*Appendix B*

# DOCUMENTATION OF THE SLX-HLA-INTERFACE

## B.1 Function Calls by Service Group

This section contains the function calls provided by the SLX-HLA-Interface software to the SLX model developer. The functions are grouped according to the management groups they belong to. The notation used in this section follows the conventions from the RTI 1.3 Programmer's Guide which is distributed with the DMSO RTI software [DOD98a], [DOD00b].

In order to give a high level access to the HLA functionality without placing the burden of detailed interface programming to the SLX developer, some functions of the HLA interface specification are implemented by aggregated or simplified SLX functions.

Please refer also to the original DMSO HLA interface specification for further details about the original RTI and federate ambassador functions.

# B.1.1 FEDERATION MANAGEMENT

## B.1.1.01 RTI_DestroyFederationExecution()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_DestroyFederationExecution tries to close down a federation execution.

### SYNOPSIS

```
Procedure RTI_DestroyFederationExecution(
          string(*) FederationName)
          returning boolean dll="rtislx10"; //resp. "slxrti13"
```

### ARGUMENTS

*FederationName*

String specifying the name of the federation execution to close down

### DESCRIPTION

RTI_DestroyFederationExecution can be used to close down a federation execution (in the RTI versions 1.0 and 1.3 from DMSO this includes the termination of the FedExec process). This function fails if other federates are still member of the federation execution.

There are no restrictions on who may destroy the federation. A federate need not be the creator of the FedExec or even have been a member of the FedExec. In the SLX-HLA-Interface it is necessary, though, to have successfully completed a call to RTI_Init. Once this has been performed, RTI_DestroyFederationExecution can be used to destroy any federation execution. It should be noted, though, that upon destroying the *same* federation execution that was specified in RTI_Init (which is usually the case), the SLX-HLA-Interface frees all memory associated with the RTI- and federate ambassador object, thus no subsequent RTI call can be made.

A more convenient way which combines the tasks of resigning from a federation and closing it down is provided by RTI_Terminate.

### RETURN VALUES

The return value is TRUE if the federation execution was closed down successfully and FALSE, if an unexpected error occurred. This does not include the case, that other federates were still member of the federation execution.

### SEE ALSO

*RTI_ResignFederationExecution, RTI_Terminate*

## B.1.1.02 RTI_FederateRestoreComplete()

**RTI 1.3**

### ABSTRACT

RTI_FederateRestoreComplete notifies the RTI that the federate has successfully completed an attempted federate restoration. The semantics of federation save and restore have changed from RTI 1.0 to RTI 1.3 This service was named RTI_RestoreAchieved in RTI 1.0 and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_FederateRestoreComplete()
          returning boolean dll="slxrti13";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_FederateRestoreComplete notifies the RTI that the federate has successfully completed an attempted federate restoration. The federate may not resume normal operation until it receives a federationRestored or federationNotRestored callback to indicate that the federation wide restoration attempt has concluded. The SLX-HLA-Interface for RTI 1.3 provides this information in the attribute RestoreCompleted which is part of the SLX_StateObject. If federationRestored has been called, the attribute RestoreCompleted is switched to “Success”. If at least one federate has not successfully completed its restoration (i.e., if federationNotRestored is called), the attribute RestoreCompleted is switched to “Failed”.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_RestoreNotAchieved, RTI_RequestRestore*

## B.1.1.03 RTI_FederateRestoreNotComplete()

**RTI 1.3**

### ABSTRACT

RTI_FederateRestoreNotComplete notifies the RTI that the federate has completed an attempted federate restoration, but without success. The semantics of federation save and restore have changed from RTI 1.0 to RTI 1.3 This service was named RTI_RestoreNotAchieved in RTI 1.0 and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_FederateRestoreNotComplete()
          returning boolean dll="slxrti13";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_FederateRestoreNotComplete notifies the RTI that the federate has unsuccessfully completed an attempted federate restoration. The federate may not resume normal operation until it receives a federationRestored or federationNotRestored callback to indicate that the federation wide restoration attempt has concluded. The SLX-HLA-Interface for RTI 1.3 provides this information in the attribute RestoreCompleted which is part of the SLX_StateObject.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_RestoreNotAchieved, RTI_FederateRestoreComplete RTI_RequestRestore*

## B.1.1.04 RTI_FederateSaveAchieved()

**RTI 1.0**

### ABSTRACT

RTI_FederateSaveAchieved notifies the RTI that the federate has successfully completed a requested save. This service is named RTI_FederateSaveComplete in RTI 1.3 and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_FederateSaveAchieved()
          returning boolean dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_FederateSaveAchieved notifies the RTI that the federate has successfully completed an attempted federate save. The RTI_FederateSaveAchieved call blocks until all other federates and the RTI have completed their saves (i.e., successfully or unsuccessfully).

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI. This indicates that the RTI save and all federate saves have completed and that the federate should resume advancing of the federate's logical time.

### SEE ALSO

*RTI_FederateSaveBegun, RTI_FederateSaveNotAchieved*

## B.1.1.05 RTI_FederateSaveComplete()

**RTI 1.3**

### ABSTRACT

RTI_FederateSaveComplete notifies the federation that the federate has successfully completed a requested save. This service is named RTI_FederateSaveAchieved in RTI 1.0 and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_FederateSaveComplete()
         returning boolean dll="slxrti13";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_FederateSaveComplete notifies the RTI that the federate has successfully completed an attempted federate save. The federate may not resume normal operation until it receives a federationSaved or federationNotSaved callback to its federate ambassador. The federate ambassador implemented in the SLX-HLA-Interface does currently not implement these callbacks, i.e., they can not be sensed by the federate.

### MIGRATION NOTE

Note that under RTI 1.0 semantics, the federate could continue operation immediately after notifying the RTI of local save success/failure. In RTI 1.3, the federate must wait until the entire federation has completed the (attempted) save before it may continue.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI. The federate should not advance with its advancement of logical time before the entire federation has completed the save.

### SEE ALSO

*RTI_FederateSaveBegun, RTI_FederateSaveNotAchieved*

## B.1.1.06 RTI_FederateSaveBegun()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_FederateSaveBegun informs the RTI that the calling federate has begun saving its internal state.

### SYNOPSIS

```
procedure RTI_FederateSaveBegun ()
         returning boolean dll="slxrti10"; //resp. "slxrti13"
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_FederateSaveBegun notifies the RTI that the federate has begun saving its internal state as per an initiateFederateSave request.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI. This indicates that the federate can proceed to save its state.

### RELEASE NOTES

*RTI 1.3*

- The 1.3 implementation of the RTI does not utilize the information provided by this call in any way; in fact, the RTI_FederateSaveBegun call may be omitted entirely. However, federates should invoke this service to assure compliance with all RTI implementations.

### SEE ALSO

*RTI_FederateSaveAchieved*

## B.1.1.07 RTI_FederateSaveNotAchieved()

**RTI 1.0**

### ABSTRACT

RTI_FederateSaveNotAchieved informs the RTI that the federate was unsuccessful in its attempt to save, but has completed the attempt. This service is named RTI_FederateSaveNotComplete in RTI 1.3 and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_FederateSaveNotAchieved()
         returning boolean dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_FederateSaveNotAchieved notifies the RTI that the federate was unable to save its internal state as requested. The RTI_FederateSaveNotAchieved call blocks until all other federates and the RTI have completed their saves (i.e., successfully or unsuccessfully). The RTI makes an entry in the federate's log file that the save failed and proceeds as if the save was successful (i.e., the internal state of the RTI will still be saved).

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI and that the federate should resume advancement of the federate's logical time.

### SEE ALSO

*RTI_FederateSaveBegun, RTI_FederateSaveAchieved*

## B.1.1.08 RTI_FederateSaveNotComplete()

**RTI 1.3**

### ABSTRACT

RTI_FederateSaveNotComplete informs the RTI that the federate was unsuccessful in its attempt to save, but has completed the attempt. This service is named RTI_FederateSaveNotAchieved in RTI 1.0 and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_FederateSaveNotComplete()
          returning boolean dll="slxrti13";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_FederateSaveNotComplete notifies the RTI that the federate was unable to save its internal state as requested. The federate may not resume normal operation until it receives a federationNotSaved callback to its federate ambassador.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_FederateSaveBegun, RTI_FederateSaveAchieved*

## B.1.1.09 RTI_Init()

**RTI 1.0** **▲RTI 1.3**

### ABSTRACT

This function has to be called as the very first function called by any SLX federate.

### SYNOPSIS

```
procedure RTI_Init( string(*) FederateName,
                    string(*) FederationName,
                    string(*) FedFileName,        ← RTI 1.3 Only
                     boolean constrained,
                     boolean regulating,
                     double Lookahead,
                     boolean SingleStepped,
                     boolean FedAmbMode,
                     boolean SaveMode,
                     boolean InteractiveStartup,
                     pointer(SLX_StateObject) StatePtr)
          returning double dll="rtislx10";   // resp. "rtislx13"
```

### ARGUMENTS

*FederateName*

String specifying the name of the Federate

*FederationName*

String specifying the name of the FedExec to create

*FedFileName* **(RTI 1.3 Only)**

String specifying the name of the fed-file to use (and the complete path, if the fed-file is not located in the RTI_CONFIG directory)

*constrained*

Boolean value specifying if the federate is supposed to be constrained in its logical time advances by other federates

*regulating*

Boolean value specifying if the federate is supposed to be regulating towards the logical time advances of other federates

*Lookahead*

Double value specifying the lookahead value of the federate

*SingleStepped*

Boolean value specifying if the SLX-HLA-Interface is supposed to run in a single stepped mode (e.g., for debugging purposes) or not. If SingleStepped is TRUE, each time advance will be prompted by popping up a message box.

*FedAmbMode*

Boolean value specifying if the SLX-HLA-Interface is supposed to prompt incoming calls to the federate ambassador (callbacks from the RTI) with a message box or not. This switch is also implemented for debugging purposes.

*SaveMode*

The parameter SaveMode provides a possibility to manually control the passing of „unsafe“ code passages, which are due to errors or instabilities in the RTI software. There are cases during the initialization phase, where accesses to the FedExec process lead to program crashes if the FedExec process is not running stable yet. If the SaveMode parameter is set TRUE, the user will be provided with the possibility to manually control the startup phase of the FedExec process.

*InteractiveStartup*

This parameter specifies if the user chooses the interactive startup mode for SLX federates or not. Using the interactive startup mode the user will be shown a message box after the initialization of the SLX federate has been completed. Only after pressing OK in this message box, the function RTI_Init will return control to SLX. The advantage of this approach is, that while the message box is shown, the HLA interface internally issues calls to the RTI_Tick method. This way other federates can join the federation and a manually coordinate startup of the entire federation becomes possible. As an alternative, the user can chose to use an non-interactive startup mode, in which RTI_Init directly returns control to the SLX model after it has completed its tasks. In this scenarios there is no guarantee that all desired federates have joined the federation. See Section 4.1.1 for alternate means of federation initialization and startup.

*StatePtr*

StatePtr should be a pointer to an SLX object instance of the class “SLX_StateObject“ defined in the SLX modules SLXRTI10.SLX and SLXRTI13.SLX, respectively. The data structure “behind” the pointer is one of the main means for communicating information back to the SLX federate. Each time the federate ambassador receives information which does not relate directly to the objects and interactions modeled by SLX, it will store the corresponding information in the SLX_StateObject. The structure of the SLX_StateObject is discussed in a separate section.

## DESCRIPTION

RTI_Init is an aggregation of several RTI ambassador methods which usually need to be called by every federate once for initialization purposes. RTI_Init performs the following tasks:

- Creating instances of the two ambassador objects (RTI ambassador and federate ambassador)
- Creating a federation execution with the name passed in the parameter FederationName (by calling the RTI ambassador function createFederationExecution)
- Joining the federation execution as the federate with the name passed in the parameter FederateName (by calling the RTI ambassador function joinFederationExecution). This associates the federate ambassador with this specific federation execution.
- Setting the initial time management parameters depending on the values specified by the parameters constrained, regulating, and lookahead. These values can be changed later on using the function RTI_SetTimeParameters (see there for a detailed explanation of the parameters).

## RETURN VALUES

The return value of this function is the initial federate time as returned by the RTI ambassador method requestFederateTime, if the initialization was successfully carried out. The return value is -1, if an error occurred during the initialization. It is the user’s task to stop the execution of the SLX models if -1 is returned.

## WINDOWS® NT NOTES

The function RTI_Init uses the RTI ambassador method createFederationExecution to start the FedExec process. To be able to do so, a proper RTI installation is required. The location of the

executable that is forked to start the FedExec is "`%RTI_HOME%\bin\win32\fedex.exe`".

### SEE ALSO

*RTI_Terminate, RTI_ResignFederationExecution, RTI_DestroyFederationExecution*

## B.1.1.10 RTI_PauseAchieved()

**RTI 1.0**

### ABSTRACT

RTI_PauseAchieved informs the federation that the federate has suspended execution. The pause/resume capability of RTI 1.0 has been replaced by the more general "synchronization point" capability in RTI 1.3.

### SYNOPSIS

```
procedure RTI_PauseAchieved(string(*) PauseLabel)
                returning boolean dll="slxrti10";
```

### ARGUMENTS

*PauseLabel*

String which can be used for specifying a federation wide identification of the pause request.

### DESCRIPTION

By calling the function RTI_PauseAchieved a federate informs the RTI that the federate has suspended execution as per most recent pause request. The federate should remain suspended until instructed to resume (see also RTI_RequestResume).

### RETURN VALUES

A return value of TRUE indicates that the RTI has been notified of the federate's successful suspension of execution.

### SEE ALSO

*RTI_RequestResume, RTI_PauseAchieved*

## B.1.1.11 RTI_RegisterFederationSynchronizationPoint()

**RTI 1.3**

### ABSTRACT

This service is used to initiate the establishment of a named checkpoint that serves to synchronize some or all federates according to federation defined semantics. The synchronization mechanism is a generalization of the pause/resume mechanism in RTI 1.0.

### SYNOPSIS

```
procedure RTI_RegisterFederationSynchronizationPoint(
         string(*) SynchronizationLabel,
         string(*) UserTag)
         returning boolean dll="slxrti13";
```

### ARGUMENTS

*SynchronizationLabel*

A string used to uniquely identify the synchronization point

*UserTag*

An application-defined string passed to remote federates when the synchronization point is announced.

### DESCRIPTION

Synchronization points provide a mechanism for federates to schedule checkpoints with federation-defined semantics, while relying on the RTI to perform the bookkeeping associated with determining

when the checkpoint is achieved.

RTI_RegisterFederationSynchronizationPoint schedules a synchronization point for all federates, including federates that join the federation while the synchronization is in progress. RTI_RegisterFederationSynchronizationPoint provides a synchronous answer about the success or failure of the registration of the synchronization point. The SLX-HLA-Interface therefore internally ticks the RTI until either one of the two callbacks “synchronizationPointRegistrationSucceeded” or “synchronizationPointRegistrationFailed” is called from the RTI.

Federates are informed about the request for a synchronization point via the federate ambassador callback announceSynchronizationPoint. The SLX-HLA-Interface informs the model about such a request by setting the switch *SynchronizationPointAnnounced* which is part of the SLX_StateObject to TRUE. The label as well as the tag associated with the announcement can be obtained via the SLX_StateObject parameters *SynchronizationLabel* and *SynchronizationTag*, respectively. A federate should respond to such a request by calling RTI_SynchronizationPointAchieved.

**RETURN VALUES**

The return value is TRUE if the RTI responded via synchronizationPointRegistrationSucceeded and FALSE if the RTI responded via synchronizationPointRegistrationFailed or if an error occurred.

**SEE ALSO**

*RTI_SynchronizationPointAchieved*

### B.1.1.12 RTI_RequestFederationRestore()

**RTI 1.3**

**ABSTRACT**

RTI_RequestFederationRestore requests that all federates re-initialize themselves based on a previous, labeled save. The semantics of federation save and restore have changed from RTI 1.0 to RTI 1.3 This service was named RTI_RequestRestore in RTI 1.0 and is discussed in a separate section.

**SYNOPSIS**

```
procedure RTI_RequestFederationRestore(string(*) RestoreLabel)
          returning boolean dll="slxrti13";
```

**ARGUMENTS**

*RestoreLabel*

String parameter which can be used to identify an associated save label

**DESCRIPTION**

The function RTI_RequestRestore can be used to request that all federates reinitialize themselves based on a labeled save state. The parameter RestoreLabel should be used for identifying the corresponding save state. A call to this function will be promoted to all other participating federates by a call to the function initiateRestore at their federate ambassador. The federate ambassador implemented in the SLX-HLA-Interface notifies the SLX model about the occurrence of such a callback by modifying the appropriate attributes in the SLX_ StateObject. The attribute RestoreRequested will be switched to TRUE and the attribute RestoreLabel will contain the corresponding label. In the current implementation of the SLX-HLA-Interface for RTI 1.3 the callbacks requestFederationRestoreFailed, requestFederationRestoreSucceeded and federationRestoreBegun, which may be invoked as a result of calling RTI_RequestFederationRestore will be ignored.

**RETURN VALUES**

The return value is TRUE if the call was successfully passed to the RTI.

**SEE ALSO**

*RTI_RequestFederationSave, RTI_RestoreAchieved*

### B.1.1.13 RTI_RequestFederationSave()

**RTI 1.0** **▲RTI 1.3**

#### ABSTRACT

RTI_RequestFederationSave requests that the federation save its state at a specified logical time. The sematics of federation save and restore have changed from RTI 1.0 to RTI 1.3.

#### SYNOPSIS

