

Simulated Railroad Framework, <http://simulrr.sourceforge.net>  
Synopsis: [100\\_SrrFramework](#)

This file valid for step 0033.10.5  
Issue Date: 2019-06-09

The Module Coordinator  
=====

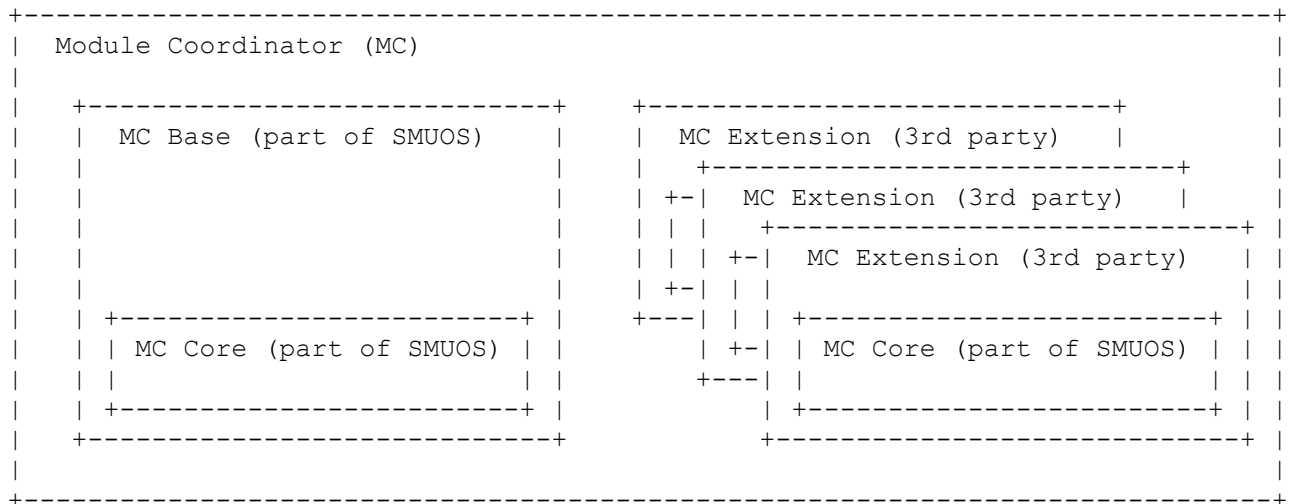
## 1 Synopsis

The Module Coordinator is the part of the SRR/SMUOS Framework that is intended to coordinate the SRR/SMUOS Framework within one module.

The present paper describes the MC Base, which is actually a part of the SMUOS Framework and it gives some basic information about how to extend the MC Base by your own MC Extensions.

The MC Extension "Train Manager" is actually a part of the SRR Framework and it is described at [221\\_ModuleCoordinatorTm](#).

The overall architecture of the MC is shown in following figure:



The "Base" of the Module Coordinator consists of an X3D prototype, which is contained in the file `SmuosMcBase.x3d` - see chapter 3 for how to use the MC Base and chapter 4.1 on how the MC Base is built internally.

Additionally, all parts of the MC may use the "MC Core" subsystem, which is contained in the file `SmuosMcCore.x3d`.

The application of MC Core is described in chapter 4.2.

## 2 The Purpose of the Module Coordinator

The module coordinator supports the module author with following aspects of an `SrrTrains` layout / Simple Multiuser Scene.

- module attach/detach
- module activation/deactivation
- initialization of MIDAS Objects
- disabling of MIDAS Objects

### 3 External View (uiMod, modParam and eiMod)

When a module has been loaded and hence the contained <ProtoInstance> "McBase.x3d" has been loaded, then the Module Coordinator (Base) is in mode of operation "LOADED" (MOO "LOADED").

After some preparation (see below), the frame sends the commParam event to the module and hence to the Module Coordinator and triggers its initialization.

