A CLASS RTI::RTIAMBASSADOR

A Class RTI::RTIambassador

A.1 Federation Management

A.1.1 createFederationExecution()

1RTI.0 RTI 1.3

ABSTRACT

This method creates a named federation execution (FedExec) and registers it with the RTI executive (RtiExec). **The RTI 1.3 version of this method has an additional argument for specifying the FED file to use for the federation.**

HLA IF SPECIFICATION

This method realizes the "Create Federation Execution" Federation Management service as specified in the *HLA Interface Specification* (§2.1 in version 1.1; §4.2 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   createFederationExecution (
      const RTI::FederationExecutionName
         executionName
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederationExecutionAlreadyExists,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,            ← RTI 1.0 Only
   RTI::SaveInProgress                ← RTI 1.0 Only
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   createFederationExecution (
      const char*   executionName,    ← Changes Type
      const char*   FED               ← RTI 1.3 Only
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederationExecutionAlreadyExists,
   RTI::RTIinternalError
)
```

ARGUMENTS

executionName

string specifying the name of the FedExec to create

FED **(RTI 1.3 Only)**

filename in the directory specified by the *RTI_CONFIG* environment variable from which to read the Federation Execution Data (FED)

DESCRIPTION

A federate uses `createFederationExecution()` to create a new federation executive (FedExec) process. [The FedExec is frequently referred to as "the federation". A FedExec is, essentially, an active federation.] Once the FedExec completes its initialization and informs the RTI Executive (RtiExec) of its existence, it is possible for federates to begin joining the new federation.

When `createFederationExecution()` completes, there is a brief period before the FedExec is ready to accept joining federates. As a result, an immediate call to `joinFederationExecution()` will probably fail. Applications should either (1) design a time delay between creating and attempting to join a new federation or (2) call `joinFederationExecution()` repeatedly until successful.

If several federates compete to create the FedExec, a race condition results. The first federate to request the new federation will succeed. Subsequent requests will encounter an exception.

RETURN VALUES

A non-exceptional exit from this method indicates that the RtiExec has approved the creation of the named Federation and that the corresponding FedExec has been forked.

A non-exceptional return does not guarantee that the federation has been successfully created. Errors that occur in the initialization of the FedExec (e.g. a bad path to the FedExec executable) are not detected. Output from the FedExec is logged to the file "`Xterm.#####`" (where ##### is a PID) in the current directory. The log should be consulted when attempting to diagnose problems with FedExec initialization.

WINDOWS® NT NOTES

The location of the executable that is forked to start the FedExec is "`%RTI_HOME%\bin\win32\fedex.exe`". Currently, output from the FedExec is not logged on Windows NT.

UNIX® NOTES

The location of the script that launches the FedExec is "`$RTI_HOME/bin/fedex.sh`".

RELEASE NOTES

RTI 1.3

The FED-file argument to `createFederationExecution()` is associated with the federation in the RtiExec and communicated to joining federates.

When migrating federates from RTI 1.0, derive the second argument by appending the string ".fed" to the first argument.

The type of RTI 1.0 string arguments is typically `const Identifier`, where *Identifier* is a typedef for `char*`. This results in an effective type of `char* const` instead of the desired `const char*`. In RTI 1.3 the typedefs have been eliminated and the type of string arguments is just `const char*`.

Note that the format of the RTI 1.3 FED file differs from the RTI 1.0 FED file. Consult the programmer's guide for details.

RTI 1.0

The 1.0 implementation does not allow an application-defined filename for FED information. The filename used is always the federation name with the ".fed" extension.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederationExecutionAlreadyExists

A FedExec for the given Federation has already been registered with the RtiExec. Note: FedExecs unregister with the RtiExec upon termination. However, an abnormal termination of a FedExec (e.g. a "kill -9") may result in a defunct FedExec that remains registered. In such cases, it is necessary to manually unregister the FedExec via the RtiExec console.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

destroyFederationExecution()

joinFederationExecution()

A.1.2 destroyFederationExecution()

RTI 1.0 RTI 1.3

ABSTRACT

This service unregisters a named federation and shuts down the associated FedExec. **The syntax of this method changes slightly from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Destroy Federation Execution" Federation Management service as specified in the *HLA Interface Specification* (§2.2 in version 1.1; §4.3 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   destroyFederationExecution (
       const RTI::FederationExecutionName
           executionName
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederatesCurrentlyJoined,
   RTI::FederationExecutionDoesNotExist,
   RTI::RestoreInProgress,             ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                 ← RTI 1.0 Only
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   destroyFederationExecution (
       const char *executionName       ← Changes Type
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederatesCurrentlyJoined,
   RTI::FederationExecutionDoesNotExist,
   RTI::RTIinternalError
)
```

ARGUMENTS

executionName

string specifying the name of the FedExec to destroy

DESCRIPTION

A federate uses `destroyFederationExecution()` to tear down an FedExec. If the prerequisites for Federation destruction are met (i.e. there are no federates joined to the Federation), the FedExec notifies the RtiExec of its intention to shut down and then exits.

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.

RETURN VALUES

A non-exceptional return indicates that the FedExec has been successfully destroyed.

NOTE ON TYPES

The type of RTI 1.0 string arguments is typically `const Identifier`, where *Identifier* is a typedef for `char*`. This results in an effective type of `char* const` instead of the desired `const char*`. In RTI 1.3 the typedefs have been eliminated and the type of string arguments is just `const char*`.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederatesCurrentlyJoined

There are still federates joined in the Federation. All federates resigned (or be manually removed via the FedExec console) before the FedExec can be destroyed. The FedExec automatically removes non-existent federates from the federation (even if they fail to resign properly). It shouldn't be necessary to manually remove federates under normal circumstances.

RTI::FederationExecutionDoesNotExist

The RTI does not have a FedExec registered for the named Federation.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

createFederationExecution()

A.1.3 federateRestoreComplete()

RTI 1.3

ABSTRACT

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

HLA IF SPECIFICATION

This method (in conjunction with `federateRestoreNotComplete()`) realizes the "Federate Restore Complete" Federation Management service as specified in the *HLA Interface Specification* (§4.20 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   federateRestoreComplete ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreNotRequested,
   RTI::RTIinternalError
)
```

DESCRIPTION

A federate invokes `federateRestoreComplete()` to indicate the successful completion of a restoration of federate-managed state, as initiated by a `initiateFederateRestore()` callback. 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. In the interim, the federate must continue to `tick()` the RTI so that synchronization messages may be processed.

The title of this method is something of a misnomer, as the method indicates not only that the restoration completed, but that it completed successfully.

RETURN VALUES

A non-exceptional return indicates that the LRC will notify the federation of the completed restore and will restart the operation of the federate when the remainder of the federation finishes restoring.

MIGRATION NOTE

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

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreNotRequested
: There is no outstanding request for a federate restoration.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::FederateAmbassador
: initiateFederateRestore()

: federationRestored()

: federationNotRestored()

RTI::RTIambassador
: federateRestoreNotComplete()

: requestFederationRestore()

A.1.4 federateRestoreNotComplete()

RTI 1.3

ABSTRACT

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

HLA IF SPECIFICATION

This method (in conjunction with `federateRestoreComplete()`) realizes the "Federate Restore Complete" Federation Management service as specified in the *HLA Interface Specification* (§4.20 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   federateRestoreNotComplete ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreNotRequested,
   RTI::RTIinternalError
)
```

DESCRIPTION

A federate invokes `federateRestoreNotComplete()` to indicate the unsuccessful completion of a restoration of federate-managed state, as initiated by a `initiateFederateRestore()` callback. The federate may not resume normal operation until it receives a `federationNotRestored()` callback to indicate that the federation-wide restoration attempt has concluded. In the interim, the federate must continue to `tick()` the RTI so that synchronization messages may be processed.

The only special action taken by RTI 1.3 based on the fact that a restore was not successful is that the `federationNotRestored()` callback will be used to restart the federates after the federation-wide restore is complete. Failure of one federate to restore does not prevent the rest of the federation from continuing to restore normally.

The title of this method is something of a misnomer, as it actually indicates that the restoration has completed, only not successfully.

RETURN VALUES

A non-exceptional return indicates that the LRC will notify the federation of the completed restore and will restart the operation of the federate when the remainder of the federation finishes restoring.

MIGRATION NOTE

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

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreNotRequested
: There is no outstanding request for a federate restoration.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::FederateAmbassador
: initiateFederateRestore()

: federationNotRestored()

RTI::RTIambassador
: federateRestoreComplete()

: requestFederationRestore()

A.1.5 federateSaveBegun()

RTI 1.0 RTI 1.3

ABSTRACT

This service informs the RtiExec that the calling federate has begun saving its internal state as per an `initiateFederateSave()` request. **The syntax of this service changes slightly between RTI 1.0 and RTI 1.3.**

HLA IF SPECIFICATION

Realizes the "Federate Save Begun" Federation Management service as specified in the *HLA Interface Specification* (§2.13 in version 1.1; §4.13 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   federateSaveBegun ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress,        ← RTI 1.0 Only
   RTI::SaveNotInitiated
)

// RTI 1.0 Only
void
RTI::RTIambassador::
   federateSaveBegun (
       RTI::FederationTime theTime
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidFederationTime,
   RTI::RTIinternalError,
   RTI::SaveNotInitiated,
   RTI::UnimplementedService   ← RTI 1.0 Only
);

// RTI 1.3 Only
void
RTI::RTIambassador::
   federateSaveBegun ( )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveNotInitiated
)
```

DESCRIPTION

A federate uses `federateSaveBegun()` to signal that it has begun saving its state in compliance with an outstanding `initiateFederateSave()` request. The federate is expected to complete the save as soon as possible after the initiation of the save. The federate is unable to advance in time (and therefore receive time-stamp-ordered events) or utilize any other service that would change the internal state of the RTI until the completion of the save.

The federate should subsequently notify the RTI via `federateSaveComplete()` on completion of a successful save or `federateSaveNotComplete()` on completion of a failed save. Currently, the RTI will save its internal state and then wait for all federates to complete their saves.

The RTI is responsible for saving and restoring its internal state. However, the RTI does not define a format or provide any facility for federates to save their external state. It is up to the federation developers to implement their own save and restore mechanisms.

RELEASE NOTES

RTI 1.0

In RTI 1.0, `federateSave[Not]Complete()` are named `federateSave[Not]Achieved()`.

Also in RTI 1.0, `federate[Not]Achieved()` block while the LRC saves its internal state and waits for the federation-wide save to complete.

RTI 1.3

The 1.3 implementation does not utilize this information in any way; in fact, the `federateSaveBegun()` call may be omitted entirely. However, federates should invoke this service to assure compliance with all RTI implementations.

In RTI 1.3, `federateSave[Not]Complete()` do not block; instead, the federate remains suspended until restarted by a `federation[Not]Saved()` callback.

RETURN VALUES

A non-exceptional return indicates that the RTI acknowledges the beginning of the federate save and that the federate should proceed to save its state.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::InvalidFederationTime

The specified time does not correspond to the valid time of an outstanding requested save.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::SaveNotInitiated

There are no current outstanding save requests of the local federate.

RTI::UnimplementedService **(RTI 1.0 Only)**

This exception is never thrown.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::

initiateFederateSave()

RTI::RTIambassador::

federateSaveAchieved()
federateSaveNotAchieved()

RTI 1.3

RTI::FederateAmbassador

initiateFederateSave()

federationSaved()

federationNotSaved()

RTI::RTIambassador

federateSaveComplete()

federateSaveNotComplete()

A.1.6 federateSaveAchieved()

RTI 1.0

ABSTRACT

This service notifies the RtiExec that the federate has successfully completed a requested save. **The semantics of federation save and restore have changed substantially from RTI 1.0 to RTI 1.3. This service is named `federateSaveComplete` in RTI 1.3; the 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method (in conjunction with `federateSaveNotAchieved()`) realizes the "Federate Save Achieved" Federation Management service as specified in the *HLA Interface Specification* (§2.14 in version).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   federateSaveAchieved ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RTIinternalError,
   RTI::SaveNotInitiated
)
```

DESCRIPTION

A Federate uses `federateSaveAchieved()` to signal a successfully completed save. Saves are conducted on receipt of an `initiateFederateSave()` request and begin with the `federateSaveBegun()` signal.

The `federateSaveAchieved()` call *blocks* until all other federates and the RTI have completed their saves (i.e., successfully or unsuccessfully). **The RTI attempts to restore** its internal state from the file "RTIxxx-n.sav" where "xxx" is the save label and "n" is the federate's federate handle. These files will be located in the configuration directory (e.g., `$RTI_CONFIG` on UNIX® platforms and `%RTI_CONFIG%` on Windows® NT platforms).

Upon return from this method, the Federate's logical time will continue advancing as prescribed by the time-advance service in effect at the time of the save.

RETURN VALUES

A non-exceptional return indicates that all federate saves and the RTI save have completed and that the Federate should resume advancement of the Federate's logical time.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveNotInitiated

There is no outstanding request for a federate save or the Federate failed to indicated the beginning of its save (i.e., via the `federateSaveBegun()` method).

SEE ALSO

RTI::FederateAmbassador::
initiateFederateSave()

RTI::RTIambassador::
federateSaveBegun()
federateSaveNotAchieved()
requestFederationSave()
requestRestore()

A.1.7 federateSaveComplete()

RTI 1.3

ABSTRACT

This service notifies the federation that the federate has successfully completed a requested save. **The semantics of federation save and restore have changed substantially from RTI 1.0 to RTI 1.3. This service was named `federateSaveAchieved` in RTI 1.0; the 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method (in conjunction with `federateSaveNotComplete()`) realizes the "Federate Save Complete" Federation Management service as specified in the *HLA Interface Specification* (§4.14 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   federateSaveComplete ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveNotInitiated
)
```

DESCRIPTION

A federate invokes `federateSaveComplete()` to indicate the successful completion of a save of federate-managed state, as initiated by an `initiateFederateSave()` callback. The federate may not resume normal operation until it receives a `federationSaved()` or `federationNotSaved()` callback to indicate that the federation-wide save attempt has concluded. In the interim, the federate must continue to `tick()` the LRC so that synchronization messages may be processed.

When the LRC receives the indication that all federates participating in the save have completed saving their federate state, it will save the state of the LRC internal components to a file. The filename the state is written to is given by **$RTI_SAVE_PATH/FED/fedex/label/type/handle**, where

- **$RTI_SAVE_PATH** is an environment variable
- **FED** is the FED filename as passed to `createFederationExecution()`
- **fedex** is the federation execution name
- **type** is the federate name as passed to `joinFederationExecution()`; this is used to match saved states to the correct type of application at restore-time
- **handle** is the current federate handle

When the LRC receives the indication that all other participating LRCs have saved their internal state, the federate will be restarted using the `federationSaved()` or `federationNotSaved()` callback.

The title of this method is something of a misnomer, as the method indicates not only that the save completed, but that it completed successfully.

RETURN VALUES

A non-exceptional return indicates that the LRC will notify the federation of the save completion and will restart the operation of the federate when the remainder of the federation has saved.

MIGRATION NOTE

Note that under the 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.

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveNotInitiated
: There is no outstanding request for a federate save or the federate failed to indicate the beginning of its save (i.e., via the `federateSaveBegun()` method).

SEE ALSO

RTI::FederateAmbassador
: initiateFederateSave()

: federationSaved()

: federationNotSaved()

RTI::RTIambassador
: federateSaveNotComplete()

: requestFederationSave()

A.1.8 federateSaveNotAchieved()

RTI 1.0

ABSTRACT

This service notifies the RtiExec that the federate was unsuccessful in its attempt to save, but has completed the attempt. **The semantics of federation save and restore have changed substantially from RTI 1.0 to RTI 1.3. This service is named `federateSaveNotComplete` in RTI 1.3; the 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method (in conjunction with `federateSaveAchieved()`) realizes the "Federate Save Achieved" Federation Management service as specified in the *HLA Interface Specification* (§2.14 in version).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   federateSaveNotAchieved ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RTIinternalError,
   RTI::SaveNotInitiated
)
```

DESCRIPTION

A Federate uses `federateSaveNotAchieved()` to signal an unsuccessfull, but completed save attempt. Saves are conducted on receipt of an `initiateFederateSave()` request and begin with the `federateSaveBegun()` signal.

The `federateSaveNotAchieved()` call *blocks* until all other federates and the RTI have completed their saves (i.e., successfully or unsuccessfully). **The RTI attempts to restore** its internal state from the file "RTIxxx-n.sav" where "xxx" is the save label and "n" is the federate's federate handle. These files will be located in the configuration directory (e.g., `$RTI_CONFIG` on UNIX® platforms and `%RTI_CONFIG%` on Windows® NT platforms).

The RtiExec 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).

Upon return from this method, the Federate's logical time will continue advancing as prescribed by the time-advance service in effect at the time of the save.

RETURN VALUES

A non-exceptional return indicates that all federate saves and the RTI save have completed and that the Federate should resume advancement of the Federate's logical time.

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveNotInitiated
: There are no current outstanding save requests of the local federate.

SEE ALSO

RTI::FederateAmbassador::
: initiateFederateSave()

RTI::RTIambassador::
: federateSaveAchieved()
federateSaveBegun()
requestFederationSave()
requestRestore()

A.1.9 federateSaveNotComplete()

RTI 1.3

ABSTRACT

This service notifies the federation that the federate has failed to successfully complete a requested save. **The semantics of federation save and restore have changed substantially from RTI 1.0 to RTI 1.3. This service was named `federateSaveNotAchieved` in RTI 1.0; the 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method (in conjunction with `federateSaveComplete()`) realizes the "Federate Save Complete" Federation Management service as specified in the *HLA Interface Specification* (§4.14 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   federateSaveNotComplete ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveNotInitiated
)
```

DESCRIPTION

A federate invokes `federateSaveNotComplete()` to indicate the unsuccessful completion of a save of federate-managed state, as initiated by an `initiateFederateSave()` callback. The federate may not resume normal operation until it receives a `federationNotSaved()` callback to indicate that the federation-wide save attempt has concluded. In the interim, the federate must continue to `tick()` the RTI so that synchronization messages may be processed.

The only special action taken by RTI 1.3 based on the fact that a save was not successful is that the `federationNotSaved()` callback will be used to restart the federates after the federation-wide save is complete. Failure of one federate to save does not prevent the rest of the federation from continuing to save normally.

When the LRC receives the indication that all federates participating in the save have completed saving their federate state, it will save the state of the LRC internal components to a file. The filename the state is written to is given by **$RTI_SAVE_PATH/FED/fedex/label/type/handle**, where

- **$RTI_SAVE_PATH** is an environment variable
- **FED** is the FED filename as passed to `createFederationExecution()`
- **fedex** is the federation execution name
- **type** is the federate name as passed to `joinFederationExecution()`; this is used to match saved states to the correct type of application at restore-time
- **handle** is the current federate handle

When the LRC receives the indication that all other participating LRCs have saved their internal state, the federate will be restarted using the `federationNotSaved()` callback.

The title of this method is something of a misnomer, as the method indicates that a save completed, only not successfully.

RETURN VALUES

A non-exceptional return indicates that the LRC will notify the federation of the save completion and will restart the operation of the federate when the remainder of the federation has saved.

MIGRATION NOTE

Note that under the 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.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveNotInitiated

There are no current outstanding save requests of the local federate.

SEE ALSO

RTI::FederateAmbassador

initiateFederateSave()

federationNotSaved()

RTI::RTIambassador

federateSaveComplete()

requestFederationSave()

A.1.10 joinFederationExecution()

RTI 1.0 RTI 1.3

ABSTRACT

This service requests permission to participate in a named federation and initializes the RTI ambassador with federation-specific data. **The types of the arguments change slightly from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Join Federation Execution" Federation Management service as specified in the *HLA Interface Specification* (§2.3 in version 1.1; §4.4 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::FederateHandle
RTI::RTIambassador::
   joinFederationExecution (
       const RTI::FederateName yourName
       const RTI::FederationExecutionName
           executionName
       RTI::FederateAmbassadorPtr
           federateAmbassadorReference
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::CouldNotOpenFED,
   RTI::ErrorReadingFED,
   RTI::FederateAlreadyExecutionMember,
   RTI::FederationExecutionDoesNotExist,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
RTI::FederateHandle
RTI::RTIambassador::
   joinFederationExecution (
       const char    *yourName,          ← Changes Type
       const char    *executionName,     ← Changes Type
RTI::FederateAmbassadorPtr     ← Changes Type
           federateAmbassadorReference
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::CouldNotOpenFED,
   RTI::ErrorReadingFED,
   RTI::FederateAlreadyExecutionMember,
   RTI::FederationExecutionDoesNotExist,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

yourName

a string specifying the name by which the federate will be known to the federation

executionName

a string specifying the name of the federation to join

federateAmbassadorReference

a pointer to an instance of a federate-supplied class implementing the *FederateAmbassador* abstract class

The calling federate is responsible for managing the storage associated with this object. The referenced object must persist until the federate resigns from the federation via `resignFederationExecution()`.

DESCRIPTION

This service negotiates entry into a named federation execution and initializes the RTI ambassador based on a Federation Execution Data (FED) file and the current execution status of the federation. The federation join process can be thought of as occurring in three phases, all of which complete during a successful invocation of `joinFederationExecution()`:

1. **The LRC establishes socket communications with global RTI processes.** A connection to the RtiExec has already been established during the construction of the *RTIambassador* object. This connection is used to query the RtiExec for a reference to the FedEx for the named federation. This reference includes the hostname and port number of the FedEx, to which the LRC attempts to connect. Once this connection is established, the FedEx communicates to the joining LRC information about the multicast channels being used for the federation. The LRC then subscribes to the appropriate multicast channels.

2. **The LRC initializes its RTI ambassador based on the federation-specific configuration found in the FED file.** The location and specific contents of the FED differ by platform and RTI version; see release notes below for details. In general, the FED file defines the hierarchy of object and interaction classes and the ordering and transportation policies for interactions and attributes. The structure of the FED file determines the assignment of RTI "handles" that are used to communicate names across the federation. It is therefore essential that all federates in a federation use an exact copy of the same FED file (if not the same physical file.)
   The Management Object Model defines a hierarchy of object and interaction classes that must appear in every FED file. Some RTI implementations also have mandatory object and interaction classes for RTI internal use. See "Federation Execution Data" in the RTI Programmer's Guide for more information on the FED.

3. **The LRC constructs the running state of the federation based on information obtained from the FedEx and the participating federates.** The FedEx communicates to the joining LRC the federate handle of all federates in the federation. To participate in the distributed time-management algorithm, the joining federate must obtain a logical time report from each federate currently participating in the federation. A solicitation message is broadcast to the federation and each of the remote LRCs will respond with the current logical time of its federate. The federate applications need not concern themselves with this process except that **`joinFederationExecution()` will block until all remote federates have ticked their LRCs sufficiently to allow for a response to the logical time solicitation.** This is one of several important reasons why federates should always call `tick()` as often as possible.
   The LRC also publishes and subscribes various MOM classes on behalf of the federate. A *Manager.Federate* object is instantiated to represent the state of the local federate. Essential MOM operation is automatically managed by the LRC without intervention by the federate.
   The FedEx and the remote federates collaborate to inform the joining federate of any outstanding save, restore, or synchronization point requests. The joining federate becomes a participant in any synchronization points that have not been fully achieved at the time of the join.

Note that an invocation of `createFederationExecution()` does not synchronously wait for the FedEx to set up and begin accepting connections. The federate creating the FedEx must account for this delay by introducing a delay between the create and the join, or by repeatedly invoking

`joinFederationExecution()` until it succeeds.

RETURN VALUES

The *FederateHandle* instance returned by this method is a numeric value that the RTI has associated with the joining federate. The handle is guaranteed to be unique to the federation during the lifetime of the federate, and is used in situations that require an individual federate to be uniquely identified. Among other things, this value is used to identify the federate as a sender/receiver of MOM objects and interactions.

RELEASE NOTES

RTI 1.0

The filename of the FED file is the supplied *FederateName* with a ".fed" extension. FED file location is operating system dependent (see below).

The RTI 1.0 uses a single multicast group per federation.

Identifiers in an RTI 1.0 FED file are case-sensitive. The Interface Specification was subsequently changed to stipulate that FED identifiers should be case-insensitive.

RTI 1.3

The filename of the FED file is communicated to the joining federate from the RtiExec and is established by the FED argument to `createFederationExecution()`.

In addition to the FedEx, RTI 1.3 also contains global processes serving as "reliable distributors". Connections may be established and broken between an LRC and reliable distributors before, during, or after the join process.

The RTI 1.3 uses potentially many multicast groups and ports per federation. See "Data Distribution Management" and "RTI Configuration" in the RTI Programmer's Guide.

The RTI 1.3 uses a checksum to enforce the use of the same FED file by all federates.

Attempts to join during a federation save or restore under RTI 1.3 will fail.

The restore functionality of RTI 1.3 uses federate names (i.e. the first argument to `joinFederationExecution()`) to map saved LRC states to currently active federates. As such, two applications should use the same name if and only if they could function correctly if the state of their LRCs were interchanged due to a save and subsequent restore. Usually, multiple instances of the same application on interchangeable platforms joined to the federation will share the same name.

WINDOWS® NT/95 NOTES

The expected location of the FED file is "`%RTI_CONFIG%\[FED-name]`".

UNIX® NOTES

The expected location of the FED file is "`$RTI_CONFIG/[FED-name]`".

NOTE ON TYPES

The type of RTI 1.0 string arguments is typically `const Identifier`, where *Identifier* is a typedef for `char*`. This results in an effective type of `char* const` instead of the desired `const char*`. In RTI 1.3 the typedefs have been eliminated and the type of string arguments is just `const char*`.

The *FederateAmbassador* class has changed substantially between RTI 1.0 and RTI 1.3; consult the Federate Ambassador reference for details.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::CouldNotOpenFED

The FED file could not be found.

RTI::ErrorReadingFED

The FED file was not in the correct format. This can occur if one of the classes or interactions used by the MOM Manager is missing or incorrect. [See the example FED files in the RTI distribution for the definitions of MOM data types.]

RTI::FederateAlreadyExecutionMember

The RTIambassador instance is already associated with a FedExec. An RTI ambassador may only be associated with one FedExec at a given time. Note: The same RTIambassador instance may be associated with different FedExec processes at different times and different RTIambassador instances may be associated with different Federation Executions at the same time.

RTI::FederationExecutionDoesNotExist

The RTI does not have a FedExec registered for the named Federation.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

createFederationExecution()

publishInteractionClass()

publishObjectClass()

resignFederationExecution()

subscribeInteractionClass()

RTI 1.0 Only

setTimeConstrained()

subscribeObjectClassAttribute()

turnRegulationOn()

RTI 1.3 Only

enableTimeConstrained()

enableTimeRegulation()

registerFederationSynchronizationPoint()

requestFederationRestore()

requestFederationSave()

subscribeObjectClassAttributes()

A.1.11 pauseAchieved()

RTI 1.0

ABSTRACT

This service informs the federation that the federate has suspended execution as per the most recent outstanding `initiatePause()` request. **The pause/resume capability of HLA 1.1 has been replaced by the more general "synchronization point" functionality of HLA 1.3.**

HLA IF SPECIFICATION

This method realizes the "Pause Achieved" Federation Management service as specified in the *HLA Interface Specification* (§2.7 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   pauseAchieved (
       const RTI::PauseLabel label
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::NoPauseRequested,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress,
   RTI::UnknownLabel
)
```

ARGUMENTS

label

a string indicating the name of the pause request to which the Federate is responding. [When a Federate requests a pause using the `requestPause()` method, a pause label is supplied. The Federation communicates this label to all federates via the `initiatePause()` callback.]

