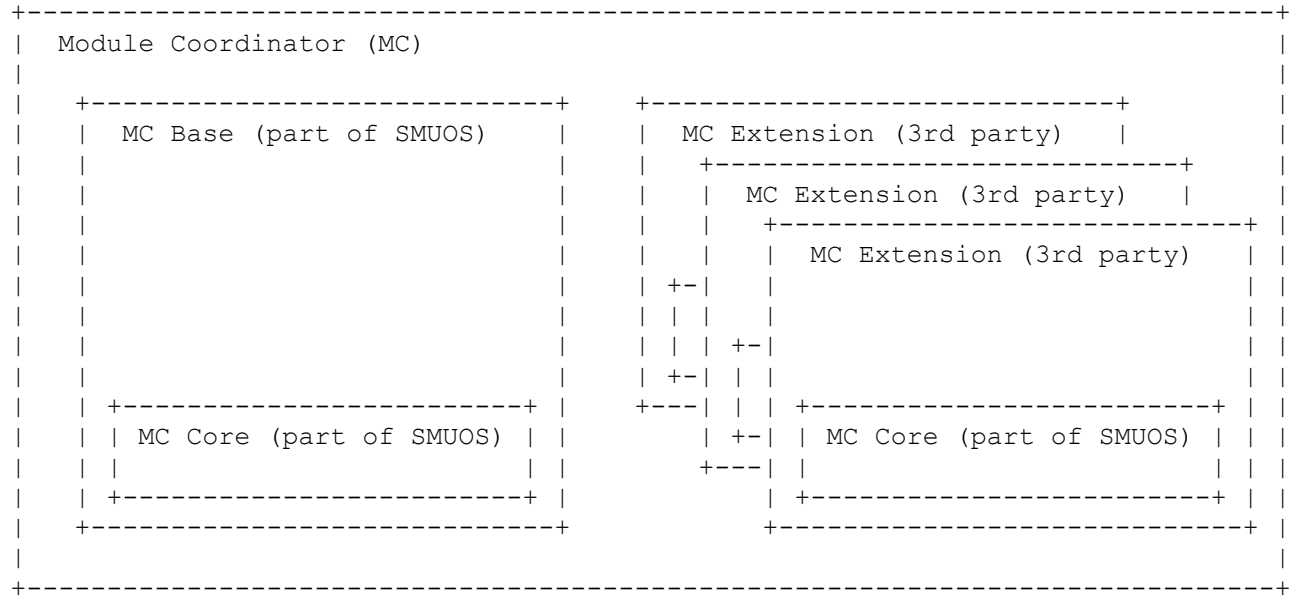
SSC Core cares about some basic topics, which are relevant for all parts of the SSC, i.e. for the SSC Base and for any SSC Extension.

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



MC Core 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 SSC Core and MC Core at 121_SimpleSceneController and 201_ModuleCoordinator, respectively.

4.2 Services

Chapter 4.1 explains possible "uses" relations. Now it is interesting to know, which services are available via these relations.

Services of SSC Core (please refer to 121_SimpleSceneController for details):

- Services for the programmer of SSC Extensions: the eiSscCore interface
  - the UOC Related SSC Dispatcher
  - the UBO Loader
  - the SscCore prototype

Services of SSC Base (please refer to 121_SimpleSceneController for details):

- Services for the programmer of SSC Extensions: the miControl interface
- Services for the frame author: the uiControl interface
- Services for the MC and for MIDAS Objects:
  - the commParam interface
  - the eiControl interface

Services of SSC Extensions (please refer to 122_BasicSscExtensions and 131_SrrControllerTm):

- Services for the frame author: the uiControl(Extension) interfaces
- Services for MC Extensions and for Extension MIDAS Objects:
  - the commParamExt(Extension) interfaces
  - the eiControl(Extension) interfaces

Services of MC Core (please refer to 201_ModuleCoordinator for details):

- Services for the programmer of MC Extensions: the eiMcCore interface
  - the McCore prototype

Services of MC Base (please refer to 201_ModuleCoordinator for details):

- Services for the programmer of MC Extensions: the miMod interface
- Services for the module author: the uiMod interface
- Services for MIDAS Objects:
  - the modParam interface
  - the eiMod interface

Services of MC Extensions (please refer to 221_ModuleCoordinatorTm for details):

- Services for the module author: the uiMod(Extension) interfaces
- Services for Extension MIDAS Objects:
  - the modParamExt(Extension) interfaces
  - the eiMod(Extension) interfaces

Services of MIDAS Base (please refer to 311_MidasBase / 312_MidasBaseNoState / 313_MidasBaseAnim):

- Services for MIDAS Objects and for Extension MIDAS Objects

Services of MIDAS Objects (please see 351_AvatarContainer, 352_BinarySwitch, 353_KeyContainer, 354_CarriedKeysLock, 355_CarouselDrive, 356_Beamer, 357_BeamerDestination, 358_ContainedKeyLock, 359_Trigger, 360_Creator, 401_TracksAndTurnouts and 402_Trains for details):

- Services for Model, Module and Frame authors: the uiObj Interface

4.3 Programming Support

If you want to implement your own MIDAS Objects (see chapter 4.1.1 and 4.2), then you can use the MIDAS Objects delivered with the SRR/SMUOS Framework as templates for your own Objects.

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 SMUOS extensions (see ch. 4.1.2 and 4.2).

5 Additional Info

None