Simulated Railroad Framework, http://simulrr.sourceforge.net
Synopsis: 100_SrrFramework

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

The Simple Scene Controller

1 Synopsis

The Simple Scene Controller is the part of the SRR/SMUOS Framework that is used by the frame of an SrrTrains layout / Simple Multiuser Scene.

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

The actual SSC Extensions "Beamer Manager", and "Key Manager", which are delivered together with the SMUOS Framework, are described in the paper 122_BasicSscExtensions.

The SSC Extension "Train Manager" is actually a part of the SRR Framework and it is described at 131_SrrControllerTm.

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

```
+------------------------------------------------------------------------------+
|  Simple Scene Controller (SSC)                                               |
|                                                                              |
|   +-----------------------------+    +-----------------------------+         |
|   |  SSC Base (part of SMUOS)   |    |  SSC Extension (3rd party)  |         |
|   |                             |    |   +-----------------------------+     |
|   |                             |    | +-|  SSC Extension (3rd party)  |     |
|   |                             |    | | |   +-----------------------------+ |
|   |                             |    | | | +-|  SSC Extension (3rd party)  | |
|   |                             |    | +-| | |                             | |
|   | +-------------------------+ |    +---| | | +-------------------------+ | |
|   | | SSC Core (part of SMUOS)| |        | +-| | SSC Core (part of SMUOS)| | |
|   | |                         | |        +---| |                         | | |
|   | +-------------------------+ |            | +-------------------------+ | |
|   +-----------------------------+            +-----------------------------+ |
|                                                                              |
+------------------------------------------------------------------------------+
```

The "Base" of the Simple Scene Controller consists of two X3D prototypes, which are contained in the files SmuosSscBase.x3d ("client" and "server") and SmuosSscBaseNs.x3d ("network sensor") - see chapter 3 for how to use the SSC Base and chapter 4.1 on how the SSC Base is built internally.

Additionally, all parts of the SSC may use the "SSC Core" subsystem, which is contained in the files SmuosSscCore.x3d, SmuosSscActivationController.x3d, SmuosSscDispatcher.x3d, SmuosSscDispatcherNs.x3d, SmuosSscDispatcherStub.x3d, SmuosSscUboLoader.x3d, SmuosSscUboLoaderNs.x3d and SmuosSscUboLoaderStub.x3d. The application of SSC Core is described in chapter 4.2.

2 The Purpose of the Simple Scene Controller

The Simple Scene Controller helps with the coordination of the SMUOS Framework in the frame of an SrrTrains Layout / SMS.

The Simple Scene Controller provides an interface to
- initialize the SMUOS Framework in multi-user-mode or in single-user-mode
- add/remove avatars
- report avatar position/orientation to the frame in relative coordinates
- request the controller role
- register/deregister modules explicitely
- activate/deactivate modules
- request MOC roles
- set trace levels and output tracer output
- display carried keys **)
- put carried keys into key containers or locks **)
- reset all keys **)
- send/receive console commands/responses
- display all beamer destinations *)
- bind one beamer destination *)
- join a remote user
- output short status messages
- read configuration data for the UBO Loaders
- support use cases of the train manager extension ***)
- support use cases of 3rd party SSC Extensions ****)

*) realized by the SSC "Beamer Manager" Extension (see 122_BasicSscExtensions)
**) realized by the SSC "Key Manager" Extension (see 122_BasicSscExtensions)
***) please refer to 131_SrrControllerTm
****) Those are currently not described in the present Concepts' Descriptions

3 External View (uiControl, commParam, eiControl)

When the <ProtoInstance> "SscBase" and all <ProtoInstances> of the SSC Extensions have been loaded, then the Simple Scene Controller is in mode of operation "LOADED" (MOO "LOADED").

After some preparation, the frame sends the "init"=true event to the uiControl interface of the SSC Base and triggers the initialization of the whole SSC.