DESCRIPTION

A Federate uses `pauseAchieved()` to signal that it has successfully suspended execution in response to an initiate pause request. Pause requests are communicated via the *FederationAmbassador* callback method `initiatePause()`. The Federate should remain suspended (in accordance with the terms of the pause label) until instructed to resume via the `initiateResume()` callback.

RETURN VALUES

A non-exceptional return indicates that the RTI acknowledges notification of the Federate's successful suspension of execution.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NoPauseRequested

There is not an outstanding pause request.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::UnknownLabel

The passed pause label does not match the label associated with the most recent outstanding pause request.

SEE ALSO

RTI::FederateAmbassador::

initiatePause()
initiateResume()

RTI::RTIambassador::

requestPause()
requestResume()

A.1.12 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 featured in HLA 1.1.**

HLA IF SPECIFICATION

This method realizes the "Register Federation Synchronization Point" Federation Management service as specified in the *HLA Interface Specification* (§4.6 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   registerFederationSynchronizationPoint (
      const char *label,
      const char *theTag
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

void
RTI::RTIambassador::
   registerFederationSynchronizationPoint (
      const char                    *label,
      const char                    *theTag,
      const RTI::FederateHandleSet& syncSet
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

label

a string used to uniquely identify the synchronization point

theTag

an application-defined string passed to remotes when the synchronization point is announced; not interpreted by the RTI

syncSet

the subset of the participating federates to synchronize

DESCRIPTION

Synchronization points are a generalization of the pause/resume capabilities featured in early revisions of HLA. They 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 by the desired set of federates.

The two variants of this service differ in the set of federates that are to be included in the synchronization point computation:

- The three-argument variation schedules a synchronization point affecting only federates explicitly represented in the handle-set. This is known as a *specified* synchronization point.
- The two-argument version schedules a synchronization point that applies to all federates in the federation *including federates that join the federation while the synchronization is in progress.* This is known as a *universal* synchronization point.

In both variants, federates that resign from the federation while a synchronization point is in progress are assumed to have achieved the synchronization point and are removed from the set.

There is no arbitrary limit on the number of synchronization points that may be in progress at a given time. However, only one synchronization point with a given label may be outstanding for the federation at a given time (even if the synchronization points apply to mutually exclusive subsets of federates.)

The RTI implements a negotiation protocol to arbitrate race conditions that may occur in the registration of synchronization points. Essentially, if two federates simultaneously attempt to register synchronization points with different federate sets, whichever request is processed first by the FedEx is the "winner". If two federates attempt to register synchronization points with the same federate set, both will succeed.

The federate is appraised of the success or failure of a synchronization point through a `synchronizationPointRegistrationSucceeded()` or `synchronizationPointRegistrationFailed()` callback, respectively. A successful indication always occurs asynchronously with respect to the registration request, i.e. during a subsequent invocation of `tick()`. An unsuccessful indication may occur asynchronously or synchronously (i.e., before the `registerFederationSynchronizationPoint()` call returns), depending on whether a race condition occurred.

If the registration succeeds, all federates to which the synchronization point is applicable will receive an announcement in the form of a `announceSynchronizationPoint()` callback. This possibly includes the federate that registered the synchronization point. When all federates included in the synchronization point (including recently-joined federates if a universal synchronization point was registered) have achieved synchronization or resigned, the relevant federates will be informed of synchronization through a `federationSynchronized()` callback.

RETURN VALUES

A non-exceptional return indicates that the federate's synchronization request has been submitted to the federation, and the federate will be appraised of success or failure of the registration through subsequent federate-ambassador callbacks.

MIGRATION NOTES

The functionality of the RTI 1.0 pause/resume services can be trivially implemented using a "pause" synchronization point, followed by a "resume' synchronization point when the pause is achieved.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador
: announceSynchronizationPoint()

: federationSynchronized()

: synchronizationPointRegistrationFailed()

: synchronizationPointRegistrationSucceeded()

RTI::RTIambassador
: joinFederationExecution()

: resignFederationExecution()

: synchronizationPointAchieved()

A.1.13 requestFederationRestore()

RTI 1.3

ABSTRACT

This service requests that all federates re-initialize themselves based on a named saved state. **The semantics of federation save and restore have changed substantially from RTI 1.0 to RTI 1.3. This service was named *requestRestore* in RTI 1.0; the 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Request Federation Restore" Federation Management service as specified in the *HLA Interface Specification* (§4.16 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   requestFederationRestore (
      const char *label
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

label

a string label associated with a previously saved named federation state

DESCRIPTION

A federate invokes `requestFederationRestore()` to initiate an attempt to reconstitute a named federation state that has previously been saved. The request to restore is communicated to the FedEx, which examines the saved state to see if a restore is possible given the current active federation. The directory expected to contain the saved state is given by **$RTI_SAVE_PATH/FED/fedex/label**, where

- **$RTI_SAVE_PATH** is an environment variable
- **FED** is the FED filename as passed to `createFederationExecution()`
- **fedex** is the federation execution name

Note that the contents of directory given by this expression must be the same for every participating federate and the FedEx for the restore to be successful. The FedEx approves the request to restore if the following criteria are met:

- the checksum computed for the FED file currently used by the federation matches the saved FED checksum
- for each type of federate (as distinguished by the "name" argument supplied to `joinFederationExecution()`), the number of federates in the saved state equals the number in the active federation
- a save or restore is not already in progress

If one or more of the criteria are not met, the requesting federate will be notified through a `requestFederationRestoreFailed()` callback. If all criteria are met, the following sequence of events takes place:

1. The FedEx enters "restore" mode, during which time all requests to join, resign, or initiate a save or restore will be denied.
2. The requesting federate receives a positive acknowledgement in the form of a `requestFederationRestoreSucceeded()` callback.
3. Each federate (including the requesting federate) receives a `federationRestoreBegun()` callback, at which time the federate is suspended from invoking any services that changes the state of the LRC or the federation.
4. When all federates have been suspended, each LRC will restore its internal state based on a saved LRC state for a federate of a matching type.
5. When an LRC has finished restoring its internal state, it invokes its federate's `initiateFederateRestore()` callback with the new federate handle as an argument. At this point, the federate should take the appropriate actions to restore its federate-managed state.
6. Each federate reports a successful or failed restoration of federate-managed state using the `federateRestoreComplete()` or `federateRestoreNotComplete()` service, respectively.
7. When all federates have reported a successful or failed restoration, all federates receive notification that the federation-wide restore has completed. If all federates reported a successful restoration, the `federationRestored()` is used, otherwise the `federationNotRestored()` callback is used. Upon receipt of such a callback, a federate is no longer suspended and may resume normal operation.
8. The FedEx returns to "running" mode, in which joins, resignations, saves, and restores are allowed.

Note that saved LRC states are not portable across platforms. Federates running on incompatible platforms should use different names to indicate that their saved states are not interchangeable.

The RTI provides no facility for saving or restoring federate-managed state. Federate developers must develop their own conventions for locating and reconstituting named federate states.

Even in a suspended state, a federate must continue to call `tick()` so that internal RTI communications may be serviced.

RETURN VALUES

A non-exceptional return indicates that the federate's desire to initiate a restoration of a named federation state has been communicated to the FedEx. The federate will be notified of the success or failure of said request through a subsequent federate ambassador callback.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::
: federationNotRestored()

: federationRestoreBegun()

: federationRestored()

: initiateFederateRestore()

: requestFederationRestoreFailed()

: requestFederationRestoreSucceeded()

RTI::RTIambassador::
: getRegion()

: requestFederationSave()

A.1.14 requestFederationSave()

RTI 1.0 RTI 1.3

ABSTRACT

This service requests that the federation save its state at a specified logical time. **The semantics of federation save and restore have changed substantially from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Request Federation Save" Federation Management service as specified in the *HLA Interface Specification* (§2.11 in versions 1.1; §4.11 in 1.3).

SYNOPSIS

```
#include <RTI.hh>
// Save at the specified time.
// RTI 1.0 Only
void
RTI::RTIambassador::
   requestFederationSave (
       const RTI::SaveLabel label
       RTI::FederationTime theTime
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   requestFederationSave (
       const char              *label,  ← Changes Type
       const RTI::FedTime&     theTime  ← Changes Type
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// Save as soon as possible.
// RTI 1.0 Only
void
RTI::RTIambassador::
   requestFederationSave (
       const RTI::SaveLabel label
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   requestFederationSave (
       const char *label                ← Changes Type
   )
throw (
       RTI::ConcurrentAccessAttempted,
       RTI::FederateNotExecutionMember,
       RTI::RestoreInProgress,
       RTI::RTIinternalError,
       RTI::SaveInProgress
)
```

ARGUMENTS

label
: a string to be associated with this particular pause request

theTime
: the desired time for the Federation save; omission of this argument implies that the save should take place as soon as possible

DESCRIPTION

A federate invokes `requestFederationSave()` to initiate an attempt to save the federation state to disk. The federate-supplied label is used to distinguish among multiple saved states for the purposes of restoration. The label is not interpreted by the RTI other than it becomes a component of the path in which RTI internal state is stored. The logical time argument (if present) specifies a federation time at which the save is to occur. The logical time is only relevant for time-constrained federates; non-time-constrained federates will always save as soon as possible. If no logical time is supplied, all federates save as soon as possible.

The following is a generic description of the save/restore process. The reader is referred to the release notes section for details specific to a particular RTI implementation.

1. At some point subsequent to a successful invocation of `requestFederationSave()`, all participating federates will receive an `initiateFederateSave()` callback. Upon such, a federate is suspended from invoking any services that would change the internal state of its LRC.
2. Each federate invokes `federateSaveBegun()` to indicate that it has begun saving its federate-managed state. Upon completion of an attempt to save federate-managed state, each federate invokes an RTI service indicating success or failure.
3. At some point during or after the invocation of the service indicating success or failure, the LRC associated with the federate saves its internal state to disk.

Even in a suspended state, a federate must continue to call `tick()` so that internal RTI communications may be serviced.

Saved LRC states may not be portable across architectures. The federation manager should ensure that the configurations of the saved and restoring federations will result in a mapping of LRC states such that a saved LRC state is always reconstituted on an identical (or compatible) platform. See the release notes for details of how this mapping is computed.

RETURN VALUES

A non-exceptional return indicates that the federation save has been initiated.

RELEASE NOTES

RTI 1.0

- The `initiateFederateSave()` callback is made as soon as possible for all federates (i.e., non-time-constrained federates will receive the callback as soon as possible.)
- The services invoked by a federate to report success or failure of a save of federate-managed state are named `federateSaveAchieved()` and `federateSaveNotAchieved()`, respectively.
- The LRC associated with a federate saves its internal state synchronously with respect to a `federateSaveAchieved()` or `federateSaveNotAchieved()` callback. Subsequent to the successful invocation of one of these services, the federate is no longer suspended and may resume normal

activity.

- The RTI is not proactive in notifying federates that the federation-wide save has completed. Federates may use the *Manager.Federate* and *Manager.Federation* object classes provided by the MOM to monitor the status of a federation save.
- Federates joining while a save is in progress are not included in the save. Federates may resign while a save is in progress.
- During a restoration, saved LRC states are mapped to active LRCs based on federate handles. That is, the state of the LRC associated with a federate with handle *n* is restored to the state of the LRC associated with the federate with handle *n* at the time of the save. It is the responsibility of the federation manager to ensure that the restored federation may operate correctly based on this mapping of federate handles.

RTI 1.3

- Federates do not receive an `initiateFederateSave()` callback until all time-constrained federates have attained the logical time at which the save is to take place.
- The services invoked by a federate to report success or failure of a save of federate-managed state are named `federateSaveComplete()` and `federateSaveNotComplete()`, respectively. The later name is somewhat of a misnomer, as the method is actually used to indicate the completion of an unsuccessful save attempt.
- The LRC associated with a federate saves its state only after it has received indication that each federate in the federation has completed (or failed) to save its federate-managed state.
- When all LRCs have finished saving their internal states, all federates will receive a callback notification that the federation-wide save has finished. If all saving federates report success, a `federationSaved()` callback is made, otherwise a `federationNotSaved()` callback is made. Upon receipt of such a callback, the federate is no longer suspended and may resume normal operation.
- Federates are prevented from joining or resigning while a save is in progress.
- During a restoration, saved LRC states are mapped to active LRCs based on the type of federate (as distinguished by the name argument provided to `joinFederationExecution()`.) A restore can only take place if there exists a function mapping federate handles in the saved state to federate handles in the active federation at the time of the restoration such that:
  - the function is total, one-to-one, and onto
  - all mappings in the function are between federates of the same type
- The `federateSaveBegun()` callback is not required by the 1.3 implementation; however, federates should still invoke this service to ensure compliance with other HLA implementations.

NOTE ON TYPES

The type of RTI 1.0 string arguments is typically `const Identifier`, where *Identifier* is a typedef for `char*`. This results in an effective type of `char* const` instead of the desired `const char*`. In RTI 1.3 the typedefs have been eliminated and the type of string arguments is just `const char*`.

RTI 1.3 encapsulates logical time in the *FedTime* class. Typically, an instance of this class is passed by reference in situations where a C++ `double` was used in RTI 1.0.

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed
: The specified logical time argument lies in the federation’s past, whereas the context of the service invocation required a future logical time.

RTI::InvalidFederationTime
: The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador
: initiateFederateSave()

RTI::RTIambassador
: federateSaveAchieved()
: federateSaveBegun()
: federateSaveNotAchieved()
: requestRestore()

RTI 1.3

RTI::FederateAmbassador
: federationNotSaved()
: federationSaved()
: initateFederateSave()

RTI::RTIambassador
: federateSaveBegun()
: federateSaveComplete()
: federateSaveNotComplete()
: requestFederationRestore()

A.1.15 requestPause()

RTI 1.0

ABSTRACT

This service requests that all federates in the Federation suspend execution as soon as possible. **The pause/resume capability of HLA 1.1 has been replaced by the more general "synchronization point" functionality of HLA 1.3.**

HLA IF SPECIFICATION

This method realizes the "Request Pause" Federation Management service as specified in the *HLA Interface Specification* (§2.5 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   requestPause (
      const RTI::PauseLabel label
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationAlreadyPaused,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

label

a string to be associated with this particular pause request

DESCRIPTION

A federate uses `requestPause()` to notify all federates in a federtion of the federate's desire to suspend federation execution. All federates participating in the federation are signaled via the `initiatePause()` callback (i.e., upon receipt of which, federates are expected to suspend their execution).

The passed label is communicated to participating federates as a part of `initiatePause()`. The label is not interpreted by the RTI. Federation developers may define different types of pauses associated with different labels in a way that makes sense in the context of a given federation. The label is simply a means for the requesting federate to specify a textual description of the reason for the pause request or any other information relevant in the context of the federation.

A paused federate should continue to participate in message exchanges (e.g., sending and receiving updates and interactions) and utilizing RTI services.

Currently, the federation does not inform the federate when a requested pause has been achieved. However, a federate can call `requestPause()` periodically until the exception *RTI::FederationAlreadyPaused* is thrown.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully communicated its desire to suspend federation execution.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationAlreadyPaused

The Federation is already paused.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

initiatePause()

RTI::RTIambassador::

initiateResume()
pauseAchieved()
requestResume()

A.1.16 requestRestore()

RTI 1.0

ABSTRACT

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

HLA IF SPECIFICATION

This method realizes the "Request Restore" Federation Management service as specified in the *HLA Interface Specification* (§2.15 in version).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   requestRestore (
       const RTI::SaveLabel label
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress,
   RTI::SpecifiedSaveLabelDoesNotExist
)
```

ARGUMENTS

label

a string used to identify the particular save that will be used to restore the state of federates

DESCRIPTION

A Federate uses `requestRestore()` to notify all federates in a Federation of the Federate's desire to initiate a restore process. All federates participating in the Federation are signaled via the `initiateRestore()` callback. Upon receipt of the `initiateRestore()` signal, federates are expected to restore a previously stored (e.g. saved in a file or database) state identified by the passed label.

Unlike the counterpart "save" operation, restoration cannot be scheduled at a specific logical time across all federates; rather, it is initiated immediately upon the receipt of the restore request. The restored logical times of all states should be the same and will override pre-restoration values. It's a good idea to pause the Federation Execution before a state restoration.

The passed label is communicated to participating Federates as a part of the `initiateRestore()` signal. The label is not interpreted by the RTI. Federation developers may define different types of saves associated with different labels in a way that makes sense in the context of a given Federation. The label should correspond to a previous "save" label.

The RTI is responsible for saving and restoring its internal state. However, the RTI does not define a format or provide any facility for federates to save their external state. It is up to the federation developers to implement their own save and restore mechanisms.

Only one restoration request may be outstanding at a given instance. Subsequent invocations of `requestRestore()` override previous requests. **The RTI attempts to restore** its internal state from the file "RTIxxx-n.sav" where "xxx" is the save label and "n" is the federate's federate handle. These files will be located in the configuration directory (e.g., `$RTI_CONFIG` on UNIX® platforms and `%RTI_CONFIG%` on Windows® NT platforms).

Federates must join the FedExec in a consistent order if the restored internal states are to match up with the same federates.

RETURN VALUES

A non-exceptional return indicates that the Federate has successfully communicated its desire to restore Federation state from a labeled state save.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::SpecifiedSaveLabelDoesNotExist

The RTI was unable to find the file where the RTI state was preserved.

SEE ALSO

RTI::FederateAmbassador::

initiateRestore()

RTI::RTIambassador::

requestFederationSave()
restoreAchieved()
restoreNotAchieved()

A.1.17 requestResume()

RTI 1.0

ABSTRACT

This service requests that a paused federation resume execution as soon as possible. **The pause/resume capability of HLA 1.1 has been replaced by the more general "synchronization point" functionality of HLA 1.3.**

HLA IF SPECIFICATION

This method realizes the "Request Resume" Federation Management service as specified in the *HLA Interface Specification* (§2.8 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   requestResume ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationNotPaused,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
)
```

DESCRIPTION

A federate uses `requestResume()` to request that the federation end the current pause and resume execution. The federation signals federates to resume by invoking the callback function `initiateResume()`. The federate requesting resumption need not be the same federate that requested the pause.

RTI 1.0 provides no easy way for a federate to determine when all federates have resumed execution. The *Manager.Federate* and *Manager.Federation* objects provided by the MOM may be used to monitor the progress of a federation-wide resume.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully communicated its desire to resume federation execution.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationNotPaused

Federation execution currently is not paused.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::
initiateResume()

RTI::RTIambassador::
requestPause()
resumeAchieved()

A.1.18 resignFederationExecution()

RTI 1.0 RTI 1.3

ABSTRACT

This services terminates the federate's participation in a federation.

HLA IF SPECIFICATION

This method realizes the "Resign Federation Execution" Federation Management service as specified in the *HLA Interface Specification* (§2.4 in version 1.1; §4.5 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

enum RTI::ResignAction {
   RELEASE_ATTRIBUTES = 1,
   DELETE_OBJECTS,
   DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES,
   NO_ACTION
};

void
RTI::RTIambassador::
   resignFederationExecution (
       RTI::ResignAction theAction
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederateOwnsAttributes,
   RTI::InvalidResignAction,
   RTI::RTIinternalError
)
```

ARGUMENTS

theAction

enumerated value indicating the desired policy for relinquishment of federate-owned attributes

DESCRIPTION

A federate uses `resignFederationExecution()` to inform a FedExec that the federate no longer wishes to participate in the federation. Before doing so, it is necessary to resolve ownership of any attribute-instances owned by the federate. The four resolution policies defined are:

RELEASE_ATTRIBUTES

The federate releases control of all owned attributes including those for which it holds the *privilege-to-delete* ownership token. Essentially, this is an unconditional divestiture of every attribute owned by the federate. The Federation will inform federates of available attributes via the `requestAttributeOwnershipAssumption()` callback.

Any ownership tokens that aren't assumed by another federate become "orphaned". Orphaned tokens continue to exist in the Federation. They are tracked by the RTI internally, by another federate process or by the FedExec and are eligible for acquisition by any interested federate. However, no additional notification is provided that an attribute is orphaned (i.e., beyond the original request for attribute ownership assumption).

DELETE_OBJECTS

The resigning federate deletes all objects for which it holds the *privilege-to-delete* ownership token. The effect of this option is the same as if the federate had explicitly called `deleteObject()` for every object for which it holds the *privilege-to-delete* token. If the federate owns attributes of objects for which it does not hold the delete privilege, these attributes become "zombies" (see *NO ACTION* below.)

DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES

The resigning federate first deletes any objects for which it holds the privilege-to-delete token and then releases ownership of any remaining owned attributes (i.e., effectively, a combination of the *DELETE_OBJECTS* and *RELEASE_ATTRIBUTES* options.) This is the recommended option for most situations.

NO_ACTION

The resigning federate suggests no action regarding attribute owndership. All attributes and objects owned by the federate become "zombies" (i.e. technically, they still exist in the Federation but they are immutable, cannot be discovered and are not eligible for acquisition by other federates.

The `resignFederationExecution()` method will not return until (1) the ownership of all federate-owned attributes is resolved as prescribed by the resign action and (2) the connection between the federate and the FedExec is terminated. On completion, the internal state of the *RTIambassador* instance is reset, allowing it to be associated with another FedExec through a subsequent invocation of `joinFederationExecution()`. The *FederateAmbassador* instance associated with the federate and previously provided to the FedExec is no longer needed at this point and may be disposed of at the federate's leisure.

Any messages queued for delivery to the federate at the time of resignation are lost.

RETURN VALUES

A non-exceptional return indicates that the resignation was successful.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederateOwnsAttributes

This exception is not thrown by the current implementations of this service.

RTI::InvalidResignAction

The parameter specifying the ownership resolution policy was not a recognized value.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::FederateAmbassador::

requestAttributeOwnershipAssumption()

RTI::RTIambassador::

deleteObject()

destroyFederationExecution()

joinFederationExecution()

requestAttributeOwnershipDivestiture()

A.1.19 restoreAchieved()

RTI 1.0

ABSTRACT

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

HLA IF SPECIFICATION

This method (in conjunction with `restoreNotAchieved()`) realizes the "Restore Achieved" Federation Management service as specified in the *HLA Interface Specification* (§2.17 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   restoreAchieved ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RTIcanNotRestore,
   RTI::RTIinternalError,
   RTI::RestoreNotRequested
)
```

DESCRIPTION

A federate uses `restoreAchieved()` to signal a successfully completed restore. Restores are conducted on receipt of an `initiateRestore()` request.

The `restoreAchieved()` call *blocks* until all other federates and the RTI have completed their restores (i.e., successfully or unsuccessfully.) The RTI attempts to restore its internal state from the file "RTIxxx-n.sav" where "xxx" is the save label and "n" is the federate's federate handle. These files will be located in the configuration directory (e.g., `$RTI_CONFIG` on UNIX® platforms and `%RTI_CONFIG%` on Windows® NT platforms).

Upon return from this method, the federate's logical time will be reset to the original save time and continue advancing as prescribed by the time-advance service in effect at the time of the save.

RETURN VALUES

A non-execptional return indicates that all federates in the Federation have finished restoring their states and that the internal state of the RTI has been restored.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreNotRequested

There is no outstanding request for a federate restoration.