```
procedure RTI_RequestFederationSave(  string(*)SaveLabel,
                                      double TimeStamp)
          returning boolean dll="slxrti10";  //resp. "slxrti13"
```

#### ARGUMENTS

*SaveLabel*

String which can be used for a federation wide identification of the save request

*TimeStamp*

Logical time at which the save is supposed to take place

#### DESCRIPTION

The function RTI_RequestFederationSave can be used to request that the federation save its state at a specified logical time. The parameter SaveLabel can be used for giving a textual description of the reason of the save request. It also identifies the request from possible other requests that might be issued. The parameter TimeStamp specifies the logical time at which the save is supposed to take place. A call to this function will result in an invocation of the callback method initiateFederateSave at the federate ambassador of all other participating federates. The federate ambassador implemented in the SLX-HLA-Interface will pass the information about such a callback to the SLX model via an entry in the SLX_StateObject. The information will be contained in the attribute SaveRequested (set to TRUE) and in the attribute SaveLabel which contains the textual description of the save request.

#### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

#### RELEASE NOTES

*RTI 1.0*

- The services invoked by a federate to report success or failure of a save of federate-managed state are named RTI_FederateSaveAchieved and RTI_FederateSaveNotAchieved, respectively.

*RTI 1.3*

- The services invoked by a federate to report success or failure of a save of federate-managed state are named RTI_FederateSaveComplete and RTI_FederateSaveNotComplete, respectively.

#### SEE ALSO

*RTI_FederationSaveASAP*

### B.1.1.14 RTI_RequestFederationSaveASAP()

**RTI 1.0** **▲RTI 1.3**

#### ABSTRACT

RTI_RequestFederationSave requests that the federation save its state as soon as possible. The sematics of federation save and restore have changed from RTI 1.0 to RTI 1.3.

#### SYNOPSIS

```
procedure RTI_RequestFederationSaveASAP(string(*)SaveLabel)
          returning boolean dll="slxrti10";  //resp. "slxrti13"
```

#### ARGUMENTS

*SaveLabel*

String which can be used for a federation wide identification of the save request.

### DESCRIPTION

The function RTI_RequestFederationSaveASAP is a variation of the function RTI_Request-FederationSave. The only difference relates to the logical time at which the save is supposed to take place. While RTI_RequestFederationSave requires the user to specify a specific time stamp for the save event, RTI_RequestFederationSaveASAP only requests that the save should take place as soon as possible (ASAP).

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### RELEASE NOTES

Please refer to RTI_RequestFederationSave.

### SEE ALSO

*RTI_RequestFederationSave*

## B.1.1.15 RTI_RequestPause()

**RTI 1.0**

### ABSTRACT

RTI_RequestPause requests that all federates in the federation suspend execution as soon as possible. The pause/resume capability of RTI 1.0 has been replaced by the more general “synchronization point” functionality of RTI 1.3.

### SYNOPSIS

```
procedure RTI_RequestPause(string(*) PauseLabel)
            returning boolean dll="rtislx10";
```

### ARGUMENTS

*PauseLabel*

String which can be used for specifying a federation wide identification of the pause request. It can also be used to give a textual description for the reason of the request.

### DESCRIPTION

RTI_RequestPause can be used to request that all federates in the federation execution suspend execution as soon as possible. A call to this function will result in invocations of the initiatePause method at the federate ambassador of all other participating federates. The federate ambassador implemented in the SLX-HLA-Interface notifies the SLX model about the occurrence of such a callback by writing the corresponding information to the SLX_StateObject. The attribute PauseRequested will be changed to TRUE, indicating the existence of a pause request. Additionally the attribute PauseLabel will contain the label passed along with the request.

### RETURN VALUES

The return value of TRUE indicates that the federate has successfully communicated its desire to suspend federation execution.

### SEE ALSO

*RTI_RequestResume, RTI_PauseAchieved*

## B.1.1.16 RTI_RequestRestore()

**RTI 1.0**

### ABSTRACT

RTI_RequestRestore requests that all federates re-initialize themselves based on a previous, labeled save. The semantics of federation save and restore have changed from RTI 1.0 to RTI 1.3. This service is named RTI_RequestFederationRestore in RTI 1.3 and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_RequestRestore(string(*) RestoreLabel)
         returning boolean dll="slxrti10";
```

### ARGUMENTS

*RestoreLabel*

String parameter which can be used to identify an associated save label

### DESCRIPTION

The function RTI_RequestRestore can be used to request that all federates reinitialize themselves based on a labeled save state. The parameter RestoreLabel should be used for identifying the corresponding save state. A call to this function will be promoted to all other participating federates by a call to the function initiateRestore at their federate ambassador. The federate ambassador implemented in the SLX-HLA-Interface notifies the SLX model about the occurrence of such a callback by modifying the appropriate attributes in the SLX_StateObject. The attribute RestoreRequested will be switched to TRUE and the attribute RestoreLabel will contain the corresponding label.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_RequestFederationSave, RTI_RestoreAchieved*

## B.1.1.17 RTI_RequestResume()

**RTI 1.0**

### ABSTRACT

RTI_RequestResume requests that a paused federation resume execution as soon as possible. The pause/resume capability of RTI 1.0 has been replaced by the more general “synchronization point” functionality of RTI 1.3.

### SYNOPSIS

```
procedure RTI_RequestResume()
         returning boolean dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_RequestResume can be used by a federate to request that a paused federation resume execution as soon as possible. A call to this function will result in an initiateResume callback to the federate ambassador of all other federates. The federate ambassador implemented in SLX-HLA-Interface informs the SLX model about the occurrence of such a callback by changing the corresponding attribute of the SLX_StateObject, i.e., the attribute ResumeRequested will be set to TRUE. The federate requesting the resumption need not be the same federate that requested the pause.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_ResumeAchieved*

## B.1.1.18 RTI_ResignFederationExecution()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_ResignFederationExecution can be used to terminate the federate's participation in a federation.

### SYNOPSIS

```
type ResignAction enum  RELEASE_ATTRIBUTES,DELETE_OBJECTS, DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES,
                        NO_ACTION;

procedure RTI_ResignFederationExecution(ResignAction Action)
          returning boolean dll="rtislx10"; //resp. "rtislx13"
```

### ARGUMENTS

*ResignAction*

Enumerated type specifying which action shall be taken upon resigning from the federation execution

### DESCRIPTION

RTI_ResignFederationExecution can be used to resign from a federation execution. It informs the federation executive that the federate no longer wishes to participate in the federation execution. This function can be used as an alternative to RTI_Terminate. RTI_ResignFederationExecution provides the possibility to determine which resign action regarding any object instances owned by the federate should be taken. Possible values for Action are

- RELEASE_ATTRIBUTES
- DELETE_OBJECTS
- DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES
- NO_ACTION.

### RETURN VALUES

The return value is TRUE if the federate successfully resigned from the federation execution.

### SEE ALSO

*RTI_DestroyFederationExecution, RTI_Terminate*

## B.1.1.19 RTI_RestoreAchieved()

**RTI 1.0**

### ABSTRACT

RTI_RestoreAchieved notifies the RTI that the federate has successfully completed an attempted federate restoration. The semantics of federation save and restore have changed from RTI 1.0 to RTI 1.3 This service is named RTI_FederateRestoreComplete in RTI 1.3 and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_RestoreAchieved()
          returning boolean dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_RestoreAchieved notifies the RTI that the federate has successfully completed an attempted federate restoration. The service will block the federate until all other federates have restored or failed to restore their states. The RTI then tries to restore its internal state and returns control to the federates.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_RestoreNotAchieved, RTI_RequestRestore*

## B.1.1.20 RTI_RestoreNotAchieved()

**RTI 1.0**

### ABSTRACT

RTI_RestoreNotAchieved notifies the RTI that the federate has completed an attempted federate restoration, but without success.

### SYNOPSIS

```
procedure RTI_RestoreAchieved()
         returning boolean dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_RestoreNotAchieved notifies the RTI that the federate was unsuccessful in restoring its internal state. The service will block the federate until all other federates have restored or failed to restore their states. The RTI then tries to restore its internal state and returns control to the federates.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_RestoreAchieved*

## B.1.1.21 RTI_ResumeAchieved()

**RTI 1.0**

### ABSTRACT

RTI_ResumeAchieved informs the RTI that the calling federate has resumed execution. The pause/resume capability of RTI 1.0 has been replaced by the more general “synchronization point” functionality of RTI 1.3.

### SYNOPSIS

```
procedure RTI_ResumeAchieved()
         returning boolean dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

The function RTI_RequestResume can be used to notify the RTI that the federate has resumed execution as per an initiate resume request.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_RequestResume*

## B.1.1.22 RTI_SynchronizationPointAchieved()

**RTI 1.3**

### ABSTRACT

This service informs the federation that the federate has met the federation-desired criteria associated with a synchronization point that has been announced to the federate. The synchronization mechanism is a generalization of the pause/resume mechanism in RTI 1.0.

### SYNOPSIS

```
procedure RTI_SynchronizationPointAchieved(
          string(*) SynchronizationLabel)
          returning boolean dll="slxrti13";
```

### ARGUMENTS

*SynchronizationLabel*

A string parameter which is the synchronization point identifier that was previously announced to the federate

### DESCRIPTION

Synchronization points provide a mechanism for federates to schedule checkpoints with federation-defined semantics, while relying on the RTI to perform the bookkeeping associated with determining when the checkpoint is achieved.

A federate uses RTI_SynchronizationPointAchieved to indicated that it has met the synchronization criteria associated with some currently outstanding synchronization point. The federate will be informed by the RTI when all federates have achieved the synchronization point by invoking “synchronizationPointAchieved”. The SLX-HLA-Interface informs the model about this callback by setting the SLX_StateObject parameter *FederationSynchronized* to TRUE.

Depending on the federation semantics associated with the synchronization point the federate may continue execution when RTI_SynchronizationPointAchieved returns or explicitly wait until all other federates have synchronized. At a minimum, all federates must continue to invoke tick() so that all internal RTI communications may be served. SLX models can achieve this by either calling RTI_Tick() or by implicitly calling tick() when a time advancement function is used.

Special care has to be taken for time constrained SLX federates, since the time advancement functions of the SLX-HLA-Interface work synchronously. If two time regulating and time constrained federates (say A and B) are to be synchronized, deadlock situations may occur. Consider the following scenario: Federate A requests and achieves a synchronization point. Federate A waits for federate B to achieve this synchronization point, too, by calling RTI_Tick and not advancing its logical time. If federate B is stuck in a time advancement function at that time, the federation deadlocks, because B is still waiting for a time advance grant and cannot respond to the announcement of the synchronization point. There is no easy way to circumvent this problem at this time other then for federate A to advance in time until federate B receives a timeAdvanceGrant.

### RETURN VALUES

The return value is TRUE if the call was successfully passed to the RTI.

### SEE ALSO

*RTI_RegisterFederationSynchronizationPoint, RTI_NextEventRequest, RTI_TimeAdvanceRequest*

## B.1.1.23 RTI_Terminate()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_Terminate can be used to combine the tasks of resigning from a federation execution and trying to close down the FedExec process that the federate has been a member of.

### SYNOPSIS

```
procedure RTI_Terminate()
          returning boolean dll="rtislx10";  //resp. "rtislx13"
```

## ARGUMENTS

None.

## DESCRIPTION

RTI_Terminate combines the process of resigning from a federation execution and trying to close it down. It can be used as an alternative to RTI_ResignFederationExecution and RTI_Destroy-FederationExecution. RTI_Terminate releases all object and attributes owned by the federate by passing the DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES option when resigning from the federation.

The function

- calls the RTI ambassador method resignFederationExecution for resigning from the federation execution and
- tries to close down the federation execution by calling the destroyFederationExecution method of the RTI ambassadors. This call fails if other federates are still member of the federation execution.

## RETURN VALUES

The return value is TRUE if the federate has successfully resigned from the federation execution.

## SEE ALSO

*RTI_ResignFederationExecution, RTI_DestroyFederationExecution*

# B.1.2 DECLARATION MANAGEMENT

## B.1.2.01 RTI_PublishInteractionClass()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_PublishInteractionClass conveys the intention of a federate to begin generating interactions of a specified class.

### SYNOPSIS

```
procedure RTI_PublishInteractionClass(
                string(*) InteractionClassName,
                pointer(*) theInteraction)
        returning boolean dll="slxrti10";  //resp. "slxrti13"
```

### ARGUMENTS

*InteractionClassName*

String specifying the name of the interaction class to publish

*theInteraction*

Pointer to an SLX object which will be associated with this interaction class. Subsequent calls of RTI_SendInteraction will automatically access the interaction parameters using *theInteraction*. It is important to note that the pointer has to be properly initialized before calling RTI_Publish-InteractionClass.

### DESCRIPTION

RTI_PublishInteractionClass conveys the intention of the federate to begin generating interactions of a given interaction class (specified by the parameter InteractionClassName). In opposition to the publishing of object classes and the registration of a concrete object instance there is no need to create an instance of an interaction, because interactions are non-persistent, i.e., they can be seen as messages.

In a call to the function RTI_PublishInteractionClass it is necessary to specify an SLX object which represents a certain interaction class (parameter theInteraction). It is extremely important NOT TO DELETE the SLX instance of the interaction during the entire simulation (unless you unpublish the interaction class before deleting it). This is somewhat in contradiction to interactions being non-persistent. The approach of defining a fixed area in SLX storage space which is associated with a specific interaction class is the most convenient way to integrate the idea of interactions/messages into SLX. In subsequent calls to RTI_SendInteraction it is thus just necessary to specify the interaction class to send. The SLX-HLA-Interface internally keeps track of which SLX object is associated with the given class name. Thus it can automatically access the proper storage space and generate the interaction.

RTI_PublishInteractionClass internally investigates the associated SLX class and checks for SLX attributes of this class matching the interaction parameters. If your interaction has a parameter called “dummy“ and you’d like to be able to generate interactions with this parameter, then your associated SLX class must have a parameter named “dummy“, too. The function automatically detects the data types of parameters. It is the users task, though, to define the data types of interactions in accordance to the data types specified in the FOM. Failure to do so may lead to program crashes (in extreme cases); usually it will just produce data garbage when sending or receiving interactions.

### RETURN VALUES

The return value is TRUE if the call was passed successfully to the RTI. The function returns FALSE if an error occurred during the call, esp. if no matches between SLX attribute names and HLA interaction parameter names were detected.

### RELEASE NOTES

*RTI 1.3*

- The RTI 1.3V7 still has some message ordering problems, related to both best_effort and reliable time stamped messages. There are cases where TSO messages may arrive after the advancement of time and be delivered as receive order. If you are having trouble with messages arriving out of order, try setting the RID parameters "tcp_bundling_toggle" and

"udp_bundling_toggle" to 0.

### SEE ALSO

*RTI_SendInteraction, RTI_UnpublishInteractionClass*

## B.1.2.02 RTI_PublishObjectClass()

**RTI 1.0** **▲RTI 1.3**

### ABSTRACT

This service conveys the intention of a federate to begin acquiring and updating instances of a set of attributes of a specified class. The semantics of this service with respect to implicitly unpublished class-attributes changes between RTI 1.0 and RTI 1.3.

### SYNOPSIS

```
procedure RTI_PublishObjectClass(
                string(*) ObjectClassName,
                string(*) AttributeList)
         returning boolean dll="rtislx10";  // resp. "rtislx13"
```

### ARGUMENTS

*ObjectClassName*

String specifying the name of the object class to publish

*AttributeList*

String containing a comma-separated list of the attribute names the federate wishes to publish for the specified object class

### DESCRIPTION

RTI_PublishObjectClass is used for publishing an object class. This class being published must be specified in the Federation Object Model (esp. in the -FED file of the federation). A call to this function indicates that the federate is capable of modeling objects of the specified class. The parameter ObjectClassName specifies the name of the object class to publish. The name must correspond with an object class name of the .FED-File of the federation. The parameter AttributeList is a comma-separated list containing the names of the attributes of this class. The attribute names also have to correspond with the names of the attribute of the specified class of the FOM. The corresponding SLX object which is later used for modeling object of this class has to contain the same attribute names (or at least a subset of them), too.

The function internally converts the names for the class and its attributes into the appropriate object class and attribute handles used internally by the RTI. The user does not have to be concerned with this process.

### RELEASE NOTES

*RTI 1.0*

- The federate retains ownership of any currently owned instances of attributes that are implicitly unpublished as a result of this service.

*RTI 1.3*

- Any locally-owned instances of attributes that are implicitly unpublished as a result of this service immediately become unowned. These instances are offered to the federation as if they had been unconditionally divested by the federate.

### RETURN VALUES

A return value of TRUE indicates that the federate has successfully published the specified set of attributes for the object class, possibly replacing an existing set of published attributes. The federate is eligible to create objects of the given object class and to acquire instances of the specified attributes via ownership management services (the latter only applies for the RTI 1.3 version of the SLX-HLA-Interface).

### SEE ALSO

*RTI_UnpublishObjectClass*

## B.1.2.03 RTI_SubscribeInteractionClass()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_SubscribeInteractionClass declares a federate's interest in receiving a specified class of interactions. Although the RTI 1.3 implementation adds a third argument for specifying active vs. passive subscription to this service, the SLX-HLA-Interface for RTI 1.3 currently does not support this feature. The subscription type is always assumed as being active.

### SYNOPSIS

