New Instance Naming

This hobby report is a "snapshot", so it will not be updated anymore!

It shows the "New Naming Rules" after their implementation at the beginning of Step 0033.11.

Version 4.3 is indicating the version at the first serious specification of the release "Arimathea" (step 0033.11), where the basic concepts of "Arimathea" seem to be stable now. No update done.

Note: this hobby report is the only place, where the new naming rules are explained that detailed!!

1 Problem to Solve

When I first started thinking about implementing UBOs (Unbound Objects), I "stumbled" on some issues I'd like to solve now.

1. When implementing the SSC Core in Step 0033.10, I had already foreseen that you could nest SSC extensions, but I had not thought about making the nesting conditional on the existence of UOCs.

   Now it would be useful to develop a "class hierarchy" for UBOs where UOCs have UOCs other than "base class" (details in chapter 1.1).

2. There are several "tracks" of naming, namely the so-called "tracer instances", the so-called "Well Known Ids" and the "Stream Names".

   It would be useful to derive the tracer instances from the Well Known IDs and match the stream names with the tracer instances so that you only have one more "track".

1.1 Class Hierarchy for Unbound Objects (UBOs) - Ideas

Furthermore, the Simple Scene Controller Base (SSC Base) is to provide the Universal Object Class (UOC) "Base".

If one implements an SSC extension, then one should now have to decide which UOC will be "refined" by this SSC extension, the Parent SSC Extension could indeed define several UOCs.

A class hierarchy could look like this:

SSC Base	**Class**	**SSC Extension**	**Class**	**SSC Extension**	**Class**
Ssc	Base	SscTrainManager	Trains	FlyingTrainMgr	FlyingTrains

Thus, "**Ssc.Base.SscTrainManager**" would be an identifier of the Train Manager Extension.

"**Ssc**" would identify SSC Base and "**Ssc.Base.SscTrainManager.Trains.FlyingTrainMgr**" would identify the Flying Train Manager.

The Flying Train Manager would also use Train Manager features, so the **Base.Trains.FlyingTrains** class would be derived from the **Base.Trains** class.

The class "**Base**" will not be used for UBOs but only for astral objects (ASOs).

The "classes" (UOCs) are only used for UBOs and ASOs, but not for Bound Objects (BDOs). BDOs are not defined by UOCs, but by modules (the modules to which they are bound).

2 Background - MMF paradigm

I have already explained the MMF paradigm so often, but I would like to do it again for the inexperienced reader here, so this "Hibernation Report" can stand for itself.

So, according to the "modular design" of model railways, I want to disassemble the landscape of the virtual model railroad in so-called "modules" and on the modules are "romping around" models. Modules should be "reloaded" as needed or "unloaded" again.

Furthermore, we need an "abstract something", a "certain infrastructure" that can be used by all modules and models to guarantee the user a virtual experience. I refer to this "abstract something" as the "frame".

In the first place, it is the case that each model is assigned to a module - because a module spans a local coordinate system to which a model can be relatively represented (rendered). This is shown in Figure 1 below:

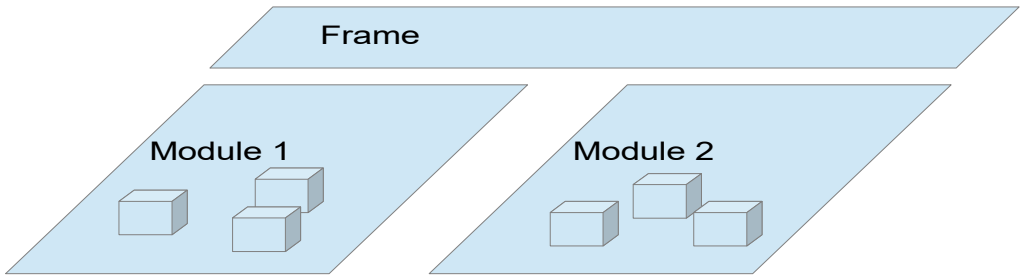

Figure 1: Objects bound to a module (bound objects - BDOs)

Pretty much from the beginning there was also an object that was not assigned to a module, which could therefore exist without a module, that was the "Avatar Container". Later I called such objects "Astral Objects (ASOs)". These are only assigned to the frame (Figure 2):