RTI::RTIcanNotRestore

The RTI internal state save file is missing or corrupt.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::FederateAmbassador::
initiateRestore()

RTI::RTIambassador::
requestFederationSave()
requestRestore()
restoreNotAcheived()

A.1.20 restoreNotAchieved()

RTI 1.0

ABSTRACT

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

HLA IF SPECIFICATION

This method (in conjunction with `restoreAchieved()`) realizes the "Restore Achieved" Federation Management service as specified in the *HLA Interface Specification* (§2.17 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   restoreNotAchieved ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreNotRequested,
   RTI::RTIcanNotRestore,
   RTI::RTIinternalError
)
```

DESCRIPTION

A Federate uses `restoreNotAchieved()` to signal a unsuccessful, but completed restore. Restores are conducted on receipt of an `initiateRestore()` request.

The `restoreNotAchieved()` call *blocks* until all other federates and the RTI have completed their restores (i.e., successfully or unsuccessfully). The RTI attempts to restore its internal state from the file "RTIxxx-n.sav" where "xxx" is the save label and "n" is the federate's federate handle. These files will be located in the configuration directory (e.g., `$RTI_CONFIG` on UNIX® platforms and `%RTI_CONFIG%` on Windows® NT platforms).

Upon return from this method, the federate's state is undetermined. The RTI's internal state is still restored. Time will continue advancing as prescribed by the time-advance service in effect at the time of the save.

RETURN VALUES

A non-execptional return indicates that all the federates in the Federation have finished restoring their states and that the internal state of the RTI has been restored.

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreNotRequested
: There is no outstanding request for a federate restoration.

RTI::RTIcanNotRestore, RTI::RTICannotRestore
: The RTI internal state save file is missing or corrupt.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::FederateAmbassador::
initiateRestore()

RTI::RTIambassador::
restoreAcheived()
requestFederationSave()
requestRestore()

A.1.21 resumeAchieved()

RTI 1.0

ABSTRACT

This service informs the RtiExec that the calling Federate has resumed execution as per the most recent `initiateResume()` request. **The pause/resume capability of HLA 1.1 has been replaced by the more general "synchronization point" functionality of HLA 1.3.**

HLA IF SPECIFICATION

This method realizes the "Resume Achieved" Federation Management service as specified in the *HLA Interface Specification* (§2.10 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   resumeAchieved ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::NoResumeRequested,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

A Federate uses `resumeAchieved()` to signal that it has successfully resumed execution in response to an initiate resume request. Resume requests are communicated via the `initiateResume()` callback method.

RETURN VALUES

A non-exceptional return indicates that the RTI acknowledges notification that the Federate has resumed execution.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NoResumeRequested

The Federation does not believe that the Federate has been requested to resume execution.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::
initiatePause()
initiateResume()

RTI::RTIambassador::
requestResume()

A.1.22 synchronizationPointAchieved()

RTI 1.3

ABSTRACT

This service informs the federation that the federate has met the federation-defined criteria associated with a synchronization point that has previously been announced to the federate. **The synchronization mechanism is a generalization of the pause/resume mechanism featured in HLA 1.1.**

HLA IF SPECIFICATION

This method realizes the "Synchronization Point Achieved" Federation Management service as specified in the *HLA Interface Specification* (§4.9 in version 1.3).

SYNOPSIS

```
void
RTI::RTIambassador::
    synchronizationPointAchieved (
        const char *label
)
throw (
    RTI::ConcurrentAccessAttempted,
    RTI::FederateNotExecutionMember,
    RTI::RestoreInProgress,
    RTI::RTIinternalError,
    RTI::SaveInProgress,
    RTI::SynchronizationPointLabelWasNotAnnounced
)
```

ARGUMENTS

label

synchronization point identifier that was previously announced to the federate

DESCRIPTION

Synchronization points are a generalization of the pause/resume capabilities featured in early revisions of HLA. They 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 by the desired set of federates.

A federate uses `synchronizationPointAchieved()` to indicate that it has met the synchronization criteria associated with some currently outstanding synchronization point. The label argument to this service must correspond to a synchronization point that has been previously announced to the federate through the `announceSynchronizationPoint()` callback.

When all federates included in the synchronization have achieved the synchronization point or resigned, each included federate will receive a `federationSynchronized()` callback to announce that synchronization has been achieved. This callback always occurs during a subsequent `tick()`; the federate will never receive a callback during a `synchronizationPointAchieved()` callback.

Depending on the semantics of a synchronization point, it may or may not be appropriate for a federate to continue operation while waiting for synchronization to be achieved. The RTI places no restrictions on a federation pending synchronization; federation developers are free to implement their own restrictions based on federation-specific synchronization semantics. **At a minimum, all federates must continue to invoke `tick()` so that internal RTI communications may be serviced.**

RETURN VALUES

A non-exceptional return indicates that the federation will be informed of the federate's attainment of the specified synchronization point.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::SynchronizationPointLabelWasNotAnnounced

The specified label does not represent a current synchronization request outstanding to the federate as indicated by `announceSynchronizationPoint()`.

SEE ALSO

RTI::FederateAmbassador

announceSynchronizationPoint()

federationSynchronized()

RTI::RTIambassador

registerFederationSynchronizationPoint()

A.2 Declaration Management

A.2.1 publishInteractionClass()

RTI 1.0 RTI 1.3

ABSTRACT

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

HLA IF SPECIFICATION

This method (in conjunction with `unpublishInteractionClass()`) realizes the "Publish Interaction Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.1 (§3.2).

This method realizes the "Publish Interaction Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.3 (§5.4).

SYNOPSIS

```
#include <RTI.hh>
void
RTI::RTIambassador::
   publishInteractionClass (
      RTI::InteractionClassHandle theInteraction
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theInteraction

the interaction class to be published

DESCRIPTION

This service informs the LRC that the federate may begin generating interactions of the specified class. Attempts by a federate to send interactions of a class that is not currently published by the federate will fail. Publication of an interaction class does *not* imply the publication of subclasses of that class.

While an interaction class is published, the LRC will advise the publishing federate of the existence of remote subscribers. These advisories are not enforced; however, it is suggested that a federate refrain from generating superfluous interactions in order to conserve resources. Invoking this service with an interaction class that is already published by the federate results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC has acknowledged the federate's intention to begin generation of the specified interaction class.

RELEASE NOTES

RTI 1.0

- The callbacks that advise the federate of the existence or absence of remote subscribing federates are `startInteractionGeneration()` and `stopInteractionGeneration()`, respectively.
- Several MOM interaction classes used managed internally by the RTI are automatically published on behalf of the federate.

RTI 1.3

- The callbacks that advise the federate of the existence or absence of remote subscribing federates are `turnInteractionsOn()` and `turnInteractionsOff()`, respectively. These advisory callbacks may be toggled on and off using the `enableInteractionRelevanceAdvisorySwitch()` and `disableInteractionRelevanceAdvisorySwitch()` services, respectively
- Only active subscriptions are considered by the advisory mechanism.
- Internal MOM publications are kept separate from federate publications. All MOM interactions that the federate intends to generate must be explicitly published.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The specified interaction class handle is not valid within the context of the current FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

startInteractionGeneration() ← **RTI 1.0 Only**

stopInteractionGeneration() ← **RTI 1.0 Only**

turnInteractionsOff() ← **RTI 1.3 Only**

turnInteractionsOn() ← **RTI 1.3 Only**

RTI::RTIambassador::

changeInteractionOrderType()

changeInteractionTransportType()

disableInteractionRelevanceAdvisorySwitch() ← **RTI 1.3 Only**

enableInteractionRelevanceAdvisorySwitch() ← **RTI 1.3 Only**

getInteractionClassHandle()

publishObjectClass()

sendInteraction()

sendInteraction()

sendInteractionWithRegion() ← **RTI 1.3 Only**

subscribeInteractionClass()

subscribeInteractionClassWithRegion() ← **RTI 1.3 Only**

unpublishInteractionClass()

A.2.2 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.**

HLA IF SPECIFICATION

This method (in conjunction with `unpublishObjectClass()`) realizes the "Publish Object Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.1 (§3.1).

This method realizes the "Publish Object Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.3 (§5.2).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   publishObjectClass (
      RTI::ObjectClassHandle          theClass
      const RTI::AttributeHandleSet&  attributeList
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object-class context of the attributes to be published

attributeList

the set of attributes to be published

DESCRIPTION

This service informs the LRC that the federate intends to acquire and update instances of the specified attributes. Subsequent to the publication of an object class and an associated set of attributes:

- The federate may register new instances of the object class. Any published attributes will be initially owned by the federate; all other attributes will be initially unowned and available for acquisition.
- The federate may bid for ownership of existing instances of the published attributes using ownership management services.
- The federate may be offered existing instances of the published attributes that are being divested by remote federates. The LRC will advise the federate of the existence or absence of remote federates that have declared a subscription interest in the published object class and attributes. These advisories are not enforced; however, it is suggested that a federate refrain from generating superfluous registrations and updates in order to conserve resources.

The *privilegeToDelete* attribute (inherently present in every federation-defined object class) is automatically added to any non-empty set of published attributes.

If the specified object class is already being published, the specified attribute set replaces the currently published attribute set. Any attributes in the currently published attribute set that are not in the specified attribute set are implicitly unpublished; see the release-specific notes for the ramifications of this policy. Invoking `publishObjectClass()` with an empty attribute set is equivalent to invoking `unpublishObjectClass()` for the object class in question. Attempts to register new instances of an unpublished object class or acquire ownership of instances of unpublished attributes will fail.

RETURN VALUES

A non-exceptional return indicates that the given set of attributes has been published 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.

RELEASE NOTES

RTI 1.0

- The callbacks that advise the federate of the presence or absence of remote subscribing federates are `startUpdates()` and `stopUpdates()` respectively.
- The federate retains ownership of any currently owned instances of attributes that are implicitly unpublished as a result of this service.
- Classes derived from the MOM-defined *Manager* object class receive special treatment. All subclasses of *Manager* are implicitly published by the MOM manager and must remain published throughout the lifetime of the FedExec. If the Federate attempts to republish a descendent of *Manager* with a different set of attributes, this set must contain all of the attributes pre-defined by the MOM. [If the Federate's publication fails to contain all such attributes, the object manager automatically adds these attributes to the new set of published attributes. Future releases may allow unpublication of non-MOM-defined attributes. For now, this functionality can be achieved by republishing the object class with an attribute set consisting of only the predefined attributes.]

RTI 1.3

- The callbacks that advise the federate of the presence or absence of remote subscribing federates are `startRegistrationForObjectClass()` and `stopRegistrationForObjectClass()`, respectively. This advisory may be toggled on and off using the `enableClassRelevanceAdvisorySwitch()` and `disableClassRelevanceAdvisorySwitch()` services, respectively.
- 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.
- If the local federate has any outstanding ownership bids for instances of attributes that would be implicitly unpublished by this service, the method will throw an exception.
- Internal MOM publications are kept separate from federate publications. The federate must explicitly publish any MOM attributes it intends to acquire and update.

subscribeObjectClassAttributes()

updateAttributeValues()

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::AttributeHandleSet

RTI::FederateAmbassador::

startUpdates()

stopUpdates()

RTI::RTIambassador::

getAttributeHandle()

getObjectClassHandle()

publishInteractionClass()

registerObject()

requestAttributeOwnershipAcquisition()

subscribeObjectClassAttribute()

updateAttributeValues()

RTI 1.3

RTI::AttributeHandleSet

RTI::FederateAmbassador::

startRegistrationForObjectClass()

stopRegistrationForObjectClass()

RTI::RTIambassador::

attributeOwnershipAcquisition()

disableClassRelevanceAdvisorySwitch()

enableClassRelevanceAdvisorySwitch()

getAttributeHandle()

getObjectClassHandle()

publishInteractionClass()

registerObjectInstance()

A.2.3 subscribeInteractionClass()

RTI 1.0 RTI 1.3

ABSTRACT

This service declares a federate's interest in receiving a specified class of interactions. **The RTI 1.3 implementation adds an optional second argument for specifying active vs. passive subscription.**

HLA IF SPECIFICATION

This method realizes the "Subscribe Interaction Class" Declaration Management service as specified in the *HLA Interface Specification* (§3.4 in version 1.1 and §5.8 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   subscribeInteractionClass (
       RTI::InteractionClassHandle theClass,
       RTI::Boolean            ← RTI 1.3 Only
           active = RTI::RTI_TRUE
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateLoggingServiceCalls,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

interaction class to be subscribed

active **(RTI 1.3 Only)**

flag specifying whether the subscription should be taken into account when advising publishing federates of interaction-class relevance

DESCRIPTION

The `subscribeInteractionClass()` service instructs the LRC to deliver interactions of a specified class to the federate. Subsequent instances of the specified interaction class occurring in the federation will be delivered to the federate in the form of `receiveInteraction()` callbacks. Interactions are never delivered to the federate ambassador of the originating federate.

If an LRC receives an interaction that is an instance of a subclass of an interaction class subscribed by the federate, the interaction will be *promoted*. The interaction will be delivered via `receiveInteraction()` as if it were an instance of the most specific subscribed superclass of the actual interaction class. Any parameters that are not present in the subscribed interaction class (i.e., are defined in subclasses of the subscribed class) will be filtered from the set of parameter values delivered to the federate.

Subscription to an interaction class entails subscription to all parameters of the interaction class. Instances of the interaction class delivered to the federate may contain values for any non-empty subset of the parameters defined for the interaction class.

Upon subscription to an interaction class, remote publishers of the interaction class (and its subclasses) will be advised of the existence of a subscriber.

Invoking `subscribeInteractionClass()` for an interaction-class that is already subscribed by the local federate results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will begin delivering instances of the specified interaction class to the federate.

RELEASE NOTES

RTI 1.0

- The `startInteractionGeneration()` callback is used to inform remote publishers of the existence of a subscriber.

RTI 1.3

- The `turnInteractionsOn()` callback is used to inform remote publishers of the existence of a subscriber.
- If the optional *active* argument is equal to *RTI::RTI_FALSE*, the federation will not be notified of the subscription. Thus, no `turnInteractionsOn()` callbacks will be made as a result of the subscription. This option is appropriate for a federate that should not have interactions generated solely for its benefit, but that should receive any interactions that would normally be generated (e.g. data-logging federates).
- Invoking `subscribeInteractionClass()` is equivalent to invoking `subscribeInteractionClassWithRegion()` with the default region (i.e., the region spanning the entire routing space bound to the interaction class) for the same interaction class.
- If a subset of a routing space is subscribed for an interaction class, interactions of that class associated with a non-intersecting region may be promoted to a more general class whose subscription intersects the interaction's region of relevance.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateLoggingServiceCalls

This exception is not thrown by the current implementations.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::

receiveInteraction()

startInteractionGeneration()

stopInteractionGeneration()

RTI::RTIambassador::

publishInteractionClass()

sendInteraction()

subscribeObjectClassAttribute()

unsubscribeInteractionClass()

RTI 1.3

RTI::FederateAmbassador

receiveInteraction()

turnInteractionsOff()

turnInteractionsOn()

RTI::RTIambassador

publishInteractionClass()

sendInteraction()

sendInteractionWithRegion()

subscribeObjectClassAttributes()

subscribeObjectClassAttributesWithRegion()

unsubscribeInteractionClass()

A.2.4 subscribeObjectClassAttribute()

RTI 1.0

ABSTRACT

This service declares a federate's interest in receiving updates for a set of attributes. **In RTI 1.3, this service is named `subscribeObjectClassAttribute`; the RTI 1.3 implementation is discussed in a separate section. The semantics of this service change from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Subscribe Object Class Attribute" Declaration Management service as specified in the *HLA Interface Specification* (§3.3 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   subscribeObjectClassAttribute (
       RTI::ObjectClassHandle theClass
       const RTI::AttributeHandleSet& attributeList
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

Class handle of the affected object class.

attributeList

List of attributes being sought.

DESCRIPTION

A Federate uses `subscribeObjectClassAttribute()` to manipulate the types of data it wishes to obtain. Subscription to an object class does not imply subscription to any subclasses of that object class. However, instances of non-subscribed subclasses are promoted to the most specific, subscribed object class and reported to the Federate. New class instances are reported to the Federate via the callback function `reflectAttributeValues()`.

Consider the following example. A *Beverage* class introduces the attributes "Volume" and "Price". The subclass *CarbonatedBeverage* extends the *Beverage* class adding the attribute "Carbonation". The subclass *Soda* extends *CarbonatedBeverage* adding the attributes "Caffeine" and "BrandName".

```
(class Beverage
   (attribute Volume)
   (attribute Price)
   (class CarbonatedBeverage
      (attribute Carbonation)
      (class Soda
         (attribute Caffeine)
         (attribute BrandName)
      )
   )
)
```

A Federate can subscribe to all attributes of the class *CarbonatedBeverage*. When another federate creates an instance of *Soda*, the new instance is reported to the subscribing Federate as an instance of *CarbonatedBeverage*. The soda-specific attributes (i.e., "Caffeine" and "BrandName") are filtered out.

Attribute subscriptions are not cumulative with respect to class hierarchies. For example, a Federate first subscribes to object class *CarbonatedBeverage* with attributes "Carbonation" and "Volume". It then subscribes to object class *Beverage* with attributes "Volume" and "Price". Subsequent attribute updates for *CarbonatedBeverage* (i.e., via `reflectAttributeValues()`) will not present the "Price" attribute. The attribute subscription set for the most-specific subscribed object class is always used.

If `subscribeObjectClassAttribute()`is invoked with an object class that is already subscribed, the new attribute set replaces the existing subscribed attribute set. A revised subscription does not affect objects that have already been discovered by the Federate. For example, if a Federate initially subscribes to the *Beverage* and later subscribes to *Soda*, *Soda* instances that have previously been discovered as *Beverage* will not be rediscovered as the more specific object class.

The Federate will not be discover objects until an attribute update is received following the subscription of a relevant object class. A Federate can request an attribute update, via `requestObjectAttributeValueUpdate()`, for pre-existing objects. This is especially useful if the attributes in question are updated sporadically.

Object classes derived from the Management Object Model (MOM) *Manager* class are treated like any other class (i.e., they are treated differently by object publication services).

RETURN VALUES

A non-exceptional return indicates that the Federate has subscribed to the given object class and attribute set (i.e., replacing the existing attribute subscription set, if any).

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

discoverObject()
removeObject()
reflectAttributeValues()

RTI::RTIambassador::

publishObjectClass()
subscribeInteractionClass()
requestObjectAttributeValueUpdate()
unsubscribeObjectClassAttribute()

A.2.5 subscribeObjectClassAttributes()

RTI 1.3

ABSTRACT

This service declares a federate's interest in receiving reflections for updates of a specified set of attributes. **The RTI 1.0 implementation of this service is named *`subscribeObjectClassAttribute`*; the RTI 1.0 implementation is discussed in a separate section**. **The RTI 1.3 implementation adds an optional third argument for specifying passive vs. active subscription.**

HLA IF SPECIFICATION

This method realizes the "Subscribe Object Class Attributes" Declaration Management service as specified in the *HLA Interface Specification* (§5.6 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   subscribeObjectClassAttribute (
       RTI::ObjectClassHandle          theClass,
       const RTI::AttributeHandleSet&  attributeList,
       RTI::Boolean
           active = RTI::RTI_TRUE
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object-class context of the attribute handles

attributeList

the set of class-attributes to be subscribed

active

flag specifying whether the subscription should be taken into account when advising publishing federates of object-registration relevance

DESCRIPTION

The `subscribeObjectClassAttributes()` service instructs the LRC to deliver updates of a specified set of class-attributes to the federate. Subsequent updates of instances of the specified class-attributes will be delivered to the federate in the form of `reflectAttributeValues()` callbacks. Updates are never delivered to the federate ambassador of the originating federate.

Invoking `subscribeObjectClassAttributes()` is equivalent to invoking `subscribeObjectClassAttributesWithRegion()` with the default region (i.e., the region spanning the entire routing space bound to the class-attributes in the FED file) for the same object class and set of class-attributes. If a federate is already subscribed to a set of class-attributes for an object class (resulting from a `subscribeObjectClassAttributes()` or a `subscribeObjectClassAttributesWithRegion()`), a subsequent invocation of `subscribeObjectClassAttributes()` for the same object class will replace the previous subscription. That is,

- any previously subscribed attributes that are not in the new attribute set will no longer be subscribed
- any regions previously associated with the attributes will be replaced by the default region of the routing space to which the attributes are bound in the FED file

An implication of this is that an invocation of `subscribeObjectClassAttributes()` with an empty attribute set is equivalent to an invocation of `unsubscribeObjectClass()` for the same object class.

Prior to receiving updates for an object instance, a federate will receive a `discoverObjectInstance()` callback conveying the instance's object handle and its object class. If an object is an instance of a subclass of an object class subscribed by a federate, the object will be discovered as if it were an instance of the most specific subscribed superclass of its actual object class. Subsequent reflections for attribute-instances of the object will be filtered by the LRC to only include those attributes that are present in the discovered object class. A federate will not receive a `discoverObjectInstance()` for a given object instance if the federate already owns attributes of the object instance.

If a federate is subscribing to some object classes with regions other than the default, object instances may be promoted beyond the most specific subscribed superclass. The discovered class of an object instance will be the most specific subscribed object class for which at least one attribute subscription region intersects the region associated with the corresponding instance-attribute at the time the discovery is delivered.

Subscription to an object class does not retroactively affect object instances that have already been discovered by the subscribing federate. That is, instances of the newly subscribed object class that have previously been discovered by the federate as a more general object class will still be treated by the LRC as instances of the previously discovered class. To force the rediscovery of instances, the federate may use the `localDeleteObjectInstance()` service.

The subscribing federate will be notified of subsequent updates of instances of the specified class-attributes using the `reflectAttributeValues()` callback. Each reflection will contain values for a non-empty subset of the currently subscribed attributes of the instance's discovered class.

Upon subscription to an interaction class, remote federates publishing the subscribed object class (or a subclass) such that the intersection of the published and subscribed class-attributes (regions notwithstanding) is non-empty may receive `startRegistrationForObjectClass()` callbacks. A federate will not receive such a callback multiple times for the same object class without an intervening `stopRegistrationForObjectClass()` callback.

If the optional *active* argument to the subscription is equal to *RTI::RTI_FALSE*, the federation will not be notified of the subscription. Thus, no `startRegistrationForObjectClass()` callbacks will be made as a result of the subscription. This option is appropriate for a federate that should not have registrations and updates made solely for its benefit, but that should receive any updates that would normally be generated (e.g. data-logging federates).

A federate will discover object instances whenever instances are registered or updated after the federate's subscription to an appropriate class. For attributes that are sporadically updated under normal circumstances, it may be desirable to explicitly request an update for the benefit of a newly-subscribed federate. The `requestObjectAttributeValueUpdate()` and `requestClassAttributeValueUpdate()` services may be used

for this purpose.

RETURN VALUES

A non-exceptional return indicates that the LRC will begin delivering reflections of instances of the specified class-attributes to the federate.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

discoverObjectInstance()

reflectAttributeValues()

removeObjectInstance()

RTI::RTIambassador::

getAttributeHandle()

getObjectClassHandle()

localDeleteObjectInstance()

publishObjectClass()

requestClassAttributeValueUpdate()

requestObjectAttributeValueUpdate()

subscribeInteractionClass()

unsubscribeObjectClass()

A.2.6 unpublishInteractionClass()

RTI 1.0 RTI 1.3

ABSTRACT

This service 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.**

HLA IF SPECIFICATION

This method (in conjunction with `publishInteractionClass()`) realizes the "Publish Interaction Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.1 (§3.2).