```
procedure RTI_SubscribeInteractionClass(
                string(*) InteractionClassName,
                pointer(*) theInteraction)
        returning boolean dll="rtislx10";  //resp. "rtislx13"
```

### ARGUMENTS

*InteractionClassName*

String specifying the name of the interaction class to subscribe

*theInteraction*

Pointer to an SLX object which will be associated with this interaction class. Any receiveInteraction calls that will be called at the federate ambassador will automatically access the interaction parameters using *theInteraction*. It is important to note that the pointer has to be properly initialized before calling RTI_SubscribeInteractionClass.

### DESCRIPTION

RTI_SubscribeInteractionClass declares the interest of a federate in receiving a given class of interactions (specified by InteractionClassName). The second parameter specifies an SLX object instance (theInteraction) which is to be associated with the interaction class. Any interactions of the specified class that will be received later on are delivered directly to the SLX object instance. Interactions of the same class which have the same time stamp will be buffered internally and can be obtained by calling RTI_ReflectNextBufferedInteraction.

The function internally examines the attributes of the specified SLX class for any matches with the parameters of the corresponding HLA interaction class. Any parameters of the interaction class that you'd like to receive should exist in the associated SLX class. SLXRTI10.DLL automatically detects the type of the SLX attributes. Care should be taken to ensure that the SLX types match the types specified in the FOM, otherwise unpredictable results may be the consequence.

It should also be noted that the user MUST NOT delete the SLX object instance associated with the interaction class unless he/she unsubscribes the class prior to that. Failure to do so may lead to crashes of the SLX program.

### RETURN VALUES

The return value is TRUE if the call was passed successfully to the RTI.

### SEE ALSO

*RTI_PublishInteractionClass, RTI_SendInteraction, RTI_ReflectNextBufferedInteraction, SetInteractionBufferMode*

## B.1.2.04 RTI_SubscribeObjectClassAttribute()

**RTI 1.0**

### ABSTRACT

This service is used to declare a federate's interest in receiving updates for a set of attribtues. In the RTI 1.3 version this service is named RTI_SubscribeObjectClassAttributes; this service is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_SubscribeObjectClassAttribute(
                string(*) objectClassName,
```

```
                string(*) attributeList)
        returning boolean dll="rtislx10";
```

### ARGUMENTS

*objectClassName*

String specifying the name of the object class to publish

*attributeList*

String containing a comma-separated list of the attribute names the federate wishes to subscribe to for the specified object class

### DESCRIPTION

RTI_SubscribeObjectClassAttribute is used to declare interest in receiving updates for a set of attributes of a certain object class. The parameter objectClassName specifies the object class name as listed in the .FED file. The parameter attributeList is a comma-separated list which specifies the set of attributes the federate is interested in. It should be noted that calls to this function are not cumulative, i.e., if this function is invoked with an object class that is already subscribed, the new attribute set replaces the existing subscribed attribute set.

### RETURN VALUES

A return value of TRUE indicates that the federate has successfully subscribed to the specified object class.

### SEE ALSO

*RTI_UnsubscribeObjectClassAttribute*

## B.1.2.05 RTI_SubscribeObjectClassAttributes()

**RTI 1.3**

### ABSTRACT

This service is used to declare a federate's interest in receiving reflections of updates for a specified set of attributes. In the RTI 1.0 version this service is named RTI_SubscribeObjectClassAttribute; this service is discussed in a separate section. The RTI 1.3 implementation adds the optional possibility for specifying passive vs. active subscription.

### HLA IF SPECIFICATION

This function provides access to the "Subscribe Object Class Attributes" Declaration Management service as specified in the HLA Interface Specification Version 1.3. The original RTI ambassador method always takes the subscription type as a third argument. In the SLX-HLA-Interface for RTI 1.3 this service is provided in too forms: the classical two-argument version known from RTI 1.0, and a three argument version as required by the IF-Specification.

### SYNOPSIS

```
procedure RTI_SubscribeObjectClassAttribute(
                string(*) objectClassName,
                string(*) attributeList)
        returning boolean dll="rtislx13";

procedure RTI_SubscribeObjectClassAttributeWithType(
                string(*) objectClassName,
                string(*) attributeList,
                boolean   subscriptionType)
        returning boolean dll="rtislx13";
```

### ARGUMENTS

*objectClassName*

String specifying the name of the object class to publish

*attributeList*

String containing a comma-separated list of the attribute names the federate wishes to subscribe to for the specified object class

*subscriptionType*

Boolean specifying whether an active or a passive subscription is desired. The flag indicates, whether the subscription should be taken into account when advising publishing federates of the relevance of its publications. The default as used by RTI_SubscribeObjectClassAttribute is an active subscription. The argument only exists in the RTI_SubscribeObjectClassAttributeWithType service of the SLX-HLA-Interface. The argument is only provided for compatibility reasons regarding the cooperation with non-SLX federates, since the SLX-HLA-Interface does not use the relevance advisory services provided by the RTI.

### DESCRIPTION

RTI_SubscribeObjectClassAttribute is used to declare interest in receiving updates for a set of attributes of a certain object class. The parameter objectClassName specifies the object class name as listed in the .FED file. The parameter attributeList is a comma-separated list which specifies the set of attributes the federate is interested in. It should be noted that calls to since function are not cumulative, i.e., if this function is invoked with an object class that is already subscribed, the new attribute set replaces the existing subscribed attribute set.

### RETURN VALUES

A return value of TRUE indicates that the federate has successfully subscribed to the specified object class.

### SEE ALSO

*RTI_UnsubscribeObjectClassAttribute*

## B.1.2.06 RTI_UnpublishInteractionClass()

**RTI 1.0** **▲RTI 1.3**

### ABSTRACT

RTI_UnpublishInteractionClass conveys the intention of a federate to cease generation of interactions of a specified class. The semantics of this service have changed from RTI 1.0 to RTI 1.3.

### SYNOPSIS

```
procedure RTI_UnpublishInteractionClass(string(*)InteractionClassName)
          returning boolean dll="rtislx10";  //resp. "rtislx13"
```

### ARGUMENTS

*InteractionClassName*

String specifying the name of the interaction class to unpublish

### DESCRIPTION

RTI_UnpublishInteractionClass conveys the intention of the federate to cease generating interactions of a given interaction class (specified by the parameter InteractionClassName).

### RETURN VALUES

The return value is TRUE if the call was passed successfully to the RTI.

### RELEASE NOTES

*RTI 1.0*

- Any subclasses of the specified interaction class will also be unpublished.

*RTI 1.3*

- Only the interaction class that is explicitly the subject of the service invocation is unpublished. Any subclasses of the specified interaction class remain published.

### SEE ALSO

*RTI_PublishInteractionClass*

## B.1.2.07 RTI_UnpublishObjectClass()

**RTI 1.0** **▲RTI 1.3**

### ABSTRACT

This service conveys the intention of a federate to cease creating instances and acquiring attributes of a specified object class. The semantics of this service has changed from RTI 1.0 to RTI 1.3.

### SYNOPSIS

```
procedure RTI_UnpublishObjectClass( string(*) ObjectClassName)
         returning boolean dll="rtislx10";   // resp. "rtislx13"
```

### ARGUMENTS

*ObjectClassName*

String specifying the name of the object class to unpublish

### DESCRIPTION

RTI_UnpublishObjectClass is used for indicating the intention of a federate to cease creating instances of a given object class. The parameter ObjectClassName specifies the name of the object class to unpublish.

### RETURN VALUES

A return value of TRUE indicates that the federate has successfully unpublished the specified object class.

### RELEASE NOTES

*RTI 1.0*

- Classes derived from the MOM-defined Manager object class receive special treatment. All subclasses are implicitly published and must remain published.
- Unpublication only affects the acquisition of new attribute-instances. It does not relieve the federate of update responsibility for any updates already owned.
- Unpublication of an object class unpublishes the specified object class and any of its subclasses.

*RTI 1.3*

- MOM object classes do not require a special treatment as in RTI 1.0.
- Upon unpublication of an object class by a federate, any locally owned instance attributes of object instances of the unpublished object class immediately become unowned. These attributes are offered to the federation as if they had been unconditionally divested by the unpublishing federate.
- Unpublication of an object class unpublishes only the specified object class (it does not unpublish subclasses).

### SEE ALSO

*RTI_PublishObjectClass*

## B.1.2.08 RTI_UnsubscribeInteractionClass()

**RTI 1.0** **▲RTI 1.3**

### ABSTRACT

RTI_UnsubscribeInteractionClass withdraws a federate's interest in receiving a specified class of interactions. The semantics of this service have changed from RTI 1.0 to RTI 1.3.

### SYNOPSIS

```
procedure RTI_UnsubscribeInteractionClass(
                string(*) InteractionClassName)
         returning boolean dll="rtislx10"; //resp. "rtislx13"
```

**ARGUMENTS**

*InteractionClassName*

String specifying the name of the interaction class to unsubscribe from

**DESCRIPTION**

RTI_UnsubscribeInteractionClass withdraws the interest of a federate in receiving a given class of interactions (specified by InteractionClassName).

**RETURN VALUES**

The return value is TRUE if the call was passed successfully to the RTI.

**RELEASE NOTES**

*RTI 1.0*

- Unsubsubscription of an interaction class also unsubscribes any subclasses of the interaction class.

*RTI 1.3*

- Unsubsubscription of an interaction class only unsubscribes the specified interaction class; it does not unsubscribe subclasses of the specified interaction class.

**SEE ALSO**

*RTI_SubscribeInteractionClass*

## B.1.2.09 RTI_UnsubscribeObjectClass()

**RTI 1.3**

**ABSTRACT**

This service withdraws a federate's interest in receiving updates for a set of attributes. In the RTI 1.0 version this service is named RTI_UnsubscribeObjectClassAttribute, the RTI 1.0 implementation is discussed in a separate section.

**SYNOPSIS**

```
procedure procedure RTI_UnsubscribeObjectClass (
                string(*) objectClassName)
         returning boolean dll="rtislx13";
```

**ARGUMENTS**

*objectClassName*

String specifying the name of the object class to unsubscribe from

**DESCRIPTION**

RTI_UnsubscribeObjectClassAttribute is used to withdraw interest in receiving updates for a certain object class. The parameter objectClassName specifies the object class name as listed in the .FED file.

**RETURN VALUES**

A return value of TRUE indicates that the federate has successfully unsubscribed from the specified object class.

**SEE ALSO**

*RTI_SubscribeObjectClassAttribute*

## B.1.2.10 RTI_UnsubscribeObjectClassAttribute()

**RTI 1.0**

**ABSTRACT**

This service withdraws a federate's interest in receiving updates for a set of attributes. In the RTI 1.3 version this service is named RTI_UnsubscribeObjectClass, the RTI 1.3 implementation is discussed

in a separate section.

## SYNOPSIS

```
procedure procedure RTI_UnsubscribeObjectClassAttribute(
                string(*) objectClassName)
          returning boolean dll="rtislx10";
```

## ARGUMENTS

*objectClassName*

String specifying the name of the object class to unsubscribe from

## DESCRIPTION

RTI_UnsubscribeObjectClassAttribute is used to withdraw interest in receiving updates for a certain object class. The parameter objectClassName specifies the object class name as listed in the .FED file.

## RETURN VALUES

A return value of TRUE indicates that the federate has successfully unsubscribed from the specified object class.

## SEE ALSO

*RTI_SubscribeObjectClassAttribute*

# B.1.3 TIME MANAGEMENT

## B.1.3.01 RTI_DisableAsynchronousDelivery()

**RTI 1.3**

### ABSTRACT

RTI_DisableAsynchronousDelivery instructs the LRC not to deliver receive-ordered events in the absense of an in-progress time-advancement service. This has only a meaning for time-constrained federates, since non-constrained federates receive all events in receive order.

### SYNOPSIS

```
procedure RTI_DisableAsynchronousDelivery()
         dll="slxrti13";
```

### ARGUMENTS

*None.*

### DESCRIPTION

RTI_DisableAsynchronousDelivery disables the delivery of receive-ordered events to a federate in the absence of a time-advancement service. This only applies to a time-constrained federate, since federates which are not time-constrained may receive events during any invocation of the RTI_Tick() service.

Since asynchronous delivery is disabled by default for a federate, this service should only be used to undo the effects of an RTI_EnableAsynchronousDelivery() service invocation.

### RETURN VALUES

None.

### SEE ALSO

*RTI_EnableAsynchronousDelivery*

## B.1.3.02 RTI_EnableAsynchronousDelivery()

**RTI 1.3**

### ABSTRACT

RTI_EnableAsynchronousDelivery instructs the LRC to begin delivering receive-ordered events to the federate even while no time-advancement service is in progress.

### SYNOPSIS

```
procedure RTI_EnableAsynchronousDelivery()
         dll="slxrti13";
```

### ARGUMENTS

*None.*

### DESCRIPTION

RTI_EnableAsynchronousDelivery enables the delivery of receive-ordered events to a federate in the absence of a time-advancement service. Subsequent to invoking this service, receive-ordered events may be delivered to the federate during any invocation of the RTI_Tick() service.

This setting is only relevant to federates that are time-constrained, since events for federates which are not time-constrained are always delivered asynchronously. The asynchronous delivery of receive-ordered events may be subsequently disabled by using the RTI_DisableAsynchronousDelivery service.

Asynchronous delivery in conjunction with SLX federates may be useful for receiving MOM updates without advancing simulation time. Section 4.1.1 gives an example for the usage of asynchronous delivery of MOM updates for synchronizing the startup of a federation.

### RETURN VALUES

None.

### SEE ALSO

*RTI_DisableAsynchronousDelivery, RTI_Tick*

## B.1.3.03 RTI_ModifyLookahead()

**RTI 1.3**

### ABSTRACT

RTI_ModifyLookahead is used to redefine the lookahead window for the federate. The RTI 1.0 implementation of this service is named RTI_SetLookahead and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_ModifyLookahead( double Lookahead )
          returning boolean dll="slxrti13";
```

### ARGUMENTS

*Lookahead*

New size of the interval extending forward from the federate's logical time at a given point in execution in which a federate will not generate any time stamp ordered events

### DESCRIPTION

RTI_ModifyLookahead can be used to redefine the lookahead window for the federate. The lookahead window is the amount of time between the logical time of the federate and the earliest allowable time stamp on a time-stamp-ordered (TSO) event generated by the federate. Lookahead is only meaningful for time-regulating federates, as non-time-regulating federates can not generate TSO events.

To minimize the overhead associated with synchronizing federation time advances, federates should make their lookahead window as large as possible.

### RETURN VALUES

The return value is TRUE, if the lookahead value was changed successfully and FALSE if an error occurred. It should be noted that decreasing the lookahead does not take effect immediately; instead the effective lookahead value will be gradually decreased in conjunction with the advancement of simulation time.

### SEE ALSO

*RTI_Init*

## B.1.3.04 RTI_NextEventRequest()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_NextEventRequest advances the federate's logical time to the time-stamp of the next relevant time-stamp-ordered event in the federation.

### SYNOPSIS

```
procedure    RTI_NextEventRequest(double NextEventTime)
             returning double dll="rtislx10";  //resp. "rtislx13"
```

### ARGUMENTS

*NextEventTime*

Time stamp of the next local event the federate wishes to advance to

### DESCRIPTION

RTI_NextEventRequest is used to advance the federate's logical time to the time-stamp of the next

TSO event in the federation. The parameter NextEventTime specifies the time-stamp of the next local event of the federate. This is the time to advance the federation logical time to in absence of an intervening TSO event.

The return value of this function is the time the federate has been granted to advance to, i.e., the new value of the federate's logical time.

The original RTI ambassador method nextEventRequest works asynchronously, i.e., the control is returned to the federate immediately. The federate is then expected to repeatedly call the RTI ambassador method tick() until the time advance request is completed by a time advance grant callback to the federate ambassador. The tick mechanism is used to trigger other callback invocations to the federate ambassador (e.g., for receiving updates or interactions). This somewhat complicated process is handled by the SLX-HLA-Interface internally.

The SLX user simply requests the time advancement (e.g., by calling RTI_NextEventRequest) and is then notified about the granted amount of time to advance to. The user is notified about any events that were received in between by the SLX_StateObject and by modification of the SLX objects corresponding to HLA interactions and objects.

### RETURN VALUES

The return value of this function is the time the federate has been granted to advance to, i.e., the new value of the federate's logical time. The time stamp can be equal or less the time specified in the parameter NextEventTime. The return value is -1 on the occurrence of an error.

### SEE ALSO

*RTI_NextEventRequestAvailable, RTI_TimeAdvanceRequest, RTI_TimeAdvanceRequestAvailable, RTI_SetTickSleepInterval*

## B.1.3.05 RTI_NextEventRequestAvailable()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_NextEventRequestAvailable is similar to RTI_NextEventRequest, except that a time advance might be granted before all time-stamp ordered events at the grat time have been delivered to the federate.

### SYNOPSIS

```
procedure RTI_NextEventRequestAvailable(double NextEventTime)
          returning double dll="rtislx10"; // resp. "rtislx13"
```

### ARGUMENTS

*NextEventTime*

Timestamp of the next local event the federate wishes to advance to

### DESCRIPTION

RTI_NextEventRequestAvailable works very much the same way RTI_NextEventRequest does. The main difference is in the meaning of the return value. In the case of RTI_NextEventRequestAvailable a return value of $t_x$ says that all TSO events with a time stamp less than $t_x$ and some, but not necessarily all events with a time stamp of $t_x$ have been delivered to the federate. Thus this function is especially attractive for zero lookahead federates that wish to be able to still send messages with the same time stamp ($t_x$).

In the case of RTI_NextEventRequest a return value $t_x$ guarantees, that ALL TSO events with a time stamp less or equal to $t_x$ have been delivered to the federate.

### RETURN VALUES

The return value of this function is the time the federate has been granted to advance to, i.e., the new value of the federate's logical time.

### SEE ALSO

*RTI_NextEventRequest, RTI_TimeAdvanceRequest, RTI_TimeAdvanceRequestAvailable, RTI_SetTickSleepInterval*

## B.1.3.06 RTI_QueryFederateTime()

**RTI 1.3**

### ABSTRACT

This service is used by a federate to obtain its current logical time. The RTI 1.0 implementation of this service is named RTI_RequestFederateTime and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_QueryFederateTime()
          returning double dll="slxrti13";
```

### ARGUMENTS

None.