After successful or unsuccessful initialization the SSC Base outputs the SFNode event "commParam", that points to the common parameters. The common parameters are hold as fields of a <Script> node, which is contained in the SSC Base. They are described in chapter 5.1.

The "commParam" event can be immediately <ROUTE>d to the module coordinators of all static modules (via interface uiModule) and to the external interface uiObj of all astral objects to trigger their initialization.

```
                              THE FRAME
                                   o
                                   | uiControl
                          +--------+--------+
  MOB, MC   eiControl o---+       SSC       |
                          +-----------------+
```

With the help of the commParam, any part of the scene can access the eiControl interface, which is mainly intended for the use by MIDAS Objects (MOBs) and by the Module Coordinator (MC). Both interfaces - uiControl and eiControl - can be extended by SSC Extensions, as described in chapters 3 of 051_Extensibility, 122_BasicSscExtensions and 131_SrrControllerTm.

3.1 MOO Diagram (MOOD)

```
                                 MOO "LOADED"
                                      |
                                      | --> commParam
                                      |
                                      V
                                    MOO I (commParam.iAmActive=true/false)
```

3.2 Use Case Matrix (uiControl)

Following use cases are available at uiControl in following MOOs:

MOO "LOADED"	MOO I (activated)	MOO I (inactive)
Preparation	-	-
Initialization	-	-
-	RemoteUserLeft	-
AddAvatar	AddAvatar	-
RemoveAvatar	RemoveAvatar	-
RequestController	RequestController	-
-	DynModRegistration	-
-	ModuleActivity	-
SetTraceLevels	SetTraceLevels	-
-	ConsoleInterface	-
-	StatusMessage	-

The Use Case Matrix displays, which use case is available via the uiControl interface in which mode of operation (MOO) of the SSC Base. An additional differentiation is done based on "commParam.iAmActive" (activated/inactive).

The following sections explain each use case in detail

3.2.1 Preparation

Before the SSC is initialized, the following fields must be set:

```
   - "mandatorySscBaseExtensions" (MFNode)
   - "optionalSscBaseExtensions" (MFNode)
        The frame instantiates all SSC Extensions and tells the SSC Base their
        addresses so that he can initialize them.
   - "uboConf" (SFNode) should refer to a <Script> node that holds UBO info
```

In case of multi-user-mode, following fields must be set additionally:

```
   - "multiuserRequest" (SFBool)
        set to TRUE to switch to multi-user-mode
   - "sessionId" (SFInt32)
        non-negative integer to identify the scene instance within the multiuser
        session
   - "useConnection" (SFString)
        value to refer to the <NetConnection> node of BS Contact, needed by the
        Event Stream Sensors
```

3.2.2 Initialization

With an event "init"=true, the frame triggers the initialization of the SSC Base and of all SSC extensions that are referenced by the fields "mandatorySscBaseExtensions" and "optionalSscBaseExtensions".

The SSC Base will answer with an event "commParam" != null, where the field "commParam.iAmActive" (SFBool) will indicate whether the initialization was successful or not.

3.2.3 RemoteUserLeft

With an event "sessionIdLeft" = <sessionId> the frame informs the SSC that a remote user has left the multiuser session.

The SSC will update the commState and re-arrange the controller role (if necessary).

3.2.4 AddAvatar/RemoveAvatar

Usually the multiuser system informs the frame, when a user has joined/has left the multiuser session.

This information is used by the frame to load/unload avatars and to forward their addresses to the SSC in order to keep the avatar containers up to date (please refer to 351_AvatarContainer for a description of avatar containers).

The events "addAvatar" and "removeAvatar" are also acceptable, when the SSC has not yet been initialized.

A remote user can be joined by entering the new position/orientation ("joined Position", "joinedOrientation") and afterwards entering the sessionId of the remote user ("joinAvatar").

3.2.5 RequestController

With an event "controllerRequest" = true the frame can request the controller role for his scene instance.

3.2.6 DynModRegistration

The frame can explicitely register/deregister dynamic modules with the help of the fields "registerModules" and "deregisterModules".