This method realizes the "Unpublish Interaction Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.3 (§5.5).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   unpublishInteractionClass (
       RTI::InteractionClassHandle theInteraction
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   unpublishInteractionClass (
       RTI::InteractionClassHandle theInteraction
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InteractionClassNotDefined,
  RTI::InteractionClassNotPublished,   ← RTI 1.3 Only
  RTI::RestoreInProgress,
  RTI::RTIinternalError
  RTI::SaveInProgress,
)
```

ARGUMENTS

theInteraction

the interaction class to be unpublished

DESCRIPTION

This service informs the LRC that the federate will no longer generate interactions of the specified class. The LRC will no longer advise the federate of the existence or absence of remote subscribing federates for the interaction class. Subsequent attempts by the federate to send interactions of the specified class will fail.

RETURN VALUES

A non-exceptional return indicates that the LRC has acknowledged the federate's intention to cease generation of the specified interaction class.

RELEASE NOTES

RTI 1.0

Any subclasses of the specified interaction class that are currently published will also be unpublished. A federate may simultaneously unpublish any published interaction class by unpublishing the *ROOT_INTERACTION_CLASS_HANDLE*. This interaction class is implicitly the superclass of all federation-defined interactions.

Invoking `unpublishInteractionClass()` with an interaction class that is not currently published and has no currently published subclasses results in a no-op.

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.

Invoking `unpublishInteractionClass()` with an interaction class that is not currently published results in a no-op.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::InteractionClassNotPublished **(RTI 1.3 Only)**

This exception is not thrown by the current implementation.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

getInteractionClassHandle()

publishInteractionClass()

A.2.7 unpublishObjectClass()

RTI 1.0 RTI 1.3

ABSTRACT

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

HLA IF SPECIFICATION

This method (in conjunction with `publishObjectClass()`) realizes the "Publish Object Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.1 (§3.1).

This method realizes the "Unpublish Object Class" Declaration Management service as specified in the *HLA Interface*

Specification version 1.3 (§5.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   unpublishObjectClass (
      RTI::ObjectClassHandle theClass
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederateOwnsAttributes,          ← RTI 1.0 Only
   RTI::ObjectClassNotDefined,
   RTI::ObjectClassNotPublished,         ← RTI 1.3 Only
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object class to unpublish

DESCRIPTION

The `unpublishObjectClass()` service conveys the intention of the federate to cease registration of new instances of a specified object class. In addition, the federate may no longer acquire attribute-instances of objects known to the federate as the unpublished class.

The specifics of this service differ dramatically between RTI 1.0 and RTI 1.3; see release notes below.

RETURN VALUES

A non-exceptional return indicates that the LRC acknowledges the unpublication of the object class.

RELEASE NOTES

RTI 1.0

- Classes derived from the MOM-defined *Manager* object class receive special treatment. All subclasses of *Manager* are implicitly published by the MOM manager and must remain published throughout the lifetime of the FedExec. The `unpublishObjectClass()` cannot be used to unpublish descendents of *Manager*.
- Unpublication only affects the acquisition of new attribute-instances. It does not relieve the federate of update responsibility for any attributes already owned.
- Unpublication of an object class removes the specified object class and any of its subclasses from the set of object classes published by the federate. The federate may no longer create objects or acquire attributes of the given object class or any of its subclasses.
- Any valid object class is a valid parameter to `unpublishObjectClass()`. The object class need not actually be published by the federate. A federate wishing to unpublish all object classes can do so by passing the parameter "`ROOT_OBJECT_CLASS_HANDLE`" (i.e., the handle of the RTI-defined object class from which all federation object classes are implicitly defined).

RTI 1.3

- There is nothing special about object classes defined by the Management Object Model. MOM internal publications and federate publications are handled independently, so a federate may publish and unpublish MOM classes as it would any other object class.
- Upon unpublication of an object class by a federate, any locally owned instance-attributes of object instances known to the federate as the unpublished 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 does *not* unpublish subclasses of the unpublished class.
- Only object classes currently published by a federate may be the subject of `unpublishObjectClass()` invocations by the federate. Attempts to unpublish object classes that are not currently published will result in `ObjectClassNotPublished` exceptions.

INTERFACE SPECIFICATION NOTES

The *HLA Interface Specification v1.3* stipulates that a federate may not unpublish an object class when it owns instances-attributes of objects known to the federate as the class being unpublished. RTI 1.3 allows this, but immediately divests any such attributes.

The *HLA Interface Specification v1.3* forbids the unpublication of an object class while a federate is participating in ownership acquisitions involving attribute-instances of objects known to the federate as the unpublished object class. The RTI 1.3 implementation does not enforce this restriction.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateOwnsAttributes **(RTI 1.0 Only)**

This exception is not thrown by the current implementation.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::ObjectClassNotPublished **(RTI 1.3 Only)**
The specified object class is not currently published by the local federate and is used in a context requiring a currently published object class.

RTI::RTIinternalError
An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::
getObjectClassHandle()
publishObjectClass()
requestAttributeOwnershipDivestiture()
← **RTI 1.3 Only**
unconditionalAttributeOwnershipDivestiture()
← **RTI 1.3 Only**

A.2.8 unsubscribeInteractionClass()

RTI 1.0 RTI 1.3

ABSTRACT

This service 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.**

HLA IF SPECIFICATION

This method (in conjunction with `subscribeInteractionClass()`) realizes the "Subscribe Interaction Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.1 (§3.4).

This method realizes the "Unsubscribe Object Class" Declaration Management service as specified in the *HLA Interface*

Specification version 1.3 (§5.9).

SYNOPSIS

```
#include <RTI.hh>

void
   unsubscribeInteractionClass (
      RTI::InteractionClassHandle theClass
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotSubscribed, ← RTI 1.3 Only
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the interaction class to unpublish

DESCRIPTION

The `unsubscribeInteractionClass()` service instructs the LRC to cease delivering interactions of the specified class to the federate. If there are no other federates subscribing to the interaction class, remote publishers of the interaction class and its subclasses may be advised to stop sending interactions of the unsubscribed class.

RELEASE NOTES

RTI 1.0

- Unsubscription of an interaction class also unsubscribes any subclasses of the interaction class.
- The specified interaction class need not actually be subscribed by the Federate. For example, the Federate may wish to unsubscribe all interaction classes by passing the argument "`ROOT_INTERACTION_CLASS_HANDLE`" (i.e., the handle of the RTI-defined interaction class from which all Federation-defined interaction classes are implicitly defined).

RTI 1.3

- An invocation of `unsubscribeInteractionClass()` only unsubscribes the specified interaction class; it does not unsubscribe subclasses of the specified interaction class.
- The specified interaction class must be currently subscribed by the federate; attempts to unsubscribe interaction classes that are not currently subscribed by the federate will result in `InteractionClassNotSubscribed` exceptions.

RETURN VALUES

A non-exceptional return indicates that the LRC acknowledges the unsubscription of the specified interaction class.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::InteractionClassNotSubscribed **(RTI 1.3 Only)**

The interaction class is not currently subscribed by the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::

receiveInteraction()

startInteractionGeneration()

stopInteractionGeneration()

RTI::RTIambassador::

getInteractionClassHandle()

publishInteractionClass()

subscribeInteractionClass()

RTI 1.3

RTI::FederateAmbassador::

receiveInteraction()

turnInteractionsOff()

turnInteractionsOn()

RTI::RTIambassador::

getInteractionClassHandle()

publishInteractionClass()

subscribeInteractionClass()

A.2.9 unsubscribeObjectClass()

RTI 1.3

ABSTRACT

This service withdraws a federate's interest in receiving updates for a specified object class. **In RTI 1.0, this service was named `unsubscribeObjectClassAttribute`; the RTI 1.0 implementation is discussed in a separate section. The semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method (in conjunction with `subscribeObjectClassAttribute()`) realizes the "Subscribe Object Class Attribute" Declaration Management service as specified in the *HLA Interface Specification* version 1.1 (§3.3).

This method realizes the "Unsubscribe Object Class" Declaration Management service as specified in the *HLA Interface Specification* version 1.3 (§5.7).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   unsubscribeObjectClass (
      RTI::ObjectClassHandle theClass
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::ObjectClassNotSubscribed,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object class to unsubscribe

DESCRIPTION

The `unsubscribeObjectClass()` service withdraws the federate's interest in receiving discoveries and update reflections for the specified object class.

Upon unsubscription from an object class, the federate will no longer receive any `discoverObjectInstance()` callbacks with a subject whose object class is the unsubscribed class. The federate will no longer receive `reflectAttributeValues()` for object instances that are known to the federate by the unsubscribed class. Such instances will *not* be promoted to a more general subscribed object class, even if such a class exists. The federate may use the `localDeleteObjectInstance()` service to force the rediscovery of object instances that have been discovered as an unpublished class.

The unsubscribing federate will receive `attributesOutOfScope()` callbacks for all previously in-scope instance-attributes of objects known to the federate by the unsubscribed class. If there are no other actively subscribing federates for the object class, remote publishers of the object class (and its subclasses) may be advised, via the `stopRegistrationForObjectClass()` callback, to stop registering object instances of the unpublished class.

Unsubscription to a specified object class does not result in unsubscritption of subclasses of the specified object class. If the subject of an `unsubscribeObjectClass()` service invocation is an object class not currently subscribed by the federate, an `ObjectClassNotSubscribed` exception will be thrown.

RETURN VALUES

A non-exceptional return indicates that the LRC will cease delivering discoveries and reflections whose subjects are object instances of the unpublished class to the federate.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::ObjectClassNotSuibscribed

The specified interaction class is not subscribed by the local federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

attributesOutOfScope()

discoverObjectInstance()

reflectAttributeValues()

stopRegistrationForObjectClass()

RTI::RTIambassador::

getObjectClassHandle()

subscribeObjectClassAttributes()

A.2.10 unsubscribeObjectClassAttribute()

RTI 1.0

ABSTRACT

This service withdraws a federate's interest in receiving updates for a set of attributes. **In RTI 1.3, this service is named `unsubscribeObjectClass`; the RTI 1.3 implementation is discussed in a separate section. The semantics of this service change from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Unsubscribe Object Class" Declaration Management service as specified in the *HLA Interface Specification* (§3.6 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   unsubscribeObjectClassAttribute (
       RTI::ObjectClassHandle theClass
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

Class handle of the affected object class.

DESCRIPTION

A Federate uses `unsubscribeObjectClassAttribute()` in conjunction with `subscribeObjectClassAttribute()` to manipulate the types of data it wishes to obtain. These two methods are used by the Federate to manipulate the types of data about which it wishes to be informed.

The `unsubscribeObjectClassAttribute()` method removes the specified object class and any of its subclasses from the set of object classes that will be presented to the Federate. The specified object class need not actually be subscribed by the Federate. For example, the Federate may unsubscribe all object classes by specifying the object class "`ROOT_OBJECT_CLASS_HANDLE`" (i.e., the handle of the RTI-defined object class from which all Federation-defined object classes are implicitly derived).

Subscription to an object class does not imply subscription to any subclasses of that object class. However, instances of non-subscribed subclasses are promoted to the most specific, subscribed object class and reported to the Federate. See `subscribeObjectClassAttribute()` for a discussion of the subscription process.

Removal of an object class from the subscription set results in the removal of all instances of that class from the set of objects known to the Federate. The Federate is informed of object instance removals through the callback function `removeObject()`. This notification is not delivered synchronously with respect to the unpublication method, but is queued up for later processing by *RTIambassador*'s `tick()` service.

If the Federate holds any attribute ownership tokens for removed objects, the object manager will automatically resolve ownership of these tokens. [See the discussion of "RELEASE_ATTRIBUTES" in the method `resignFederationExecution()`.]

Object classes derived from the Management Object Model (MOM) *Manager* class are treated like any other class (i.e., they are treated differently by object publication services).

RETURN VALUES

A non-exceptional return indicates that the given object class and any of its subclasses has been removed from the set of subscribed classes. All discovered instances of the affected classes have been queued for deletion from the set of objects known by the Federate. The object manager will attempt to divest attribute ownership tokens of any removed objects "behind the scenes". Updates for the affected attributes will no longer be presented to the Federate.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

discoverObject()

removeObject()

reflectAttributeValues()

RTI::RTIambassador::

publishObjectClass()

requestObjectAttributeValueUpdate()

subscribeInteractionClass()

subscribeObjectClassAttribute()

A.3 Object Management

A.3.1 changeAttributeTransportType()

RTI 1.0 RTI 1.3

ABSTRACT

This service specifies the transportation policy for a specified set of instance-attributes of a specified object instance to use for updates made by the local federate. **The type (but not semantics) of one parameter changes between RTI 1.0 and RTI 1.3.**

HLA IF SPECIFICATION

Realizes the "Change Attribute Transport Type" Object Management service as specified in the *HLA Interface Specification* (§4.10 in version 1.1;§6.11 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
enum RTI::TransportType {
   RELIABLE = 1,
   BEST_EFFORT
};

// RTI 1.0 Only
void
RTI::RTIambassador::
   changeAttributeTransportType (
      RTI::ObjectID theObject
      const RTI::AttributeHandleSet& theAttributes
      RTI::TransportType theType
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotOwned,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidTransportType,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   changeAttributeTransportType (
      RTI::ObjectHandle theObject,   ← Changes Types
      const RTI::AttributeHandleSet&
         theAttributes,
      RTI::TransportationHandle theType ← Changes Types
   )
throw (
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InvalidTransportationHandle,     ← Changes Types
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the object-instance whose instance-attributes are affected

theAttributes

the instance-attributes whose transportation policy is being set

theType

the transportation policy to use for subsequent updates of the specified instance-attributes by the local federate

DESCRIPTION

This service specifies the transportation mechanism to be used for subsequent updates of the specified instance-attributes made by the local federate. The current RTI implementations offer a choice between reliable and best-effort transportation:

- **Reliable** transportation uses the TCP protocol to guarantee that updates will not be discarded by the underlying network. This level of service is necessary for essential updates, which must be delivered to all subscribers (e.g. missile detonations, collision notifications.) Reliable updates experience relatively high latency and overhead and generally consume more network bandwidth than best-effort updates. The reliable service may cause federates to block in an `updateAttributeValues()` service invocation.
- **Best-effort** delivery uses the UDP multicast protocol to achieve efficient delivery to a large number of recipients. It is possible that best-effort updates will be discarded by the network and fail to be delivered to all intended recipients. This service may be appropriate for non-essential updates such a routine position reports. Best-effort service typically offers relatively low latency and utilizes bandwidth efficiently.

The default transportation service for an instance-attribute is the transportation service specified for the corresponding class-attribute in the FED file. The transportation service for an instance-attribute will revert to the default if the instance-attribute is transferred between federates using ownership management services.

This service will allow a transportation policy to be set for any instance-attributes of any objects known to the federate; however, such a specification is only meaningful for instance-attributes that are owned by the local federate.

RETURN VALUES

A non-exceptional exit indicates the specified transportation service will be used for future updates of the specified object-attributes.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotOwned

This exception is not thrown by the current implementations.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidTransportationHandle **(RTI 1.3 Only)**

The specified transportation service handle is not a valid handle as returned by the `getTransportationHandle()` service.

RTI::InvalidTransportType **(RTI 1.0 Only)**

The transport type specified is not recognized.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the Federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal

state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::RTIambassador::
changeAttributeOrderType()

changeInteractionTransportType()

updateAttributeValues()

A.3.2 changeInteractionTransportType()

RTI 1.0 RTI 1.3

ABSTRACT

Change the transportation mechanism used by the federate for interactions of a specified class.

HLA IF SPECIFICATION

Realizes the "Change Interaction Transport Type" Object Management service as specified in the *HLA Interface Specification* (§4.12 in version 1.1; §6.12 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>
// RTI 1.0 Only
enum RTI::TransportType {
   RELIABLE = 1,
   BEST_EFFORT
};

// RTI 1.0 Only
void
RTI::RTIambassador::
   changeInteractionTransportType (
       RTI::InteractionClassHandle theClass
       RTI::TransportType theType
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotPublished,
   RTI::InvalidTransportType,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
// RTI 1.3 Only
void
RTI::RTIambassador::
   changeInteractionTransportType (
       RTI::InteractionClassHandle theClass,
       RTI::TransportationHandle        ← Changes Types
           theType
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InteractionClassNotDefined,
  RTI::InteractionClassNotPublished,
  RTI::InvalidTransportationHandle,     ← Changes Types
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the interaction class affected by the change in transportation service

theType

the transportation service to use for the specified interaction class

DESCRIPTION

This service specifies the transportation mechanism to be used for subsequent interactions of the specified interaction class sent by the local federate. The current RTI implementations offer a choice between reliable and best-effort transportation:

- **Reliable** transportation uses the TCP protocol to guarantee that interactions will not be discarded by the underlying network. This level of service is necessary for essential interactions, which must be delivered to all subscribers (e.g. missile detonations, collision notifications.) Reliable interactions experience relatively high latency and overhead and generally consume more network bandwidth than best-effort interactions. The reliable service may cause federates to block in an `sendInteraction()` service invocation.
- **Best-effort** delivery uses the UDP multicast protocol to achieve efficient delivery to a large number of recipients. It is possible that best-effort interactions will be discarded by the network and fail to be delivered to all intended recipients. This service may be appropriate for non-essential interactions such a routine position reports. Best-effort service typically offers relatively low latency and utilizes bandwidth efficiently.

The default transportation service for an interaction instance is the transportation service specified for the corresponding interaction class in the FED file.

RETURN VALUES

A non-exceptional return indicates that future interactions of the specified class will be sent using the specified transportation service.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::InteractionClassNotPublished

The operation attempted requires that the interaction class be currently published by the federate.

RTI::InvalidTransportationHandle

The specified transportation service handle is not a valid handle as returned by the `getTransportationHandle()` service.

RTI::InvalidTransportType

The transport type specified is not recognized.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

changeAttributeTransportType()

changeInteractionOrderType()

publishInteractionClass()

sendInteraction()

A.3.3 deleteObject()

RTI 1.0

ABSTRACT

This service removes an object from the federation. **The RTI 1.3 implementation of this service is named `deleteObjectInstance`; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Delete Object" Object Management service as specified in the *HLA Interface Specification* (§4.8 in versions 1.1).

SYNOPSIS

```
#include <RTI.hh>

RTI::EventRetractionHandle
RTI::RTIambassador::
   deleteObject (
       RTI::ObjectID objectID
       RTI::FederationTime theTime
       const RTI::UserSuppliedTag theTag
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::DeletePrivilegeNotHeld,
   RTI::FederateNotExecutionMember,
   RTI::InvalidFederationTime,
   RTI::ObjectNotKnown,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
)
```

ARGUMENTS

objectID

Object to be deleted from the FedExec.

theTime

Time at which the object deletion is to become effective.

theTag

A string passed to federates that describes the interaction or provides some other meaningful data.

DESCRIPTION

A Federate uses `deleteObject()` to remove an object from the FedExec. The Federate must own the object – i.e., hold the *privilege-to-delete* attribute. The RTI-defined handle "`PRIVILEGE_TO_DELETE_HANDLE`" identifies the privilege-to-delete attribute. Delete privilege is initially granted to the Federate that registers the object.

If the Federate is "time regulating" and any of the instance's attributes are being sent time-stamp-ordered, the object deletion message will be designated for time-stamp-ordered delivery. Otherwise, the object deletion message is sent receive-ordered.

A successful invocation of this service causes the Federation to send `removeObject()` callbacks to federates that have *discovered* the specified object.

When an object is removed from the FedExec, its ID cannot be reused.

RETURN VALUES

A non-exceptional return indicates that an object-deletion message has been sent to the other federates in the FedExec.

An *RTI::EventRetractionHandle* instance is returned. It can be used as an argument to the `retract()` method to reinstate the object.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::DeletePrivilegeNotHeld

The Federate does not hold the *privilege-to-delete* attribute.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidFederationTime

The specified time value is not a legal time for a time-stamp-ordered update. It is less than the Federate's logical time plus its lookahead. Not thrown in RTI 1.0 and dropped entirely in RTI 1.3.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the Federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::
removeObject()

RTI::RTIambassador::
queryAttributeOwnership()

registerObject()

retract()

A.3.4 deleteObjectInstance()

RTI 1.3

ABSTRACT

This service removes an object instance from the federation. **The RTI 1.3 implementation of this service is named `deleteObjectInstance`; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Delete Object Instance" Federation Management service as specified in the *HLA Interface Specification* (§6.8 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::EventRetractionHandle
RTI::RTIambassador::
   deleteObjectInstance (
       RTI::ObjectHandle    objectHandle,
       const RTI::FedTime&  theTime,
       const char           *theTag
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::DeletePrivilegeNotHeld,
   RTI::FederateNotExecutionMember,
   RTI::InvalidFederationTime,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
);

void
RTI::RTIambassador::
   deleteObjectInstance (
       RTI::ObjectHandle    objectHandle,
       const char           *theTag
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::DeletePrivilegeNotHeld,
   RTI::FederateNotExecutionMember,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

objectHandle

the object-instance to be removed from the federation

theTime

the time at which to schedule the deletion for time-stamp-ordered delivery

theTag

a string that is passed to resulting invocations of `removeObjectInstance()`; this is not interpreted by the RTI and may be used to communicate federation-specific information about the deletion

DESCRIPTION

This service initiates the deletion of an object instance from the federation. Only a federate owning the *privilegeToDelete* instance-attribute of an object instance may initiate the deletion of that instance. The deletion is processed immediately by the LRC of the initiating federate. The deletion is communicated to remote LRCs and is queued for time-stamp-ordered (TSO) or receive-ordered processing for relevant federates. Processing of a deletion event by an LRC entails the following:

- All records of the object instance are removed from the internal structures of the LRC.
- If the object instance is known to the federate through registration or discovery, a `removeObjectInstance()` callback is made.
- Any instance-attributes owned by the federate no longer exist in the federation. The federate may receive a `turnUpdatesOffForObjectInstance()` advisory callback for any locally owned instance-attributes for which updates are currently turned on.

If an LRC processes an update for an object instance that has previously been deleted at the LRC, the object may be rediscovered by the federate, possibly as a different object class. This may include updates that were queued for delivery or in-transit at the point in execution when the deletion was processed. This is usually undesirable, and is recommended that the federation take precautions to prevent this situation (e.g., by using time-stamp-ordering to ensure that the deletion is the last event processed for an object instance.)

A `removeObjectInstance()` callback resulting from a `deleteObjectInstance()` service invocation will be delivered TSO if and only if

- a logical time argument is provided to the `deleteObjectInstance()` service invocation
- a TSO delivery policy is in effect for the *privilegeToDelete* attribute of the specified object instance at the initiating federate
- the federate initiating the deletion is time regulating
- the federate receiving the deletion is time constrained at the point in execution at which the deletion is queued for delivery *and* the point in execution at which the deletion is delivered to the federate

No logical time argument will be provided to `removeObjectInstance()` callbacks that are not delivered in TSO order, even if a logical time was specified to the `deleteObjectInstance()` service invocation which initiated the deletion.

A deletion notification is distributed to the federation using whichever transportation service is in effect for the *privilegeToDelete* instance-attribute of the deleted object instance.

RETURN VALUES

A non-exceptional return indicates that the object-instance deletion has been initiated in the federation.

The timed variant of this service returns

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::DeletePrivilegeNotHeld

The "privilege to delete" attribute is not owned by the local federate.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidFederationTime
The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::ObjectNotKnown
The specified object ID is not valid within the current FedExec or is not known to the Federate.

RTI::RestoreInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::
discoverObjectInstance()
removeObjectInstance()
turnUpdatesOffForObjectInstance()

RTI::RTIambassador::
changeAttributeOrderType()
changeAttributeTransportType()
enableTimeConstrained()
enableTimeRegulation()
localDeleteObjectInstance()
registerObjectInstance()
resignFederationExecution()
retract()
updateAttributeValues()

A.3.5 localDeleteObjectInstance()

RTI 1.3

ABSTRACT

This service may be used by a federate to cause a specified object instance to be rediscovered by the federate.

HLA IF SPECIFICATION

This service realizes the “Local Delete Object Instance” Federation Management service as specified in the *HLA Interface Specification* (§6.10 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador
   localDeleteObjectInstance (
      RTI::ObjectHandle    objectHandle
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::FederateOwnsAttributes,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

objectHandle

the object instance to rediscover

DESCRIPTION

This service causes the LRC to treat the specified object instance as if it had never been discovered by the federate. If any of the instance-attributes of the object instance match the current subscription interests of the federate, the federate will receive a `discoverObjectInstance()` callback for the object instance and an `attributesInScope()` callback for the instance-attributes matching the subscription. The newly discovered class will be the most specific object class at which the federate has subscribed to a class-attribute with a region overlapping the region associated with the corresponding instance-attribute. These callbacks will occur synchronously with respect to the `localDeleteObjectInstance()` service invocation.

If none of the instance-attributes of the specified instance match the federate’s current subscription interests, the instance remains undiscovered by the federate. It may be subsequently rediscovered as a result of changing subscriptions or associations. The federate will not receive an `attributeOutOfScope()` or `removeObjectInstance()` callback.

RETURN VALUES

A non-exceptional return indicates that the specified object instance has been undiscovered and possibly rediscovered.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederateOwnesAttributes

One or more attributes of the specified object are owned by the local federate.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the Federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

attributeOutOfScope()

attributesInScope()

discoverObjectInstance()

RTI::RTIambassador::

deleteObjectInstance()

A.3.6 registerObject()

RTI 1.0

ABSTRACT

This service introduces a new object instance into the federation. **The RTI 1.3 implementation of this service is named `registerObjectInstance`; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Register Object" Object Management service as specified in the *HLA Interface Specification* (§4.2 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   registerObject (
       RTI::ObjectClassHandle theClass
       RTI::ObjectID theObjectID
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidObjectID,
   RTI::ObjectAlreadyRegistered,
   RTI::ObjectClassNotDefined,
   RTI::ObjectClassNotPublished,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object class for which a new instance is being registered

TheObjectID

the object ID to associate with the specified class (should have been previously obtained via a call to `requestID()`.)

DESCRIPTION

A Federate uses `registerObject()` to identify a unique identification number for a new instance of an object class that is *under construction*. [The `registerObject()` method does not announce the existence of a newly created object. The object cannot be discovered by other federates until an attribute update notification is sent out.] The Federate must publish the object class (i.e., *ObjectClassHandle*) prior to calling `registerObject()`. The registration process differs between the RTI 1.0 and RTI 1.3 releases.

The Federate first obtains one or more object IDs via `requestID()`. It then uses the method `registerObject()` to asscociate an object class type with the object ID. [The Federate supplies both the object class handle and the object ID.] The `registerObject()` call binds the class type to the ID number.

The initial transportation mechanism and update ordering policy for a given attribute are set to the values defined in the FOM file, "`[federation name].fed`".

RETURN VALUES

A non-exceptional return indicates that the object has been successfully registered with the RTI. The Federate may begin generating updates for the attributes it publishes.

WINDOWS® NT NOTES

On Windows NT, the location of the FOM file is "`%RTI_CONFIG%\[federation].fed`".

UNIX® NOTES

On UNIX platforms, the location of the FOM file is "`$RTI_CONFIG/[federation name].fed`".

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidObjectID

The specified object ID has not been reserved for use by the Federate.

RTI::ObjectAlreadyRegistered

An object has already been registered using the specified object ID.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::ObjectClassNotPublished

The specified object class has not been published by the Federate.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

discoverObject()

RTI::RTIambassador::

changeAttributeOrderType()

changeAttributeTransportType()

deleteObject()

publishObjectClass()

requestID()

updateAttributeValues()

A.3.7 registerObjectInstance()

RTI 1.3

ABSTRACT

TBD

HLA IF SPECIFICATION

This method realizes the "Register Object Instance" Object Management service as specified in the *HLA Interface Specification* (§6.2 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::ObjectHandle
RTI::RTIambassador::
   registerObjectInstance (
      RTI::ObjectClassHandle  theClass,
      const char              *theObject
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectAlreadyRegistered,
   RTI::ObjectClassNotDefined,
   RTI::ObjectClassNotPublished,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

RTI::ObjectHandle
RTI::RTIambassador::
   registerObjectInstance (
      RTI::ObjectClassHandle theClass
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::ObjectClassNotPublished,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object class of which the new object is an instance

theObject

a symbolic name associated with the new instance; this is not interpreted by the RTI except that it must be unique to the federation at a given point in execution

DESCRIPTION

This service creates a new object instance in the federation. Instance-attributes of the newly created object which correspond to class-attributes published by the federate at the level of the specified object class are initially owned by the registering federate. All other instance-attributes are initially unowned and are available for acquisition by any federate.