### DESCRIPTION

RTI_QueryFederateTime requests the current logical time of the federate, i.e., the most recent time requested by the federate. If the federate is time regulating, its logical time plus lookahead constitutes the minimum allowable time stamp of time-stamp-ordered messages subsequently sent by the federate. If the federate is time constrained, the logical time represents the maximum time stamp of time-stamp-ordered events that will be delivered to the federate prior to the next time advance request.

### RETURN VALUES

The returned value is the current federate logical time or -1 if an error has occurred.

### SEE ALSO

*RTI_QueryLBTS*

## B.1.3.07 RTI_QueryLBTS()

**RTI 1.3**

### ABSTRACT

RTI_RequestLBTS requests the current effective federation lower-bound time stamp for the federate. The RTI 1.0 version of this service was named RTI_RequestLBTS and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_QueryLBTS()
          returning double dll="slxrti13";
```

### ARGUMENTS

None.

### DESCRIPTION

This function requests the current effective federation lower-bound time stamp (LBTS) for the federate. The federation LBTS is defined as the minimum time-stamp such that it can be guaranteed that no federate will generate any more time-stamp-ordered events with a lower time-stamp.

A time-regulating federate's LBTS is its current logical time plus its current lookahead; a non-time-regulating federate's LBTS is positive infinity (as it can not generate any TSO messages).

Non-time-constrained federates cannot receive TSO events, so their effective federation LBTS is infinity.

### RETURN VALUES

The return value is the current LBTS of the federation or -1 if an error has occurred.

### SEE ALSO

*RTI_QueryFederateTime, RTI_QueryMinNextEventTime, RTI_TimeAdvanceRequest*

## B.1.3.08 RTI_QueryLookahead()

**RTI 1.3**

### ABSTRACT

RTI_QueryLookahead is used to obtain the current lookahead window being used for the federate. The RTI 1.0 implementation of this service was named RTI_RequestLookahead and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_QueryLookahead()
          returning double dll="slxrti13";
```

### ARGUMENTS

None.

### DESCRIPTION

This function queries the current lookahead value of the federate. The effective lookahead at a given time is at least as great as the current lookahead as specified by RTI_ModifyLookahead.

### RETURN VALUES

The return value is the current effective lookahead for the federate or -1 if an error occurred during the service invocation.

### SEE ALSO

*RTI_ModifyLookahead*

## B.1.3.09 RTI_QueryMinNextEventTime()

**RTI 1.3**

### ABSTRACT

RTI_QueryMinNextEventTime requests the minimum possible time-stamp of the earliest time-stamp-ordered event that will ever be delivered in the federation's future. The RTI 1.0 implementation of this service was named RTI_RequestMinNextEventTime and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_QueryMinNextEventTime()
          returning double dll="slxrti13";
```

### ARGUMENTS

None.

### DESCRIPTION

This function requests the minimum possible time-stamp of the earliest time-stamp-ordered event that will ever be delivered in the federation's future.

The minimum next event time is defined as the largest time-stamp such that the RTI can guarantee that no time-stamp-ordered (TSO) events will be delivered to the federate with a smaller time-stamp value. This is defined as the minimum of the federation lower-bound time stamp an the time-stamp of the earliest TSO event (if any) in the federate's event queue.

In the case of a non-constrained federate, this is always infinity (i.e., no TSO events and an infinite LBTS). A time advance grant can never be made to a federation time greater than the minimum next event time.

### RETURN VALUES

The returned value is the current minimum next event time for the federate or -1 if an error has occurred.

### SEE ALSO

*RTI_RequestFederateTime, RTI_RequestFederationTime, RTI_RequestLBTS*

## B.1.3.10 RTI_RequestFederateTime()

**RTI 1.0**

### ABSTRACT

This service requests the current federate logical time. The RTI 1.3 implementation of this service is named RTI_QueryFederateTime and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_RequestFederateTime()
         returning double dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

RTI_RequestFederateTime requests the current logical time of the federate, i.e., the most recent time requested by the federate. If the federate is time regulating, its logical time plus lookahead constitutes the minimum allowable time stamp of time-stamp-ordered messages subsequently sent by the federate. If the federate is time constrained, the logical time represents the maximum time stamp of time-stamp-ordered events that will be delivered to the federate prior to the next time advance request.

### RETURN VALUES

The returned value is the current federate logical time or -1 if an error has occurred.

### SEE ALSO

*RTI_RequestFederationTime, RTI_RequestLBTS*

## B.1.3.11 RTI_RequestFederationTime()

**RTI 1.0**

### ABSTRACT

RTI_RequestFederationTime requests the current federation time.

### SYNOPSIS

```
procedure RTI_RequestFederationTime()
         returning double dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

This function requests the current federation time. It should be noted that this is the federation time as perceived by the federate. Federation time for a given federate is defined as the minimum of the current federation lower-bound time stamp and the federate's logical time.

### RETURN VALUES

The return value is the current federation time as perceived by the federate or -1 if an error has occurred.

### SEE ALSO

*RTI_RequestFederateTime, RTI_RequestLBTS, RTI_RequestMinNextEventTime*

### B.1.3.12 RTI_RequestLBTS()

**RTI 1.0**

#### ABSTRACT

RTI_RequestLBTS requests the current effective federation lower-bound time stamp for the federate. The RTI 1.3 version of this service is named RTI_QueryLBTS and is discussed in a separate section.

#### SYNOPSIS

```
procedure RTI_RequestLBTS()
          returning double dll="slxrti10";
```

#### ARGUMENTS

None.

#### DESCRIPTION

This function requests the current effective federation lower-bound time stamp (LBTS) for the federate. The federation LBTS is defined as the minimum time-stamp such that it can be guaranteed that no federate will generate any more time-stamp-ordered events with a lower time-stamp.

A time-regulating federate's LBTS is its current logical time plus its current lookahead; a non-time-regulating federate's LBTS is positive infinity (as it can not generate any TSO messages).

Non-time-constrained federates cannot receive TSO events, so their effective federation LBTS is infinity.

#### RETURN VALUES

The return value is the current LBTS of the federation or -1 if an error has occurred.

#### SEE ALSO

*RTI_RequestFederateTime, RTI_RequestFederationTime, RTI_RequestMinNextEventTime, RTI_TimeAdvanceRequest*

### B.1.3.13 RTI_RequestLookahead()

**RTI 1.0**

#### ABSTRACT

RTI_RequestLookahead is used to obtain the current lookahead window being used for the federate. The RTI 1.3 implementation of this service is named RTI_QueryLookahead and is discussed in a separate section.

#### SYNOPSIS

```
procedure RTI_RequestLookahead()
          returning double dll="slxrti10";
```

#### ARGUMENTS

None.

#### DESCRIPTION

This function queries the current lookahead value of the federate. The effective lookahead at a given time is at least as great as the current lookahead as specified by RTI_SetLookahead.

#### RETURN VALUES

The return value is the current effective lookahead for the federate or -1 if an error occurred during the service invocation.

#### SEE ALSO

*RTI_SetLookahead*

## B.1.3.14 RTI_RequestMinNextEventTime()

**RTI 1.0**

### ABSTRACT

RTI_RequestMinNextEventTime requests the minimum possible time-stamp of the earliest time-stamp-ordered event that will ever be delivered in the federation's future. The RTI 1.3 implementation of this service is named RTI_QueryMinNextEventTime and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_RequestMinNextEventTime()
          returning double dll="slxrti10";
```

### ARGUMENTS

None.

### DESCRIPTION

This function requests the minimum possible time-stamp of the earliest time-stamp-ordered event that will ever be delivered in the federation's future.

The minimum next event time is defined as the largest time-stamp such that the RTI can guarantee that no time-stamp-ordered (TSO) events will be delivered to the federate with a smaller time-stamp value. This is defined as the minimum of the federation lower-bound time stamp an the time-stamp of the earliest TSO event (if any) in the federate's event queue.

In the case of a non-constrained federate, this is always infinity (i.e., no TSO events and an infinite LBTS). A time advance grant can never be made to a federation time greater than the minimum next event time.

### RETURN VALUES

The returned value is the current minimum next event time for the federate or -1 if an error has occurred.

### SEE ALSO

*RTI_RequestFederateTime, RTI_RequestFederationTime, RTI_RequestLBTS*

## B.1.3.15 RTI_Retract()

**RTI 1.3**

### ABSTRACT

This service cancels an update, interaction, or deletion previously scheduled by the federate.

### SYNOPSIS

```
procedure RTI_Retract(int EventRetractionHandle)
          returning boolean dll="slxrti13";
```

### ARGUMENTS

*EventRetractionHandle*

The event retraction handle as obtained from RTI_SendInteraction, RTI_UpdateAttributeValues, or RTI_DeleteObjectInstance.

### DESCRIPTION

A federate can utilize this service to withdraw an update, interaction, or deletion it has previously scheduled, A successful invocation will result in the issuance of an event retraction message to every federate in the federation. If the specified event is currently queued for delivery to a given remote federate, it is automatically removed from its queue.

If the specified event has been recently delivered to the federate (the current RTI maintains a history of the last 50,000 events delivered), the appropriate callback is invoked and the federate is responsible for rolling back its state as appropriate. The current version of the SLX-HLA-Interface does not provide the information about requested retractions to the SLX federate yet, i.e., there is no way at this time for a SLX federate to know when to roll back its internal state. An implementation of

this functionality will be provided once SLX offers a built-in state saving technique, which is a pre-requisite for using optimistic synchronization schemes, the main application of the retract-service.

**RETURN VALUES**

The returned value is TRUE if the call was passed successfully to the RTI and FALSE otherwise.

**SEE ALSO**

*RTI_UpdateAttributeValues, RTI_SendInteraction, RTI_DeleteObjectInstance*

### B.1.3.16 RTI_SetLookahead()

**RTI 1.0**

**ABSTRACT**

RTI_SetLookahead is used to redefine the lookahead window for the federate. The RTI 1.3 implementation of this service is named RTI_ModifyLookahead and is discussed in a separate section.

**SYNOPSIS**

```
procedure RTI_SetLookahead(double Lookahead)
         returning boolean dll="slxrti10";
```

**ARGUMENTS**

*Lookahead*

New lookahead value to use for the federate

**DESCRIPTION**

RTI_SetLookahead can be used to redefine the lookahead window for the federate. The lookahead window is the amount of time between the logical time of the federate and the earliest allowable time stamp on a time-stamp-ordered (TSO) event generated by the federate. Lookahead is only meaningful for time-regulating federates, as non-time-regulating federates can not generate TSO events.

To minimize the overhead associated with synchronizing federation time advances, federates should make their lookahead window as large as possible.

**RETURN VALUES**

The return value is TRUE, if the lookahead value was changed successfully and FALSE if an error occurred. It should be noted that the lookahead change does not necessarily takes effect immediately, esp. If the new lookahead value is smaller than the old one.

**SEE ALSO**

*RTI_Init*

### B.1.3.17 RTI_SetTimeParameters()

**RTI 1.0** **RTI 1.3**

**ABSTRACT**

RTI_SetTimeParameters can be used to alter the different parameters related to the HLA time management. These parameters are initially set in the call to RTI_Init.

**SYNOPSIS**

```
procedure RTI_SetTimeParameters(
                  boolean constrained,
                  boolean regulation,
                  double Lookahead)
         returning double dll="rtislx10"; //resp. "rtislx13"
```

### ARGUMENTS

*constrained*

specifies whether the federate is time constrained or not

*regulation*

specifies whether the federate is time regulating or not

*Lookahead*

specifies the new lookahead value of the federate

### DESCRIPTION

RTI_SetTimeParameters sets the different parameters related to the HLA time management. It is suggested to read the HLA Time Management Design Document to learn more about the parameters that can be influenced by this function.

### RETURN VALUES

The return value is the current federate time as returned by requestFederateTime in RTI 1.0 and queryFederateTime in RTI 1.3.

### SEE ALSO

*RTI_Init*

## B.1.3.18 RTI_TimeAdvanceRequest()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_TimeAdvanceRequest requests an advancement of the logical time of the federate to a specified federation time.

### SYNOPSIS

```
procedure RTI_TimeAdvanceRequest(double FederationTime)
         returning double dll="rtislx10"; // resp. "rtislx13"
```

### ARGUMENTS

*FederationTime*

Timestamp representing the point on the federation time axis to which to advance the federate's logical time.

### DESCRIPTION

RTI_TimeAdvanceRequest requests an advance of the logical time of the federate to a specified federation time (FederationTime). The service will not return until the advance is achieve, i.e., the RTI can guarantee that all time-stamp-ordered (TSO) events delivered to the federate in future will have a time stamp greater than the new federate logical time.

The main difference to the RTI_NextEventRequest functions is that RTI_TimeAdvanceRequest will always return FederationTime.

By requesting a time advance, the federate is agreeing to not generate any time stamp ordered events with a time stamp less that the requested time plus the current federate lookahead.

### RETURN VALUES

The return value is the new logical federate time as requested (FederationTime) or -1 if a error has occurred.

### SEE ALSO

*RTI_NextEventRequest, RTI_NextEventRequestAvailable, RTI_TimeAdvanceRequestAvailable, RTI_SetTickSleepInterval*

## B.1.3.19 RTI_TimeAdvanceRequestAvailable()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_TimeAdvanceRequestAvailable is similar to the RTI_TimeAdvanceRequest service, except that not all events occurring at exactly the requested time will necessarily be delivered before a timeAdvanceGrant is made.

### SYNOPSIS

```
procedure RTI_TimeAdvanceRequestAvailable(double FederationTime)
          returning double dll="rtislx10"; // resp. "rtislx13"
```

### ARGUMENTS

*NextEventTime*

Timestamp of the next local event the federate wishes to advance to

### DESCRIPTION

RTI_TimeAdvanceRequestAvailable works the almost the same way RTI_TimeAdvanceRequest does. The only difference is the meaning of the return value: In the case of RTI_TimeAdvanceRequestAvailable the RTI can only guarantee that all time stamp ordered events delivered to the federate in the future will have a time stamp not less that the new federate time. In essence this means that the federate may still receive events with the same time stamp; thus this function is esp. important for zero lookahead federates.

### RETURN VALUES

The return value is the new logical federate time as requested (FederationTime) or -1 if a error has occurred.

### SEE ALSO

*RTI_NextEventRequest, RTI_NextEventRequestAvailable, RTI_TimeAdvanceRequest, RTI_SetTickSleepInterval*

# B.1.4 OBJECT MANAGEMENT

## B.1.4.01 RTI_DeleteObject()

**RTI 1.0**

### ABSTRACT

This service removes an object from the federation. The RTI 1.3 implementation of this service is named RTI_DeleteObjectInstance and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_DeleteObject(
                int ObjectID,
                double TimeStamp,
                string(*) UserSuppliedTag)
          returning boolean dll="slxrti10";
```

### ARGUMENTS

*ObjectID*

Object to be deleted from the federation execution

*TimeStamp*

Time at which the object deletion is to become effective

*UserSuppliedTag*

A string passed to federates which can be used for federation specific purposes

### DESCRIPTION

RTI_DeleteObject removes an object from the federation execution. The object is specified in the parameter ObjectID. The parameter TimeStamp states the time at which the object deletion is to become effective. The parameter UserSuppliedTag can be used to give a textual description for the reason for the object removal or some other federation specific tasks.

From a methodological point of view it would be natural to call RTI_DeleteObject in the final-property of SLX objects. Unfortunately, this approach will not work for reasons of providing crash safety to SLX federates. Internally, SLX keeps track of the number of references to objects by using a use-count mechanism. An object can only be deleted. if the use-count is zero.

The SLX-HLA-Interface takes advantage of this approach by internally incrementing the use-count of objects when RTI_RegisterObject or RTI_RegisterGhostedObject for this object is called. This prevents the user from accidentally deleting the object while it is still known to the RTI and the SLX-HLA-Interface. The major drawback to this approach is, that the invocation of final-property of this SLX object will be delayed until RTI_DeleteObjectI is called. Thus, if RTI_DeleteObject is placed into the final-property, the invocation will be delayed forever.

### RETURN VALUES

The return value is TRUE if the call was passed successfully to the RTI.