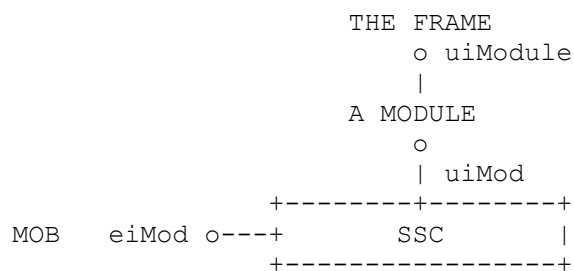
After successful initialization the Module Coordinator outputs the SFNode event "initialized", that points to some internal node.

An event initialized=null indicates unsuccessful initialization.

The "initialized" event should be forwarded to the frame (via the interface uiModule) - at least in case of dynamic modules - to indicate success/failure.

After successful or unsuccessful initialization, the module coordinator outputs the SFNode event "modParam", that points to the module parameters - see chapter 5.1 for a detailed description of the module parameters.

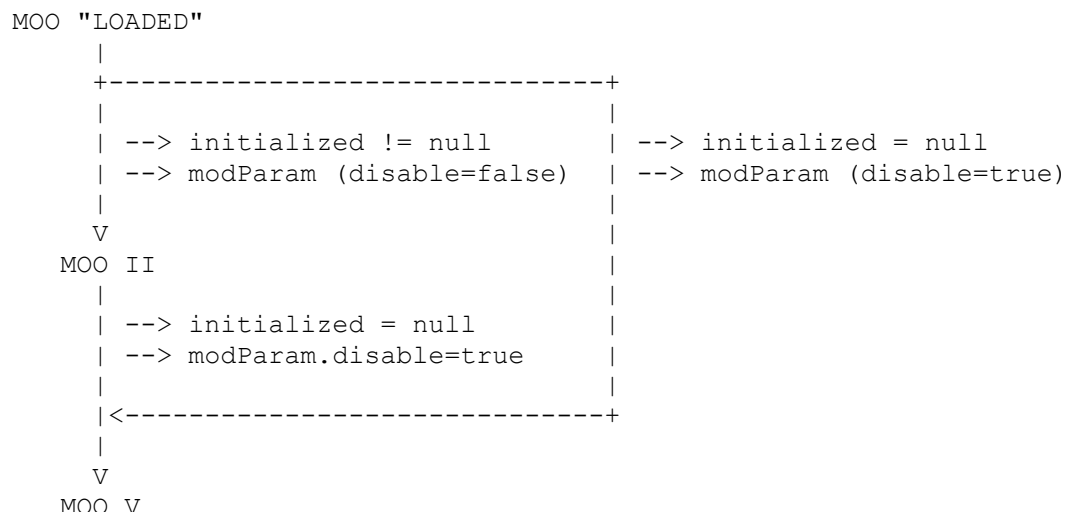
The "modParam" event can be immediately <ROUTE>d to the external interfaces of all bound models and MIDAS objects (via the interfaces uiObj) to trigger their initialization and attachment.



Above figure sketches the "user interface" uiMod and the "external interface" eiMod. They are used by the module and by other parts of the SRR/SMUOS Framework, respectively.

Both interfaces are extended by MC Extensions, as described at chapters 3 of [051\\_Extensibility](#) and [221\\_ModuleCoordinatorTm](#).

### 3.1 MOO Diagram (MOOD)



### 3.2 Use Case Matrix (uiMod Interface)

Following use cases are available via uiMod in following MOOs:

MOO "LOADED"	MOO II	MOO V	
Preparation	-	-	
Initialization	-	-	
ModuleActivity	ModuleActivity	-	
	DisableModule	-	

Table 1: State Event Matrix of the Module Coordinator

The State Event Matrix displays, which use case is available in which mode of operation (MOO), either MOO "LOADED" or MOO II (attached) or MOO V (disabled).

The following sections provide a detailed description of all use cases.

#### 3.2.1 Preparation

Before triggering the initialization of the module coordinator, the module instantiates all module coordinator extensions and provides their addresses in the MFNode field "mandatoryModCoordExtensions" and "optionalModCoordExtensions".