All instance-attributes are initially associated with the default region (i.e., the region spanning the entire routing space to which the corresponding class-attributes are bound in the FED file.) All instance-attributes are initially associated with the transportation and ordering service categories to which the corresponding class-attributes are bound in the FED file.

The creation of a new object instance is immediately announced to the federation, resulting in `discoverObjectInstance()` callbacks for any federates whose subscription interests include at least one class-attribute of the registered object class. The object instance may also be discovered as a result of subsequent updates and data-distribution management operations affecting instance-attributes of the object.

The symbolic name must be unique to the federation over the lifetime of the object instance. The `getObjectInstanceHandle()` and `getObjectInstanceName()` services are used to convert between symbolic object names and numeric identifiers, and may be used to implement global named objects. If no symbolic name is specified, the RTI will assign a symbolic name consisting of the concatenation of

1. the string "HLA"
2. the hexadecimal IP address of the federate's host machine
3. the process ID of the federate process
4. the time in seconds since the epoch at which the federate process was started
5. an integer corresponding to the sequence number of the registration relative to other registrations by the same federate

Note that symbolic names generated using this algorithm are guaranteed to be unique over the lifetime of the federation (not just the lifetime of the object instance.)

The LRC will provide a `turnUpdatesOnForObjectInstance()` advisory for any instance-attributes of the newly registered instance for which there are active subscribers.

RETURN VALUES

A successful invocation of this service returns a numeric handle that uniquely identifies the newly created instance in the current federation execution. This handle is guaranteed to be unique over the lifetime of the registering federate.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectAlreadyRegistered

The symbolic name associated with the object has already been registered within the federation.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::ObjectClassNotPublished

The operation attempted requires that the object class be currently published by the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

discoverObject()

startRegistrationForObjectClass()

turnUpdatesOnForObjectInstance()

RTI::RTIambassador::

associateRegionForUpdates()

attributeOwnershipAcquisition()

changeAttributeOrderType()

changeAttributeTransportType()

deleteObjectInstance()

enableAttributeRelevanceAdvisorySwitch()

getAttributeRoutingSpaceHandle()

getObjectClassHandle()

getObjectInstanceHandle()

getObjectInstanceName()

negotiatedAttributeOwnershipDivestiture()

publishObjectClass()

registerObjectInstanceWithRegion()

subscribeObjectClassAttributes()

updateAttributeValues()

A.3.8 requestClassAttributeValueUpdate()

RTI 1.0 RTI 1.3

ABSTRACT

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

HLA IF SPECIFICATION

This method (in conjunction with `requestObjectAttributeValueUpdate()`), realizes the intent of the "Request Attribute Value Update" Object Management service as specified the *HLA Interface Specification* (§4.14 in version 1.1; §6.15 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   requestClassAttributeValueUpdate (
      RTI::ObjectClassHandle theClass
      const RTI::AttributeHandleSet& theAttributes
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object class for which updates are requested

theAttributes

the set of class-attributes of the specified class for which updates are requested

DESCRIPTION

This method solicits an update of the specified class-attributes from the federation. Remote federates will receive a `provideAttributeValueUpdate()` callback for each instance of the requested object class (or a subclass) for which the federate owns one or more instance-attributes corresponding to the solicited class-attributes. Upon receipt of such a callback, the remote federate is expected to comply with the request by issuing an `updateAttributeValues()` for the solicited instance-attributes.

This service may be used by a late-arriving (or late-subscribing) federate to solicit updates for all existing object instances. It is particularly useful for instance-attributes that are updated infrequently (or not at all) after instantiation.

RETURN VALUES

A non-exceptional return indicates that updates of all existing instances of the specified class-attributes have been solicited from the federation.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::

provideAttributeValueUpdate()

reflectAttributeValues()

RTI::RTIambassador::

discoverObject()

requestObjectAttributeValueUpdate()

RTI 1.3

RTI::FederateAmbassador::

provideAttributeValueUpdate()

reflectAttributeValues()

RTI::RTIambassador::

discoverObjectInstance()

requestClassAttributeValueUpdateWithRegion ()

requestObjectAttributeValueUpdate()

A.3.9 requestID()

RTI 1.0

ABSTRACT

This service obtains a range of unique IDs for use in registering objects with the Federation. **In RTI 1.3, object identifiers are assigned and managed by the LRC.**

HLA IF SPECIFICATION

This method realizes the "Request ID" Object Management service as specified in the *HLA Interface Specification* (§4.1 in versions 1.1 and 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   requestID (
       RTI::ObjectIDcount idCount
       RTI::ObjectID& firstID
       RTI::ObjectID& lastID
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::IDsupplyExhausted,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress,
   RTI::TooManyIDsRequested
)
```

ARGUMENTS

idCount
the number of identification numbers (IDs) to reserve

firstID
(returned) the first ID in the range of reserved IDs

lastID
(returned) the last ID in the range of reserved IDs

DESCRIPTION

A Federate uses `requestID()` to obtain a set of unique object IDs that may be used in subsequent calls to `registerObject()`. The IDs are taken from a per Federation set of unique IDs assigned to each federate in a FedExec. Object IDs are unique within the Federate and within the FedExec at any given time. [The same object ID may be used by different federates at different times within the same FedExec.] IDs are not recycled upon the deletion of their associated object.

The number of unique IDs available to a federate is configurable via the "MAX_OBJECTS_PER_FEDERATE" entry in the "`RTI.rid`" file.

RETURN VALUES

Upon a non-exceptional completion, the arguments "firstID" and "lastID" define the endpoints of an inclusive range of object IDs that may be used by the Federate for the registration of new Federation objects.

WINDOWS® NT NOTES

On Windows NT, the path of the RTI configuration file is "`%RTI_CONFIG%/RTI.rid`".

UNIX® NOTES

On UNIX platforms, the path of the RTI configuration file is "`$RTI_CONFIG/RTI.rid`" file.

EXCEPTIONS

RTI::IDsupplyExhausted
The Federate has exhausted its supply of unique object IDs.

RTI::FederateNotExecutionMember
The RTIambassador instance is not currently associated with a FedExec.

RTI::ConcurrentAccessAttempted
An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::RTIinternalError
An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TooManyIDsRequested
The request cannot be granted with a single, continuous range of object IDs. The request should be broken up into multiple, smaller requests. [**This exception is never actually thrown in RTI 1.0.**]

SEE ALSO

RTI::RTIambassador::
registerObject()

A.3.10 requestObjectAttributeValueUpdate()

RTI 1.0 RTI 1.3

ABSTRACT

This service stimulates the generation of instance-attribute updates for a specified object instance. **The syntax of this service changes slightly between RTI 1.0 and RTI 1.3.**

HLA IF SPECIFICATION

This method (in conjunction with `requestClassAttributeValueUpdate()`), realizes the intent of the "Request Attribute Value Update" Object Management service as specified the *HLA Interface Specification* (§4.14 in version 1.1; §6.15 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   requestObjectAttributeValueUpdate (
      RTI::ObjectID theObject
      const RTI::AttributeHandleSet& theAttributes
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   requestObjectAttributeValueUpdate (
      RTI::ObjectHandle theObject      ← Changes Types
      const RTI::AttributeHandleSet& theAttributes
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectNotKnown,                ← RTI 1.3 Only
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the object instance whose instance-attributes are to be solicited

theAttributes

the instance-attributes of the specified object instance to solicit

DESCRIPTION

This method solicits an update of the specified instance-attributes from the federation. A remote federate will receive a `provideAttributeValueUpdate()` callback for any solicited instance-attributes owned by the federate. Upon receipt of such a callback, the remote federate is expected to comply with the request by issuing an `updateAttributeValues()` for the solicited instance-attributes.

This service may be used by a late-arriving (or late-subscribing) federate to solicit updates for all existing object instances. It is particularly useful for instance-attributes that are updated infrequently (or not at all) after instantiation.

RETURN VALUES

A non-exceptional return indicates that updates of the specified instance-attributes have been solicited from the federation.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown **(RTI 1.3 Only)**

The specified object ID is not valid within the current FedExec or is not known to the Federate.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

provideAttributeValueUpdate()

reflectAttributeValues()

RTI::RTIambassador::

discoverObject()

requestClassAttributeValueUpdate()

A.3.11 sendInteraction()

RTI 1.0 RTI 1.3

ABSTRACT

This service generates an interaction event in the federation. **The syntax of this service changes from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Send Interaction" Object Management service as specified in the *HLA Interface Specification* (§4.6 in version 1.1; §6.6 and 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::EventRetractionHandle
RTI::RTIambassador::
   sendInteraction (
       RTI::InteractionClassHandle theInteraction
       const RTI::ParameterHandleValuePairSet&
           theParameters
       RTI::FederationTime theTime
       const RTI::UserSuppliedTag theTag
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotPublished,
   RTI::InteractionParameterNotDefined,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
RTI::EventRetractionHandle
RTI::RTIambassador::
   sendInteraction (
       RTI::InteractionClassHandle     theInteraction,
       const RTI::ParameterHandleValuePairSet&
           theParameters,
       const RTI::FedTime& theTime,     ← Changes Types
       const char* theTag               ← Changes Types
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotPublished,
   RTI::InteractionParameterNotDefined,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   sendInteraction (
       RTI::InteractionClassHandle   theInteraction,
       const RTI::ParameterHandleValuePairSet&
           theParameters,
       const char                    *theTag
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotPublished,
   RTI::InteractionParameterNotDefined,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theInteraction
: the class of the interaction

theParameters
: a set of associated values for a subset of the parameters of the specified interaction class

theTime
: a logical time used to determine the time-stamp-ordering of the interaction

theTag
: a string passed to resulting invocations of `receiveInteraction()`; this value is not interpreted by the RTI and may be used to communicate federation-specified information about the interaction

DESCRIPTION

This service may be used by the federate to communicate an interaction to the federation. Interactions are used to represent transient events in the federation (e.g. collisions among objects) or any other information that is not appropriately represented as persistent attribute state.

Interactions sent using this service are associated with the default region (i.e. the region spanning the entire routing space to which the interaction class is bound in the FED file.) The transportation and ordering services used to communicate the interaction are those bound to the interaction class statically in the FED file or dynamically using the `changeInteractionTransportType()` or `changeInteractionOrderType()` service, respectively.

Interaction instances will be delivered, using the `receiveInteraction()` callback, to remote federates subscribing to the associated interaction class or a superclass of the associated interaction class. If an interaction is *promoted* to a more general interaction class, only values for parameters present in the delivered class will be presented to the federate.

An interaction instance will be delivered as time-stamp-ordered (TSO) to a remote subscribing federate if any only if:

- The federate initiating the interaction is time-regulating at the time the interaction is sent.
- A logical time argument is provided to the `sendInteraction()` service invocation resulting in the interaction. (In RTI 1.0, this is always the case.)
- The interaction class is associated with a TSO ordering service in the FED file or through a subsequent `changeInteractionOrderType()` service invocation.
- The remote federate is time-constrained at the point at which the interaction is received and the point at which the interaction is delivered.

The RTI does not enforce the inclusion of values for all parameters of an interaction class for instances of that class. The parameter set associated with an interaction instance may contain values for any subset of the parameters of the specified interaction class and its superclasses.

RETURN VALUES

A non-exceptional return indicates that the interaction will be delivered, using the `receiveInteraction()` callback, to remote federates whose subscription interests match the class and region of the interaction instance.

The timed variant of this service returns an event handle which uniquely identifies the event for purposes of retraction.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::InteractionClassNotPublished

The operation attempted requires that the interaction class be currently published by the federate.

RTI::InteractionParameterNotDefined

One or more of the specified parameter handles is not valid in the context of the specified interaction class.

RTI::InvalidFederationTime

The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::ParameterHandleValuePairSet

RTI::FederateAmbassador::

receiveInteraction()

startInteractionGeneration()

RTI::RTIambassador::

changeInteractionOrderType()

changeInteractionTransportType()

publishInteractionClass()

retract()

subscribeInteractionClass()

tick()

turnRegulationOn()

RTI 1.3

RTI::ParameterHandleValuePairSet

RTI::FederateAmbassador::

receiveInteraction()

turnInteractionsOn()

RTI::RTIambassador::

changeInteractionOrderType()

changeInteractionTransportType()

enableTimeRegulation()

publishInteractionClass()

retract()

subscribeInteractionClass()

subscribeInteractionClassWithRegion()

tick()

A.3.12 updateAttributeValues()

RTI 1.0 RTI 1.3

ABSTRACT

This service notifies the federation of a change in values for one or more instance-attributes of an object instance. **The syntax of this service changes from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Update Attribute Values" Object Management service as specified in the *HLA Interface Specification* (§4.3 in version 1.1;§6.4 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::EventRetractionHandle
RTI::RTIambassador::
   updateAttributeValues (
       RTI::ObjectID theObject
       const RTI::AttributeHandleValuePairSet&
          theAttributes
       RTI::FederationTime theTime
       const RTI::UserSuppliedTag theTag
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotOwned,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidFederationTime,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
RTI::EventRetractionHandle
RTI::RTIambassador::
   updateAttributeValues (
       RTI::ObjectHandle theObject,     ← Changes Types
       const AttributeHandleValuePairSet&
          theAttributes,
       RTI::FedTime&theTime,            ← Changes Types
       const char* theTag               ← Changes Types
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotOwned,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidFederationTime,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   updateAttributeValues (
       RTI::ObjectHandle                theObject,
          const RTI::AttributeHandleValuePairSet&
             theAttributes,
       const char                       *theTag
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotOwned,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theObject
: the object instance whose instance-attributes are being updates

theAttributes
: the set of instance-attributes being updated

theTime
: a logical time used for time-stamp-ordering of the update

theTag
: a string passed to resulting invocations of `reflectAttributeValues()`; this value is not interpreted by the RTI and may be used to communicate federation-specified information about the update

DESCRIPTION

This service may be used by the federate to communicate a change in state of an object instance to the federation. Attributes are used to represent persistent characteristics of federation state (e.g. the position of an entity.)

An update of an instance-attribute sent using this service is associated with the region associated with the instance-attribute through a local invocation of `registerObjectInstanceWithRegion()` or `associateRegionForUpdates()`, or with the default region if no region has been specifically associated with the instance-attribute. (The default region is the region spanning the entire routing space to which the class-attribute is bound in the FED file.) Instance-attributes that are transferred between federates using ownership management services revert to the default region.

The transportation and ordering services used to communicate an update of an instance-attribute initiated using this service are those bound to the corresponding class-attribute statically in the FED file or dynamically using the `changeAttributeTransportType()` or `changeAttributeOrderType()` service, respectively.

If instance-attributes subject to an `updateAttributeValues()` service invocation differ in their characteristic region/transport/order tuples, multiple updates (and subsequently multiple reflections) will be generated as a result of the invocation. One update will be generated for each region/transport/order tuple; it will contain all instance-attributes that share that characteristic tuple.

If an update is received by a federate whose subscription matches the associated class/region for some instance-attribute, the update will result in a `discoverObjectInstance()` callback immediately prior to the reflection.

Instance-attribute updates will be delivered, using the `reflectAttributeValues()` callback, to remote federates subscribing to the instance-attributes at the level of the class by which the object instance is discovered by the federate. Only values for those instance-attributes whose corresponding class-attributes are present in the discovered object class of the object instance will be presented to a recipient of the update. Only values for those instance-attributes whose associated update region intersects the remote federate's subscription region for the class-attribute at the level of the discovered object class will be presented to the federate as a result of the update.

An instance-attribute update will be delivered as time-stamp-ordered (TSO) to a remote subscribing federate if any only if:

- The federate initiating the update is time-regulating at the time the update is sent.
- A logical time argument is provided to the `updateAttributeValues()` service invocation resulting in

the interaction. (In RTI 1.0, this is always the case.)

- The instance-attribute class is associated with a TSO ordering service in the FED file or through a subsequent `changeAttributeOrderType()` service invocation.
- The remote federate is time-constrained at the point at which the update is received and the point at which the update is delivered.

The RTI does not enforce the inclusion of values for all class-attributes of an object class for updates of instances of that class. The attribute set associated with an update may contain values for any subset of the attributes of the registered class of the object instance and its superclasses.

RETURN VALUE

A non-exceptional return indicates that the update will be delivered, using the `reflectAttributeValues()` callback, to remote federates whose subscription interests match the class and regions of the update.

The timed variant of this service returns an event handle which uniquely identifies the event for purposes of retraction.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotOwned

One or more of the specified attribute-instances is not owned by the local federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidFederationTime

The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the Federate.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::AttributeHandleValuePairSet

RTI::FederateAmbassador::

discoverObject()

reflectAttributeValues()

RTI::RTIambassador::

changeAttributeOrderType()

changeAttributeTransportType()

publishObjectClass()

queryAttributeOwnership()

registerObject()

retract()

subscribeObjectClassAttributes()

tick()

turnRegulationOn()

RTI 1.3

RTI::AttributeHandleValuePairSet

RTI::FederateAmbassador::

attributesInScope()

discoverObjectInstance()

reflectAttributeValues()

turnUpdatesOnForObjectInstance()

RTI::RTIambassador::

associateRegionForUpdates()

changeAttributeOrderType()

changeAttributeTransportType()

enableTimeRegulation()

publishObjectClass()

registerObjectInstance()

registerObjectInstanceWithRegion()

retract()

subscribeObjectClassAttributes()

subscribeObjectClassAttributesWithRegion()

tick()

A.4 Ownership Management

A.4.1 attributeIsOwnedByFederate()

RTI 1.0

ABSTRACT

This service queries the RTI as to which federate, if any, holds the attribute ownership token for a given attribute. **The RTI 1.3 version of this service is named `isAttributeOwnedByFederate`; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Is Attribute Owned By Federate" Ownership Management service as specified in the *HLA Interface Specification* (§5.9 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

RTI::Boolean
RTI::RTIambassador::
   attributeIsOwnedByFederate (

       RTI::AttributeHandle theAttribute
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotKnown,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theObject

Object whose attribute ownership status is being queried.

theAttribute

Attribute whose ownership status is being queried.

DESCRIPTION

`attributeIsOwnedByFederate()` provides a facility for the federate to quickly and synchronously determine whether a given ownership token is locally held.

Note that an attribute for which a federate has outstanding negotiated divestiture requests is still considered to be held by the federate until ownership is assumed by another federate.

RETURN VALUES

A successful invocation of this service returns *RTI::RTI_TRUE* if the specified ownership token is held by the local federate or *RTI::RTI_FALSE* if the ownership token is held by another federate, unowned, or no longer exists.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotKnown

(Not thrown in 1.0.)

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateInternalError

An error internal to the federate has occurred; this exception will result in an entry being made to the federate's RTI log.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

requestAttributeOwnershipAcquisition()

requestAttributeOwnershipDivestiture()

A.4.2 attributeOwnershipAcquisition()

RTI 1.3

ABSTRACT

This service initiates an attempt to acquire one or more instance-attributes of a specified object instance. **The RTI 1.0 implementation of this service is named *`requestAttributeOwnershipAcquisition`*; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Attribute Ownership Acquisition" Ownership Management service as specified in the *HLA Interface Specification* (§7.7 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   attributeOwnershipAcquisition (
         RTI::ObjectHandle        theObject,
  const RTI::AttributeHandleSet& desiredAttributes,
  const char                     *theTag
   )
throw (
  RTI::AttributeNotDefined,
  RTI::AttributeNotPublished,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::FederateOwnsAttributes,
  RTI::ObjectClassNotPublished,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the object instance whose instance-attributes are requested

desiredAttributes

the instance-attributes of the specified instance 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

DESCRIPTION

This service initiates a request to transfer ownership of a specified set of instance-attributes of a specified object instance to a federate. No instance-attributes in an acquisition request may be owned by the requesting federate, and all class-attributes must be published by the requesting federate at the level of the instance's discovered object class. Failure to meet these two conditions will result in a `FederateOwnsAttributes` or `ObjectClassNotPublished` exception, respectively.

Some of the requested instance-attributes may exist in the federation as unowned attributes. This can occur if

- a federate registers an instance of an object class for which it is not publishing all class-attributes
- a federate resigns using the release-attributes resignation policy
- a federate unconditionally divests attributes
- a federate implicitly or explicitly unpublishes a class-attribute for which it owns instances
- an attribute acquisition request by a federate succeeds, but the federate has unpublished the class-attribute in the interim

Ownership tokens associated with unowned attributes are tracked internally by the RTI. Unowned ownership tokens typically "reside" in the LRC associated with some federate in the federation. These attributes may be acquired by any federate in the federation.

If an instance-attribute is unowned and its ownership token is tracked by the local LRC, the acquiring federate will receive an `attributeOwnershipAcquisitionNotification()` callback synchronously with respect to the `attributeOwnershipAcquisition()` service invocation. If an instance-attribute is unowned and its ownership token is tracked by a remote LRC, the acquiring federate will receive an `attributeOwnershipAcquisitionNotification()` callback during a subsequent invocation of `tick()`.

If an instance-attribute is owned by a remote federate, the remote federate will be requested, in the form of a `requestAttributeOwnershipRelease()` callback, to release ownership of the instance-attribute. (A given remote federate will actually receive a single callback for all requested instance-attributes that it owns.) The remote federate may release ownership of the instance-attribute (using `unconditionalAttributeOwnershipDivestiture()`, `negotiatedAttributeOwnershipDivestiture()`, or `attributeOwnershipReleaseResponse()`), or it may do nothing.

If a remote federate positively responds to a release request, the ownership of the instance-attribute is transferred to the requesting federate, and the requesting federate is advised of such using the `attributeOwnershipAcquisitionNotification()` callback. If a remote federate does not respond to a release request, the request remains outstanding until it is cancelled by the requesting federate. (The remote federate will only receive a single `requestAttributeOwnershipRelease()` callback regardless of how long the request is outstanding.)

Some instance-attributes may no longer exist in the federation as a result of federates crashing or resigning without releasing ownership of locally owned instance-attributes. The RTI does not attempt to detect and report this condition; an attempt to acquire non-existent instance-attributes will silently fail to succeed.

An instance-attribute may be *temporarily* effectively non-existent in the federation if an ownership transfer message is in-transit between a releasing and acquiring federate.

A requesting federate may wish to withdraw an acquisition request using `cancelAttributeOwnershipAcquisition()` to prevent attribute acquisition requests from succeeding at arbitrary times in the future. A federate should cancel any outstanding attribute acquisition requests before unpublishing a class-attribute (implicitly or explicitly).

If an instance-attribute in an acquisition request is already the subject of an acquisition request by the federate, the acquisition request will be reiterated to the owing federate, if any (i.e., `requestAttributeOwnershipRelease()` will be invoked again.)

If an instance-attribute in an acquisition request is already the subject of an acquisition request *if available* by the federate, the request is changed from *if available* to a regular request and conveyed to the owning federate, if any.

If an instance-attribute in an acquisition request is the subject of an outstanding acquisition cancellation by the federate, the acquisition request is reinstated and reiterated to the owning federate, if any.

Note that this service results in zero or one `requestAttributeOwnershipRelease()` callbacks for each remote federate in the federation, and zero to min(*n, f-1*) `attributeOwnershipAcquisitionNotification()` callbacks for the requesting federate (where *n* is the number of instance-attributes requested and *f* is the number of federates in the federation.) One `attributeOwnershipAcquisitionNotification()` may occur synchronously with respect to the `attributeOwnershipAcquisition()` service invocation.

RETURN VALUES

A non-exceptional return indicates that the remote federate(s) owning the specified instance-attributes will be requested to relinquish ownership. The requesting federate will be notified of successful acquisitions via subsequent `attributeOwnershipAcquisitionNotification()` callbacks.

INTERFACE SPECIFICATION NOTES

The *HLA Interface Specification* version 1.3 states that a federate must continue publishing an object class while the federate has outstanding attribute ownership acquisition requests for instance-attributes of instances known to the federate as the published object class. The RTI 1.3 does not enforce this restriction. Instead, if an attribute acquisition request succeeds subsequent to the unpublication of the discovered object class, the LRC of the acquiring federate will give the attribute-instances away as if they had been unconditionally divested by the federate.

EXCEPTIONS

RTI::AttributeNotDefined
: One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotPublished
: One or more of the specified attributes are not currently published by the local federate.

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::FederateOwnsAttributes
: One or more attributes of the specified object are owned by the local federate.

RTI::ObjectClassNotPublished
: The operation attempted requires that the object class be currently published by the federate.

RTI::ObjectNotKnown
: The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::
: attributeOwnershipAcquisitionNotification()

: requestAttributeOwnershipRelease()

RTI::RTIambassador::
: attributeOwnershipAcquisitionIfAvailable()

: cancelAttributeOwnershipAcquisition()

: getAttributeHandle()

A.4.3 attributeOwnershipAcquisitionIfAvailable()

RTI 1.3

ABSTRACT

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

HLA IF SPECIFICATION

This method realizes the "Attribute Ownership Acquisition If Available" Ownership Management service as specified in the *HLA Interface Specification* (§7.8 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   attributeOwnershipAcquisitionIfAvailable (
        RTI::ObjectHandle        theObject,
  const RTI::AttributeHandleSet& desiredAttributes
   )
throw (
  RTI::AttributeAlreadyBeingAcquired,
  RTI::AttributeNotDefined,
  RTI::AttributeNotPublished,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::FederateOwnsAttributes,
  RTI::ObjectClassNotPublished,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the object instance whose instance-attributes are requested

desiredAttributes

the instance-attributes of the specified instance for which ownership is requested

DESCRIPTION

This service is similar to `attributeOwnershipAcquisition()`, except that remote federates are not asked to release ownership of any instance-attributes that are currently owned:

- The requesting federate will receive an `attributeOwnershipUnavailable()` callback for any instance-attributes that are currently owned by remote federates.
- The requesting federate will receive an `attributeOwnershipAcquisitionNotification()` callback for any instance-attributes that are not currently owned by any federate. This callback may occur synchronously with respect to the `attributeOwnershipAcquisitionIfAvailable()` service invocation.
- The requesting federate will receive no callback for any instance-attributes that no longer exist in the federation. Canceling the acquisition request after waiting an appropriate period may be in order.

If one or more instance-attributes requested using this service is already the subject of an `attributeOwnershipAcquisition()` or `attributeOwnershipAcquisitionIfAvailable()` request (including a request that is in the process of being cancelled), the `AttributeAlreadyBeingAcquired` exception will be thrown.

RETURN VALUES

A non-exceptional return indicates that the acquisition request has been announced to the federation. The requesting federate will be appraised of success or failure through subsequent callbacks.

EXCEPTIONS

RTI::AttributeAlreadyBeingAcquired

The federate is already in the process of acquiring one or more of the specified attribute-instances.

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotPublished

One or more of the specified attributes are not currently published by the local federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederateOwnsAttributes

One or more attributes of the specified object are owned by the local federate.

RTI::ObjectClassNotPublished

The operation attempted requires that the object class be currently published by the federate.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

attributeOwnershipAcquisitionNotification()

attributeOwnershipUnavailable()

RTI::RTIambassador::

attributeOwnershipAcquisition()

cancelAttributeOwnershipAcquisition()

getAttributeHandle()

A.4.4 attributeOwnershipReleaseResponse()

RTI 1.3

ABSTRACT

This service releases ownership of a set of instance-attributes for a specified instance, in compliance with a `requestAttributeOwnershipRelease()` request. **In RTI 1.0, this information was communicated via the return value of the *`requestAttributeOwnershipRelease`* callback.**

HLA IF SPECIFICATION

This method realizes the "Attribute Ownership Release Response" Ownership Management service as specified in the *HLA Interface Specification* (§7.11 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::AttributeHandleSet*
RTI::RTIambassador::
   attributeOwnershipReleaseResponse (
         RTI::ObjectHandle        theObject,
  const RTI::AttributeHandleSet& theAttributes
   )
throw (
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::FederateWasNotAskedToReleaseAttribute,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the object instance whose instance-attributes are being released

theAttributes

the instance-attributes to release

DESCRIPTION

This service is used to provide a positive response to a remote `attributeOwnershipAcquisition()` request that has been communicated to the local federate using the `requestAttributeOwnershipRelease()` callback. The instance-attributes specified to the service invocation must be the a subset of the instance-attribute subjects of previous `requestAttributeOwnershipRelease()` callbacks.

Upon a successful return from this method, the federate no longer owns the specified instance-attributes. No `attributeOwnershipDivestitureNotification()` callback is made to the federate. The federate(s) requesting ownership will be notified of ownership transfer using the `attributeOwnershipAcquisitionNotification()` callback. If more than one federates have outstanding acquisition requests for the same instance-attribute, ownership will be transferred to the federate whose request was received most recently.

If the `FederateWasNotAskedToReleaseAttribute` exception is thrown unexpectedly, it probably means that the ownership acquisition was cancelled. Currently, HLA makes no provision for informing a federate that a previously requested instance-attribute release has been cancelled.

The `unconditionalAttributeOwnershipDivestiture()` or `negotiatedAttributeOwnershipDivestiture()` service may be used instead of this service to respond to a release request.

RETURN VALUES

A non-exceptional return indicates that the federate has released ownership the specified instance-attributes. The requesting federate will be notified via the `attributeOwnershipAcquisitionNotification()` callback.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotOwned

One or more of the specified attribute-instances is not owned by the local federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederateWasNotAskedToReleaseAttribute

One or more of the attribute-instances are not the subject of a currently outstanding `requestAttributeOwnershipRelease()` notification.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

requestAttributeOwnershipRelease()

RTI::RTIambassador::

getAttributeHandle()

negotiatedAttributeOwnershipDivestiture()

unconditionalAttributeOwnershipDivestiture()

A.4.5 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 method realizes the "Cancel Attribute Ownership Acquisition" Ownership Management service as specified in the *HLA Interface Specification* (§7.13 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   cancelAttributeOwnershipAcquisition (
         RTI::ObjectHandle        theObject,
  const RTI::AttributeHandleSet& theAttributes
   )
throw (
  RTI::AttributeAcquisitionWasNotRequested,
  RTI::AttributeAlreadyOwned,
  RTI::AttributeNotDefined,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the instance for which instance-attribute acquisition is being cancelled

theAttributes

the instance-attributes for which acquisition is being cancelled

DESCRIPTION

This service is used to request the cancellation of an attribute acquisition request previously made by the federate using the `attributeOwnershipAcquisition()` service. The instance-attribute 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 to guard against the race condition that would occur if an acquisition request were cancelled while an acquisition response was in-transit.

Upon receipt of the cancellation request by a remote LRC, the LRC will confirm the cancellation of any of the specified instance-attributes that

- the remote federate has been asked to release using `requestAttributeOwnershipRelease()`
- the remote federate has not yet released using `attributeOwnershipReleaseResponse()`, `unconditionalAttributeOwnershipDivestiture()`, or `negotiatedAttributeOwnershipDivestiture()`

This confirmation is delivered, in the form of a `confirmAttributeOwnershipAcquisitionCancellation()` callback, to the federate requesting the cancellation.

The canceling federate may still receive some `attributeOwnershipAcquisitionNotification()` callbacks for instance-attributes for which an acquisition cancellation has been attempted. This occurs when a remote federate has already responded to the acquisition request when the cancellation request arrives.

If an instance-attribute no longer exists in the federation, the canceling federate will never receive a cancellation confirmation for that instance-attribute.

A remote federate is not informed of the fact that an acquisition request made using the `requestAttributeOwnershipRelease()` callback has been cancelled. An attempt by a remote federate to respond to a cancelled acquisition request using `attributeOwnershipReleaseResponse()` will result in a `FederateWasNotAskedToReleaseAttribute` exception. An attempt by a remote federate to respond to a cancelled acquisition request using `unconditionalAttributeOwnershipDivestiture()` or `negotiatedAttributeOwnershipDivestiture()` will proceed as they would in the absence of the acquisition request.

Note that this service will result in zero to min(*n, f-1*) `confirmAttributeOwnershipAcquisitionCancellation()` callbacks, where *n* is the number of instance-attributes for which a cancellation is requested and *f* is the number of federates in the federation.

RETURN VALUES

A non-exceptional return indicates that the federation has been notified of the cancellation request. The federate will be notified of successful cancellations using the `confirmAttributeOwnershipAcquisitionCancellation()` callback.

EXCEPTIONS

RTI::AttributeAcquisitionWasNotRequested

One or more of the instance-attributes is not the subject of a currently outstanding `attributeOwnershipAcquisition()` request.

RTI::AttributeAlreadyOwned

One or more of the instance-attributes is already owned by the local federate.

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal

state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

requestAttributeOwnershipRelease()

confirmAttributeOwnershipAcquisitionCancellation()

RTI::RTIambassador::

attributeOwnershipAcquisition()

attributeOwnershipReleaseResponse()

getAttributeHandle()

negotiatedAttributeOwnershipDivestiture()

unconditionalAttributeOwnershipDivestiture()

A.4.6 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 method realizes the "Cancel Negotiated Attribute Ownership Divestiture" Ownership Management service as specified in the *HLA Interface Specification* (§7.12 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   cancelNegotiatedAttributeOwnershipDivestiture (
        RTI::ObjectHandle        theObject,
  const RTI::AttributeHandleSet& theAttributes
   )
throw (
  RTI::AttributeDivestitureWasNotRequested,
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the instance for which instance-attribute divestiture is being cancelled

theAttributes

the instance-attributes for which divestiture is being cancelled

DESCRIPTION

This service 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 `negotiatedAttributeOwnershipDivestiture()` request by the local federate
- still owned by the local federate (i.e., has not been the subject of a `attributeOwnershipDivestitureNotification()` callback)

If these two criteria are not met, an invocation of `cancelNegoatiedAttributeOwnershipDivestiture()` will result in an `AttributeDivestitureWasNotRequested` or `AttributeNotKnown` exception, respectively.

If these criteria are met, ownership of the specified instance-attributes is no longer being divested and will not be transferred to a remote federate without explicit approval of the owning federate. Remote federates are not notified that the divestiture has been cancelled. If a remote federate responds to the divestiture (or already has a response in-transit at the time of the divestiture cancellation), it is resolved as if the divestiture had never occurred:

- If the remote federate responds using `attributeOwnershipAcquisition()`, the owing federate will receive a `requestAttributeOwnershipRelease()` callback.
- If the remote federate responds using `attributeOwnershipAcquisitionIfAvailable()`, the remote federate will receive an `attributeOwnershipUnavailable()` callback.

RETURN VALUES

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

EXCEPTIONS

RTI::AttributeDivestitureWasNotRequested

One or more of the instance-attributes is not the subject of a currently outstanding `negotiatedAttributeOwnershipDivestiture()` request.

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotOwned

One or more of the specified attribute-instances is not owned by the local federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

attributeOwnershipDivestitureNotification()

attributeOwnershipUnavailable()

requestAttributeOwnershipAssumption()

RTI::RTIambassador::

attributeOwnershipAcquisitionIfAvailable()

getAttributeHandle()

negotiatedAttributeOwnershipDivestiture()

A.4.7 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.

HLA IF SPECIFICATION

This method realizes the “Is Attribute Owned By Federate” Federation Management service as specified in the *HLA Interface Specification* (§7.17 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::Boolean
RTI::RTIambassador::
   isAttributeOwnedByFederate (
       RTI::ObjectHandle    theObject,
       RTI::AttributeHandle theAttribute
   )
throw (
  RTI::AttributeNotDefined,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
);
```