"registerModules" will only allocate a moduleIx (that will be used in all scene instances for this module name). The actual loading and initialization of the module shall be performed by the frame. This can be supported by the SMS Module Loader (see 101_SmsBase for a description of the SMS Module Loader).

A dynamic module may be loaded and initialized without explicit registration, an implicit registration will be done during module initialization (as it is done for static modules). In this case, the frame does not know the moduleIx prior to module initialization but only afterwards.

"deregisterModules" will deregister the module and detach the module coordinator in all scene instances. The SSC will inform the frame about the deregistration of the modules ("registeredModules" field). The actual unloading of the module has then to be performed by the frame (maybe supported by the SMS Module Loader, see above).

A frame can locally unload one instance of a dynamic module (refer to chapter "Use Case DisableModule" in 201_ModuleCoordinator).

3.2.7 ModuleActivity

Module activity is usually changed by the module itself. To achieve this, the module sends an event to the external interface uiMod of the module coordinator.

Please refer to chapter "Use Case ModuleActivity" in 201_ModuleCoordinator for a short introduction to module activity.

Additionally, the frame has the possibility to influence module activity via the fields "activateRequest", "deactivateRequest" and "mocRequest" of the external interface uiControl of the Simple Scene Controller.

3.2.8 SetTraceLevels

The frame can set all trace levels of his scene instance. Please refer to 101_SmsBase for a more detailed description.

3.2.9 ConsoleInterface

The frame can request the handling of a console command (an SFString value, i.e. a command line of a given syntax).
Each console command will be answered with exactly one console response (an MFString value, i.e. a list of response lines), unless the console is already active (in which case the console command will be silently discarded).
The syntax of the console response is kept as simple as possible, hence easing the automatic handling by some GUI controller (MVC pattern).
Please refer to 014_ConsoleInterface for more details.

3.2.10 StatusMessage

The Simple Scene Controller outputs short status messages via the fields "statusMessage" (SFString), "statusTrigger" (SFTime), "statusIsActive" (SFBool) and "statusSoftState" (SFFloat).

3.3 Interworking with Module Coordinator and MIDAS Objects (commParam/eiControl)

The SSC can use the "common parameters" (see chapter 5.1) to broadcast information to all parts of the scene. This is actually used for the Avatar Container MIDAS Object (see 351_AvatarContainer).

When a part of the scene knows the "commParam", then it can access the eiControl interface of the SSC Base.

Also the Avatar Container requires some fields at the eiControl interface (see 351_AvatarContainer) and the module coordinator (MC Base) is supported by fields of the eiControl interface (see 201_ModuleCoordinator).

The SSC Base provides the field "displayStatusMessage" (SFString) at the eiControl interface, to enable the triggering of the output of short status messages by any part of the scene.

The field "getSscBaseExtension" can be used to assert the presence of an SSC Extension in the "commParam.extensions" field.

3.4 Other Interfaces of the SSC Subsystem

The SSC has an interface between SSC Dispatcher and SMS Dispatcher Stub. This interface is an internal interface and will not be specified here.

The SSC has an interface between SSC UBO Loader and SMS UBO Loader Stub. This interface is an internal interface and will not be specified here.

4 Internal View

4.1 Internal Structure of the SSC Base

Tbd.

4.2 How to Implement an SSC Extension (SscExt)

Chapter 3 describes the external interfaces uiControl, commParam and eiControl – i.e. how frames, the MC and MOBs can use the SSC.