Additionally, the module forwards the module name and the module wrapper parameters from the uiModule interface to the module coordinator (SFString "moduleName" and SFNode "mwParam", respectively).

### 3.2.2 Initialization

-----

The initialization of a module coordinator is triggered by the commParam event, which is forwarded by the frame and the module from the Simple Scene Controller to the external interface uiMod of the module coordinator.

Following things will happen during the initialization of a module coordinator:

- 1) the module coordinator extensions will be initialized
  - 2) the module coordinator will announce itself at the Simple Scene Controller
  - 3) the SSC will send a CSCR "attachModule" to the central controller
  - 4) if not yet registered, the module name and moduleIx will be registered in the commState implicitly
  - 5) the module will be attached in the commState
  - 6) as soon as the commState has been distributed, the SSC Base will physically attach the announced module coordinator
  - 6a) the module coordinator extensions will receive the moduleIx to be able to attach themselves at their SSC extensions
  - 7) the attached module will output a reference to the "module parameters", modParam, which will be forwarded to the MIDAS Objects of the module\*)
  - 8) additionally the module coordinator will indicate successful initialization with an event initialized!=null
- \*) The parameter "modParam.disable" will be "false" in case of successful initialization, which will lead to an initialization of the MIDAS Objects. In case of unsuccessful initialization, "modParam.disable" will be "true", leading to disabled MIDAS Objects (see also the use case "DisableModule").

### 3.2.3 ModuleActivity

-----

A module can request a change of its own activity with one of the two events activateRequest and deactivateRequest at the external interface uiMod of the module coordinator.

Additionally, the SSC Base has got some fields at his user interface, so that the frame can influence module activity, too (please refer to chapter "Use Case ModuleActivity" in [121\\_SimpleSceneController](#)).

The intention of module activity is to save CPU resources, when a module is not within the focus of a user (e.g. if he doesn't look at the module).

Module activity is a matrix, i.e. it can be different in each scene instance (depending on the behaviours of the different users).

It's up to the module author, how he triggers module activation/deactivation (probably with some kind of environment sensor, e.g. a proximity sensor).

The module coordinator will inform all MIDAS Objects in his responsibility about the activity of the module and it will give feed back to the module about the actual activity state.

It's a rule that MIDAS Objects should avoid extensive calculations (e.g. calculated animations), when their parent module is inactive.

One scene instance of all scene instances that have a distinct module activated, gets the so-called MOC role (module controller role). This scene instance is dedicated to perform calculations, that must be done centrally for a module (e.g. maintaining the global states of MIDAS Objects).

The Object Controller roles of the MIDAS Objects ("OBCO" Roles) follow the MOC roles of their parent modules (see [301\\_MidasObjects](#)).

The frame can request the MOC role for modules in his scene instance (otherwise the central controller will assign MOC roles, so that one and only one MOC exists for each module, which is active in at least one scene instance).

### 3.2.4 DisableModule

While module deactivation is a means to save CPU resources, dynamic modules can be unloaded completely to save CPU resources *and* memory.

Loading and unloading of dynamic modules is left to the frame, but before a module is unloaded, it should be detached from the Simple Scene Controller and as a consequence all MIDAS Objects of the module will be disabled.

Detachment of a module can be triggered by one of two events:

- 1) a frame decides to locally unload a dynamic module and as a preparation sets `disable="now"` at the external interface `uiModule` of the module.
- 2) a module is deregistered globally and the Simple Scene Controller detaches the module locally, after having received the `commState`.

ad 1) the `disable="now"` event will be forwarded by the module to the external interface `uiMod` of the module coordinator and trigger a detachment at the SSC

ad 2) Please refer to chapter "Use Case DynModRegistration" in [121\\_SimpleSceneController](#) and to the field `"deregisterModules"`, which is described there.

ad 1) + 2)

The result of detachment will be a `moduleIx=-1` sent from the SSC to the module coordinator and hence triggering the disabling of the MIDAS Objects of the module.