ARGUMENTS

theObject

the object instance for which instance-attribute ownership is being queried

theAttribute

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

DESCRIPTION

This service 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 `negotiatedAttributeOwnershipDivestiture()` service invocation are still considered owned by the divesting federate until the delivery of a `attributeOwnershipDivestitureNotification()` callback.

RETURN VALUES

A successful invocation of this service returns *RTI::RTI_TRUE* if the specified instance-attribute is owned by the local federate, otherwise it returns *RTI::RTI_FALSE*.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

getAttributeHandle()

queryAttributeOwnership()

A.4.8 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 the absence of an acquiring federate, the instance-attributes will continue to be owned by the divesting federate. **The RTI 1.0 equivalent of this service is named *`requestAttributeOwnershipDivestiture`*; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Negotiated Attribute Ownership Divestiture" Ownership Management service as specified in the *HLA Interface Specification* (§7.3 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   negotiatedAttributeOwnershipDivestiture (
         RTI::ObjectHandle              theObject,
  const RTI::AttributeHandleSet&        theAttributes,
  const char                            *theTag
   )
throw (
  RTI::AttributeAlreadyBeingDivested,
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the object instance whose instance-attributes are to be divested

theAttributes

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

DESCRIPTION

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

- currently owned by the federate initiating the divestiture
- not already the subject of an outstanding `negotiatedAttributeOwnershipDivestiture()` request

If the specified instance-attributes fail to meet these criteria, the service invocation will result in an `AttributeNotOwned` or `AttributeAlreadyBeingDivested` exception, respectively.

If an instance-attribute that is the subject of a `negotiatedAttributeOwnershipDivestiture()` service invocation is the subject of a currently outstanding `attributeOwnershipAcqusition()` 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. Instance-attributes that are immediately divested in this fashion will result in a `attributeOwnershipDivestitureNotification()` callback made synchronously with respect to the `negotiatedAttributeOwnershipDivestiture()` service invocation.

An implication of this is that `negotiatedAttributeOwnershipDivestiture()` may be used instead of `attributeOwnershipReleaseResponse()` to respond to an ownership release request.

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 at the level of the class by which the object instance is discovered by that federate.
2. One or more remote federates may respond to the assumption request using the `attributeOwnershipAcquisition()` or `attributeOwnershipAcquisitionIfAvailable()` service.
3. The LRC of the divesting federate will transfer ownership of an instance-attribute to the first remote federate for which such a response is received. At this time, the divesting federate will receive an `attributeOwnershipDivestitureNotification()` callback to inform it that ownership of the instance-attribute has been divested.

If a remote federate's response is received after the ownership of the requested instance-attribute has already been divested, the request will be ignored by the LRC of the federate that divested ownership. Depending on the timing, a late response may result in the federate that acquired that instance-attribute receiving an `requestAttributeOwnershipRelease()` callback (if the response was in the form of `attributeOwnershipAcquisition()`) or the requesting federate receiving an `attributeOwnershipUnavailable()` callback (if the response was in the form of `attributeOwnershipAcquisitionIfAvailable()`). If the acquisition request is received by the federate to which ownership is being divested *while* the divestiture notification is in-transit, no callbacks will be made (i.e., it will be as if the instance-attribute is temporarily non-existent in the federation).

If no remote federates immediately respond to a divestiture request, the instance-attributes remain in a state of divestiture indefinitely. They may be subsequently acquired by any federate; however, no additional `requestAttributeOwnershipAssumption()` callbacks will be made after the initial divestiture attempt.

The divesting federate may subsequently cancel the negotiated divestiture of any instance-attributes that have not already been successfully divested using the `cancelNegotiatedAttributeOwnershipDivestiture()` service. The divesting federate may unconditionally divest

instance-attributes that are the subject of outstanding negotiated divestiture requests.

RETURN VALUES

A non-exceptional return indicates that a negotiated divestiture of the specified instance-attributes has been initiated. The federate may receive `attributeOwnershipDivestitureNotification()` callbacks for some or all of the divested instance-attributes.

EXCEPTIONS

RTI::AttributeAlreadyBeingDivested

A negotiated divestiture is already in progress for one or more of the attribute-instances.

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotOwned

One or more of the specified attribute-instances is not owned by the local federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

attributeOwnershipDivestitureNotification()

requestAttributeOwnershipAssumption()

RTI::RTIambassador::

cancelNegotiatedAttributeOwnershipDivestiture()

getAttributeHandle()

unconditionalAttributeOwnershipDivestiture()

A.4.9 queryAttributeOwnership()

RTI 1.0 RTI 1.3

ABSTRACT

This service determines which federate (if any) holds the attribute ownership token for a given instance-attribute. **The names and semantics of the resulting callbacks change from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Query Attribute Ownership" Ownership Management service as specified in the *HLA Interface Specification* (§5.7 in version 1.1; §7.15 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   queryAttributeOwnership (
       RTI::ObjectID theObject
       RTI::AttributeHandle theAttribute
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotKnown,              ← RTI 1.0 Only
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectNotKnown,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
 )

// RTI 1.3 Only
void
RTI::RTIambassador::
   queryAttributeOwnership (
       RTI::ObjectHandle theObject      ← Changes Types
       RTI::AttributeHandle theAttribute
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectNotKnown,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
 )
```

ARGUMENTS

theObject

the object-instance of the instance-attribute whose ownership is being queried

theAttribute

the instance-attribute whose ownership is being queried

DESCRIPTION

This service queries the federation as to the ownership status of a specified instance-attribute. If the LRC of the querying federate can provide an answer immediately, i.e.,

- the instance-attribute is owned by the querying federate
- the instance-attribute is owned internally by the LRC associated with the querying federate
- the instance-attribute is unowned but tracked by the local LRC

the LRC will provide a response via a callback occurring synchronously with respect to the `queryAttributeOwnership()` service invocation. One of three callbacks will result from this query:

`attributeIsNotOwned()`,

`attributeOwnedByRTI()`, or

`informAttributeOwnership()`,

as appropriate.

If the LRC cannot provide a response immediately, a query is sent to the federation. Upon receipt of the query by a remote LRC, the LRC will send a response if the instance-attribute exists at the LRC (whether it is owned by the federate, the RTI, or unowned). This response will be delivered to the querying federate in the form of a federate ambassador callback during some subsequent invocation of `tick()`.

If an instance-attribute no longer exists in the federation (i.e., a federate has crashed or resigned without releasing ownership), the querying federate will never receive a response. Note that instance-attributes being transferred among federates will temporarily be effectively non-existent while the transfer message is in-transit.

RETURN VALUES

A non-exceptional return indicates that the federate has been informed of the ownership status of the instance-attribute via a federate ambassador callback, or that the ownership query has been sent to the federation.

RELEASE NOTES

RTI 1.0

- Instance-attributes of MOM objects that are registered on behalf of a federate by an LRC will be reported as being owned by the federate.
- The querying federate will never receive a response if the instance-attribute is unowned.
- The `attributeIsOwnedByFederate()` service may be used as a quick-test to synchronously determine whether an instance-attribute is owned by the local federate.

RTI 1.3

- Instance-attributes of MOM objects that are registered on behalf of a federate by an LRC will be reported as being owned by the RTI.
- The querying federate will receive a `attributeIsNotOwned()` callback if the instance-attribute exists in the federation but is currently unowned (i.e., it is tracked by some LRC in the federation).
- The `isAttributeOwnedByFederate()` service may be used as a quick-test to synchronously determine whether an instance-attribute is owned by the local federate.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotKnown **(RTI 1.0 Only)**

This exception is not thrown by the RTI 1.0 implementation.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown
The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::

informAttributeOwnership()

RTI::RTIambassador::

attributeIsOwnedByFederate()

getAttributeHandle()

RTI 1.3

RTI::FederateAmbassador::

attributeIsNotOwned()

attributeOwnedByRTI()

informAttributeOwnership()

RTI::RTIambassador::

getAttributeHandle()

isAttributeOwnedByFederate()

A.4.10 requestAttributeOwnershipAcquisition()

RTI 1.0

ABSTRACT

This service informs the federation of the federate's desire to acquire ownership of a specified set of attributes for a specified object. **The RTI 1.3 implementation of this service is named *`attributeOwnershipAcquisition`*; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Request Attribute Ownership Acquisition" Ownership Management service as specified in the *HLA Interface Specification* (§5.5 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   requestAttributeOwnershipAcquisition (
       RTI::ObjectID theObject
       const RTI::AttributeHandleSet&
           desiredAttributes
       const RTI::UserSuppliedTag theTag
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotPublished,
   RTI::AttributeNotSubscribed,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederateOwnsAttributes,
   RTI::ObjectClassNotPublished,
   RTI::ObjectClassNotSubscribed,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theObject

Object whose attributes the federate wishes to acquire ownership of.

theAttributes

Attributes of the specified object that the federate wishes to acquire ownership of. The caller is responsible for freeing the storage associated with this set and may do so at any time after the completion of the call.

theTag

String value that can contain a description of the reason for the acquisition request or any other data that is meaningful for a particular federation. The caller is responsible for freeing the storage associated with this string and may do so at any time after the completion of the call.

DESCRIPTION

This method informs the federation that the federate wishes to acquire ownership of the specified attributes of the specified object. If the ownership token for a given attribute exists but is not held by any federate (technically, the token is held by the object manager of some federate but not owned by the associated federate ambassador), the federate is automatically granted ownership of the attribute. If the token is held by a federate, that federate is requested to relinquish control via its *FederateAmbassador::requestAttributeOwnershipRelease* method.

If the federate successfully acquires some or all of the requested attributes, it will be notified via its *FederateAmbassador::attributeOwnershipAcquisitionNotification* method. It is possible that a single acquisition request result in multiple acquisition notifications, as different subsets of the set of requested attributes may be held by different federates. These notifications do not occur synchronously with respect to the acquisition request method, but will occur at a later time in response to an *RTIambassador::tick* invocation.

There is no negative acknowledgement of ownership acquisition requests; if a given attribute ownership token no longer exists or if it is held by a federate that declines to relinquish control, then no further actions result from the ownership acquisition request for that attribute.

A federate must publish and subscribe a given attribute and its associated object class before attempting to acquire tokens for that attribute. (Note that here "object class" refers to the class by which the federate knows an object, which may differ from the actual class of the object.)

RETURN VALUES

A non-exceptional return indicates that a request has been made on behalf of the federate to obtain ownership of the specified attributes. It is important to note that a successful return of this method does not imply acquisition of the requested attributes, only that the request has been successfully issued.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotPublished

One or more of the specified attribute handles is not published by the federate.

RTI::AttributeNotSubscribed

One or more of the specified attribute handles is not subscribed by the federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederateOwnsAttributes

The federate already holds ownership tokens for one or more of the attributes specified.

RTI::ObjectClassNotPublished

The object class of the specified object is not published by the federate.

RTI::ObjectClassNotSubscribed

The object class of the specified object is not subscribed by the federate.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log

file for more details.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::
attributeOwnershipAcquisitionNotification()
requestAttributeOwnershipRelease()

RTI::RTIambassador::
publishObjectClass()
queryAttributeOwnership()
subscribeObjectClassAttribute()

A.4.11 requestAttributeOwnershipDivestiture()

RTI 1.0

ABSTRACT

This service informs the federation that the federate wishes to relinquish ownership of a specified set of object attributes and solicits bids to assume ownership of said attributes. **The equivalent RTI 1.3 functionality is provided by the *unconditionalAttributeOwnershipDivestiture* and *negotiatedAttributeOwnershipDivestiture* services; the RTI 1.3 implementation is discussed in separate sections.**

HLA IF SPECIFICATION

This method realizes the "Request Attribute Ownership Divestiture" Ownership Management service as specified in the *HLA Interface Specification* (§5.1 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

enum RTI::OwnershipDivestitureCondition {
   NEGOTIATED = 1,
   UNCONDITIONAL
};

void
RTI::RTIambassador::
   requestAttributeOwnershipDivestiture (
       RTI::ObjectID theObject
       const RTI::AttributeHandleSet& theAttributes
       RTI::OwnershipDivestitureCondition theCondition
       const RTI::UserSuppliedTag theTag
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotOwned,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidDivestitureCondition,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

A variation allows the divesting federate to designate which federates may submit bids for attribute ownership:

```
void
RTI::RTIambassador::
   requestAttributeOwnershipDivestiture (
       RTI::ObjectID theObject
       const RTI::AttributeHandleSet& theAttributes
       RTI::OwnershipDivestitureCondition theCondition
       const RTI::UserSuppliedTag theTag
       const RTI::FederateHandleSet& theCandidates
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotOwned,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateDoesNotExist,
   RTI::FederateNotExecutionMember,
   RTI::InvalidDivestitureCondition,
   RTI::ObjectNotKnown,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
)
```

ARGUMENTS

theObject

Object whose attributes the federate wishes to divest ownership

theAttributes

Set of attributes that the federate wishes to divest ownership of. The caller is responsible for freeing the memory associated with the set and may do so at any time after the completion of the call.

theCondition

Enumerated value indicating whether the federate wishes to be relieved of attribute publication responsibility even in the absence of an accepting federate.

theTag

String value that can contain a description of the reason for the divestiture request or any other data that is meaningful for a particular federation. The caller is responsible for freeing the storage associated with this string and may do so at any time after the completion of the call.

theCandidates

Set of federates that are to be offered ownership of the attributes being divested. The caller is responsible for freeing the storage associated with this set and may do so at any time after the completion of the call.

DESCRIPTION

The federate uses this method to request to be relieved of attribute publication responsibility. The federate must hold the ownership token for all attributes it attempts to divest; multiple negotiated divestiture requests may be made for the same object attribute in the absence of an acquiring federate.

The request sent out by this method will result in the invocation of the *FederateAmbassador::requestAttributeOwnershipAssumption* method in each of the candidate federates (or all federates join in the execution if no candidate federates were specified.) This may result in one or more of these federates making a bid for some or all of the offered attributes; the divesting federate's object manager will transfer the ownership token for a given attribute to the submitter of the first bid (or attribute acquisition request) received for the attribute.

If the federate specifies a negotiated divestiture, it will retain ownership of a given attribute until the receipt of a notification that another federate is willing to assume ownership (this notification can consist of an ownership bid made in response to the attribute assumption request or an attribute acquisition request made independently.) At this point, the federate's object manager will send an attribute assumption notification to the assuming federate and inform the divesting federate, via a *FederateAmbassador::attributeOwnershipDivestitureNotification* callback, that it has been relieved of ownership of the attribute.

If the federate specifies an unconditional divestiture, it is immediately relieved of ownership of all divested attributes and is notified of this via a *IFederateAmbassador::attributeOwnershipDivestitureNotification* callback (this callback does not occur synchronously with respect to the attribute divestiture request, but is queued up for future processing by the *RTIambassador::tick* service.) At this point, the ownership tokens are not held by any federate (technically, the tokens still reside in the object manager of the divesting federate) and will be transferred to the first federate expressing interest (i.e. responding to the attribute assumption request or submitting an independent request for attribute acquisition.)

Note that the candidate set only defines which federates are explicitly offered ownership of the tokens via the *FederateAmbassador::requestAttributeOwnershipAssumption* method; this does not prevent non-candidate federates from assuming ownership of the tokens by making an attribute

acquisition request via the *RTIambassador::requestAttributeOwnershipAcquisition* service.

There is no negative acknowledgement of a negotiated attribute divestiture request; if the federate receives no positive response within a reasonable amount of time it may wish to issue another attribute divestiture request.

Multiple ownership divestiture notifications may result from a single divestiture request, as different federates may assume ownership of different subsets of the offered attributes.

RETURN VALUES

A non-exceptional return indicates that the candidate federates (or all federates, if no candidate federates were specified) have been offered ownership of the specified attributes and that the calling federate's object manager has been instructed to transfer the attribute ownership tokens to the first federate willing to assume them. If an unconditional divestiture was specified, an *attributeOwnershipDivestitureNotification* has been queued for delivery to the federate ambassador on a subsequent *RTIambassador::tick* (if a negotiated divestiture was specified, these notifications are given only after a given attribute has been divested.)

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotOwned

One or more of the specified attribute-instances is not owned by the local federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederateDoesNotExist

One or more of the specified federate handles is not valid in the context of the current active federation.

RTI::InvalidDivestitureCondition

The specified divestiture condition was not a recognized enumerated value.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet
RTI::FederateHandleSet
RTI::FederateAmbassador::
attributeOwnershipDivestitureNotification()
requestAttributeOwnershipAssumption()
RTI::RTIambassador::
queryAttributeOwnership()
requestAttributeOwnershipAcquisition()

A.4.12 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. **The RTI 1.0 equivalent of this service is named *`requestAttributeOwnershipDivestiture`*; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Unconditional Attribute Ownership Divestiture" Ownership Management service as specified in the *HLA Interface Specification* (§7.2 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   unconditionalAttributeOwnershipDivestiture (
         RTI::ObjectHandle            theObject,
  const RTI::AttributeHandleSet&      theAttributes
   )
throw (
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the object instance whose instance-attributes are to be divested

theAttributes

the instance-attributes to divest

DESCRIPTION

This service immediately releases ownership of the specified instance-attributes.

No `attributeOwnershipDivestitureNotification()` callback is made to the federate.

If a released instance-attribute is the subject of an outstanding `attributeOwnershipAcquisition()` request, ownership is immediately transferred to a requesting federate. The requesting federate will receive an `attributeOwnershipAcquisitionNotification()` callback to inform it of the ownership transfer. If more than one remote federates have currently outstanding acquisition requests for the same instance-attribute, ownership is transferred based on whichever request was received most recently.

If a released instance-attribute is not the subject of an outstanding `attributeOwnershipAcquisition()` request, the instance-attribute becomes unowned. Each remote federate that is publishing the corresponding class-attribute at the level of the object class by which the instance is discovered by the remote federate will be offered ownership using the `requestAttributeOwnershipAssumption()` callback. If multiple federates respond to the assumption request, ownership will be transferred to the federate whose response is received first. If no federates respond, the instance-attribute continues to be unowned and may be acquired using the `attributeOwnershipAcquisition()` or `attributeOwnershipAcquisitionIfAvailable()` services. No `requestAttributeOwnershipAssumption()` callbacks will be made after the initial divestiture.

RETURN VALUES

A non-exceptional return indicates that the specified instance-attributes are no longer owned by the federate. Other federates may be offered ownership using the `requestAttributeOwnershipAssumption()` callback.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotOwned

One or more of the specified attribute-instances is not owned by the local federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::FederateAmbassador::

requestAttributeOwnershipAssumption()

RTI::RTIambassador::

attributeOwnershipReleaseResponse()

getAttributeHandle()

negotiatedAttributeOwnershipDivestiture()

A.5 Time Management

A.5.1 changeAttributeOrderType()

RTI 1.0 RTI 1.3

ABSTRACT

This service specifies the ordering policy for a specified set of instance-attributes of a specified object instance to use for updates made by the local federate. **The type (but not semantics) of one parameter changes between RTI 1.0 and RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Change Attribute Order Type" service as specified in the *HLA Interface Specification* (§4.11 (Object Management) in version 1.1; §8.23 (Time Management) in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
enum RTI::OrderType {
   RECEIVE = 1,
   TIMESTAMP
};

// RTI 1.0 Only
void
RTI::RTIambassador::
   changeAttributeOrderType (
      RTI::ObjectID theObject,
      const RTI::AttributeHandleSet& theAttributes,
      RTI::OrderType theType
   )
throw (
   RTI::AttributeNotDefined,
   RTI::AttributeNotOwned,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidOrderType,
   RTI::ObjectNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void changeAttributeOrderType (
        RTI::ObjectHandle theObject,    ← Changes Types
  const RTI::AttributeHandleSet& theAttributes,
        RTI::OrderingHandle theType     ← Changes Types
   )
throw (
  RTI::AttributeNotDefined,
  RTI::AttributeNotOwned,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InvalidOrderingHandle,           ← Changes Types
  RTI::ObjectNotKnown,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theObject

the object-instance whose instance-attributes are affected

theAttributes

the instance-attributes whose ordering policy is being set

theType

the ordering policy to use for subsequent updates of the specified instance-attributes by the local federate

DESCRIPTION

This service specifies the ordering policy to be associated with subsequent updates of the specified instance-attributes made by the local federate. Recall that an update of an instance-attribute is delivered in time-stamp order to a subscribing federate if and only if

- the sending federate is time-regulating
- the receiving federate is time-constrained
- a time-stamp-ordered ordering policy is in effect for the specified instance-attribute at the sending federate
- a time-stamp is specified to the `updateAttributeValues()` service invocation (in RTI 1.0, this is necessarily the case)

The default ordering policy for an instance-attribute is the ordering policy specified for the corresponding class-attribute in the FED file. The ordering policy for an instance-attribute will revert to the default if the instance-attribute is transferred between federates using ownership management services.

This service will allow an ordering policy to be set for any instance-attributes of any objects known to the federate; however, such a specification is only meaningful for instance-attributes which are owned by the local federate.

RETURN VALUES

A non-exceptional return indicates that subsequent updates of the specified instance-attributes will be sent using the specified ordering-policy (if the sending federate is time regulating).

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::AttributeNotOwned

This exception is not thrown by the current implementations.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidOrderingHandle **(RTI 1.3 Only)**

The specified ordering policy is not a valid handle as returned by the `getOrderingHandle()` service.

RTI::InvalidOrderType **(RTI 1.0 Only)**

The specified ordering policy is not a valid member of the `OrderType` enumeration.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::AttributeHandleSet

RTI::RTIambassador::

changeAttributeTransportType()

changeInteractionOrderType()

setTimeConstrained()

tick()

turnRegulationOn()

updateAttributeValues()

RTI 1.3

RTI::AttributeHandleSet

RTI::RTIambassador::

changeAttributeTransportType()

changeInteractionOrderType()

enableTimeConstrained()

enableTimeRegulation()

getOrderingHandle()

tick()

updateAttributeValues()

A.5.2 changeInteractionOrderType()

RTI 1.0 RTI 1.3

ABSTRACT

This service specifies the ordering policy for a specified class of interactions to use for interactions sent by the local federate. **The type of one argument to this service method changes between RTI 1.0 and RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Change Interaction Order Type" service as specified in the *HLA Interface Specification* (§4.13 (Object Management) in version 1.1; §8.24 (Time Management) in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
enum RTI::OrderType {
   RECEIVE = 1,
   TIMESTAMP
};

// RTI 1.0 Only
void
RTI::RTIambassador::
   changeInteractionOrderType (
       RTI::InteractionClassHandle theClass
       RTI::OrderType theType
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotPublished,
   RTI::InvalidOrderType,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress
)

// RTI 1.3 Only
void
RTI::RTIambassador::
   changeInteractionOrderType (
       RTI::InteractionClassHandle theClass,
       RTI::OrderingHandle theType      ← Changes Types
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InteractionClassNotDefined,
  RTI::InteractionClassNotPublished,
  RTI::InvalidOrderingHandle,          ← Changes Types
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the interaction class affected by the ordering policy

theType

the ordering policy to use for the specified interaction class

DESCRIPTION

This service specifies the ordering policy to be associated with subsequent interactions of the specified interaction class sent by the local federate. Recall that an interaction is delivered in time-stamp order to a subscribing federate if and only if

- the sending federate is time-regulating
- the receiving federate is time-constrained
- a time-stamp-ordered ordering policy is in effect for the interaction class at the sending federate
- a time-stamp is specified to the `sendInteraction()` service invocation (in RTI 1.0, this is necessarily the case)

The default ordering policy for an interaction class is the ordering policy specified in the FED file.

The federate must be publishing the interaction class subject of a `changeInteractionOrderType()` service invocation. If the federate subsequently invokes `publishInteractionClass()` again for the specified interaction class, the ordering policy will revert to the default, even if there has been no intervening `unpublishInteractionClass()` service invocation.

RETURN VALUES

A non-exceptional return indicates that future interactions of the specified class will be sent using the specified ordering policy (if the federate is time regulating).

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::InteractionClassNotPublished

The operation attempted requires that the interaction class be currently published by the federate.

RTI::InvalidOrderingHandle **(RTI 1.3 Only)**

The specified ordering policy is not a valid handle as returned by the `getOrderingHandle()` service.

RTI::InvalidOrderType **(RTI 1.0 Only)**

The specified ordering policy is not a valid member of the `OrderType` enumeration.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI 1.0

RTI::RTIambassador::

changeAttributeOrderType()

changeInteractionTransportType()

sendInteraction()

setTimeConstrained()

tick()

turnRegulationOn()

RTI 1.3

RTI::RTIambassador::

changeAttributeOrderType()

changeInteractionTransportType()

enableTimeConstrained()

enableTimeRegulation()

sendInteraction()

tick()

A.5.3 disableAsynchronousDelivery()

RTI 1.3

ABSTRACT

This service instructs the LRC not to deliver receive-ordered events in the absence of an in-progress time-advancement service, if the federate is time-constrained. **The closest RTI 1.0 equivalent to this service is `dequeueFIFOasynchronously`; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the “Disable Asynchronous Delivery” Time Management service as specified in the *HLA Interface Specification* (§8.15 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   disableAsynchronousDelivery( )
throw (
   RTI::AsynchronousDeliveryAlreadyDisabled,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
);
```

DESCRIPTION

This service disables the delivery of receive-ordered events to the federate in the absence of a time-advancement service. This only applies to a time-constrained federate: if a federate is not time-constrained, events may be delivered during any invocation of the `tick()` service.

Asynchronous delivery is disabled by default for a federate, so this service should only be used to undo the effects of an `enableAsynchronousDelivery()` service invocation.

RETURN VALUES

A non-exceptional return indicates that receive-ordered events will subsequently be delivered only during an in-progress time-advancement service.

EXCEPTIONS

RTI::AsynchronousDeliveryAlreadyDisabled

Asynchronous delivery of FIFO events is already disabled for the local federate.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

reflectAttributeValues()

RTI::RTIambassasor::

enableAsynchronousDelivery()

enableTimeConstrained()

flushQueueRequest()

nextEventRequest()

timeAdvanceRequest()

A.5.4 disableTimeConstrained()

RTI 1.3

ABSTRACT

This service instructs the LRC not to constrain the advancement of the federate's time based on the federation's time, and to deliver all events in receive order. **The closest RTI 1.0 equivalent to this service is `setTimeConstrained`; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Disable Time Constrained" Time Management service as specified in the *HLA Interface Specification* (§8.7 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   disableTimeConstrained ( )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::TimeConstrainedWasNotEnabled
)
```

DESCRIPTION

This service instructs the LRC not to constrain the advancement of the federate's time based on the federation's time. Subsequent invocations of time-advancement services such as `nextEventRequest()` and `timeAdvanceRequest()` will trivially succeed, as there are no longer *any* time-stamp-ordered events in the federation from the perspective of a non-time-constrained federate. (That is, for any arbitrary point on the federation time axis, it can be guaranteed that no time-stamp-ordered events will be delivered with an earlier time stamp.)

Any events subsequently received by the LRC of a non-time-constrained federate will be queued for receive-order delivery to the federate, regardless of the time-ordering policy associated with the event by its sender. Events may be delivered to the federate during any invocation of the `tick()` service, obviating the need to invoke time-advancement services.

If there are any events remaining in the time-stamp-ordered queue at the point time-constraint is disabled, they will be delivered to the federate, in ascending order by time-stamp, before any receive-ordered events are delivered.

Time-constraint is disabled by default for a federate. This service should only be used after time-constraint has been enabled using `enableTimeConstrained()` *and* has been achieved, as indicated by a `timeConstrainedEnabled()` callback.

RETURN VALUES

A non-exceptional return indicates that the LRC will treat all subsequently received events as receive-ordered and will not constrain the advancement of federate time based on the federation logical time.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TimeConstrainedWasNotEnabled

Time-constraint is not currently enabled for the local federate.

SEE ALSO

RTI::FederateAmbassador::
timeConstrainedEnabled()

RTI::RTIambassador::
enableTimeConstrained()

A.5.5 disableTimeRegulation()

RTI 1.3

ABSTRACT

This service instructs the federation to disregard the federate's logical time for purposes of regulating advancement of federation logical time. All events generated subsequently by the federate will be sent as receive-ordered. **The closest RTI 1.0 equivalent to this service is *`turnRegulationOff`*; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Disable Time Regulation" Time Management service as specified in the *HLA Interface Specification* (§8.4 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   disableTimeRegulation ( )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::TimeRegulationWasNotEnabled
)
```

DESCRIPTION

Time-regulation is disabled by default for a federate. This service should only be used after time-regulation has been enabled using `enableTimeRegulation()` *and* has been achieved, as indicated by a `timeRegulationEnabled()` callback.

RETURN VALUES

A non-exceptional return indicates that time regulation has been turned off for the local federate.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TimeRegulationWasNotEnabled

Time-regulation is not currently enabled for the local federate.

SEE ALSO

RTI::FederateAmbassador::
timeConstrainedEnabled()

RTI::RTIambassador::
enableTimeConstrained()

A.5.6 enableAsynchronousDelivery()

RTI 1.3

ABSTRACT

This service instructs the LRC to begin delivering receive-ordered events to the federate even while no time-advancement service is in progress. **The closest RTI 1.0 equivalent to this service is `dequeueFIFOasynchronously`; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the “Enable Asynchronous Delivery” Time Management service as specified in the *HLA Interface Specification* (§8.14 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   enableAsynchronousDelivery( )
throw (
   RTI::AsynchronousDeliveryAlreadyEnabled,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to deliver receive-ordered events to the federate in the absence of an in-progress time-advancement service. Subsequent to invoking this service, receive-ordered events may be delivered to the federate during any invocation of `tick()`. This setting is only relevant for federates that are time-constrained: all events are always delivered asynchronously to non-time-constrained federates.

The asynchronous delivery of receive-ordered events may be subsequently disabled using the `disableAsynchronousDelivery()` service.

RETURN VALUES

A non-exceptional return indicates that asynchronous delivery of receive-ordered events has been enabled for the local federate.

EXCEPTIONS

RTI::AsynchronousDeliveryAlreadyEnabled
: Asynchronous delivery of FIFO events is already enabled for the local federate.

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::
: disableAsynchronousDelivery()

: disableTimeConstrained()

: enableTimeConstrained()

: tick()

A.5.7 enableTimeConstrained()

RTI 1.3

ABSTRACT

This service instructs the LRC to constrain the advancement of the federate's time based on the federation's time, and to deliver time-stamp-ordered events in the correct order. **The closest RTI 1.0 equivalent to this service is *`setTimeConstrained`*; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Enable Time Constrained" Time Management service as specified in the *HLA Interface Specification* (§8.5 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   enableTimeConstrained ( )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::EnableTimeConstrainedPending,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::TimeAdvanceAlreadyInProgress,
  RTI::TimeConstrainedAlreadyEnabled
)
```

DESCRIPTION

This service instructs the LRC to deliver time-stamp-ordered events to the federate in non-decreasing order according to their associated time stamp. Events occurring simultaneously in logical time are subject to an arbitrary order of delivery.

Types of events that are potentially subject to time-stamp ordering are updates, interactions, object deletions, and federation saves. For a particular event to be subject to time-stamp ordering, the following criteria must be met:

- For all events except saves, the sender must be time regulating at the point in execution at which the event is generated.
- For updates and interactions, the ordering policy (as established by the FED file and subsequent invocations of `changeAttributeOrderType()` and `changeInteractionOrderType()`, respectively) in effect for the relevant instance-attributes or interaction class at the sender must be a time-stamp-ordered policy.
- For deletions, the ordering policy (as established by the FED file and subsequent invocations of `changeAttributeOrderType()`) in effect for some instance-attribute of the object at the sender must be a time-stamp-ordered policy.
- The receiver must be time-constrained at the point in execution at which the event is received by the LRC *and* the point in execution at which it delivered to the federate (though not necessary at all subsequent points in-between.)
- A time-stamp argument must be provided to the service invocation resulting in the federation event.

The federate will be notified through its `timeConstrainedEnabled()` callback when time constraint has taken effect. This callback occurs asynchronously to the `enableTimeConstrained()` service invocation, during a subsequent invocation of `tick()`.

Time-stamp-ordered events will only be delivered to a time-constrained federate when a time-advancement service (e.g., `timeAdvanceRequest()` or `nextEventRequest()`) is in progress. A time-stamp-ordered event will not be delivered until the LRC can guarantee that no events will be received with an earlier time-stamp than the event to be delivered. The `timeAdvanceGrant()` service informs the federate that all events with time-stamps less than or equal to (or strictly less than, depending on the time-advancement service used) a specified time have been delivered to the federate.

No time-stamp ordered events will be delivered to a time-constrained federate unless a time-advancement service is in progress. No receive-ordered events will be delivered to a time-constrained federate unless a time-advancement service is in progress *or* the federate has enabled asynchronous delivery of receive ordered events, using the `enableAsynchronousDelivery()` service.

RETURN VALUES

A non-exceptional return indicates that the LRC will enable time constraint for the federate as soon as possible and notify the federate using the `timeConstrainedEnabled()` callback.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::EnableTimeConstrainedPending

Another `enableTimeConstrained()` request is currently outstanding.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TimeAdvanceAlreadyInProgress

The time-management disposition of a federate may not be changed while a time-advancement service is currently in progress.

RTI::TimeConstrainedAlreadyEnabled

Time-constraint is already enabled for the local federate.

SEE ALSO

RTI::FederateAmbassador::

timeConstrainedEnabled()

timeAdvanceGrant()

RTI::RTIambassador::

disableTimeConstrained()

enableAsynchronousDelivery()

enableTimeRegulation()
flushQueueRequest()
nextEventRequest()
nextEventRequestAvailable()
timeAdvanceRequest()
timeAdvanceRequestAvailable()

A.5.8 enableTimeRegulation()

RTI 1.3

ABSTRACT

This service instructs the federation to consider the federate's logical time for the purposes of governing the advancement of federation logical time. **The closest RTI 1.0 equivalents to this service are *`turnRegulationOn`* and *`turnRegulationOnNow`*; the RTI 1.0 implementation is discussed in separate sections.**

HLA IF SPECIFICATION

This method realizes the "Enable Time Regulation" Time Management service as specified in the *HLA Interface Specification* (§8.2 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void enableTimeRegulation (
   const RTI::FedTime& theFederateTime,
   const RTI::FedTime& theLookahead
)
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::EnableTimeRegulationPending,
  RTI::FederateNotExecutionMember,
  RTI::InvalidFederationTime,
  RTI::InvalidLookahead,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::TimeAdvanceAlreadyInProgress,
  RTI::TimeRegulationAlreadyEnabled
);
```

ARGUMENTS

theFederateTime

the minimum logical time to set the federate's logical time to when turning regulation on

theLookahead

the lookahead to use for the federate

DESCRIPTION

This service instructs the federation to consider the federate's logical time in controlling the advancement of federation time. The specified lookahead represents the length of the logical-time interval extending forward from the federate's logical time at any given point in execution in which the federate will not generate events. That is, the minimum permissible time-stamp for a time-stamp-ordered event generated by a time regulating federate is the federate's current logical time plus its lookahead. The sum of a federate's logical time and its lookahead may be referred to as the *effective* logical time of the federate.

The federate's logical time upon enabling time regulation will be the minimum of the specified *effective* federate logical time and the current lower-bound time-stamp of the federation.

If the specified lookahead is less than the federate's current lookahead, the specified lookahead will be phased in such that the effective logical time of the federate remains strictly non-decreasing. The new lookahead will completely take effect when the federate has advanced its logical time from the time at which regulation was enabled by an amount exceeding the difference between the new and old lookahead values.

Enabling time regulation allows the federate to generate events that are subject to time-stamp-ordering. While a federate is time regulating, events sent by the federate may be designated as time-stamp ordered, provided

- a time-stamp ordering policy is in effect for the relevant interaction class or instance-attributes, as defined by the FED file and subsequent invocations of `changeAttributeOrderType()` and `changeInteractionOrderType()`
- a time-stamp argument is provided to the service invocation generating the event

The federate will be notified through its `timeRegulationEnabled()` callback when time constraint has taken effect. This callback occurs asynchronously to the `enableTimeRegulation()` service invocation, during a subsequent invocation of `tick()`.

RETURN VALUES

A non-exceptional return indicates that time regulation will be enabled for the federate as soon as possible.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::EnableTimeRegulationPending

Another `enableTimeRegulation()` request is currently outstanding.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidFederationTime

The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::InvalidLookahead

The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TimeAdvanceAlreadyInProgress

The time-management disposition of a federate may not be changed while a time-advancement service is currently in progress.

RTI::TimeRegulationAlreadyEnabled

Time-regulation is already enabled for the local federate.

SEE ALSO

RTI::FederateAmbassador::
timeRegulationEnabled()

RTI::FedTime
RTI::RTIambassador::
disableTimeRegulation()
enableTimeConstrained()

A.5.9 flushQueueRequest()

RTI 1.0 RTI 1.3

ABSTRACT

This service initiates a flush of the federate's event queues, violating the ordering of time-stamp-ordered messages if necessary. **The type of the argument has changed in syntax, but not in semantics, between RTI 1.0 and RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Flush Queue Request" Time Management service as specified in the *HLA Interface Specification* (§6.9 in version 1.1; §8.12 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   flushQueueRequest (
      RTI::FederationTime theTime
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress,
   RTI::TimeAdvanceAlreadyInProgress
 )

// RTI 1.3 Only
void
RTI::RTIambassador::
   flushQueueRequest (
      const RTI::FedTime& theTime      ← Changes Types
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::EnableTimeConstrainedPending,
  RTI::EnableTimeRegulationPending,
  RTI::FederateNotExecutionMember,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::TimeAdvanceAlreadyInProgress
)
```

ARGUMENTS

theTime

the maximum logical time to which to advance upon completion of the flush

DESCRIPTION

This service designates all events currently in the federate's event queue as eligible for presentation to the federate. Subsequent invocations of `tick()` will first process any receive-ordered events that have arrived, then will process events in the TSO queue without regard for the federation lower-bound time stamp. For any given invocation of `tick()`, the earliest available TSO event is processed; however, RTI may not be able to guarantee that TSO events with a lower time-stamp will not arrive in the future.

A time advance is granted when the federate has processed all TSO events that were queued at the time of the request. The grant time is the minimum of the minimum-next-event time and the specified cutoff time. Note that this is trivial in the case of a non-time-constrained federate, which by definition has no events in its TSO queue; in this case, a grant to the specified cutoff time will be made upon the next invocation of `tick()`.

If time-stamp-ordered events arrive while a flush is in progress, they will be delivered in order with respect to the events remaining in the queue. Late-arriving events may replace events with a later time-stamp in the set of events affected by the flush. That is, a `flushQueueRequest()` invocation will flush a *number* of time-stamp-ordered events equal to the number of events queued for time-stamp-ordered delivery at the time of the service invocation. However, the actual events delivered as a result of the flush may not be the same events that were in the queue at the time of the `flushQueueRequest()` service invocation.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully announced its desire to advance in time; the federate will be notified of the successful completion of the request via a subsequent `timeAdvanceGrant()` callback.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::EnableTimeConstrainedPending

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::EnableTimeRegulationPending

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed

The specified federation time is less than the current logical time of the federation.

RTI::InvalidFederationTime

Not thrown in 1.0.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TimeAdvanceAlreadyInProgress

A previous time advance request, next event request, or flush queue request has not yet been completed.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::
timeAdvanceGrant()

RTI::RTIambassador::
nextEventRequest()
requestFederateTime()
requestLBTS()

setTimeConstrained()

tick()

timeAdvanceRequest()

turnRegulationOn()

RTI 1.3

RTI::FederateAmbassador::

timeAdvanceGrant()

RTI::FedTime

RTI::RTIambassador::

enableTimeConstrained()

enableTimeRegulation()

nextEventRequest()

queryFederateTime()

queryLBTS()

tick()

timeAdvanceRequest()

A.5.10 modifyLookahead()

RTI 1.3

ABSTRACT

This service changes the size of the interval extending forward from the federate's logical time at a given point in execution in which the federate will not generate any time-stamp-ordered events. **The RTI 1.0 equivalent of this method is `setLookahead`; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Modify Lookahead" Time Management service as specified in the *HLA Interface Specification* (§8.19 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   modifyLookahead (
       const RTI::FedTime& theLookahead
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InvalidLookahead,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theLookahead

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

DESCRIPTION

This service specifies a new lookahead value for the local federate. The lookahead represents the length of the logical-time interval extending forward from the federate's logical time at any given point in execution in which the federate will not generate events. That is, the minimum permissible time-stamp for a time-stamp-ordered event generated by a time regulating federate is the federate's current logical time plus its lookahead. The sum of a federate's logical time and its lookahead may be referred to as the *effective* logical time of the federate.

If the specified lookahead is less than the federate's current lookahead, the specified lookahead will be phased in such that the effective logical time of the federate remains strictly non-decreasing. The new lookahead will completely take effect when the federate has advanced its logical time from the time at which regulation was enabled by an amount exceeding the difference between the new and old lookahead values.

The lookahead setting is only meaningful for a time regulating federate: non-time-regulating federates may always generate events with any time stamp.

RETURN VALUES

A non-exceptional return indicates that the federate's lookahead will be adjusted to the specified value as soon as possible.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidLookahead

The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FedTime

RTI::RTIambassador::

enableTimeRegulation()

A.5.11 nextEventRequest()

RTI 1.0 RTI 1.3

ABSTRACT

This service advances the federate's logical time to the time-stamp of the next relevant time-stamp-ordered event in the federation. **The type (but not semantics) of the parameter changes from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

The RTI 1.0 implementation of this method (in conjunction with `nextEventRequestAvailable()`) realizes the "Next Event Request" Time Management service as specified in the *HLA Interface Specification* (§6.8 in version 1.1).

The RTI 1.3 implementation of this method realizes the "Next Event Request" Time Management service as specified in the *HLA Interface Specification* (§8.10 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   nextEventRequest (
       RTI::FederationTime theTime
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress,
   RTI::TimeAdvanceAlreadyInProgress
 )

// RTI 1.3 Only
void
RTI::RTIambassador::
   nextEventRequest (
       const RTI::FedTime& theTime      ← Changes Types
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::EnableTimeConstrainedPending,   ← RTI 1.3 Only
  RTI::EnableTimeRegulationPending,    ← RTI 1.3 Only
  RTI::FederateNotExecutionMember,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::TimeAdvanceAlreadyInProgress
)
```

ARGUMENTS

theTime

the "cutoff point" at which to stop advancing logical time in the absence of any intervening relevant time-stamp-ordered events

DESCRIPTION

This service allows the federate to advance in time to the time-stamp of the next time-stamp-ordered event occurring in the federation which matches the federate's current subscription interests. A `timeAdvanceGrant()` callback will occur after one or more time-stamp-ordered events have been delivered or the federation lower-bound time stamp (LBTS) advances past the specified cutoff time.