Now,
- the uiControl interface can be extended by SSC Extensions (uiControl(WKI))
- the eiControl interface can be extended by SSC Extensions (eiControl(WKI))
- the commParam must be extended by commParamExt for each SSC Extension
- each SSC Extension must provide a minimum interface miControl to the parent and it should use the SSC Core via its external interface eiSscCore

```
                           T   H   E       F   R   A   M   E

                          uiControl           uiControl(WKI)
                              o                      o
                              |                      |
                      +-------|----------------------|---------+
                      | SSC   |                      |         |
                      |  +----+----+          +------+------+  |
                      |  | SscBase |          | SscExt(WKI) |  |
eiControl(WKI) o------------------------------+             |  |
                      |  |         |       eiSscCore o      |  |
     eiControl o---------+         |miControl | +----|----+ |  |
                      |  |         +-(o---------+ SscCore | |  |
                      |  |         |          | |         | |  |
                      |  |         |          | +---------+ |  |
                      |  +---------+          +-------------+  |
                      +----------------------------------------+
```

Note: WKI is the "well-known-id", which identifies the SSC Extension (please refer to 011_NamingRules for details).

4.2.1 The Extension of uiControl and eiControl

Each SSC Extension (SscExt) is free to add fields to the uiControl and to the eiControl interface, we denote these extensions as uiControl(WKI) and eiControl(WKI), where WKI is the "well-known-id" of the SSC Extension.

uiControl(WKI) and eiControl(WKI) of the "Key Manager" and of the "Beamer Manager" are described at 122_BasicSscExtensions, the "Train Manager" is described at 131_SrrControllerTm.

4.2.2 The Minimum Interface miControl of all SSC Extensions

Each SSC Extension MUST provide following fields to its parent:

Following fields are handled directly by the contained "SscCore" prototype and should be <connect>ed to the interface eiSscCore:

- 'parentInstance' (SFString), 'parentWku' (SFString), 'parentUoc' (SFString),
- 'commParam' (SFNode), 'disable' (SFTime), 'startActivity' (SFBool),
- 'initializeStateAndStartActivity' (SFBool) and 'enabledOut' (SFBool).

The field "wellKnownId" (SFString) must be set by the SscExt during basic initialization, which must be finished by "basicallyInitialized" (SFBool).

The fields "activityStarted" (SFNode) and "initialized" (SFNode) must be used to report successful/unsuccessful initialization/activation (more details at chapter 4.2.4).

The fields "traceLevelControl" (MFInt32) and "traceLevelComm" (MFInt32) may be used to inherit trace levels from the SSC Base.

4.2.3 The external Interface of SscCore (eiSscCore)

The subsystem "SSC Core" consists of following prototypes:

- SmuosSscCore.x3d
  is the "Core Prototype" that coordinates the MOOs of all parts of the SSC.

- SmuosSscActivationController.x3d
  is internally used by SscCore and by SscUboLoader.

- SmuosSscDispatcher.x3d / SmuosSscDispatcherNs.x3d (see 123_SscDispatcher)
  is internally used by SscBase and must be used by each SSC Extension,
  that wants to define one or more UOCs.
  One SSC Dispatcher must be instantiated for each UOC.

- SmuosSscDispatcherStub.x3d (see 123_SscDispatcher for a description)
  For each UOC that shall provide SSC Parameters, the SSC Extension must
  instantiate one SMS Dispatcher Stub.