Active nodes other than MIDAS Objects must be disabled by the module author (this can be achieved by <ROUTE>ing the `"enabledOut"` field of the module coordinator to the `"enabled"` fields of those nodes).

### 3.3 Interworking with MIDAS Objects (modParam/eiMod)

The MC can use the "module parameters" (see chapter 5.1) to broadcast information to all parts of the module. This is actually used to inform MIDAS objects about the activity of the module.

When a part of the module knows the `"modParam"`, then it can access the `eiMod` interface of the MC Base.

The field `"getModCoordExtension"` can be used to assert the presence of an MC Extension in the `"modParam.extensions"` field.

## 4 Internal View

### 4.1 Internal Information about the Module Coordinator

The Simple Scene Controller supports the module coordinator.

To support the module coordinator, the SSC Base

- provides some fields at the eiControl interface (refer to section 4.1.1)
- expects some fields at the eiControl interface (refer to section 4.1.2)

#### 4.1.1 Fields of the SSC Base at eiControl

"announceModule" (SFNode)

see 3.2.2. "Initialization": during initialization the module coordinator sends a "this"-pointer to the SSC Base. The SSC Base stores the pointer in a temporary list of modCoords and attaches the module at the commState (central controller).

After the central controller has answered, the SSC Base stores the pointer in the list of modCoords and sends an event "registered"=<moduleIx> to the module coordinator.

In the unsuccessful case the SSC Base answers with "registered"="-1".

"deannounceModule" (SFNode)

see 3.2.4 "DisableModule": when a frame decides to unload a module, it sets disable="now" at the uiModule interface. As a reaction, the module coordinator sends a "deannounceModule"="this" event to the SSC Base and hence triggers the SSC Base to detach the module coordinator (delete it from the list of modCoords and send "registered"="-1" to the detached modCoord).

Additionally, the SSC Base will send a detach indication to the central controller, that will detach the module in the commState.

"activateModRequest" (SFInt32)

see 3.2.3 "ModuleActivity": an attached module coordinator can send the own moduleIx to the SSC Base to trigger it's activation.

"deactivateModRequest" (SFInt32)

see 3.2.3 "ModuleActivity": an attached module coordinator can send the own moduleIx to the SSC Base to trigger it's DEactivation.

#### 4.1.2 Fields of the Module Coordinator at eiControl

"registered" (SFInt32)

With this field, the SSC Base reports the moduleIx to the module coordinator, in one of the following situations

- <moduleIx> during the initialization of the module coordinator, after the module has been attached.  
Reporting the moduleIx leads to initialization of the MIDAS Objects.
- "-1" after the module has been detached (on own request or on foreign request), or after unsuccessful attachment.  
Reporting "-1" leads to disabling the MIDAS Objects

"setActivated" (SFInt32)

see 3.2.3 "ModuleActivity": "setActivated" is a bit field, where the bits have following meaning:

- 1 (bit 0): scene instance took the MOC role for this module
- 2 (bit 1): scene instance lost the MOC role for this module
- 4 (bit 2): scene instance activated this module
- 16 (bit 4): scene instance deactivated this module

This information will be forwarded to all MIDAS Objects of the module (via modParam) and to the module (via uiMod).

"sessionIds" (MFInt32)

see 3.2.3 "ModuleActivity": the SSC Base updates the list of all sessionIds, in whose scene instances the module is active. Some MIDAS Objects need this information, which will be forwarded to all MIDAS Objects via modParam.