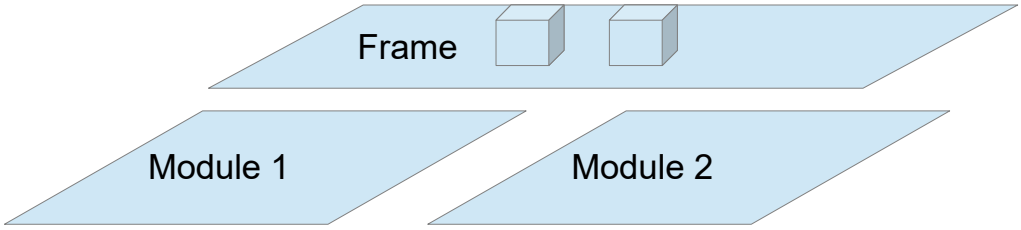

Figure 2: Objects that are not bound to any module (Astral Objects - AOBs)

Since astral objects are not assigned to a module, they can not be rendered. According to my nomenclature ("object" is either "model" or "MIDAS object") so only MIDAS objects can act as astral objects, models can not do that (a model that can never be rendered makes no sense).

Now there are the unbound objects (UBOs) - that is, they will exist - which are normally assigned to a module, so that you can render them, but they can also change the module (which I call "handover") and they can even temporarily act as astral objects - eg if you "pull away the module under the butt".

The transition from a module to the frame is called a "deAttachment" - the object is then "detached" - and the transition from the frame to a module is an "attachment".

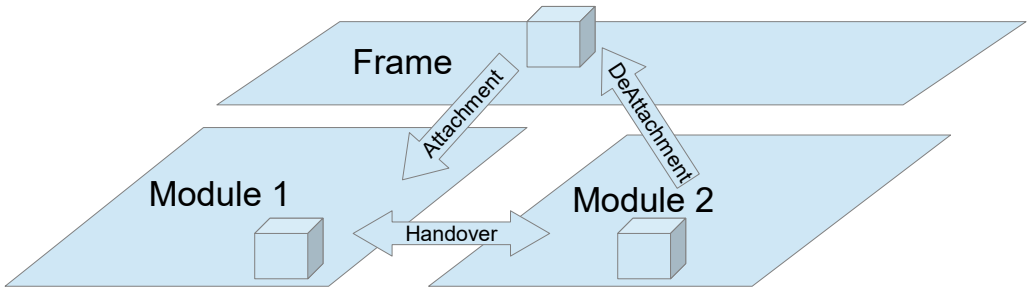

Abbildung 3: Ungebundene Objekte (Unbound Objects - UBOs)

From all these considerations, the following architecture necessarily results for the SRR framework:

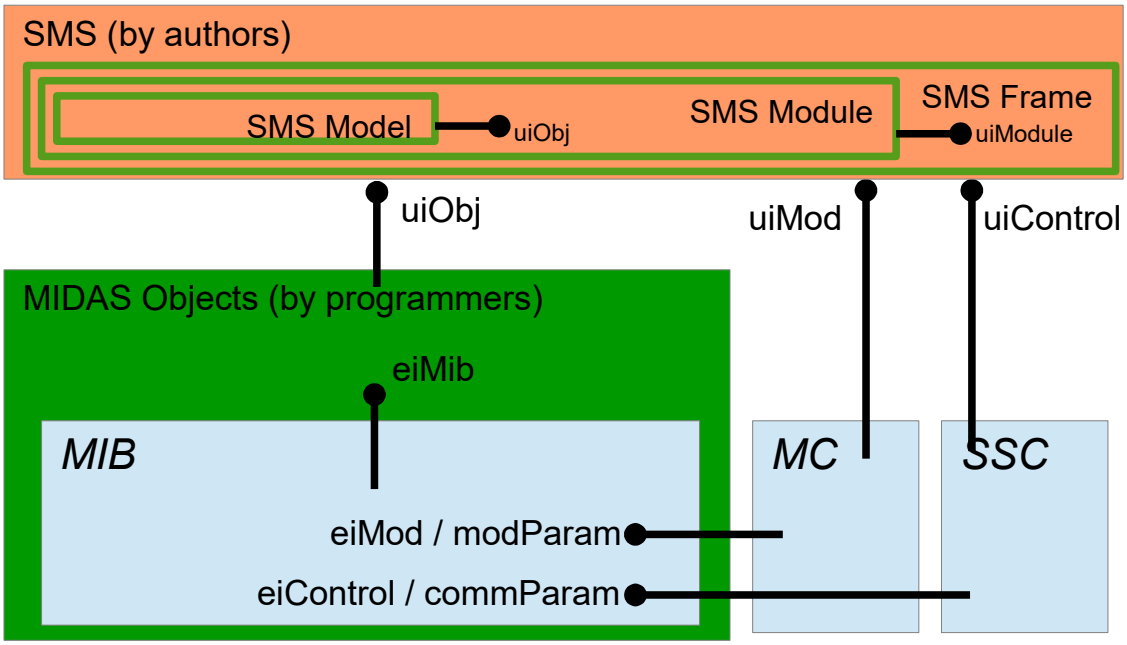

Figure 4: The parts of the SRR Framework: SSC, MC and MIB

The Simple Scene Controller (SSC) exists exactly once (per scene instance) and is used within the frame to control the central features of the simulation.

It is used by the frame via the user interface @uiControl and also offers its services to the other parts of the scene via the external interface @eiControl. We call @eiControl an external interface because it can be used from outside the subsystem "SimpleSceneController".

The Module Coordinator (MC) exists exactly once per module (and per scene instance). It is used within each module to coordinate module parameters and all objects of the module.

It is used by the module via the user interface @uiMod and offers its services also to the other parts of the module, including all objects - but not to the frame - via the external interface @eiMod. We refer to @eiMod as an external interface because it can be used from outside the ModuleCoordinator subsystem.

The MIDAS Base (MIB) helps programmers develop MIDAS objects.

One or more MIDAS objects can be used to instrument a model. They offer their services via the user interface @uiObj, which always has to provide a certain basic set of functionalities (otherwise the object would not be an MIDAS object), but depending on the type the MIDAS object can offer any additional functions (depending on what the Programmer has programmed).

This hibernation report now looks at what software instances could be in the SSC, the MC, and the MIB after Step 0033.11 of the SrrTrains v0.01 project will be (may be) completed.

3 SimpleSceneController – Instances and Interfaces

There are several types of instances in the SimpleSceneController subsystem

- Singleton instances (as part of the scene instance)
- Instances "via registered module" (in the context of the scene instance)
- Instances "by UOC" (as part of the scene instance)
- Instances "per object" (in an object - actually in an object instance)

3.1 Singleton Instances

Both the SSC Base and all SSC Extensions are singleton instances. They only exist once per scene instance and contain central functions that are needed in the frame.

SSC Base:

"<SscInstance>" = "Ssc"

SSC Extension:

"<SscInstance>" = "<ParentClassPath>.<wki>

Example: the "<SscInstance>" for Train Manager is "Ssc.Base.SscTrainManager"

SSC instances each provide a @uiControl interface and a @eiControl interface.

3.2 "Per Module" Instances

The SSC Dispatcher, actually the "Module Related SSC Dispatcher" is included in the context of the scene instance - since it is needed if the corresponding module is not (yet) loaded. It is instantiated "per module" and used by bound objects.

"<DispInstance>" = "**Bdo**.<moduleName>"

Example: "**Bdo**.City"

3.3 "Per UOC" Instances

Each UOC needs exactly one SSC Dispatcher, strictly speaking a "UOC Related SSC Dispatcher". This is also part of the scene instance.

"<DispInstance>" = "**Uoc**.<uocName>"

Beispiel: "**Uoc**.Base.Keys"

3.4 "Per Object" Instances

The SSC Dispatcher Stub, which allows each object to access the correct SSC dispatcher, is instantiated "per object" and also within the object.

"<StubInstance>" = "<extObjId>" = "<DispInstance>-<objId>"

Each Dispatcher stub provides an @eiConsole interface for a specific object.