### SEE ALSO

*RTI_RegisterObject*

## B.1.4.02 RTI_DeleteObjectInstance()

**RTI 1.3**

### ABSTRACT

This service removes an object instance from the federation. The RTI 1.0 implementation of this service is named RTI_DeleteObject and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_DeleteObjectInstance(
                int ObjectID,
```

```
            double TimeStamp,
            string(*) UserSuppliedTag)
    returning int dll="slxrti13";
```

### ARGUMENTS

*ObjectID*

Object to be deleted from the federation execution

*TimeStamp*

Time at which the object deletion is to become effective. If -1 is specified as the logical time the deletion is treated as a receive order event.

*UserSuppliedTag*

A string passed to federates which can be used for federation specific purposes

### DESCRIPTION

RTI_DeleteObjectInstance removes an object from the federation execution. The object is specified in the parameter ObjectID. The parameter TimeStamp states the time at which the object deletion is to become effective. The parameter UserSuppliedTag can be used to give a textual description for the reason for the object removal or some other federation specific tasks.

From a methodological point of view it would be natural to call RTI_DeleteObjectInstance in the final-property of SLX objects. Unfortunately, this approach will not work for reasons of providing crash safety to SLX federates. Internally, SLX keeps track of the number of references to objects by using a use-count mechanism. An object can only be deleted. if the use-count is zero.

The SLX-HLA-Interface takes advantage of this approach by internally incrementing the use-count of objects when RTI_RegisterObjectInstance or RTI_RegisterGhostedObjectInstance for this object is called. This prevents the user from accidentally deleting the object while it is still known to the RTI and the SLX-HLA-Interface. The major drawback to this approach is, that the invocation of final-property of this SLX object will be delayed until RTI_DeleteObjectInstance is called. Thus, if RTI_DeleteObjectInstance is placed into the final-property, the invocation will be delayed forever.

### RETURN VALUES

If the call was successfully processed the return value is the event retraction handle returned from the RTI. This is a non-negative integer, which can be used by the federate to retract the deletion if appropriate. If the call failed for some reason, -1 is returned. Note that receive order deletions (which can be initiated by passing -1 as the time stamp) cannot be retracted and therefore do not return an event retraction handle. In this case 0 is returned for a successful call and -1 otherwise.

### SEE ALSO

*RTI_RegisterObjectInstance, RTI_Retract*

## B.1.4.03 RTI_ReflectControlVariableChanges()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_ReflectControlVariableChanges is used to make SLX note control variable changes that may have taken place during a time advance function.

### SYNOPSIS

```
procedure    RTI_ReflectControlVariableChanges()
             dll="slxrti10"; //resp. "slxrti13"
```

### ARGUMENTS

None.

### DESCRIPTION

By invoking this function the user initiates an SLX-internal re-evaluation of all control variables that might have been changed from the SLX-HLA-Interface. It is important for the logical correctness of your model to invoke this function after a request for advancing the federate time has been granted and the model has advanced to the granted time stamp (see also Section 3.6.3). If your model does not use any control variables that might be changed from the outside you can ignore this function.

### RETURN VALUES

This function does not return a value.

## B.1.4.04 RTI_RegisterGhostedObject()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

RTI_RegisterGhostedObject is used to register a local copy of a remote object with the SLX-HLA-Interface. Copies of remote objects must be created inside the SLX model and then be registered using this service.

### SYNOPSIS

```
procedure RTI_RegisterGhostedObject(
             string(*) ObjectClassName,
             pointer(*) theObject)
          returning int dll="rtislx10"; //resp. "rtislx13"
```

### ARGUMENTS

*ObjectClassName*

String specifying the name of the object class for which a ghosted object instance will be generated

*theObject*

Pointer to an SLX object which will be the SLX representative (“proxy”) of the ghosted object instance. All subsequent callbacks for receiving updates for this object instance will access the SLX object instance directly. It is important to note that the pointer has to be properly initialized before calling RTI_RegisterGhostedObject.

### DESCRIPTION

RTI_RegisterGhostedObject should be used to provide a storage space for incoming attribute updates. This process is referred to as ghosting a remote object (i.e., creating a local copy).

Once you indicated interest in a certain object class (by invoking RTI_SubscribeObjectClassAttributes) the RTI will inform you when an object instance of this class is detected. To do this the RTI calls the discoverObject callback at the federate ambassador of all interested federates. The federate ambassador of the SLX-HLA-Interface notifies the SLX model about the detection of such an object class via an entry in the SLX_StateObject. The counter of DiscoveredObjects will be increased and the parameter DiscoveredObjectClass will show the name of the discovered object class. This is the place where RTI_RegisterGhostedObject should be invoked.

RTI_RegisterGhostedObject takes two parameters: the name of the object class and a pointer to a corresponding SLX object instance. The SLX instance must not be deleted during the entire simulation, unless you unsubscribe from the object class.

If more than one object instance is detected at the same time, the parameter DiscoveredObjectClass will only show the oldest object class that was discovered but not yet ghosted. After invoking RTI_RegisterGhostedObject the information in this attribute will be updated.

The SLX-HLA-Interface buffers any updates that might be received for discovered objects that are not ghosted yet.

### RETURN VALUES

The return value is the object ID of the ghosted object if the call was successful. In most cases this ID can be discarded since the SLX model does not need it for receiving updates for the object. The return value is -1 if an error occurred during the function call.

### SEE ALSO

*RTI_RegisterObject*

## B.1.4.05 RTI_RegisterObject()

**RTI 1.0**

### ABSTRACT

RTI_RegisterObject is used to introduce a new object instance into the federation. The RTI 1.3 implementation of this service is named RTI_RegisterObjectInstance and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_RegisterObject( string(*) ObjectClassName,
                pointer(*) theObject)
          returning int dll="rtislx10";
```

### ARGUMENTS

*ObjectClassName*

String specifying the name of the object class for which an object instance will be generated

*theObject*

Pointer to an SLX object which will be the SLX representative of the object instance to register. All subsequent calls for sending updates for this object instance will access the SLX object instance directly. It is important to note that the pointer has to be properly initialized before calling RTI_RegisterObject.

### DESCRIPTION

RTI_RegisterObject can be used for registering object instances with the RTI. It is necessary to specify the object class name as listed in the FOM (ObjectClassName) and a pointer to a corresponding SLX object (theObject). The corresponding SLX object should contain all attributes that the federate wishes to update subsequently (the ones that were specified in the corresponding call to RTI_PublishObjectClassAttribute). The order of the attributes in the SLX object is not relevant; it can also contain additional (non-HLA) attributes. The SLX-HLA-Interface automatically detects the data types of the attributes of the SLX object. The object can contain simple types like integer, double, boolean, string, enum, and their control variants. Complex objects are currently supported for up to one hierarchy level (e.g., an SLX attribute can be another object, this object must consist of simple types, though). Objects containing sets or arrays are not supported.

### RETURN VALUES

The return value is the object ID assigned by the RTI if the object was registered successfully and -1 if the registration was not successful. The federate should store the return value to be able to send updates for this object class in subsequent calls of RTI_UpdateAttributeValues.

### SEE ALSO

*RTI_PublishObjectClass, RTI_UpdateAttributeValues*

## B.1.4.06 RTI_RegisterObjectInstance()

**RTI 1.3**

### ABSTRACT

RTI_RegisterObjectInstance is used to introduce a new object instance into the federation. The RTI 1.0 implementation of this service is named RTI_RegisterObject and is discussed in a separate section.

### SYNOPSIS

```
procedure RTI_RegisterObjectInstance(
                string(*) ObjectClassName,
                pointer(*) theObject)
          returning int dll="rtislx13";
```

### ARGUMENTS

*ObjectClassName*

String specifying the name of the object class for which an object instance will be generated

*theObject*

Pointer to an SLX object which will be the SLX representative of the object instance to register. All subsequent calls for sending updates for this object instance will access the SLX object instance directly. It is important to note that the pointer has to be properly initialized before calling RTI_RegisterObject.

### DESCRIPTION

RTI_RegisterObject can be used for registering object instances with the RTI. It is necessary to specify the object class name as listed in the FOM (ObjectClassName) and a pointer to a corresponding SLX object (theObject). The corresponding SLX object should contain all attributes that the federate wishes to update subsequently (the ones that were specified in the corresponding call to RTI_PublishObjectClassAttribute). The order of the attributes in the SLX object is not relevant; it can also contain additional (non-HLA) attributes. The SLX-HLA-Interface automatically detects the types of the attributes of the SLX object. It can contain simple types like integer, double, boolean, string, enum, and their control variants. Complex objects are currently supported for up to one hierarchy level (e.g., an SLX attribute can be another object, this object must consist of simple types, though). Objects containing sets or arrays are not supported.

The creation of a new object instance is immediately announced to the federation, resulting in discoverObjectInstance callbacks to the federate ambassador of remote federates which are subscribed to at least one attribute of the registered object class.

### RETURN VALUES

The return value is the object ID assigned by the RTI if the object was registered successfully and -1 if the registration was not successful. The federate should store the return value to be able to send updates for this object class in subsequent calls of RTI_UpdateAttributeValues. The returned object ID is guaranteed to be unique over the lifetime of the registering federate. Under RTI 1.3, the object ID cannot be used as a federation wide identifier for the object instance, since different federates may know the same objects by different IDs. If a unique object instance identifier is needed, the instance name assigned by the RTI has to be used.

### SEE ALSO

*RTI_PublishObjectClass, RTI_UpdateAttributeValues*

## B.1.4.07 RTI_RequestClassAttributeValueUpdate()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service stimulates the generation of attribute updates for a given class of objects.

### SYNOPSIS

```
procedure RTI_RequestClassAttributeValueUpdate(
                    string(*) ClassName,
                    string(*) AttributeList)
          returning boolean dll="slxrti10";  //resp. "rtislx13"
```

### ARGUMENTS

*ClassName*

String containing the name of the object class for which updates are requested

*AttributeList*

String containing a comma-separated list of the names of the class attributes for which an update is requested

### DESCRIPTION

RTI_RequestClassAttributeValueUpdate requests an attribute update for all attributes specified via AttributeList for all instances of the object class specified via ClassName. Remote federates will receive a provideAttributeValueUpdate callback for each instance of the requested object class. The federate ambassador of the SLX-HLA-Interface handles these requests automatically by providing the updates as requested.

This service may be used by late-arriving federates to solicit updates for all existing object instances. This is particularly useful for instance attributes which are updated infrequently.

### RETURN VALUES

The return value is TRUE if the call was passed successfully to the RTI.

### SEE ALSO

*RTI_RequestObjectAttributeValueUpdate*

## B.1.4.08 RTI_RequestObjectAttributeValueUpdate()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service stimulates the generation of instance attribute updates for a specified objects instance.

### SYNOPSIS

```
procedure RTI_RequestClassAttributeValueUpdate(
                    int   ObjectID,
                    string(*)   AttributeList)
          returning boolean dll="slxrti10"; //resp. "rtislx13"
```

### ARGUMENTS

*ObjectID*

Object ID of the object instance whose instance attributes are to be solicited

*AttributeList*

String containing a comma-separated list of the names of the instance attributes for which an update is requested

### DESCRIPTION

RTI_RequestObjectAttributeValueUpdate requests an attribute update for all instance attributes specified via AttributeList for the object instance specified by ObjectID. Remote federates will receive a provideAttributeValueUpdate callback for any solicited instance attributes owned by the federate. The federate ambassador of the SLX-HLA-Interface handles these requests automatically by providing the updates as requested.

This service may be used by late-arriving federates to solicit updates for all existing object instances. This is particularly useful for instance attributes which are updated infrequently.

### RETURN VALUES

The return value is TRUE if the call was passed successfully to the RTI.

### SEE ALSO

*RTI_RequestClassAttributeValueUpdate*

## B.1.4.09 RTI_SendInteraction()

**RTI 1.0** **▲RTI 1.3**

### ABSTRACT

RTI_SendInteraction generates an interaction event in the federation.

### SYNOPSIS

```
procedure RTI_SendInteraction(
                    string(*) InteractionClassName,
                    string(*) ParameterList,
                    double TimeStamp)
          returning boolean dll="rtislx10";
```

```
procedure RTI_SendInteraction(
                  string(*) InteractionClassName,
                  string(*) ParameterList,
                  double TimeStamp)
         returning int dll="rtislx13";  ← Changes return type
```

### ARGUMENTS

*InteractionClassName*

String specifying the name of the interaction class

*ParameterList*

String containing a comma-separated list of the names of the interaction parameters which will be sent along with the interaction

*TimeStamp*

Time stamp associated with the interaction. Under RTI 1.3, if the time stamp is passed as -1, the interaction will be sent as a receive order event.

### DESCRIPTION

RTI_SendInteraction sends an interaction of the specified class (parameter InteractionClassName) to all federates that have subscribed to the interaction class. ParameterList is a comma-separated list which specifies the parameters that will be sent along with the interaction. TimeStamp specifies the federation time at which the interaction occurs.

The values of the parameters to send with the interaction are taken directly from the corresponding SLX attributes as specified in RTI_PublishInteractionClass.

### RETURN VALUES

RTI 1.0

The return value is TRUE, if the call was passed successfully to the RTI, otherwise FALSE.

RTI 1.3

If the call was successfully processed the return value is the event retraction handle returned from the RTI. This is a non-negative integer, which can be used by the federate to retract the interaction if appropriate. If the call failed for some reason, -1 is returned. Note that receive order interactions (which can be sent by passing -1 as the time stamp) cannot be retracted and therefore do not return an event retraction handle. In this case 0 is returned for a successful call and -1 otherwise.

### SEE ALSO

*RTI 1.0*

RTI_PublishInteractionClass

*RTI 1.3*

RTI_PublishInteractionClass, RTI_Retract

## B.1.4.10 RTI_UpdateAttributeValues()

**RTI 1.0** **▲RTI 1.3**

### ABSTRACT

RTI_UpdateAttributeValues notifies the federation of a change in values for one or more instance attributes of an object instance.

### SYNOPSIS

```
procedure RTI_UpdateAttributeValues( int Object_ID,
                      string(*) AttributeList,
                      double TimeStamp)
         returning boolean dll="rtislx10";

procedure RTI_UpdateAttributeValues( int Object_ID,
                      string(*) AttributeList,
                      double TimeStamp)
         returning int dll="rtislx13"      ← Changes return type
```

## ARGUMENTS

*ObjectID*

The object ID of the object instance as assigned by the RTI. The object ID is returned upon registering an object by calling RTI_RegisterObject (or RTI_RegisterObjectInstance under RTI 1.3).

*AttributeList*

String containing a comma-separated list of the names of the instance attributes for which an update will be sent

*TimeStamp*

Time stamp associated with the update. Under RTI 1.3, if the time stamp is passed as -1, the update will be sent as a receive order event.

## DESCRIPTION

RTI_UpdateAttribtueValues notifies the federation of a change in value of one or more attributes of an object. ObjectID identifies the object instance that the update relates to (this is the ID that is obtained when registering the object instance with the RTI). AttributeList is a comma-separated list which specifies the attributes which are to be updated. The corresponding attribute values are taken directly from the corresponding SLX object. TimeStamp specifies the time stamp associated with the update. The minimum value for TimeStamp is determined by the current federate time plus its current lookahead value.

## RETURN VALUES

RTI 1.0

The return value is TRUE, if the call was passed successfully to the RTI, otherwise FALSE.

RTI 1.3

If the call was successfully processed the return value is the event retraction handle returned from the RTI. This is a non-negative integer, which can be used by the federate to retract the update if appropriate. If the call failed for some reason, -1 is returned. Note that receive order updates (which can be sent by passing -1 as the time stamp) cannot be retracted and therefore do not return an event retraction handle. In this case 0 is returned for a successful call and -1 otherwise.

## RELEASE NOTES

*RTI 1.3*

- The RTI 1.3 (currently in version 7) still has some message ordering problems, related to both best_effort and reliable timestamped messages. There are cases where TSO messages may arrive after the advancement of time and be delivered as receive order. If you are having trouble with messages arriving out of order, try setting the RID parameters "tcp_bundling_toggle" and "udp_bundling_toggle" to 0.

## SEE ALSO

*RTI 1.0*

RTI_SubscribeObjectClassAttribute, RTI_RegisterObject

*RTI 1.3*

RTI_SubscribeObjectClassAttributes, RTI_RegisterObjectInstance, RTI_Retract

# B.1.5 OWNERSHIP MANAGEMENT

## B.1.5.01 RTI_AttributeIsOwnedByFederate()

**RTI 1.0**

### ABSTRACT

This service queries the RTI to determine whether a specified instance-attribute of a specified object instance is currently owned by the local federate. The RTI 1.3 implementation of this service is named RTI_IsAttributeOwnedByFederate and is discussed in a separate section.

### HLA IF SPECIFICATION

This function realizes the “Is Attribute Owned By Federate” Ownership Management service.