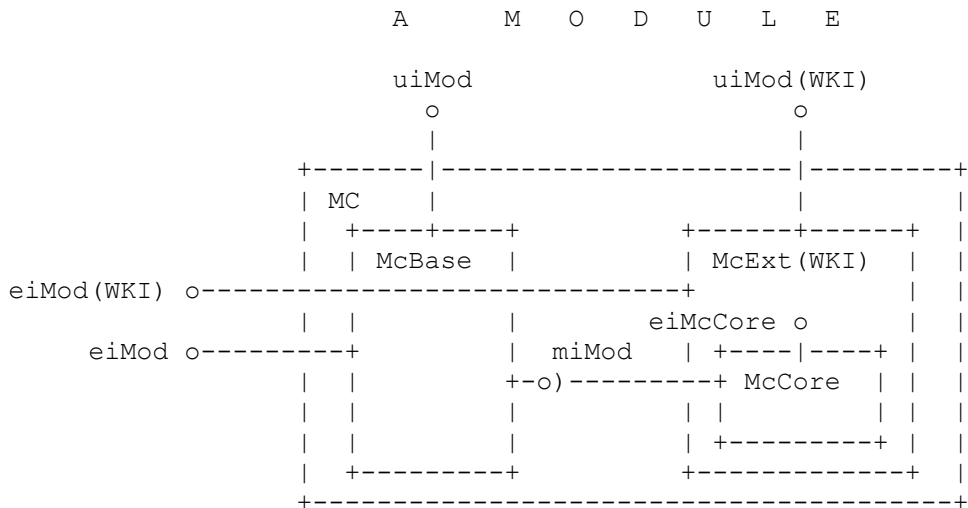
"setTraceLevel" (MFString)

The module coordinator maintains a dynamic route from commParam.traceLevelModules to this input field. Everytime, when the SSC Base updates the MFString, it will be scanned for entries, that can apply to the current module name and the actual trace levels of the module will be updated.

-----

100

- the eiMod interface can be extended by MC Extensions (eiMod(WKI))
- the MC Base itself and the MC Core provide services to the MC Extension which can be described by describing the miMod interface and by describing the eiMcCore interface, respectively

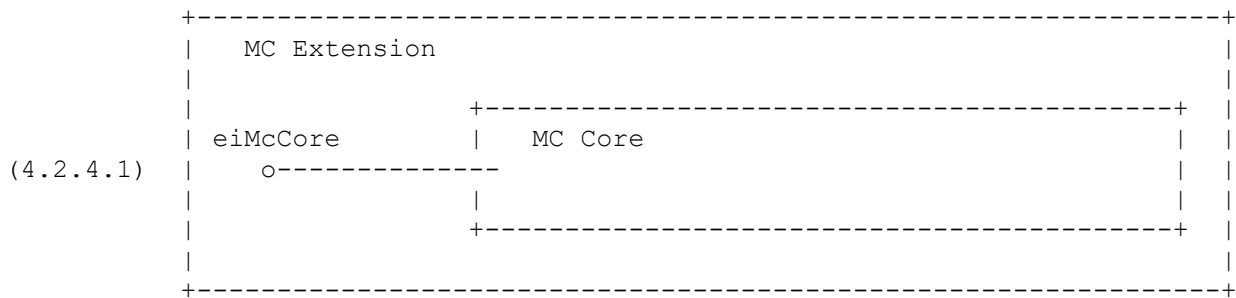


iii. Mod (WKT) of the "Train Manager" is described at 231 Module Coordinator Em

---

of Mod (WKT) of the "Train Manager" is described at 221 ModuleCoordinator™

#### 4.2.4 The external Interface of McCore (eiMcCore)



##### 4.2.4.1 The Fields of the McCore Prototype

###### 4.2.4.1.1 Fields for the external interface of an MC Extension

- get information from the parent part of the MC

```
<field accessType='inputOutput' name='parentInstance' type='SFString' value=''/>
<field accessType='inputOutput' name='refinedClassPaths' type='MFString'
value='\"McBase\"'/>
<field accessType='inputOutput' name='mandatoryMcExtensionsIn'
type='MFNode'></field>
<field accessType='inputOutput' name='optionalMcExtensionsIn'
type='MFNode'></field>
<field accessType='inputOnly' name='modParamIn' type='SFNode'/'>
<field accessType='inputOnly' name='disableIn' type='SFTime'/'>
<field accessType='inputOnly' name='startAttachment' type='SFInt32'/'>
- report the state of the MC Extension to the parent part of the MC
<field accessType='inputOutput' name='disabled' type='SFBool' value='false'/'>
<field accessType='inputOutput' name='enabledOut' type='SFBool' value='true'/'>
<field accessType='inputOutput' name='initialized' type='SFNode' value='\"NULL\"'/>
<field accessType='inputOutput' name='attachmentFinished' type='SFNode'
value='\"NULL\"'/>
```