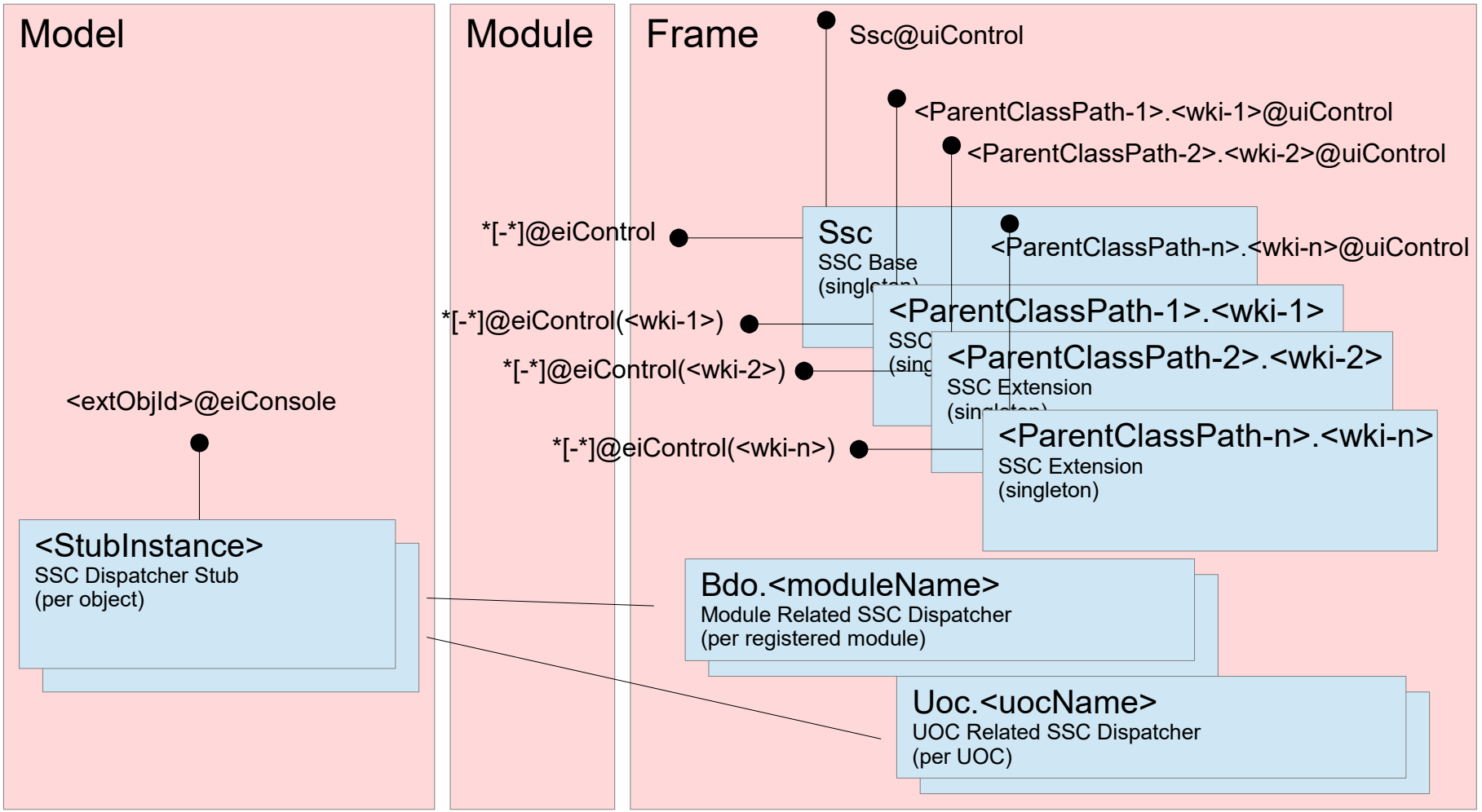

Figure 5: Subsystem "SimpleSceneController" - Instances and Interfaces

4 ModuleCoordinator – Instances and Interfaces

In subsystem "ModuleCoordinator" there are only instances

- by loaded module

to be instantiated.

So these are instances of MC Base or instances of any MC Extensions.

"<McInstance>" = "**Mod**.<ModuleName>-<McClassPath>"

For example, the Train Manager Extension in the "City" module would look like

"<McInstance>" = "Mod.City-McBase.McTrainManager"

The MC base in the module "Hill" would look like

"<McInstance>" = "Mod.Hill-McBase"

The module coordinator uses the @eiControl interface of the SimpleSceneController and offers the @uiMod interface and the @eiMod interface