- SmuosSscUboLoader.x3d / SmuosSscUboLoaderNs.x3d / SmuosSscUboLoaderStub.x3d
  For each UOC that shall provide Unbound Objects (UBOs), the SSC Extension
  must instantiate one SSC UBO Loader.

```
             +----------------------------------------------------------------+
             |   SSC Extension                                                |
             |                                                                |
             |                 +-------------------------------------------+  |
             | e               |   SSC Core                                |  |
  (4.2.3.1)  |    o--------------                                          |  |
             |                 |                                           |  |
             | i          +---------------------------------------------+  |  |
  (4.2.3.2)  |    o---------  Common Parameter Extension                |  |  |
             |            |                                             |  |  |
             | S          +---------------------------------------------+  |  |
             |            +---------------------------------------------+  |  |
             |            |   SSC Dispatcher (defines one UOC)          |  |  |
  (4.2.3.3)  | s  o---------                                            |  |  |
             |            |                                             |  |  |
             |       +----------------------------------------+         |  |  |
  (4.2.3.4)  | c  o----  SSC Dispatcher Stub (SSC Parameters) |         |  |  |
             |       |                                        |         |  |  |
             |       +----------------------------------------+         |  |  |
             | C          |                                             |  |  |
             |       +----------------------------------------+         |  |  |
  (4.2.3.5)  |    o----  SSC UBO Loader (Unbound Objects)     |         |  |  |
             | o     |                                        |         |  |  |
             |       +----------------------------------------+         |  |  |
             |            |                                             |  |  |
             | r          |                                             |  |  |
             |            +---------------------------------------------+  |  |
             |                 |                                           |  |
             | e               +-------------------------------------------+  |
             +----------------------------------------------------------------+
4.2.3.1 The Fields of the SscCore Prototype
-------------------------------------------
4.2.3.1.1 Fields for the external interface of an SSC Extension
---------------------------------------------------------------
- get information from the parent part of the SSC
<field accessType='inputOutput' name='parentInstance' type='SFString' value=""/>
<field accessType='inputOutput' name='parentWku' type='SFString' value=""/>
<field accessType='inputOutput' name='parentUoc' type='SFString' value="Uoc"/>
<field accessType='inputOutput' name='refinedClassPaths' type='MFString'
value='"Ssc.Base"'/>
<field accessType='inputOnly' name='commParamIn' type='SFNode'/>
<field accessType='inputOnly' name='disableIn' type='SFTime'/>
<field accessType='inputOnly' name='startActivity' type='SFBool'/>
<field accessType='inputOnly' name='initializeStateAndStartActivity'
type='SFBool'/>
- report the state of the SSC Extension to the parent part of the SSC
<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='activityStarted' type='SFNode'
value="NULL"/>
```
Explain!!!: "refinedClassPaths", "disabled", "initialized", "activityStarted"

```
4.2.3.1.2 fields provided internally by/to the SSC Extension
------------------------------------------------------------
<field accessType='inputOutput' name='traceLevelControl' type='MFInt32'
value='1,1'/>
<field accessType='inputOutput' name='traceLevelComm' type='MFInt32'
value='1,1'/>
<field accessType='inputOutput' name='commParamExtIn' type='SFNode'
value='NULL'/>
<field accessType='inputOutput' name='networkSensorsIn' type='MFNode'></field>
<field accessType='inputOutput' name='sscDispatchersIn' type='MFNode'></field>
<field accessType='inputOutput' name='noState' type='SFBool' value="true"/>
- output lifetime parameters towards the SSC Extension
<field accessType='inputOutput' name='sscInstance' type='SFString'
value='[SscInstance]'/>
<field accessType='inputOutput' name='tracerInstanceIdClient' type='SFString'
value='[instanceIdClient]'/>
<field accessType='inputOutput' name='tracerInstanceIdServer' type='SFString'
value='[instanceIdServer]'/>
<field accessType='inputOutput' name='assertedOptionalSscExtensions'
type='MFNode'></field>
- output dynamic parameters towards the SSC Extension -->
<field accessType='outputOnly' name='heartbeat' type='SFFloat'/>
<field accessType='inputOutput' name='commParamExt' type='SFNode' value="NULL"/>
<field accessType='inputOutput' name='oldCommParamExt' type='SFNode'
value="NULL"/>
<field accessType='inputOutput' name='currentProcedure' type='SFString'
value="nil"/>
<field accessType='outputOnly' name='startProcedure' type='SFBool'/>
<field accessType='inputOnly' name='procedureFinished' type='SFBool'/>
- coordinate the activation of this SSC Extension
<field accessType='inputOutput' name='nsIntd' type='SFInt32' value='0'/>
<field accessType='outputOnly' name='initializeState' type='SFBool'/>
<field accessType='inputOutput' name='globalStateStored' type='SFBool'
value='false'/>
<field accessType='outputOnly' name='handleGlobalState' type='SFBool'/>
<field accessType='inputOnly' name='globalStateHandled' type='SFBool'/>
<field accessType='outputOnly' name='activityStart' type='SFBool'/>
<field accessType='inputOnly' name='startedActivity' type='SFBool'/>
```
Explain!!!: "noState", tracer, network sensors, ssc dispatchers, commParamExt