###### 4.2.4.1.2 Fields provided internally by/to the MC Extension

```
<field accessType='inputOutput' name='traceLevelControl' type='MFInt32'
value='1,1'/'>
<field accessType='inputOutput' name='modParamExtIn' type='SFNode'
value='\"NULL\"'/>
<field accessType='inputOutput' name='requiredSscExtensionsIn' type='MFString'
value='\"'/>
<field accessType='inputOnly' name='setModuleName' type='SFNode'/'>
- output lifetime parameters towards the MC Extension
<field accessType='inputOutput' name='mcInstance' type='SFString'
value='[McInstance]'/>
<field accessType='inputOutput' name='tracerInstanceIdCoord' type='SFString'
value='[moduleName]-[instanceIdMod].Coord'/'>
<field accessType='inputOutput' name='assertedOptionalMcExtensions'
type='MFNode'></field>
<field accessType='inputOutput' name='assertedSscExtensions'
type='MFNode'></field>
- output dynamic parameters towards the MC Extension
<field accessType='outputOnly' name='heartbeat' type='SFFloat'/'>
<field accessType='inputOutput' name='modParamExt' type='SFNode' value='\"NULL\"'/>
<field accessType='inputOutput' name='oldModParamExt' type='SFNode'
value='\"NULL\"'/>
<field accessType='inputOutput' name='moduleIx' type='SFInt32' value='\"-1\"/'>
<field accessType='inputOutput' name='oldModuleIx' type='SFInt32' value='\"-1\"/'>
<field accessType='inputOutput' name='currentProcedure' type='SFString'
value='\"nil\"'/>
- coordinate the execution of procedures by the MC Extension -->
<field accessType='outputOnly' name='startProcedure' type='SFBool'/'>
<field accessType='inputOnly' name='procedureFinished' type='SFBool'/'>
```



-----

-----

-----

[illegible]

---

The modParam of each module contain an SFNode that points to the "Common Parameters" of the Simple Scene Controller (refer to [121 SimpleSceneController](#))

```
commParam(SFNode) .....points to the common parameters
```

---

The following parameters give general information about the module, which contains this module coordinator.

```
version(SFFloat).....version of the module coordinator base module
moduleName(SFString).....the valid module name, that has been set by
                           the frame
mwParam(SFNode).....the module wrapper parameters of the module
gravity(SFVec3f).....the current gravity within the module
```

#### 5.1.4 Data Distribution to all MIDAS Objects

---

The following parameters are used to distribute information from the module coordinator to all MIDAS Objects of the module.

disable(SFBool).....an SFBool event "true" at this field disables  
all MIDAS Objects of the module  
sessionIds(MFInt32).....this field distributes a list of all scene  
instances, in which this module is active  
(used by "animated" MIDAS Objects)  
sessionIdLeft(SFInt32).....this field reports every scene instance that  
has left the game (used by MIDAS Objects to  
handle OBCO roles correctly)  
takeMOC(SFBool).....a "true" event at this field reports having  
got the MOC role  
grantMOC(SFBool).....a "true" event at this field reports having  
lost the MOC role  
activate(SFBool).....a "true" event at this field reports having  
been activated  
deactivate(SFBool).....a "true" event at this field reports having  
been deactivated

#### 5.1.4 Address of the Module Coordinator

---

smsModCoord(SFNode).....Address of the Module Coordinator <Script>.  
If a module coordinator extension module or  
a MIDAS Object wants to send an event to the  
Module Coordinator, then it can address the  
input fields of the Module Coordinator via  
modParam.smsModCoord.<fieldName>