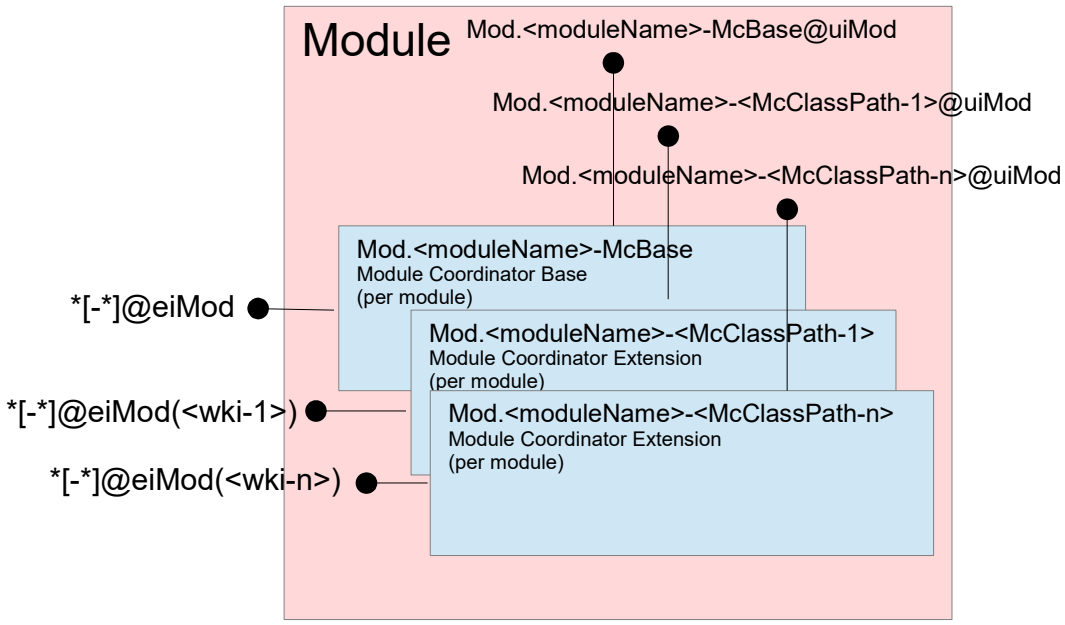

Figure 6: Subsystem "ModuleCoordinator" - Instances and Interfaces

5 MidasBase – Instances and Interfaces

In the MIDAS base, there are only instances that are instantiated "per object". These join the instance of the SSC Dispatcher stub, which is also instantiated "per object".

An object is uniquely defined by an Extended Object ID, which results as follows:

"<extObjId>" = "<DispInstance>-<objId>"

The <objId> is unique within a UOC or within a module.

The difference is made with the prefix "**Uoc**." or "**Bdo**." in the <DispInstance>.

A UOC Related <DispInstance> is used for astral objects and unbound objects, a Module Related <DispInstance> for bound ones.

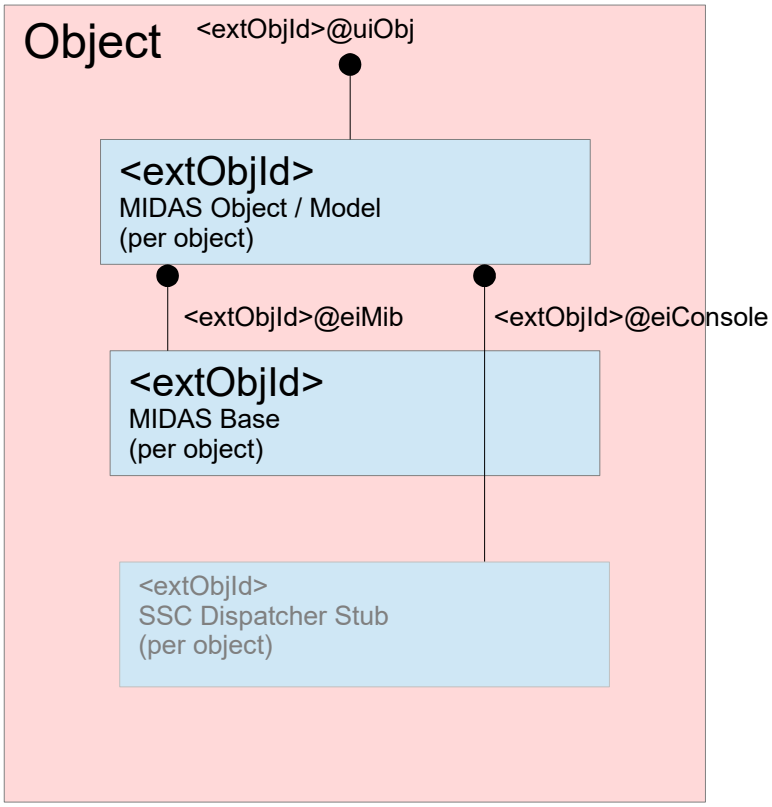

Abbildung 7: Instances and Interfaces that exist "per Object"

6 Basic elements for the "New naming rules"

6.1 Labels for Software Types

6.1.1 "<wki>"

A well-known id identifies an SSC extension or an MC extension. <wki> s are unique identifiers of SMUOS based software worldwide

For example, "<wki>" = "SscTrainManager" shows the software of the SSC Train Manager, which is part of the SRR Framework

6.1.2 "<wku>"

A well-known uoc identifies a UOC relative to the SSC extension that defines the UOC.