### SYNOPSIS

```
procedure RTI_AttributeIsOwnedByFederate (
                     int       ObjectID
                     string(*) AttributeName)
          returning boolean dll="rtislx10";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which instance-attribute ownership is being queried

*AttributeName*

Name of the instance attribute of the object instance for which ownership is being queried

### DESCRIPTION

This function may be used to synchronously determine whether a specified instance-attribute is owned by the local federate. A positive (*true*) response indicates that the local federate owns the specified instance attribute. A negative (false) response indicates, that the specified instance attribute is unowned, non-existent, or owned by a remote federate.

### RETURN VALUES

A successful invocation of this service returns TRUE if the specified instance-attribute is owned by the local federate, otherwise it returns false.

### SEE ALSO

*RTI_QueryAttributeOwnership*

## B.1.5.02 RTI_AttributeOwnershipAcquisition()

**RTI 1.3**

### ABSTRACT

This service initiates an attempt to acquire ownership of a specified set of instance-attributes for a specified object instance.

### HLA IF SPECIFICATION

This function provides access to the “Attribute Ownership Acquisition” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_AttributeOwnershipAcquisition (
                     int       ObjectID
                     string(*) AttributeList,
                     string(*) theTag)
          returning boolean dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance whose instance-attributes are requested

*AttributeList*

List of the names of the instance-attributes for which ownership is requested

*theTag*

a string that is passed to resulting invocations of requestAttributeOwnershipRelease(); this argument is not interpreted by the RTI and may be used to communicate federation-specific information about the ownership request. Note: The argument is only provided for compatibility reasons for co-operation with non-SLX federates. The argument can only be sent along with this call, in the resulting requestAttributeOwnershipRelease() call this argument will be ignored by the SLX-HLA-Interface.

### DESCRIPTION

This function initiates a request to transfer ownership of the specified instance-attributes to a federate. No instance-attribute in an acquisition request may be owned by the requesting federate, and all class-attributes must be published by the requesting federate.

If an instance-attribute is unowned the federate will receive a synchronous response, otherwise the response may occur asynchronously during some subsequent invocation of the tick-service. Since the invocation of the tick service is performed automatically during the time advancement services there is no need to call tick from within SLX. The only exception is an explicit need for the ownership transfer to happen without advancing the simulation clock. In this case the RTI_Tick() function can be called until the ownership transfer has succeeded.

For attributes which are currently owned by remote federates the call to RTI_AttributeOwnership-Acquisition will result in the requestAttributeOwnershipRelease() callback invocation at the Federate Ambassador of remote federates. A remote SLX federate will be informed of this callback by placing the associated SLX object into the set OwnershipReleaseRequests, which is part of the SLX_StateObject. The remote federate can then query the attributes which it has been requested to release by invoking RTI_GetAttributesRequestedToRelease().

If the remote federate positively responds to a release request (by calling RTI_AttributeOwnership-ReleaseResponse), the ownership of the instance-attribute is transferred to the requesting federate, and the requesting federate is advised of such using the attributeOwnershipAcquisition callback. An SLX federate will be notified by placing the associated SLX object into the set OwnershipAcquisition-Notification, which is part of the SLX_StateObject. The SLX federate can then query which instance-attribute have been transferred by calling RTI_GetAcquiredAttributes.

### RETURN VALUES

A return value of TRUE indicates that the remote federate(s) owning the specified instance-attributes will be requested to relinquish ownership. The requesting federate will be notified about of successful acquisitions via the SLX_StateObject.

### SEE ALSO

*RTI_AttributeOwnershipAcquisitionIfAvailable, RTI_CancelAttributeOwnershipAcquisition*

## B.1.5.03 RTI_AttributeOwnershipAcquisitionIfAvailable()

**RTI 1.3**

### ABSTRACT

This service initiates an attempt to acquire ownership of a specified set of instance-attributes for a specified object instance. Only object instances that exist in the federation but are currently unowned will be acquired.

### HLA IF SPECIFICATION

This function provides access to the “Attribute Ownership Acquisition If Available” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_AttributeOwnershipAcquisitionIfAvailable (
                    int       ObjectID
```

```
                    string(*) AttributeList)
          returning boolean dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance whose instance-attributes are requested

*AttributeList*

List of the names of the instance-attributes for which ownership is requested

### DESCRIPTION

This function is similar to RTI_AttributeOwnershipAcquisition(), except that remote federates are not asked to release ownership of any instance-attributes that are currently owned.

The federate will receive the Federate Ambassador callback attributeOwnershipUnavailable() for any instance-attributes that are currently owned by remote federates. SLX federates will be informed of such a callback by placing the associated SLX object into the set OwnershipUnavailable, which is part of the SLX_StateObject. SLX federates can then query the unavailable attributes by calling RTI_GetUnavailableAttributes.

The federate will receive the Federate Ambassador callback attributeOwnershipAcquisitionNotification() for any instance-attributes that are not currently owned by any federate. SLX federates will be informed of such a callback by placing the associated SLX object into the set OwnershipAcquisitionNotification, which is part of the SLX_StateObject. SLX federates can then query the acquired attributes by calling RTI_GetAcquiredAttributes.

### RETURN VALUES

A return value of TRUE indicates that the acquisition request has been announced to the federation. The requesting federate will be appraised of success or failure subsequently.

### SEE ALSO

*RTI_AttributeOwnershipAcquisition, RTI_CancelAttributeOwnershipAcquisition*

## B.1.5.04 RTI_AttributeOwnershipReleaseResponse()

**RTI 1.3**

### ABSTRACT

This service releases ownership of a set of instance-attributes for a specified object instance. This service should be called as a positive response to a previously received request to release ownership of the specified instance-attributes.

### HLA IF SPECIFICATION

This function provides access to the “Attribute Ownership Release Response” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_AttributeOwnershipReleaseResponse (
                    int       ObjectID
                    string(*) AttributeList)
          returning boolean dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance whose instance-attributes are being released

*AttributeList*

List of the names of the instance-attributes to release

### DESCRIPTION

This function is used to provide a positive response to a remote RTI_AttributeOwnershipAcquisition request that has been communicated to the local federate via the SLX_StateObject (set OwnershipReleaseRequests) and the query function RTI_GetAttributesRequestedToRelease().

Upon a successful return from this function, the federate no longer owns the specified instance-

attributes.

### RETURN VALUES

A return value of TRUE indicates that the federate has released ownership of the specified instance-attributes.

### SEE ALSO

*RTI_NegotiatedAttributeDivestiture, RTI_UnconditionalAttributeOwnershipDivestiture*

## B.1.5.05 RTI_CancelAttributeOwnershipAcquisition()

**RTI 1.3**

### ABSTRACT

This service requests the cancellation of a previously requested ownership acquisition for a specified set of instance-attributes of a specified object instance.

### HLA IF SPECIFICATION

This function provides access to the “Cancel Attribute Ownership Acquisition” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_CancelAttributeOwnershipAcquisition (
                     int       ObjectID
                     string(*) AttributeList)
          returning boolean dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which instance-attributes divestiture is being canceled

*AttributeList*

List of the names of the instance-attributes for which divestiture is being canceled

### DESCRIPTION

This function is used to request the cancellation of an attribute acquisition request previously made by the federate using the RTI_AttributeOwnershipAcquisition() service. The instance attributes subjects of such a cancellation request must be instance-attributes that the local federate has requested to acquire but has not yet been given ownership. Such a cancellation must be negotiated with the rest of the federation. A confirmation of the cancellation is delivered in the form of the confirmAttributeOwnershipAcquisitionCancellation() callback. An SLX federate will be notified of such a confirmation by placing the associated SLX object into the set OwnershipAcquisitionCancellation, which is part of the SLX_StateObject. The SLX federate can then query the attributes for which the acquisition request has been successfully canceled by calling RTI_GetCanceledAttributes.

### RETURN VALUES

A return value of TRUE indicates that the federation has been notified of the cancellation request. The federate will be notified of successful cancellations via the SLX_StateObject.

### SEE ALSO

*RTI_AttributeOwnershipAcquisition, RTI_AttributeOwnershipReleaseResponse, RTI_NegotiatedAttributeOwnershipDivestiture, RTI_UnconditionalAttributeOwnershipDivestiture*

## B.1.5.06 RTI_CancelNegotiatedAttributeOwnershipDivestiture()

**RTI 1.3**

### ABSTRACT

This service cancels a previously requested negotiated ownership divestiture for a specified set of instance-attributes of a specified object instance.

### HLA IF SPECIFICATION

This function provides access to the “Cancel Negotiated Attribute Ownership Divestiture” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_CancelNegotiatedAttributeOwnershipDivestiture(
                     int       ObjectID
                     string(*) AttributeList)
          returning boolean dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which instance-attributes divestiture is being canceled

*AttributeList*

List of the names of the instance-attributes for which divestiture is being canceled

### DESCRIPTION

This function cancels the effects of previous requests to negotiate an ownership divestiture of the specified instance-attributes of the specified object instance. An instance-attribute eligible for a divestiture cancellation must be

- the subject of a previous RTI_NegotiatedAttributeOwnershipDivestiture request by the local federate
- still owned by the local federate.

### RETURN VALUES

A return value of TRUE indicates that ownership of the specified instance-attributes will not be transferred without the explicit approval of the federate.

### SEE ALSO

*RTI_AttributeOwnershipAcquisitionIfAvailable, RTI_NegotiatedAttributeOwnershipDivestiture*

## B.1.5.07 RTI_GetAcquiredAttributes()

**RTI 1.3**

### ABSTRACT

This service is used to query for which instance-attributes of a specified object instance the local federate has acquired ownership. This service can be used to sense the information that was supplied to the local federate by a previous attributeOwnershipAcquisitionNotification callback to its Federate Ambassador. Since SLX-Federates cannot access this information directly, the query function is supplied.

### HLA IF SPECIFICATION

This function provides access to the information that is supplied by the “Attribute Ownership Acquisition Notification” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_GetAcquiredAttributes (
                     int       ObjectID)
          returning string(1024) dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which the acquired instance-attributes are being queried

### DESCRIPTION

This function queries the information that has been provided by a previous attributeOwnership-AcquisitionNotification callback. If such a callback occurs (most likely in response to a previous RTI_AttributeOwnershipAcquisition() request), the object instance for which attributes have been successfully acquired is placed into the set “OwnershipAcquisitionNotification”, which is part of the

SLX_StateObject. The SLX-Federate is then supposed to use RTI_GetAcquiredAttributes to query as for which set of attributes it has been given ownership.

### RETURN VALUES

The function returns a string containing a comma-separated list of the attribute names which the SLX-Federate has acquired.

### SEE ALSO

*RTI_AttributeOwnershipAcquisition, RTI_NegotiatedAttributeOwnershipDivestiture*

## B.1.5.08 RTI_GetAttributesRequestedToRelease()

**RTI 1.3**

### ABSTRACT

This service is used to query for which instance-attributes of a specified object instance the local federate has been requested to release ownership. This service can be used to sense the information that was supplied to the local federate by a previous requestAttributeOwnershipRelease callback to its Federate Ambassador. Since SLX-Federates cannot access this information directly, the query function is supplied.

### HLA IF SPECIFICATION

This function provides access to the information that is supplied by the “Request Attribute Ownership Release” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_GetAttributesRequestedToRelease (
                      int       ObjectID)
          returning string(1024) dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which the instance-attributes which have been requested to release are being queried

### DESCRIPTION

This function queries the information that has been provided by a previous requestAttributeOwnershipRelease callback. If such a callback occurs (most likely in response to a previous RTI_AttributeOwnershipAcquisition() request by a remote federate), the object instance for which attributes are being requested to release is placed into the set “OwnershipReleaseRequests”, which is part of the SLX_StateObject. The SLX-Federate is then supposed to use RTI_GetAttributesRequestedToRelease to query as for which set of attributes the release request was issued.

### RETURN VALUES

The function returns a string containing a comma-separated list of the attribute names which the SLX-Federate has been requested to release.

### SEE ALSO

*RTI_AttributeOwnershipAcquisition, RTI_NegotiatedAttributeOwnershipDivestiture*

## B.1.5.09 RTI_GetCanceledAttributes()

**RTI 1.3**

### ABSTRACT

This service is used to query for which instance-attributes of a specified object instance the local federate has successfully canceled a previously issued ownership acquisition request. This service can be used to sense the information that was supplied to the local federate by a previous confirmAttributeOwnershipAcquisitionCancellation callback to its Federate Ambassador. Since SLX-Federates cannot access this information directly, the query function is supplied.

### HLA IF SPECIFICATION

This function provides access to the information that is supplied by the “Confirm Attribute Ownership Acquisition Cancellation” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_GetCanceledAttributes (
                    int       ObjectID)
         returning string(1024) dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which the instance-attributes are being queried

### DESCRIPTION

This function queries the information that has been provided by a previous confirmAttribute-OwnershipAcquisitionCancellation callback. If such a callback occurs (most likely in response to a previous RTI_CancelAttributeOwnershipAcquisition() request), the object instance for which attribute acquisition has been successfully canceled is placed into the set “OwnershipAcquisitionCancelation”, which is part of the SLX_StateObject. The SLX-Federate is then supposed to use RTI_GetCanceled-Attributes to query as for which set of attributes ownership acquisition has been canceled.

### RETURN VALUES

The function returns a string containing a comma-separated list of the attribute names for which the SLX-Federate has canceled acquisition.

### SEE ALSO

*RTI_CancelAttributeOwnershipAcquisition*

## B.1.5.10 RTI_GetOfferedAttributes()

**RTI 1.3**

### ABSTRACT

This service is used to query for which instance-attributes of a specified object instance the local federate has been offered ownership. This service can be used to sense the information that was supplied to the local federate by a previous requestAttributeOwnershipAssumption callback to its Federate Ambassador. Since SLX-Federates cannot access this information directly, the query function is supplied.

### HLA IF SPECIFICATION

This function provides access to the information that is supplied by the “Request Attribute Ownership Assumption ” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_GetOfferedAttributes (
                    int       ObjectID)
         returning string(1024) dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which the offered instance-attributes are being queried

### DESCRIPTION

This function queries the information that has been provided by a previous requestAttribute-OwnershipAssumption callback. If such a callback occurs (most likely in response to a previous RTI_NegotiatedAttributeOwnershipDivestiture() request by a remote federate), the object instance for which attributes have been offered is placed into the set “OwnershipAssumptionRequests”, which is part of the SLX_StateObject. The SLX-Federate is then supposed to use RTI_GetOfferedAttributes to query as for which set of attributes it has been offered ownership.

### RETURN VALUES

The function returns a string containing a comma-separated list of the attribute names which the SLX-Federate has been offered.

### SEE ALSO

*RTI_AttributeOwnershipAcquisition, RTI_NegotiatedAttributeOwnershipDivestiture*

## B.1.5.11 RTI_GetReleasedAttributes()

**RTI 1.3**

### ABSTRACT

This service is used to query for which instance-attributes of a specified object instance the local federate has successfully released ownership. This service can be used to sense the information that was supplied to the local federate by a previous attributeOwnershipDivestitureNotification callback to its Federate Ambassador. Since SLX-Federates cannot access this information directly, the query function is supplied.

### HLA IF SPECIFICATION

This function provides access to the information that is supplied by the “Attribute Ownership Divestiture Notification” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_GetReleasedAttributes (
                      int       ObjectID)
          returning string(1024) dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which the instance-attributes that have been released are queried

### DESCRIPTION

This function queries the information that has been provided by a previous attributeOwnership-DivestitureNotification callback. If such a callback occurs (most likely in response to a previous RTI_NegotiatedAttributeOwnershipDivestiture() request), the object instance for which attributes have been successfully released is placed into the set “OwnershipDivestitureNotification”, which is part of the SLX_StateObject. The SLX-Federate is then supposed to use RTI_GetReleasedAttributes to query as for which set of attributes it has been given ownership.

### RETURN VALUES

The function returns a string containing a comma-separated list of the attribute names which the SLX-Federate has successfully released.

### SEE ALSO

*RTI_AttributeOwnershipAcquisition, RTI_NegotiatedAttributeOwnershipDivestiture*

## B.1.5.12 RTI_GetUnavailableAttributes()

**RTI 1.3**

### ABSTRACT

This service is used to query for which instance-attributes of a specified object instance the local federate has not been given ownership. This service can be used to sense the information that was supplied to the local federate by a previous attributeOwnershipUnavailable callback to its Federate Ambassador. Since SLX-Federates cannot access this information directly, the query function is supplied.

### HLA IF SPECIFICATION

This function provides access to the information that is supplied by the “Attribute Ownership

Unavailable” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_GetOwnershipUnavailable (
                      int       ObjectID)
          returning string(1024) dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which the unavailable instance-attributes are being queried

### DESCRIPTION

This function queries the information that has been provided by a previous attributeOwnership-Unavailable callback. If such a callback occurs (this can only occur in response to a previous RTI_AttributeOwnershipAcquisitionIfAvailable() request of the local federate), the object instance for which attributes can not be acquired is placed into the set “OwnershipUnavailable”, which is part of the SLX_StateObject. The SLX-Federate is then supposed to use RTI_GetUnavailableAttributes to query as for which set of attributes it has not been given ownership.

### RETURN VALUES

The function returns a string containing a comma-separated list of the attribute names which were not available for acquisition.

### SEE ALSO

*RTI_AttributeOwnershipAcquisitionIfAvailable*

## B.1.5.13 RTI_IsAttributeOwnedByFederate()

**RTI 1.3**

### ABSTRACT

This service queries the LRC to determine whether a specified instance-attribute of a specified object instance is currently owned by the local federate. The RTI 1.0 implementation of this service is named RTI_AttributeIsOwnedByFederate and is discussed in a separate section.

### HLA IF SPECIFICATION

This function realizes the “Is Attribute Owned By Federate” Ownership Management service.

### SYNOPSIS

```
procedure RTI_IsAttributeOwnedByFederate (
                      int       ObjectID
                    string(*) AttributeName)
          returning boolean dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which instance-attribute ownership is being queried

*AttributeName*

Name of the instance attribute of the object instance for which ownership is being queried

### DESCRIPTION

This function may be used to synchronously determine whether a specified instance-attribute is owned by the local federate. A positive (*true*) response indicates that the local federate owns the specified instance attribute. A negative (false) response indicates, that the specified instance attribute is unowned, non-existent, owned by a remote federate, or owned by the RTI.