4.2.3.2 The Minimum Fields of a Common Parameter Extension

```
<!--  ....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='initializeOnly' name='wellKnownId' type='SFString'
value='MyWKI'/>
```

4.2.3.3 The Fields of the SscDispatcher Prototype

```
<field accessType='inputOutput' name='parentInstance' type='SFString'
value="[sscInstance]"/>
<field accessType='inputOutput' name='uocName' type='SFString'
value="[uocName]"/>
<field accessType='inputOutput' name='wellKnownUoc' type='SFString'
value="[wellKnownUoc]"/>
<field accessType='inputOutput' name='dispatcherName' type='SFString' value=""/>
<field accessType='inputOutput' name='mandatorySscExtensionsIn'
type='MFNode'></field>
<field accessType='inputOutput' name='optionalSscExtensionsIn'
type='MFNode'></field>
<field accessType='inputOutput' name='commParam' type='SFNode' value="NULL"/>
<field accessType='inputOutput' name='basicallyInitialized' type='SFBool'
value='false'/>
<field accessType='outputOnly' name='initialized' type='SFNode'/>
<field accessType='inputOnly' name='disable' type='SFTime'/>
<field accessType='inputOutput' name='traceLevel' type='SFInt32' value="1"/>
<field accessType='inputOutput' name='moduleIx' type='SFInt32' value='-1'/>
<field accessType='inputOutput' name='dispatcherStub' type='SFNode'
value='NULL'/>
<field accessType='inputOutput' name='uboLoader' type='SFNode' value='NULL'/>
```

Explain!!!: "wellKnownUoc", "uocName", (basic) initialization, "dispatcherStub", "uboLoader"

4.2.3.4 The Fields of the SmsDispatcherStub Prototype

The SMS Dispatcher Stub is described at 123_SscDispatcher.

```
4.2.3.5 The Fields of the SscUboLoader Prototype
------------------------------------------------
<field accessType='inputOutput' name='parentInstance' type='SFString'
value="[sscInstance]"/>
<field accessType='inputOutput' name='wellKnownUoc' type='SFString'
value="[wellKnownUoc]"/>
<field accessType='inputOutput' name='dispatcherName' type='SFString' value=""/>
<field accessType='inputOutput' name='commParam' type='SFNode' value="NULL"/>
<field accessType='inputOutput' name='basicallyInitialized' type='SFBool'
value='false'/>
<field accessType='outputOnly' name='initialized' type='SFNode'/>
<field accessType='inputOnly' name='disable' type='SFTime'/>
<field accessType='inputOnly' name='startActivity' type='SFBool'/>
<field accessType='inputOnly' name='initializeStateAndStartActivity'
type='SFBool'/>
<field accessType='inputOutput' name='iAmActive' type='SFBool' value='false'/>
<field accessType='inputOutput' name='activityStarted' type='SFNode'
value="NULL"/>
<field accessType='inputOutput' name='traceLevelControl' type='SFInt32'
value="1"/>
<field accessType='inputOutput' name='traceLevelComm' type='SFInt32' value="1"/>
<field accessType='initializeOnly' name='typicalDynElemSpace' type='SFInt32'
value="50"/>
<field accessType='outputOnly' name='dynElemSpace' type='SFInt32'/>
<field accessType='inputOutput' name='uboConf' type='SFNode' value="NULL"/>
<field accessType='inputOnly' name='registerInitialObjectTypes' type='SFBool'/>
<field accessType='inputOutput' name='registeredObjectTypes' type='MFString'
value=""/>
```
Explain!!!: ???

5 Additional Info

5.1 The Common Parameters

The term "common parameters" refers to a <Script> node, which is a part of the SscBase prototype and which holds in its fields a lot of common data, that is accessible by all parts of the scene (especially by all parts of the SRR/SMUOS Framework).