For example, defines "<wku>" = "Trains" the UOC "Trains" relative to the Train Manager SSC Extension.

"SscTrainManager.Trains" is therefore a worldwide unique identification of the UOC "Trains"

Note: The "<wku>" can not be changed, but the "<uocName>" (s.u.) can be different in different SMSn (the Frame Author can influence that)

6.1.3 "<SscClassPath>"

"<SscClassPath>" = "Ssc.Base[.<Wki>.<Wku>]..."

The <SscClassPath> identifies a UOC within a specific hierarchy of SSC Base and SSC Extensions in an SMS.

The <SscClassPath> can be used as <ParentClassPath> in a <SscInstance> (see there).

In our example, "<SscClassPath>" = "Ssc.Base.SscTrainManager.Trains" is the <ParentClassPath> for the SSC extension "Ssc.Base.SscTrainManager.Trains.FlyingTrainMgr" in this SMS.

6.1.4 "<McClassPath>"

"<McClassPath>" = "McBase[.<Wki>]..."

The <McClassPath> identifies either the MC Base or the type of MC Extension within a specific hierarchy of MC Base and MC Extensions.

Together with the <moduleName>, the <McClassPath> in a <McInstance> is used to identify an instance of the MC Base or an MC Extension (see there).

For example, could an MC extension in our example scene the

"<McClassPath>" = "McBase.McTrainManager"

receive.

6.2 Labels for elements of the SMS according to the MMF paradigm

6.2.1 "<moduleName>"

A <moduleName> identifies a module within a Simple Multiuser Scene.

Example:

"<moduleName>" = "City" ..... "second module" of the "official demo layout"

6.2.2 "<uocName>"

A <uocName> identifies a Universal Object Class (UOC) within a Simple Multiuser Scene.

Example:

"<uocName>" = "Base" ..... the UOC "Base" of the SSC Base

"<uocName>" = "Base.Keys" ..... a UOC managed by the SscKeyManager

"<uocName>" = "Base.Trains" ..... a UOC with UBO loader in the SscTrainManager

6.2.3 "<extObjId>", "<objId>"

An <extObjId> identifies an object (that is, a MIDAS object or a model). The <objId> is the local object ID (within a UOC or within a module)

Bound objects:

"<extObjId>" = "<DispInstance>-<objId>" = "**Bdo**.<moduleName>-<objId>"

Astral objects:

"<extObjId>" = "<DispInstance>-<objId>" = "**Uoc**.Base-<objId>"

Unbound objects:

"<extObjId>" = "<DispInstance>-<objId>" = "**Uoc**.<uocName>-<objId>"

6.3 Designations for Software Instances within an SMS

Instances are names used in (1) the Tracer, (2) Stream Names, and (3) the console interface to identify elements of the SMS.

6.3.1 "<SscInstance>"

An <SscInstance> identifies either the SSC base or an SSC extension. It ALWAYS starts with the letter sequence "Ssc".

SSC Base:

"<SscInstance>" = "Ssc"

SSC Extension:

"<SscInstance>" = "<ParentClassPath>.<Wki>

Example: the "<SscInstance>" for Train Manager is "Ssc.Base.SscTrainManager"

6.3.2 "<DispInstance>"

A <DispInstance> identifies either a UOC Related SSC Dispatcher or a Module Related SSC Dispatcher. It is also used in the included UBO Loader

Module Related SSC Dispatcher:

"<DispInstance>" = "**Bdo**. <ModuleName>"

Example: "Bdo.City"

UOC Related SSC Dispatcher:

"<DispInstance>" = "**Uoc**. <UocName>"

Example: "Uoc.Base.Keys"

6.3.3 "<McInstance>"

A <McInstance> identifies an instance of the MC Base or an instance of an MC Extension for a specific module.