Note that instance-attributes that have been the subject of an outstanding RTI_NegotiatedAttribute-OwnershipDivestiture service invocation are still considered owned by the divesting federate until the delivery of a attribtueOwnershipDivestitureNotification() callback. Please refer to the description of RTI_NegotiatedAttributeOwnershipDivestiture for how a SLX-Federate can sense the occurrence of such a callback.

### RETURN VALUES

A successful invocation of this service returns TRUE if the specified instance-attribute is owned by the local federate, otherwise it returns false.

### SEE ALSO

*RTI_QueryAttributeOwnership*

## B.1.5.14 RTI_NegotiatedAttributeOwnershipDivestiture()

**RTI 1.3**

### ABSTRACT

This service initiates an attempt to release ownership of a specified set of instance-attributes for a specified object instance. In absence of an acquiring federate, the instance-attributes will continue to be owned by the divesting federate.

### HLA IF SPECIFICATION

This function provides access to the “Negotiated Attribute Ownership Divestiture” Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_NegotiatedAttributeOwnershipDivestiture (
                     int       ObjectID
                     string(*) AttributeList,
                     string(*) theTag)
          returning boolean dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance whose instance-attributes are to be divested

*AttributeList*

List of the names of the instance-attributes to divest

*theTag*

a string that is passed to resulting invocations of requestAttributeOwnershipAssumption(); this argument is not interpreted by the RTI and may be used to communicate federation-specific information about the divestiture request.

Note: The argument is only provided for compatibility reasons for co-operation with non-SLX federates. The argument can only be sent along with this call, in the resulting requestAttributeOwnershipAssumption() call this argument will be ignored.

### DESCRIPTION

This function initiates a negotiated ownership divestiture of the specified instance-attributes. For an instance-attribute to be valid subject of a RTI_NegotiatedAttributeOwnershipDivestiture() invocation, it must be

- currently owned by the federate
- not already the subject of an outstanding RTI_NegotiatedAttributeOwnershipDivestiture() request.

If an instance-attribute that is subject of a RTI_NegotiatedAttributeOwnershipDivestiture() service invocation is the subject of a currently outstanding RTI_AttributeOwnershipAcquisition() request by one or more remote federates, ownership will be immediately transferred to a requesting federate. If multiple federates have outstanding requests for the same instance attribute, ownership will be transferred to the federate whose request was received most recently.

For instance-attributes that are not already the subject of an acquisition request, ownership divestiture will be coordinated with the federation to locate an acquiring federate, as follows:

1. Each remote federate will receive a requestAttributeOwnershipAssumption() callback for any divesting instance-attributes whose corresponding class-attributes are published by the remote federate. An SLX federate will be informed about the occurrence of this callback by placing the associated object into the set “OwnershipAssumptionRequests”. This set is part of the SLX_StateObject. SLX federates are then expected to use the query function RTI_GetOfferedAttributes() to determine the attributes being offered.

2. One or more federates may respond to the assumption request. SLX federates ma do so using the RTI_AttributeOwnershipAcquisiton() or RTI_AttributeOwnershipAcqusitionIfAvailable() service.
3. The LRC of the divesting federate will transfer ownership of an instance-attribute to the first remote federate for which a response was received.

After step 3 has been completed, the divesting federate will be informed via the attributeOwnershipDivestitureNotification callback to its Federate Ambassador that it has successfully been released of (parts of) the instance-attributes it wishes to release. A federate can use the "OwnershipDivestitureNotification" attribute of the SLX_StateObject and the query function RTI_GetReleasedAttributes() to sense such a notification. Please refer to the description of the SLX_StateObject and the query function RTI_GetReleasedAttributes to learn more about how an SLX federate can sense this callback.

### RETURN VALUES

A return value of TRUE indicates that a negotiated divestiture of the specified instance-attributes has been initiated. The federate may be notified about the successful divestiture of some or all of the divested attributes via the SLX_StateObject.

### SEE ALSO

*RTI_CancelNegotiatedAttributeOwnershipDivestiture,*
*RTI_UnconditionalAttributeOwnershipDivestiture*

## B.1.5.15 RTI_UnconditionalAttributeOwnershipDivestiture()

**RTI 1.3**

### ABSTRACT

This service releases ownership of a specified set of instance-attributes for a specified object instance. The attributes immediately become unowned and are available for acquisition by any federate.

### HLA IF SPECIFICATION

This function provides access to the "Unconditional Attribute Ownership Divestiture" Ownership Management service of the RTI.

### SYNOPSIS

```
procedure RTI_UnconditionalAttributeOwnershipDivestiture (
                    int       ObjectID
                  string(*) AttributeList)
         returning boolean dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance whose instance-attributes are to be divested

*AttributeList*

List of the names of the instance-attributes to divest

### DESCRIPTION

This function immediately releases ownership of the specified instance-attributes. The behavior and involved procedures of this function are equal to the RTI_NegotiatedAttributeOwnershipDivestiture service with respect to the following items:

- Since no attributeOwnershipDivestitureNotification callback is made to the Federate Ambassador, the SLX_StateObject remains unchanged. No further notification about the divestiture is made.
- If no federates qualify for assuming ownership of the specified instance-attributes, they remain unowned.

### RETURN VALUES

A return value of TRUE indicates that the specified instance-attributes are no longer owned by the federate.

### SEE ALSO

*RTI_AttributeOwnershipReleaseResponse, RTI_NegotiatedAttribtueOwnershipDivestiture*

## B.1.5.16 RTI_QueryAttributeOwnership()

**RTI 1.3**

### ABSTRACT

This service determinates which federate (if any) holds the attribute ownership token for a given instance-attribute.

### HLA IF SPECIFICATION

This function provides access to the “Query Attribute Ownership” Ownership Management service of the RTI. The version for SLX differs in that it always provides a synchronous answer, while the original RTI ambassador method may provide asynchronous results or no results at all. The behavior of the SLX version of this function is achieved by bundling the RTI ambassador call to queryAttributeOwnership with invocations of the tick-method. Thus an answer will be obtained via one of the Federate Ambassador callbacks attributeIsNotOwned, attributeOwnedByRTI, attributeOwnershipUnavailable, or informAttributeOwnership.

### SYNOPSIS

```
procedure RTI_QueryAttributeOwnership (
                      int       ObjectID
                      string(*) AttributeName)
          returning string(1024) dll="rtislx13";
```

### ARGUMENTS

*ObjectID*

RTI ID of the object instance for which instance-attribute ownership is being queried

*AttributeName*

Name of the instance-attribute whose ownership is being queried

### DESCRIPTION

This function queries the federation as to the ownership status of a specified instance-attribute. If the LRC cannot provide the answer synchronously, a query is sent to the federation. The function waits for a certain amount of time for an answer to occur (at least 100 ms). If the federation does not respond during this time interval, the function returns “QueryTimeout”. This can indicate that the instance-attribute no longer exists in the federation (i.e., a federate has crashed or resigned without releasing ownership).

### RETURN VALUES

A successful invocation of this service returns the federate handle of the federate that holds the ownership token of the instance-attribute. Since the return value is provided as a string, the user may need to convert the return value to an integer, if appropriate.

The return value may also contain one of the following values:

- The value “QueryTimeout” is returned, if no answer from the federation within the timeout interval of 100 ms was obtained.
- The value “OwnershipUnavailable” is returned, if an invalid object or attribute was specified as argument to the function RTI_QueryAttributeOwnership.
- The values “OwnedByRTI” and “NotOwned” are returned, if the answer from the federation occurred via the Federate Ambassador callbacks “attributeOwnedByRTI” and “attributeNotOwned”, respectively.

### SEE ALSO

*RTI_IsAttributeOwnedByFederate*

# B.1.6 SPECIAL SUPPORT SERVICES FOR SLX

## B.1.6.01 RTI_ReflectNextBufferedInteraction()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service is used to retrieve the next buffered interaction for a given interaction class

### HLA IF SPECIFICATION

This function does not correspond to any function in the HLA Interface Specification.

### SYNOPSIS

```
procedure    RTI_ReflectNextBufferedInteraction(
                    string(*) InteractionClassName )
             dll="slxrti10"; //resp. "slxrti13"
```

### ARGUMENTS

*InteractionClassName*

Name of the interaction class for which the next buffered interaction is to be reflected

### DESCRIPTION

This function can be used to retrieve the next buffered interaction for a given interaction class. Per default, incoming interactions will be buffered, if, and only if, during any invocation of a time advancement service, more than one interaction is received for a given interaction class. This behavior is necessary because if there was no such buffering mechanism, interactions with the same time stamp and the same interaction class could overwrite each other when they are stored in the associated SLX object instance.

An SLX model is notified about the fact that an interaction has been buffered by placing the SLX object class associated with this interaction into the set “BufferedInteractionClasses” which is part of the SLX_StateObject. If this is the case, the interaction can be retrieved by calling RTI_ReflectNextBufferedInteraction.

Interaction buffering can be turned off by calling SetInteractionBufferMode.

### RETURN VALUES

None.

### SEE ALSO

*SetInteractionBufferMode*

## B.1.6.02 RTI_Tick()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service passes control from a federate to its LRC. Under normal conditions the SLX-HLA-Interface performs issues tick calls automatically during time advance functions and at other occasions where necessary (e.g., in conjunction with ownership management services). RTI_Tick is provided for receiving data without a time-advancement service being in progress and for experimental purposes.

### HLA IF SPECIFICATION

This function provides access to the tick method of the RTI ambassador.

### SYNOPSIS

```
procedure    RTI_Tick()
             returning int dll="slxrti10";  //resp. "slxrti13"
```

### ARGUMENTS

None.

### DESCRIPTION

The RTI_Tick service temporarily passes execution control from the federate to the LRC. The LRC will perform periodic federation maintenance and process incoming traffic from the network. Uder normal conditions the SLX-HLA-Interface issues tick calls automatically.

### RETURN VALUES

The return value is zero if no further processing needs to be done by the LRC and larger than zero if not.

### SEE ALSO

*RTI_NextEventRequest, RTI_NextEventRequestAvailable, RTI_TimeAdvanceRequest, RTI_TimeAdvanceRequestAvailable, RTI_EnableAsynchronousDelivery*

## B.1.6.03 SetTickSleepInterval()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service can be used to override the default tick sleep interval applied by the SLX-HLA-Interface (which is set to 10 ms).

### HLA IF SPECIFICATION

This function does not correspond to any function in the HLA Interface Specification.

### SYNOPSIS

```
procedure SetTickSleepInterval (
                     int       SleepInterval)
          dll="rtislx10"; //resp. "rtislx13"
```

### ARGUMENTS

*SleepInterval*

The new sleep interval in Milliseconds that will be applied

### DESCRIPTION

This function sets the sleep interval that will be used in the invocation of Sleep() after a series of tick-calls is finished and no events remain to be processed. This is usually the case, when a time advancement function has been called and the federate is waiting for other federates to advance their logical time.

The federate will only go into sleep mode, if a timeAdvanceGrant has not yet been received and if tick returns with no more events to process. If in that case no Sleep statements were applied, the processor usage would constantly be up at 100 % because of constantly calling tick() (active polling). If more then one federate is executed on one machine, the application of Sleep statements between multiple invocations of tick can significantly speed up federation execution.

If each federate has a designated machine on its own, the Sleep interval can be set to zero without decreasing the overall federation performance. It can, in fact, slightly increase performance depending on the tick-interval that was used before. If tick intervals are too large, a timeAdvanceGrant might be ready for delivery while the federate is still sleeping, thus decreasing the federation performance. This can also apply for the situation of running multiple federates on one machine. If performance problems occur it is therefore useful to experiment with different sleep intervals.

### RETURN VALUES

None.

### SEE ALSO

*RTI_Tick, RTI_NextEventRequest, RTI_NextEventRequestAvailable, RTI_TimeAdvanceRequest, RTI_TimeAdvanceRequestAvailable*

## B.1.6.04 SetErrorMessageMode()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service can be used to set the operation mode of the SLX-HLA-Interface regarding runtime errors.

### HLA IF SPECIFICATION

This function does not correspond to any function in the HLA Interface Specification.

### SYNOPSIS

```
procedure    SetErrorMessageMode(
                boolean ErrorMessageMode
                )
             dll="slxrti10"; //resp. "slxrti13"
```

### ARGUMENTS

*ErrorMessageMode*

The boolean switch ErrorMessageMode indicates whether error message mode is turned on or off.

### DESCRIPTION

This function can be used to override the default operation mode of the SLX-HLA-Interface regarding runtime error messages. Per default, any runtime errors are prompted with a message box giving a textual description of the error (and in case of an RTI exception, the reason for this message). Only after the user presses OK in this error box, the error is reflected back into the SLX model via the return value of the function that had been called. The SLX model can then continue or abort the execution.

Under certain circumstances it may not be desirable to have error message boxes popping up (e.g., if all error handling is done inside the SLX model). In that case these error messages can be turned off by calling SetErrorMessageMode with “FALSE” as argument.

### RETURN VALUES

None.

## B.1.6.05 SetInteractionBufferMode()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service can be used to override the default mode in which interactions are received by the SLX-HLA-Interface and how (and when) they are promoted to SLX.

### HLA IF SPECIFICATION

This function does not correspond to any function in the HLA Interface Specification.

### SYNOPSIS

```
procedure    SetInteractionBufferMode(boolean ON_OFF)
             dll="slxrti10"; //resp. "slxrti13"
```

### ARGUMENTS

*ON_OFF*

The boolean switch ON_OFF indicates whether interaction buffering will be turned on or off.

### DESCRIPTION

This function can be used to override the default operation mode of the SLX-HLA-Interface regarding the buffering of received interactions. Per default, incoming interactions will be buffered, if, and only if, during any invocation of a time advancement service, more than one interaction is received for a given interaction class.

This behavior is necessary because if there was no such buffering mechanism, interactions with the

same time stamp and the same interaction class could be overwritten when they are stored in the associated SLX object instance.

If interaction buffering is turned off by calling SetInteractionBufferMode, and there will be more than one receiveInteraction callback during a time advance request for the same interaction class, the parameters received in the second receiveInteraction callback will overwrite the parameters received in the first one.

If interaction buffering is “on” (the default) and interactions have been buffered, they can be retrieved by calling RTI_ReflectNextBufferedInteraction(). The SLX model is notified about the fact that an interaction has been buffered by placing the SLX object class associated with this interaction into the set “BufferedInteractionClasses” which is part of the SLX_StateObject.

### RETURN VALUES

None.

### SEE ALSO

*RTI_ReflectNextBufferedInteraction*

## B.1.6.06 SetEndianConversion()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service can be used to override the mode in which object attributes and interaction parameters are transferred and received over the network.

### HLA IF SPECIFICATION

This function does not correspond to any function in the HLA Interface Specification.

### SYNOPSIS

```
procedure    SetEndianConversion(
                        boolean RTIAmb,
                        boolean FedAmb)
             dll="slxrti10"; //resp. "slxrti13"
```

### ARGUMENTS

*RTIAmb*

The boolean switch RTIAmb indicates whether endian conversions will be performed for outgoing data.

*FedAmb*

The boolean switch FedAmb indicates whether endian conversions will be performed for incoming data.

### DESCRIPTION

This function can be used to override the default operation mode of the SLX-HLA-Interface regarding endian conversions. Per default, no endian conversions will be performed.

Endian conversions are necessary if a federation consists of federates implemented on platforms which use different endian types. The Endianess of a platform relates to the order in which bytes for numerical values are stored (little endian vs. big endian).

HLA does not offer an automated mechanism to convert between the different endian types of federates. Therefore federation specific agreements have to be made regarding who performs endian conversions. The SLX-HLA-Interface allows the most flexible way of doing so. The user can select whether all outgoing data, all incoming data, or both will be converted.

### RETURN VALUES

None.

## B.1.6.07 ToggleDebuggingSwitches()

**RTI 1.0** **RTI 1.3**

### ABSTRACT

This service can be used to alter the operation mode of the SLX-HLA-Interface debug features.

### HLA IF SPECIFICATION

This function does not correspond to any function in the HLA Interface Specification.

### SYNOPSIS

