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

This file valid for step 0033.11
Issue Date: tbd.

Unbound Objects (*not yet implemented* - planned for step 0033.11 - *TODO11*)

1 Synopsis

Unbound models - or more generally unbound objects (UBOs) - are an advanced concept of the SRR/SMUOS Framework. It's not sufficient to use the SMUOS Framework, if you like to have unbound objects. You need (an) extension(s) of the SMUOS Framework that support(s) unbound objects.

Extensions of the SMUOS Framework may choose to support so-called "universal object classes" (UOCs). Each UOC may provide for

- SSC parameters (parameters of the SSC Extension that are accessible via the console interface) - please refer to 014_ConsoleInterface for details
- a new class of unbound objects (rail vehicles, trains, helicopters, ...)

This paper examplifies the concept of unbound objects and describes them from the point of view of the "Train Manager Extension" (TME) of the SMUOS Framework.

2 Purpose

Unbound objects

- can be created/deleted on demand during the runtime of the scene
- are still rendered relative to a module (as all other rendered objects are)
- may change their being attached to another module (handover)
- are identified by a "universal object class name" (UOC name) and by an "object ID" (the UOC name is defined by the SMUOS extension)
- being attached to this or that module may change, but the "universal object class" of an unbound object never changes during lifetime

2.1 Glossary

An object type ID (OTI) can be "registered" at the SRR/SMUOS Framework, hence making known the URL(s) and the categories of an object type via UOC + OTI.

An object type can be instantiated more than once, hence "creating" unbound objects (UBOs) of one and the same UOC + OTI, but with different objIds.

An OT is identified by UOC + OTI, a UBO is identified by UOC + objId.

Whether a UBO is "created" or not, is a global property, which is stored in the "existence state" (ES) of the UOC. In case of the TME, we call the existence state the "train/vehicle state (tvState)".

When a UBO is globally "created", then it will become "loaded" in each scene instance. Being "loaded" or "unloaded" is a local property.

An SSC Extension may implement "delayed loading" of UBOs, where the UBOs are not loaded immediately upon creation, but later. In this case the SSC Extension must define and implement some criterion, when the loading SHOULD take place.

A UBO is "assigned" to a module, which is a global property. Changing the being assigned to another module is called "global handover".

When the module of a UBO exists in a scene instance, then the "positioning" MIDAS Objects will try to get "positioned", which is a local property.

When a UBO is "positioned" in a scene instance, then it gets "attached" to the "current module", and hence it gets "visible" locally.

A UBO can be deleted anytime. Being deleted is a global category. If a UBO is deleted, then it will be unloaded in each scene instance.

An SSC Extension may implement "precurring unloading" of UBOs, where the UBO may be unloaded in a scene instance, although it is still in global state "created". In this case the SSC Extension must define and implement a criterion, when the unloading SHOULD take place.

If an OT is UNREGISTERED, then all instances of this OT (all UBOs) are automatically deleted, before the OT is unregistered.

2.2 Architecture for UBOs

If you implement an SSC Extension, then you have to consider a few things.

Among others:
- Does my SSC Extension need to define one or more UOCs?
- Does my SSC Extension need to support SSC parameters?
- Does my SSC Extension need to load and unload unbound objects?

The key prototype to define UOCs is the SscDispatcher.
The key prototype to handle SSC parameters is the SmsDispatcherStub.
And the key prototypes to handle UBOs are the SscUboLoader and the Creator.

All of these prototypes can be instantiated and used from the SMUOS Framework and they are coordinated by the SscCore prototype.

Now the basic architecture for the support of UBOs can be drawn as follows:

```
            M  O  D  U  L  E                             F  R  A  M  E
            ================                             =============
                                          +------------------------------------+
                                          | SSC Extension supporting UBOs      |
                                          |                                    |
                                          | +------------------------+         |
                                          | | commParamExtension     |         |
                                          | |                        |         |
   +----------------------------------------- - ptr (points to UBO Loader)     |
  +|---------------------------------+    | +------------------------+         |
  ||Extension MIDAS Object           |    |           +======================+ |
  ||                                 |    |           | UBO Loader           | |
  ||           +===================+ |    |           |                      | |
  |+---------->| Creator           |<---UBO Support-->| loads / unloads UBOs | |
  |            +===================+ |    |           +======================+ |
  +----------------------------------+    +------------------------------------+
```

In the beginning, the Creator - which cares for loading/unloading of UBOs via the UBO Loader and which is provided as a part of the SMUOS framework - cannot access the UBO Loader. The UBO Loader is instantiated as a part of an SSC Extension - which is per se unknown by the Creator (a basic MIDAS Object).

But the SSC Extension must provide an Extension MIDAS Object that can access the "ptr" in the commParamExtension and therefore the Creator receives the "ptr" and can register at the UBO Loader. Now the Creator can care for UBOs.

2.3 Use Cases with UBOs

2.3.1 Statically Registering Object Types

The SSC Base provides an SFNode field "uboConf" at its user interface uiControl. The frame can load a configuration file and e.g. provide a <Script> node at this field, before it starts the initialization of the SRR/SMUOS Framework.

During initialization, each UBO Loader will look for Object Type IDs (OTIs) in that "uboConf". If it finds them - together with their URLs and categories -, then it will store the categories and URLs and it will globally register the OTIs.

OTIs are globally registered, categories and URLs are stored locally in each scene instance. Hence the categories and the URLs might differ in different scene instances. In this case we say, the scene instances have got different "views" on that object types.

2.3.2 Registering an Object Type Dynamically

Will not be implemented

2.3.3 Creating a UBO from a Registered OT

Not yet implemented

2.3.4 Establishing a UBO (Dynamically Registering and then Creating)

Will not be implemented

2.3.5 Deleting a UBO

Not yet implemented

2.3.6 Beaming a UBO

Will not be implemented

2.3.7 Deregistering an OT

Not yet implemented

3 External View

tbd

4 Internal View

tbd

5 Additional Info

tbd