"<McInstance>" = "**Mod**.<ModuleName>-<McClassPath>"

For example, the Train Manager Extension in the "City" module would look like

"<McInstance>" = "Mod.City-McBase.McTrainManager"

6.3.4 "<ObjInstance>"

An <ObjInstance> identifies an instance of a MIDAS object or model.

Bound objects:

"<ObjInstance>" = "<extObjId>"

Example: "Bdo.City-StationHouse.Door.Switch.Lock"

Astral objects:

"<ObjInstance>" = "<extObjId>"

Example: "Uoc.Base-DefAvaCon" is the default avatar container

Unbound objects:

"<ObjInstance>" = "<extObjId>"

Example: "Uoc.Base.Trains-tgv0001"

7 Tracer instances and interfaces

There are some trace points in the Tracer for which instance naming (<instance>) is extremely important. The most important are "messageReceived", "sendMessage", "eventReceived" and "sendEvent".

SMS: trace level = 2, timestamp = <timestamp>, ownSessionId = **<sessionId>**
SMS: instanceId = **<instance>**, subsystem = <ss>, fileName = <fileName>, ssVersion = <ssVersion>
SMS: action = **messageReceived**; sender = **<sender>**; messageType = <mt>; messageId = <mid>
SMS: <free text from the programmer>
SMS: END

SMS: trace level = 2, timestamp = <timestamp>, ownSessionId = **<sessionId>**
SMS: instanceId = **<instance>**, subsystem = <ss>, fileName = <fileName>, ssVersion = <ssVersion>
SMS: action = **sendMessage**; receiver = **<receiver>**; messageType = <mt>; messageId = <mid>
SMS: <free text from the programmer>
SMS: END

A message is thus always sent from an <instance> within a <sessionId> to an <instance> within a <sessionId>:

<sender> = <instance>@<sessionId>
<receiver> = <instance>@<sessionId>

An event can only be sent within one <sessionId> from one <instance> to another <instance>, where the <field> has the same name at both ends.

Events between two subsystems are routed via so-called interfaces (<ifInstance>@<ifName>).

SMS: trace level = 2, timestamp = <timestamp>, ownSessionId = **<sessionId>**
SMS: instanceId = **<instance>**, subsystem = <ss>, fileName = <fileName>, ssVersion = <ssVersion>
SMS: action = sendEvent; destination = **<destination>**; field = <field>
SMS: <free text from the programmer>
SMS: END

SMS: trace level = 2, timestamp = <timestamp>, ownSessionId = **<sessionId>**
SMS: instanceId = **<instance>**, subsystem = <ss>, fileName = <fileName>, ssVersion = <ssVersion>
SMS: action = eventReceived; origin = **<origin>**; field = <field>
SMS: <free text from the programmer>
SMS: END

It applies

<origin> = <instance>
<destination> = <instance>

if Origin and Destination are in the same subsystem, and it applies

<origin> = <ifInstance>@<ifName>
<destination> = <ifInstance>@<ifName>,

if that is not the case.

This is to be made clear in the following two subchapters.

7.1 Tracer Instances

7.1.1 Tracer Instances for SSC Base and SSC Extensions

Controller / Central Controller:

"<SscInstance> .Control" ..... 1x per scene instance (singleton)

"<SscInstance> .CentralSrv" .... 1x per multiuser session, manages CommState / Global State

7.1.2 Tracer Instances for SSC Dispatcher and SSC UBO Loader

Client / Server / UOC Server:

"<DispInstance> .Client" ..... 1x per scene instance and per module / UOC

"<DispInstance> .Server" ..... 1x by scene instance and module / UOC, processes SMS requests

"<DispInstance> .UocSrv" .... 1x per multiuser session and module / UOC, manages UBO State

7.1.3 Tracer instances for MC Base and MC Extensions

Coordinator:

"<McInstance> .Coord" ..... 1x by scene instance and by loaded module, coordinates module

7.1.4 Tracer instances for objects

Client Software / Object Controller / Console Server

"<ObjInstance> .Control" ..... 1x by scene instance and loaded object

"<ObjInstance> .ObCo" ..... 1x per multiuser session and object, manages Global State

"<ObjInstance> .Server" ..... 1x by scene instance and object, processes SMS requests