```
procedure ToggleDebuggingSwitches(
            boolean SinlgeStepped,
            boolean FedAmbMessageMode,
            boolean SaveMode
    boolean SaveMode) dll="slxrti10";  //resp. "slxrti13"
```

### ARGUMENTS

*SingleStepped*

Boolean switch which determines whether to run the SLX-HLA-Interface in single step mode or not.

*FedAmbMessageMode*

Boolean switch which determines whether the federate ambassador of the SLX-HLA-Interface prompts each callback with a message box or not.

*SaveMode*

Boolean switch which determines whether to run the SLX-HLA-Interface in save mode or not.

### DESCRIPTION

This function can be used to override the default operation mode of the SLX-HLA-Interface regarding debug features. Initially these switches are set by calling RTI_Init. ToggleDebuggingSwitches can be used at a later point inside the model to override these settings. A detailed explanation of each switch can be found in the section about RTI_Init.

### RETURN VALUES

None.

### SEE ALSO

*RTI_Init*

## B.2 Documentation of the SLX_StateObject

The SLX_StateObject is a data structure which stores different information received via the federate ambassador of the SLX-HLA-Interface.

Since SLX does not allow the user to write callback functions inside SLX (which are required for implementing a federate ambassador) a mail box principle is used to transfer information from the federate ambassador to the SLX model. The SLX_StateObject is the data structure of this mail box. It contains all information which do not directly relate to attribute updates or interaction reflections. This information is stored directly in the associated SLX objects.

The SLX_StateObject has the following declaration:

```
class SLX_StateObject
{
   int            ObjectsDiscovered;
   string(50)     DiscoveredObjectClass;
   int            InteractionsReceivedCount;
   control set(*) ReceivedInteractionClasses;
   set(*)         BufferedInteractionClasses;
   int            AttributeUpdatesReceived;
   set(*)         UpdatedObjectClasses;
   boolean        SynchronizationPointAnnounced;
   string(256)    SynchronizationLabel;
   string(256)    SynchronizationTag;
   boolean        FederationSynchronized;
   boolean        SaveRequested;
   string(256)    SaveLabel;
   double         SaveRequestedForTime;
   boolean        RestoreRequested;
   string(256)    RestoreLabel;
   string(50)     RestoreCompleted;
   set(*)         OwnershipReleaseRequests;
   set(*)         OwnershipDivestitureNotifications;
   set(*)         OwnershipAssumptionRequests;
   set(*)         OwnershipAcquisitionNotifications;
   set(*)         OwnershipUnavailable;
   set(*)         OwnershipAcquisitionCancellation;
 }
```

The following section gives an explanation for each attribute of the SLX_StateObject and gives references to RTI_-functions which are possibly associated with the attribute.

### ObjectsDiscovered

**ABSTRACT**

Counter for the number of discovered object instances.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

Each discoverObjectInstance callback increases this counter.

**ASSOCIATED RTI_-FUNCTION**

RTI_RegisterGhostedObject should be called to associate an SLX object instance with the discovered object instance.

**CONVENTIONS**

The SLX model should decrease the counter after a successful invocation of RTI_RegisterGhostedObject.

## DiscoveredObjectClass

**ABSTRACT**

Name of the oldest discovered object class which is not ghosted yet.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

Each discoverObjectInstance invocation adds the name of the discovered object class to a list of objects which have been discovered but not yet ghosted via RTI_RegisterGhostedObject. The name of the oldest object in that list is shown in the attribute DiscoveredObjectClass.

**ASSOCIATED RTI_-FUNCTION**

RTI_RegisterGhostedObject should be called to associate an SLX object instance with the discovered object instance.

**CONVENTIONS**

The SLX model should not change this attribute. It gets adjusted automatically whenever RTI_RegisterGhostedObject is called.

## InteractionsReceivedCount

**ABSTRACT**

Counter for the reflect interaction invocations received by the federate ambassador.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

Each reflectInteraction invocation increases this attribute by 1.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The SLX model should decrease this counter if it is used within the model, e.g., to detect that an interaction has been received.

## ReceivedInteractionClasses

**ABSTRACT**

Set which contains pointers to all interaction classes which have been received.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

Each reflectInteraction invocation puts the pointer of the SLX associated with this interaction into this set.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The SLX model should remove the pointer to the SLX object representing the interaction after the interaction has been processed by the model. If the set is not used within the model there is no need to remove the pointer.

## BufferedInteractionClasses

**ABSTRACT**

Set which contains pointers to all buffered interaction classes.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

Per default, incoming interactions, which are received via the reflectInteraction method of the federate ambassador, will be buffered, if, and only if, during any invocation of a time advancement service, more than one interaction is received for a given interaction class. Pointers to these interaction classes are put into this set.

**ASSOCIATED RTI_-FUNCTION**

RTI_ReflectNextBufferedInteraction is used to retrieve the next interaction of a buffered class. SetInteractionBufferMode can be used to turn interaction buffering on or off.

**CONVENTIONS**

The SLX model should not remove any members of this set. The members are adjusted automatically whenever RTI_ReflectNextBufferedInteraction is called, i.e., if all buffered interactions have been retrieved, the pointer to the class is removed automatically, otherwise it will stay in the set.

## AttributeUpdatesReceived

**ABSTRACT**

Counter for the received attribute updates.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

Each reflectAttributeValues callback increases this counter. Please note that within one callback more than one attribute update may be received, if updates are bundled. Therefore the value of this counter does not necessarily correspond to the number of updated instance attributes.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The SLX model should decrease the counter once the update has been processed. If the model does not use this counter to detect attribute updates (in most cases there are more convenient ways for doing this, e.g., by using control variables in the ghosted object classes) the counter can be safely ignored.

## UpdatedObjectClasses

**ABSTRACT**

Set containing pointers to all updated object instances.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The SLX object associated with any reflectAttributeValues callback will be placed into this set.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The SLX model should remove the pointer to the SLX object from this set after the update has been processed by the model. If the set is not used within the model there is no need to remove the pointer.

## SynchronizationPointAnnounced

**ABSTRACT**

Boolean switch which indicates the occurrence of a request to synchronize the federation.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

This switch will be set to true if the announceSynchronizationPoint callback is received by the federate ambassador. This callback will also set the attributes SynchronizationLabel and SynchronizationTag.

**ASSOCIATED RTI_-FUNCTION**

The SLX model should respond to the announcement of a synchronization point by calling the function RTI_SynchronizationPointAchieved, once the federate specific requirements for the synchronization point have been met.

An SLX federate can use RTI_RegisterFederationSynchronizationPoint to register a federation wide synchronization point.

**CONVENTIONS**

After detecting the announcement of a synchronization point via this attribute of SLX_StateObject, it is the models task to reset the attribute to FALSE.

## SynchronizationLabel

**ABSTRACT**

Label of the latest synchronization request.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

This string will be set by the announceSynchronizationPoint callback.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The model can reset this string at its discretion.

## SynchronizationTag

**ABSTRACT**

Tag supplied for the latest synchronization request.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

This string will be set by the announceSynchronizationPoint callback.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The model can reset this string at its discretion.

## FederationSynchronized

**ABSTRACT**

Boolean switch which indicates that all federates have completed a request to achieve a synchronization point.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The federationSynchronized callback will switch this attribute to TRUE.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The model can reset this attribute at its discretion.

## SaveRequested

**ABSTRACT**

Boolean switch which indicates the occurrence of a SaveRequest.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The initiateFederateSave callback will switch this attribute to TRUE. This callback will also set the attributes SaveRequestedForTime and SaveLabel.

**ASSOCIATED RTI_-FUNCTION**

RTI_FederateSaveBegun should be called to indicate that the federate has begun saving its internal state as requested. The functions RTI_FederateSaveNotComplete and RTI_FederateSaveComplete should be after the completion of the save-operation depending on its success.

**CONVENTIONS**

The model should reset this attribute after the save operation has been completed.

## SaveLabel

**ABSTRACT**

Label that is associated with the latest save request.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The initiateFederateSave callback will set this attribute.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The model can reset this attribute at its discretion.

## SaveRequestedForTime

**ABSTRACT**

Double value representing the time for which the save operation was requested. This value only exists in RTI 1.0.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The RTI 1.0 version of initiateFederateSave has the optional argument of a time stamp at which the save should take effect. This parameter is not really necessary since the RTI schedules the save initiation at the appropriate time. Therefore this parameter has been eliminated in RTI 1.3 and RTI 1.3NG. The SLX-HLA-Interface for these RTI versions sets the time stamp to –1.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The model can reset this attribute at its discretion.

## RestoreRequested

**ABSTRACT**

Boolean switch which indicates the occurrence of a restore request.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The initiateFederateRestore callback will switch this attribute to TRUE. This callback will also set the attribute RestoreLabel.

**ASSOCIATED RTI_-FUNCTION**

The functions RTI_FederateRestoreNotComplete or RTI_FederateRestoreComplete should be called after the completion of the restore operation depending on its success.

**CONVENTIONS**

The model should reset this attribute after the restore operation has been completed.

## RestoreLabel

**ABSTRACT**

Label that is associated with the latest restore request.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The initiateFederateRestore callback will set this attribute.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The model can reset this attribute at its discretion.

## RestoreCompleted

**ABSTRACT**

String which indicates the completion of a federation restoration.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The federationRestored and federationNotRestored callbacks will set this attribute to “Success” and “Failed”, respectively.

**ASSOCIATED RTI_-FUNCTION**

None.

**CONVENTIONS**

The model can reset this attribute (empty string) at its discretion, e.g., after a federation restoration has been completed.

## OwnershipReleaseRequests

**ABSTRACT**

Set which contains pointers to all objects for which an ownership release request has been received.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The requestAttributeOwnershipRelease callback will place the SLX object associated with the attribute which has been requested to be release into this set.

**ASSOCIATED RTI_-FUNCTION**

RTI_GetAttributesRequestedToRelease can be used to retrieve specific information as to which attributes of the object have been requested to be released.

**CONVENTIONS**

The model should remove the object from the set after responding to the request.

## OwnershipDivestitureNotifications

**ABSTRACT**

Set which contains pointers to all object instances for which an attribute ownership divestiture notification has been received.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The attributeOwnershipDivestitureNotification callback will place the SLX object associated with the notification into this set.

**ASSOCIATED RTI_-FUNCTION**

RTI_GetReleasedAttributes can be used to retrieve specific information about the notification.

**CONVENTIONS**

The model should remove the object from the set after adjusting its internal state.

## OwnershipAssumptionRequests

**ABSTRACT**

Set which contains pointers to all object instances for which an attribute ownership assumption request has been received.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The requestAttributeOwnershipAssumption callback will place the SLX object associated with the request into this set.

**ASSOCIATED RTI_-FUNCTION**

RTI_GetOfferedAttributes can be used to retrieve specific information about the request.

**CONVENTIONS**

The model should remove the object from the set after adjusting its internal state.

## OwnershipAcquisitionNotifications

**ABSTRACT**

Set which contains pointers to all object instances for which an attribute acquisition notification has been received.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The attributeOwnershipAcquisitionNotification callback will place the SLX object associated with the request into this set.

**ASSOCIATED RTI_-FUNCTION**

RTI_GetAcquiredAttributes can be used to retrieve specific information about the request.

**CONVENTIONS**

The model should remove the object from the set after adjusting its internal state.

## OwnershipUnavailable

**ABSTRACT**

Set which contains pointers to all object instances for which an attribute instance was not available for acquisition via the RTI_AttributeOwnershipAcquisitionIfAvailable() function.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The attributeOwnershipAcquisitionNotification callback will place the SLX object associated with the request into this set.

**ASSOCIATED RTI_-FUNCTION**

RTI_GetUnavailableAttributes can be used to retrieve specific information about the attributes which were not available for acquisition.

**CONVENTIONS**

The model should remove the object from the set after adjusting its internal state.

## OwnershipAcquisitionCancellation

**ABSTRACT**

Set which contains pointers to all object instances for which an attribute instance ownership acquisition has been successfully canceled.

**ASSOCIATED FEDERATE AMBASSADOR METHOD**

The attributeOwnershipAcquisitionNotification callback will place the SLX object associated with the request into this set.

**ASSOCIATED RTI_-FUNCTION**

RTI_GetCanceledAttributes can be used to retrieve specific information about the cancellation.

**CONVENTIONS**

The model should remove the object from the set after adjusting its internal state.

## B.3 Installation and Troubleshooting of the SLX-HLA-Interface

The SLX-HLA-Interface uses the RTI software provided by the Defense Modeling and Simulation Office (DMSO) of the U.S. Department of Defense. The SLX-HLA-Interface is currently available for RTI 1.0.3, RTI 1.3v7, and RTI 1.3 NGv3.

The RTI versions provided by DMSO are compatible with other RTI releases from DMSO for different platforms and languages within their respective RTI families.

Please note that at the current status of the HLA development there is no specified wire compatibility between RTI implementations of different vendors/developers. Please check this and the RTI implementation used by your potential HLA partners prior to the investment of manpower and resources.

### *B.3.1 Installation*

The SLX-HLA-Interface is implemented in form of a Window's Dynamic Link Library (DLL) and is provided in form of compiled code. This enables the user to create HLA federates without having to program in C++. The DLL itself is written in C++.

The SLX-HLA-Interface requires a proper installation of the DMSO RTI 1.0.3 (RTI 1.3v7 and RTI 1.3NGv3 respectively) on a Windows NT 4.0/2000 operating system. The installation under Windows NT 4 also requires the installation of service pack 4 or higher.

The RTI software has to be obtained separately via the HLA-homepage. A good way to test an RTI installation is to run a federation of at least two HelloWorld federates on the machine the RTI has been installed on.

The SLX-HLA-Interface consists of the following files:

| Files for RTI 1.0/RTI 1.3(NG) | Description |
|---|---|
| SLXRTI10.DLL/<br>SLXRTI13.DLL/<br>SLXRTI13NG.DLL | Library file that provides the connection to the RTI. The DLL implements the federate ambassador (which cannot be done inside SLX) and provides access to the RTI ambassador functions (by wrapping the C++-methods with normal C-functions which can be accessed via the DLL-interface of SLX) |
| SLXRTI10.SLX/<br>SLXRTI13.SLX/<br>SLXRTI13NG.SLX | Include or header-file which all SLX-federates have to import. This file defines the function prototypes of the functions contained in the SLXRTI1x.DLL. It also defines some of the basic data types that should be used in conjunction with the RTI. Especially important is the SLX_StateObject class, which is used by the federate ambassador of the SLX-HLA-Interface to notify the SLX model about some general information received from the RTI. |

Table 11: Files belonging to the installation the SLX-HLA-Interface

To install the package it is necessary to simply copy the files which match your RTI version to your SLX directory. For a proper execution of more than one federate it is necessary to increase the stacksize of SLX to at least 2048K. This can be done via the options menu of SLX.

### *B.3.2 Troubleshooting*

1) Stacksize

An error which is frequently encountered on a new installation is the failure to increase the SLX stacksize. For the SLX-HLA-Interface to work properly, the stacksize has to be at least 2048K. If your SLX models are also using other DLL's (e.g., Proof for Windows), you need to increase stacksize further. Problems resulting from failure to increase stacksize are not always obvious to notice. If federates behave strange in certain situations, it is always advisable to first check the stacksize.

2) Out-Of-Order-Delivery of TSO messages (RTI 1.3v7 only)

The RTI 1.3v7 still has message ordering problems when TCP/UDP bundling is turned on. Up to RTI 1.3v5 this option was turned off by default. RTI 1.3v7 per default turns this option on. It is therefore a good idea to disable this option right after installing the RTI. You can do this by opening the RTI.rid file located in the config subdirectory of your RTI installation with a normal text editor. Locate the entries:

```
;; TCP bundling on or off (1 || 0)
   (tcp_bundling_toggle 1)

 ;; UDP bundling on or off (1 || 0)
   (udp_bundling_toggle 1)
```

and replace them by the following:

```
;; TCP bundling on or off (1 || 0)
   (tcp_bundling_toggle 0)

 ;; UDP bundling on or off (1 || 0)
   (udp_bundling_toggle 0)
```

Message ordering should work fine now. According to DMSO's RTI help desk the entire problem is a matter of perspective:

„*It was decided that the problem is a matter of preceptive. After shipping RTI1.3V5, we received numerous complaints that bundling was disabled by default. A very vocal set of users seem to prefer performance to accuracy.*
*Ideally, the RTI would be able to bundle and maintain TSO event order, but that really can't be guaranteed since bundling forces messages to be held until the queue is full or timed out. With multiple federates withholding messages at different rates, the RTI has no way of knowing when all relevant messages have arrived. Any TSO message that arrives after a time advance is delivered as RO.*"

3) Memory leakage when running a federate multiple times

When you are running SLX federates multiple times (i.e., restarting them after they have finished without closing the SLX window), memory leakage problems can occur. Each time the federate is restarted, the amount of virtual memory used by SLX increases. It is therefore advisable to close the SLX environment from time to time.

4) Federates hang when restarting a federate (RTI 1.3 only)

In addition to the problem stated above federates that are immediately restarted after they have finished may appear to be hanging. This applies to RTI 1.3 only and seems to be a network specific problem. If the amount of time between the restarting of a federate is significantly large enough, the whole process works fine.

In addition to that, the FedExec process under RTI 1.3 behaves differently then the one from RTI 1.0. Federates are not automatically removed from a federation when they crash as it was the case under RTI 1.0. They have to be removed manually from the FedExec. There can also be problems with re-using an existing FedExec process multiple times. It may be necessary to frequently close the FedExec manually (just type "kill" into the log window).

5) Messages and Error Codes

The SLX-HLA-Interface performs all RTI-related exception and error handling. An exception will most likely be the result of an improper service invocation (e.g., misspelled attribute names, invalid update times)

Exceptions will be shown to the user per default by message boxes popping up. In addition to that, the exceptions are reflected into the SLX model via the return code of the wrapper function. Usually –1 or FALSE is returned if an error occurred. It is envisioned for future releases to provide more detailed error reflection using more differentiated return codes. Message boxes popping up can be turned off using SetErrorMessageMode.