During initialization, the SFNode event "commParam" is forwarded from the Simple Scene Controller to the Module Coordinators and to the astral objects.

5.1.1 Core Information

These four fields are set by the SscCore prototype.

enabled(SFBool)	remains true, as long as the SSC Base is enabled
initialized(SFBool)	set by SSC Core after initialization of the SSC. A value "true" indicates successful initialization of the SSC Base and of all mandatory SSC Extensions. Now the field "extensions" contains pointers to all successfully initialized SSC Extensions and the SSC Base can send an Access Request to the Central Controller
iAmActive(SFBool)	set by SSC Core after activation. A value "true" indicates successful activation and initialization of the SSC Base (now all common parameters are valid). Many of the functions of the SSC Base are only available, when iAmActive = "true"
extensions(MFNode)	Pointers to all successfully initialized extensions of the Common Parameters (please refer to chapter 4.2 **and ??????**)

5.1.2 Static Information

The common parameters contain besides others some static information, which does not change during the lifetime of the scene.

sscBaseVersion(SFFloat).........	version of the SSC Base prototype
neededBrowsers(MFString)........	Browser.name of all supported browsers
neededVersions(MFFloat).........	needed version of all supported browsers
blaxxunIx(SFInt32)..............	index of blaxxunCC3D within "neededBrowsers" and "neededVersions"
octagaIx(SFInt32)...............	index of Octaga Player within "neededBrowsers" and "neededVersions"
instantIx(SFInt32)..............	index of Avalon within "neededBrowsers" and "neededVersions"
errorStrings(MFString)..........	errorStrings[errorNo] is the last occured error

5.1.3 Networking Parameters

These parameters are accessible as global SSC parameters

maxRTT(SFFloat).................	network parameter: maximum roundtrip time
randomFactor(SFFloat)...........	another network parameter

5.1.4 Status Information

Following common parameters indicate basic status about the SSC Base.

browserIx(SFInt32)..............	set during initialization, either blaxxunIx or octagaIx or instantIx
browserVersion(SFFloat).........	set during initialization, actual version of used browser
useConnection(SFString).........	DEF name of the <NetConnection> (BS Contact), needed by the network sensors
connection(SFNode)..............	Pointer to the <Connection> (Octaga), needed by the network sensors
errorNo(SFInt32)................	last occured error. Initialized to 0. The text of the last occured error can be accessed by errorStrings[errorNo]
multiuser(SFBool)...............	true, if multiuser mode is actually chosen

5.1.5 Local Scene Information

The local scene information is derived from the commState. So the source is the reception of a "communication state" (commState) - see below.

ownSessionId(SFInt32)...........	the sessionId, which was delivered by the <BSCollaborate> node, "-1" in case of single-user-mode
attachedModules(SFString).......	a comma-separated list of all modules that are currently attached in this scene instance
activeModules(SFString).........	a comma-separated list of all modules that are currently active in this scene instance
attachedModulesBoolArr(MFInt32).	a list of quasi-boolean values ("0" or "1"), i.e., if attachedModulesBoolArr[moduleIx] then this module is attached in this scene instance
activatedModulesBoolArr(MFInt32).	a list of quasi-boolean values ("0" or "1"), i.e., if activatedModuleBoolArr[moduleIx] then this module is active in this scene instance
mocRolesBoolArr(MFInt32)........	a list of quasi-boolean values ("0" or "1"), i.e., if mocRolesBoolArr[moduleIx] then this module has its MOC role in this scene instance
ownMocRoles(SFString)...........	a comma-separated list of module names that have their MOC role in this scene instance

5.1.6 Communication State

The communication state is stored on the collaboration server and distributed to all scene instances as a rather complicated MFString state value, whose reception should be performed as an atomic action.