7.2 Tracer Interface

7.2.1 @commParam Interface

* [- *]@commParam SscBase distributes trace levels and events to avatars

<mcInstance>@commParam McBase receives Trace Levels from SscBase

<objInstance>@commParam MoosAvatarContainer receives events from SscBase

7.2.2 @eiControl Interface

* [- *]@eiControl SscBase receives request from somewhere [and answers]

<mcInstance>@eiControl Exchange between SscBase and McBase

<objInstance>@eiControl Exchange between SscBase and some MIDAS objects:
MoosAvatarContainer, MoosCreator,
MoosSwitchA, MoosSwitchB

7.2.3 @commParam(<wki>) interface

*[- *]@commParam(<wki>)	SscBeamerManager distributes Beamer Destinations
<mcInstance>@commParam(<wki>)	is not used (receive)
<objInstance>@commParam(<wki>)	MoosBeamer (receives Beamer Destinations)

7.2.4 @eiControl(<wki>) interface

* [- *]@eiControl(<wki>)	is not used (request from somewhere)
<mcInstance>@eiControl(<wki>)	Exchange between SrrControlTm and SrrModCoordTm
<objInstance>@eiControl(<wki>)	Exchange between SscBeamerManager / SscKeyManager / SrrControlTm and some MIDAS objects MoosBeamer / MoosBeamerDestination / MoosKeyContainer / MoosLockA / MoosLockB / SrrReplicator

7.2.5 @modParam interface

Modules.<ModuleName>-*@ modParam	McBase distributes MAM
Modules.<ModuleName>-<objId>@modParam	MibAnim / MibNoStateOsm / MibStandardOsm receive the MAM

7.2.6 @eiMod Interface

Modules.<ModuleName>-*@eiMod	McBase receives request from somewhere [and answers]

7.2.7 @modParam(<wki>) interface

Not used at the moment

7.2.8 @eiMod(<wki>) interface

<objInstance> @eiMod (<wki>)	Exchange between SrrModCoordTm and the MIDAS Objects SrrBasicTrackSection / SrrBasicTurnout2Way / SrrTrackEdge / SrrTrackNode

7.2.9 @eiConsole Interface

<objInstance>@eiConsole	Exchange between SscDispatcherStub and their Users SscBase / SscKeyManager / MoosCreator / MoosDriveA / MoosKeyContainer / MoosLockB / MoosSwitchA / MoosSwitchB

7.2.10 @eiMib interface

<objInstance>@eiMib	Importing MIB through a MOB

7.2.11 @uiObj interface

<objInstance>@uiObj is offered by:
MibCore / MoosAvatarContainer / MoosBeamer / MoosBeamerDestination / MoosCreator / MoosDriveA / MoosKeyContainer / MoosLockA / MoosLockB / MoosSwitchA / MoosSwitchB / MoosTrigger / SrrBasicTrackSection / SrrBasicTurnoutr2Way / SrrReplicator / SrrTrackEdge / SrrTrackNode

is used by
SscBase (Avatar Container) / Model / Module / Frame

7.2.12 @uiMod interface

<mcInstance>@uiMod is offered by:
McBase / SrrModCoordTm

is used by:
modules

7.2.13 @uiModule Interface

Module.<ModuleName>@uiModule is used by:
SmsModuleLoader / Module Wrapper / Frame

7.2.14 @uiControl Interface

<sscInstance>@uiControl is used by: Frame

8 StreamNames

Stream names have the general form

"<StreamName>" = "Sms-<mainIdentifier>-<User>.<Suffix>"

where <suffix> can be freely chosen by the programmer (everything except points and dashes)

<User> is specified by the SMUOS Framework and will be explained together with the <mainIdentifier> in the next chapters

8.1 SSC Base and SSC Extensions

"<mainIdentifier>" = "<SscInstance>"

"<User>" = "Base", if it is a network sensor of SSC Base and

"<User>" = "Ext", if it is a network sensor of an SSC extension

8.2 SSC Dispatcher and SSC UBO Loader

"<mainIdentifier>" = "<DispInstance>"

"<User>" = "SmsDispatcher" if it is a network sensor of the SSC Dispatcher and

"<User>" = "UboLoader" if it is a Network Sensor of the SSC UBO Loader

8.3 Objects

Stream Names:

"<mainIdentifier>" = "<ObjInstance>"

"<User>" = "Obj" if it is a Network Sensor of the MIDAS object or the model

"<User>" = "Mib", if it is a Network Sensor of MIDAS Base

9 Console interface

<DispInstance>-<objId>-<parameter name>