The communication state is stored in the common parameters in two sections, first the "Master Communication State", i.e. the parameters, which contain all the information, and which can be transformed 1:1 to the MFString value; second, the "Communication State Helpers", which contain redundant information, which is derived from the master communication state in each scene instance separately

5.1.6.1 Master Communication State

commStateSequence(SFInt32)........	sequence number of the commState message
sessionIds(MFInt32)...............	ordered list of all sessionIds of all scene instances
avaConBinding(MFString)..........	has same dimension as sessionIds, each element contains the extObjId of the bound avatar container of that scene instance or an empty str.
controllerIx(SFInt32).............	sessionIds[controllerIx] has the central controller role
ownIx(SFInt32)....................	sessionIds[ownIx] is the same as ownSessionId
moduleActivity(MFInt32)..........	module activity matrix
modulesAttachedState(MFInt32).....	module attachment matrix
unregisteredModules(MFInt32)......	all free moduleIxs
registeredModules(MFString).......	module names of all registered modules (can be empty string - gaps are possible)
implicitelyRegistered(MFInt32)....	"1", if module is implicitely registered

5.1.6.2 Communication State Helpers

iAmController(SFBool)............	if "true", then this scene instance has the central controller role
attachedModulesStrings(MFString)..	has same dimension as sessionIds, each element contains a comma-separated list of attached modules of the addressed scene instance
activatedModulesStrings(MFString).	has same dimension as sessionIds, each element contains a comma-separated list of activated modules of the addressed scene instance
mocRolesStrings(MFString).........	has same dimension as sessionIds, each element contains a comma-separated list of MOC roles of the addressed scene instance

5.1.7 List of sessionIds of Sessions that Left During the Last 20 Seconds

With the help of the following parameters, the SSC Base maintains a list of sessionIds of sessions that have left during the last 20 seconds. This list is used by the MIDAS Base to perform a correct handling of assignment of OBCO roles.

leftSessionIds(MFInt32)..........	List of sessionIds that left during the last 20 seconds
leftSessionIdsTimestamps(MFTime)..	has the same dimension as leftSessionIds. Keeps the associated timer values

5.1.8 SSC Dispatchers

sscDispatchersGroup(MFNode).......	**to be described**
modulesOffset(SFInt32)............	**to be described**

5.1.9 Address of the SSC Base

```
sscBase(SFNode)..................Address of the SSC Base Client <Script>.
                                 If a module coordinator [extension module] or
                                 a MIDAS Object wants to send an event to the
                                 SSC Base Client, then it can address the
                                 input fields of the SSC Base Client via
                                 [modParam.]commParam.sscBase.<fieldName>
```

3.1.10 Distribute Commands to all Avatar Containers
----

setPosOriReporting(SFString)......to be described
joinedPosition(SFVec3f)...........to be described
joinedOrientation(SFRotation).....to be described
joinAvatar(SFInt32)...............to be described

3.1.11 Tracer Support

The basic support of the Tracer is implemented directly in the common parameters.

Where we have to distinguish
- the "classic" tracer (deprecated but still supported)
- the SMS Tracer

The "classic tracer" can be directly accessed via the common parameters, without additional necessities, on the other hand it supports only one trace level, which is common to all parts of the scene.

The "SMS Tracer" is more sophisticated. It is implemented in the X3D prototype SmsTracer.x3d, but it has to be additionally supported by the common parameters.

traceLevel(SFInt32)...............to be described
traceLevelSscBase(MFInt32)   .....to be described
traceLevelCommControl(MFInt32)....to be described
traceLevelModules(MFString).......to be described
traceLevelObjects(MFString).......to be described
command(SFNode)...................to be described
command.ErrLog(MFString)..........to be described
command.InfoLog(MFString).........to be described
command.Debug(MFString)...........to be described
command.ErrLogNew(MFString).......to be described
command.InfoLogNew(MFString)......to be described
command.DebugNew(MFString)........to be described
command.ErrLogNew2(MFString)......to be described
command.traceOutput(MFString).....to be described
command.traceBuffer(MFString).....to be described