In the presence of an event, all relevant events with the same time-stamp will be delivered before the grant. The grant time will be equal to the time-stamp of the event(s) delivered to the federate.

In the absence of a relevant event, the grant time will be the specified cutoff time.

Any number of receive-ordered events may be delivered while the `nextEventRequest()` is in progress.

Note that if the federate is not time-constrained, the grant criteria are trivially met (i.e. the effective federation LBTS for a non-constrained federate is always infinity), so a `timeAdvanceGrant()` to the cutoff time will be immediately scheduled for delivery by a subsequent invocation of `tick()`.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully announced its desire to advance in time; the federate will be notified of the successful completion of the request (as described above) via a subsequent `timeAdvanceGrant()` callback.

RELEASE NOTES

RTI 1.3

Time-advancement services may not be initiated while a request to enable time-regulation or time-constraint is pending.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::EnableTimeConstrainedPending **(RTI 1.3 Only)**

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::EnableTimeRegulationPending **(RTI 1.3 Only)**

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed

The specified federation time is less than the current logical time of the federation.

RTI::InvalidFederationTime

This exception is not thrown by the current implementations.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TimeAdvanceAlreadyInProgress

A previous time advance request, next event request, or flush queue request has not yet been completed.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::

timeAdvanceGrant()

RTI::RTIambassador::

flushQueueRequest()

nextEventRequestAvailable()

requestFederateTime()

requestLBTS()

setTimeConstrained()

tick()

timeAdvanceRequest()

turnRegulationOn()

RTI 1.3

RTI::FederateAmbassador::

timeAdvanceGrant()

RTI::FedTime

RTI::RTIambassador::

enableTimeConstrained()

enableTimeRegulation()

flushQueueRequest()

nextEventRequestAvailable()

queryFederateTime()

queryLBTS()

tick()

timeAdvanceRequest()

A.5.12 nextEventRequestAvailable()

RTI 1.0 RTI 1.3

ABSTRACT

This service is similar to `nextEventRequest()`, except that a time advance might be granted before all time-stamp-ordered events at the grant time have been delivered to the federate.

HLA IF SPECIFICATION

The RTI 1.0 implementation of this method (in conjunction with `nextEventRequest()`) realizes the "Next Event Request" Time Management service as specified in the *HLA Interface Specification* (§6.8 in version 1.1).

The RTI 1.3 implementation of this method realizes the "Next Event Request Available" Federation Management service as specified in the *HLA Interface Specification* (§8.11 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   nextEventRequestAvailable (
      RTI::FederationTime theTime
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::InvalidFederationTime,
   RTI::RTIinternalError,
   RTI::RestoreInProgress,
   RTI::SaveInProgress,
   RTI::TimeAdvanceAlreadyInProgress
 )

// RTI 1.3 Only
void
RTI::RTIambassador::
   nextEventRequestAvailable (
      const RTI::FedTime& theTime      ← Changes Types
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::EnableTimeConstrainedPending,   ← RTI 1.3 Only
  RTI::EnableTimeRegulationPending,    ← RTI 1.3 Only
  RTI::FederateNotExecutionMember,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::TimeAdvanceAlreadyInProgress
)
```

ARGUMENTS

theTime

the "cutoff point" at which to stop advancing logical time in the absence of any intervening relevant time-stamp-ordered events

DESCRIPTION

This service allows the federate to advance in time to the time-stamp of the next time-stamp-ordered event occurring in the federation which matches the federate's current subscription interests. A `timeAdvanceGrant()` callback will occur after one or more time-stamp-ordered events have been delivered or the federation lower-bound time stamp (LBTS) advances past the specified cutoff time.

In the presence of an event, all relevant events with the same time-stamp *that have been received by the LRC* will be delivered before the grant. The grant time will be equal to the time-stamp of the event(s) delivered to the federate. It is possible that other time-stamp-ordered events with a time stamp equal to the grant time will be delivered to the federate during a subsequent time-advancement service.

In the absence of a relevant event, the grant time will be the specified cutoff time.

Any number of receive-ordered events may be delivered while the `nextEventRequestRequest()` is in progress.

Note that if the federate is not time-constrained, the grant criteria are trivially met (i.e. the effective federation LBTS for a non-constrained federate is always infinity), so a `timeAdvanceGrant()` to the cutoff time will be immediately scheduled for delivery by a subsequent invocation of `tick()`.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully announced its desire to advance in time; the federate will be notified of the successful completion of the request (as described above) via a subsequent `timeAdvanceGrant()` callback.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::EnableTimeConstrainedPending **(RTI 1.3 Only)**

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::EnableTimeRegulationPending **(RTI 1.3 Only)**

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed

The specified federation time is less than the current logical time of the federation.

RTI::InvalidFederationTime

This exception is not thrown by the current implementations.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TimeAdvanceAlreadyInProgress

A previous time advance request, next event request, or flush queue request has not yet been completed.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::
timeAdvanceGrant()

RTI::RTIambassador::
- flushQueueRequest()
- nextEventRequest()
- requestFederateTime()
- requestLBTS()
- setTimeConstrained()
- tick()
- timeAdvanceRequest()
- turnRegulationOn()

RTI 1.3

RTI::FederateAmbassador::
- timeAdvanceGrant()

RTI::FedTime

RTI::RTIambassador::
- enableTimeConstrained()
- enableTimeRegulation()
- flushQueueRequest()
- nextEventRequest()
- queryFederateTime()
- queryLBTS()
- tick()
- timeAdvanceRequest()

A.5.13 queryFederateTime()

RTI 1.3

ABSTRACT

This service is used by a federate to obtain its current logical time. **The equivalent RTI 1.0 service is named *`requestFederateTime`*; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the “Query Federate Time” Time Management service as specified in the *HLA Interface Specification* (§8.17 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   queryFederateTime (
       RTI::FedTime& theTime
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theTime

out-parameter set to indicate the federate’s current logical time

DESCRIPTION

This service allows the federate to its current logical time. If time-regulation is enabled, this value (plus the federate’s *lookahead*) represents the minimum time-stamp that may be associated with time-stamp-ordered events generated subsequently by the federate.

If a `timeAdvanceRequest()` (or `timeAdvanceRequestAvailable()`) service is in progress, the federate’s logical time is the federation time subject of the advancement request. If a `nextEventRequest()` (or `nextEventRequestAvailable()`) service is in progress, the federate’s logical time may be any value that is less than the current federation lower-bound time-stamp. If a `flushQueueRequest()` service is in progress, or if no time-advancement service is in progress, the federate’s logical time is the time associated with the last `timeAdvanceGrant()` callback.

RETURN VALUES

A non-exceptional return indicates that the *theTime* out-parameter has been set to reflect the federate’s current logical time.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::
timeAdvanceGrant()

RTI::FedTime

RTI::RTIambassador::
enableTimeRegulation()

flushQueueRequest()

nextEventRequest()

queryLBTS()

queryLookahead()

queryMinNextEventTime()

timeAdvanceRequest()

A.5.14 queryLBTS()

RTI 1.3

ABSTRACT

This service is used by a federate to obtain the current federation lower-bound time-stamp. **The equivalent RTI 1.0 service is named `requestLBTS`; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the “Query LBTS” Time Management service as specified in the *HLA Interface Specification* (§8.16 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   queryLBTS (
       RTI::FedTime& theTime
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theTime

out-parameter set to indicate the federation’s current lower-bound time-stamp

DESCRIPTION

This service reports the federation lower-bound time-stamp (LBTS) effective for the local federate. The LBTS is the greatest time-stamp such that it can be guaranteed that no time-stamp-ordered events will be subsequently generated in the federation with a lesser time-stamp.

For a time-constrained federate, the effective LBTS is the minimum of the most recently reported effective logical times (i.e., federate logical time plus federate lookahead) for all time-regulating federates in the federation.

For a non-time-constrained federate, the effective LBTS is infinity: there are no time-stamp-ordered events in the federation from the perspective of a non-time-constrained federate.

Note that events with time stamps earlier than the LBTS may still be queued for time-stamp-ordered delivery to a federate; the LBTS merely indicates that no time-stamp-ordered events will be subsequently *generated* with an earlier time stamp. A federate may use the `queryMinNextEventTime()` service to determine the minimum time-stamp of all time-stamp-ordered events that may be subsequently delivered to the federate.

RETURN VALUES

A non-exceptional return indicates that the *theTime* argument has been set to indicate the current federation lower-bound time-stamp.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FedTime

RTI::RTIambassador::

enableTimeConstrained()

enableTimeRegulation()

queryFederateTime()

queryLookahead()

queryMinNextEventTime()

A.5.15 queryLookahead()

RTI 1.3

ABSTRACT

This service queries the size of the interval extending forward from the federate's logical time at a given point in execution in which the federate will not generate any time-stamp-ordered events. **The RTI 1.0 equivalent of this method is `requestLookahead`; the RTI 1.0 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the "Query Lookahead" Time Management service as specified in the *HLA Interface Specification* (§8.20 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   queryLookahead (
      RTI::FedTime& theTime
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
);
```

ARGUMENTS

theTime

out-parameter that is set to indicate the current lookahead interval of the local federate

DESCRIPTION

This service queries the length lookahead interval in effect for the local federate. The lookahead represents the length of the logical-time interval extending forward from the federate's logical time at any given point in execution in which the federate will not generate events. That is, the minimum permissible time-stamp for a time-stamp-ordered event generated by a time regulating federate is the federate's current logical time plus its lookahead. The sum of a federate's logical time and its lookahead may be referred to as the *effective* logical time of the federate.

If the `modifyLookahead()` service has been used to decrease the length of the lookahead interval, the new lookahead value will not take effect immediately. The specified lookahead will be phased in such that the effective logical time of the federate remains strictly non-decreasing. The new lookahead will completely take effect when the federate has advanced its logical time from the time at which the lookahead was decreased by an amount exceeding the difference between the new and old lookahead values.

The lookahead setting is only meaningful for a time regulating federate: non-time-regulating federates may always generate events with any time stamp.

RETURN VALUES

A non-exceptional return indicates that the *theTime* argument has been set to indicate the current lookahead interval of the federate.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FedTime

RTI::RTIambassador::

enableTimeRegulation()

modifyLookahead()

queryFederateTime()

queryLBTS()

queryLookahead()

queryMinNextEventTime()

A.5.16 queryMinNextEventTime()

RTI 1.3

ABSTRACT

This service obtains the minimum time-stamp of all time-stamp-ordered events that may be subsequently delivered to the federate.

HLA IF SPECIFICATION

This method realizes the “Query Minimum Next Event Time” Time Management service as specified in the *HLA Interface Specification* (§8.18 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   queryMinNextEventTime (
      RTI::FedTime& theTime
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

ARGUMENTS

theTime

out-parameter set to indicate the minimum time stamp of the next (potential) time-stamp-ordered event that may be subsequently delivered to the federate

DESCRIPTION

This service may be used to obtain the largest time stamp such that the LRC can guarantee that no time-stamp-ordered events will be subsequently delivered to the federate with an earlier time stamp. This is computed by taking the minimum of the effective federation lower-bound time stamp (see `queryLBTS()`) and the time stamps of all time-stamp-ordered events currently queued for delivery to the federate (if any.)

Note that the minimum next-event time for a non-time-constrained federate will generally be infinity. The effective LBTS of a non-time-constrained federate is infinity and no events will be queued for time-stamp-ordered delivery (although there may be some time-stamp-ordered events already queued when time-constraint was disabled.)

RETURN VALUES

A non-exceptional return indicates that the *theTime* argument has been set to reflect the minimum time stamp of the next (potential) time-stamp-ordered event that may be subsequently delivered to the federate.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FedTime

RTI::RTIambassador::

enableTimeConstrained()

queryLBTS()

A.5.17 requestFederateTime()

RTI 1.0

ABSTRACT

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

HLA IF SPECIFICATION

This method realizes the "Request Federate Time" Time Management service as specified in the *HLA Interface Specification* (§6.3 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

RTI::FederationTime
RTI::RTIambassador::
   requestFederateTime ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

This service allows the federate to obtain its current logical time, i.e. the most recent time requested by the federate via *RTIambassador::timeAdvanceRequest*. If the federate is time-regulating, its logical time plus its 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.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

requestFederationTime()

requestLBTS()

requestMinNextEventTime()

tick()

timeAdvanceRequest()

turnRegulationOn()

A.5.18 requestFederationTime()

RTI 1.0

ABSTRACT

This service requests the current federation time.

HLA IF SPECIFICATION

This method realizes the “Request Federation Time” Time Management service as specified in the *HLA Interface Specification* (§6.1 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

RTI::FederationTime
RTI::RTIambassador::
   requestFederationTime ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

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. This value represents the maximum time-stamp value that is eligible for delivery to the federation at this particular instance in time.

RETURN VALUES

The returned value is the current federation time, as perceived by the federate.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

requestFederateTime()

requestLBTS()

requestMinNextEventTime()

tick()

timeAdvanceRequest()

A.5.19 requestLBTS()

RTI 1.0

ABSTRACT

This service requests the current effective federation lower-bound time stamp (LBTS) for the federate. **The RTI 1.3 implementation of this service is named *queryLBTS*; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the “Request LBTS” Time Management service as specified in the *HLA Interface Specification* (§6.2 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

RTI::FederationTime
RTI::RTIambassador::
   requestLBTS ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

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 cannot generate any time-stamp-ordered messages. The federation LBTS is the minimum of the LBTS's of all participating federates.

Time-stamp ordered messages with a time-stamp less than LBTS may still be queued for processing, and may therefore be delivered to the federate as a result of *RTIambassador::tick* invocations; the LBTS simply guarantees that no new messages with a lower-time stamp will be queued for processing (to find out the absolute minimum time-stamp of all messages eligible for future delivery, use *RTIambassador::requestMinNextEventTime*.)

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

RETURN VALUES

The returned value is the current federation lower-bound time stamp.

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

requestFederateTime()

requestFederationTime()

requestMinNextEventTime()

tick()

timeAdvanceRequest()

turnRegulationOn()

A.5.20 requestLookahead()

RTI 1.0

ABSTRACT

This service obtains the current lookahead window being used for the federate. **The RTI 1.3 implementation of this service is named *queryLookahead*; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the “Request Lookahead” Time Management service as specified in the *HLA Interface Specification* (§6.6 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

RTI::FederationTimeDelta
RTI::RTIambassador::
   requestLookahead ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

This service allows the federate to obtain the value of its effective lookahead, i.e. the time window between its logical time and the minimum allowable time-stamp of a time-stamp-ordered event generated by the federate. The effective lookahead at a given time is at least as great as the current lookahead as specified by the *RTIambassador::setLookahead* service (see the section on this service for a discussion of why this is true.)

RETURN VALUES

The return value is the current effective lookahead for the federate.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

requestFederateTime()

requestFederationTime()

requestLBTS()

setLookahead()

timeAdvanceRequest()

A.5.21 requestMinNextEventTime()

RTI 1.0

ABSTRACT

This service 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 *queryMinNextEventTime*; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the “Request Minimum Next Event Time” Time Management service as specified in the *HLA Interface Specification* (§6.4 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

RTI::FederationTime
RTI::RTIambassador::
   requestMinNextEventTime ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

The minimum next event time is defined as the largest time-stamp such that 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 and the time-stamp of the earliest time-stamp-ordered event (if any) in the federate's event queue. Note that 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.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

requestFederateTime()

requestFederationTime()

requestLBTS()

setTimeConstrained()

tick()

timeAdvanceRequest()

A.5.22 retract()

RTI 1.0 RTI 1.3

ABSTRACT

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

HLA IF SPECIFICATION

This method realizes the "Retract" service as specified in the *HLA Interface Specification* (§4.16 (Object Management) in version 1.1; §8.21 (Time Management) in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   retract (
       RTI::EventRetractionHandle theHandle
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidRetractionHandle,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theHandle

the event-retraction handle (as obtained from `updateAttributeValues()`, `sendInteraction()`, `deleteObject()`) of the event to unschedule

DESCRIPTION

The federate can utilize this service to withdraw an update, interaction, or deletion it has previously scheduled (or, in the current implementations, one that has been scheduled by another federate.) A successful invocation will result in the issuance of an event-retraction message to every federate in the Federation Execution. If the specified event is currently queued for delivery to a given remote federate, it is removed from its queue. If the specified event has been recently delivered to the federate (the current implementations maintain 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.

While the RTI event retraction services don't do very much in and of themselves, they provide a cornerstone upon which optimistic simulations can be built using such techniques as "anti-messages."

RETURN VALUES

A non-exceptional return indicates that the other federates in the federation have been advised to cancel the event specified by the retraction handle.

RELEASE NOTES

RTI 1.0

Retractions are delivered to federates using the `reflectRetraction()` callback.

RTI 1.3

Retractions are delivered to federates using the `requestRetraction()` callback.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidRetractionHandle

The event-retraction handle does not correspond to an event previously scheduled by the federate. (Not thrown in 1.0.)

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::EventRetractionHandle

RTI::FederateAmbassador::

reflectRetraction() ← **RTI 1.0 Only**

requestRetraction() ← **RTI 1.3 Only**

RTI::RTIambassador::

deleteObject() ← **RTI 1.0 Only**

deleteObjectInstance() ← **RTI 1.3 Only**

sendInteraction()

updateAttributeValues()

A.5.23 setLookahead()

RTI 1.0

ABSTRACT

This service redefines the lookahead window for the federate. **The RTI 1.3 implementation of this service is named *`modifyLookahead`*; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method realizes the “Set Lookahead” Time Management service as specified in the *HLA Interface Specification* (§6.5 in version 1.1).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   setLookahead (
      RTI::FederationTimeDelta theLookahead
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidLookahead,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theLookahead

new lookahead value to use for the federate

DESCRIPTION

This service allows the federate to dynamically modify its lookahead window, i.e. the amount of time between the federate logical time 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 do not generate TSO events. To minimize the overhead associated with synchronizing federation time advances, federates should make their lookahead window as large as is feasible.

If the specified lookahead is smaller than the current lookahead, the new lookahead does not go into effect immediately, as this would result in the federate breaking an earlier "promise" not to generate TSO events before a given federation time. In this case, the federate's actual lookahead is gradually decreased as the federate's logical time is increased (to preserve a constant value of "logical time + lookahead") until it becomes possible to use the specified lookahead value. If the specified lookahead is greater than the current federation lookahead, it goes into effect immediately.

Obviously, lookahead values must be non-negative. A federate's lookahead defaults to *EPSILON* as defined in *$RTI_HOME/include/RTItypes.h*.

Time-constrained zero-lookahead federates are an interesting "special case"; see `timeAdvanceRequestAvailable()` and `nextEventRequestAvailable()` for discussion of special considerations for such federates.

RETURN VALUES

A non-exceptional return indicates that the federate lookahead will be adjusted to the specified value as soon as possible.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidLookahead

The specified lookahead interval must be a non-negative size.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

nextEventRequest()

nextEventRequestAvailable()

requestLBTS()

requestLookahead()

timeAdvanceRequest()

timeAdvanceRequestAvailable()

A.5.24 setTimeConstrained()

RTI 1.0

ABSTRACT

This service specifies whether a federate wishes to process time-stamp-ordered events in time-stamp order. If so, the federate's logical time will be constrained by the federation logical time. .
The RTI 1.3 equivalents of this service are named *enableTimeConstrained* and *disableTimeConstrained*; the RTI 1.3 implementation is discussed in separate sections.

HLA IF SPECIFICATION

This method does not directly correspond to a service specified in the *HLA Interface Specification* version 1.1.

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   setTimeConstrained (
       RTI::Boolean state
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

state

Whether or not the federate wishes to be time constrained.

DESCRIPTION

This method allows the federate to dynamically specify whether or not RTI should take time into consideration when determining when to present events to the federation. If a federate is not time-constrained, all incoming events are processed in receive-order, i.e. they are immediately made available for processing by an *RTIambassador::tick* service call. Events only become eligible for presentation to a time constrained federate when it can be guaranteed that no time-stamp-ordered events with a lower time-stamp will be received. This ordering only applies to events that are designated by the sender as being time-stamp-ordered; events designated as receive-ordered will always be made eligible for presentation immediately.

Turning time constraints on affects only events received subsequently; it does not affect any time-stamp-ordered events that may have already been received and placed in the receive-order queue.

Federates are non-time-constrained by default.

RETURN VALUES

A non-exceptional return indicates that the federate's time constraints have been turned on or off as requested.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederationExecutionDoesNotExist

The RTI does not have a FedExec registered for the named Federation.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

changeAttributeOrderType()
changeInteractionOrderType()
tick()
timeAdvanceRequest()
turnRegulationOn()

A.5.25 timeAdvanceRequest()

RTI 1.0 RTI 1.3

ABSTRACT

This service requests an advance of the logical time of the federate to a specified federation time. **The syntax of this service has changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This RTI 1.0 implementation of this method (in conjunction with `timeAdvanceRequestAvailable()`) realizes the "Time Advance Request" Time Management service as specified in the *HLA Interface Specification* (§6.7 in version 1.1).

The RTI 1.3 implementation of this method realizes the "Time Advance Request" Time Management service as specified in the *HLA Interface Specification* (§8.8 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   timeAdvanceRequest (
       RTI::FederationTime theTime
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress,
   RTI::TimeAdvanceAlreadyInProgress
 )

// RTI 1.3 Only
void timeAdvanceRequest (
   const RTI::FedTime& theTime          ← Changes Types
)
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::EnableTimeConstrainedPending,    ← RTI 1.3 Only
  RTI::EnableTimeRegulationPending,     ← RTI 1.3 Only
  RTI::FederateNotExecutionMember,
  RTI::FederationTimeAlreadyPassed,
  RTI::InvalidFederationTime,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress,
  RTI::TimeAdvanceAlreadyInProgress
)
```

ARGUMENTS

theTime

the time-stamp representing the point on the federation time axis to which to advance the federate's logical time

DESCRIPTION

This service releases all time-stamp-ordered (TSO) events between the federate's current logical time and the requested time, inclusive, for delivery to the federate. Each relevant TSO event occurring in this interval will be delivered to the federate as soon as it can be guaranteed that all relevant TSO events with earlier time stamps have already been delivered. That is, an event will be delivered if and only if

- all events currently queued for TSO delivery to the federate have a time stamp that is not greater than the event's time stamp
- the federation LBTS is not less than the time stamp of the event (i.e., the LRC can guarantee that no TSO events with an earlier time stamp will arrive subsequently)

When all relevant TSO events with time stamps less than or equal to the requested time have been delivered to the federate (i.e., the *minimum next-event time* is greater than the requested time), the federate will receive a `timeAdvanceGrant()` callback indicating that the time-advancement has completed. Only after receiving such a callback may the federate proceed to initiate another time-advancement service.

Subsequent to initiating a `timeAdvanceRequest()`, the federate may not generate TSO events whose time stamps are less than the requested time plus the federate's lookahead. That is, the logical time of the federate is immediately set equal to the requested time upon a `timeAdvanceRequest()` service invocation.

For non-time-constrained federates, time advances are trivial: by definition such federates do not receive any TSO events, so a time-advance grant is immediately scheduled for delivery by a subsequent invocation of the `tick()` service.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully initiated the time-advancement process. TSO events from the current time through the requested time (inclusive) may now be delivered to the federate, and the federate may no longer generate TSO events with a time-stamp less than the requested time plus the federate lookahead. The federate will receive notification of the successful completion of the time advance (as described previously) via its `timeAdvanceGrant()` callback.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::EnableTimeConstrainedPending **(RTI 1.3 Only)**

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::EnableTimeRegulationPending **(RTI 1.3 Only)**

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed

The requested time is less than the current logical time of the federate.

RTI::InvalidFederationTime

The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal

state of the RTI, which is not permitted during a "save" operation.

RTI::TimeAdvanceAlreadyInProgress
A previous time advance request, next event request, or flush queue request has not yet been completed.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::
timeAdvanceGrant()

RTI::RTIambassador::
flushQueueRequest()
nextEventRequest()
requestFederateTime()
requestLBTS()
setTimeConstrained()
tick()
timeAdvanceRequestAvailable()
turnRegulationOn()

RTI 1.3

RTI::FederateAmbassador::
timeAdvanceGrant()

RTI::FedTime

RTI::RTIambassador::
enableTimeConstrained()
enableTimeRegulation()
flushQueueRequest()
nextEventRequest()
queryFederateTime()
queryLBTS()
tick()
timeAdvanceRequestAvailable()

A.5.26 timeAdvanceRequestAvailable()

RTI 1.0 RTI 1.3

ABSTRACT

This service is similar to the `timeAdvanceRequest()` service, except that not all events occurring at exactly the requested time will necessarily be delivered before a `timeAdvanceGrant()` is made. **The syntax of this service has changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This RTI 1.0 implementation of this method (in conjunction with `timeAdvanceRequest()`) realizes the "Time Advance Request" Time Management service as specified in the *HLA Interface Specification* (§6.7 in version 1.1).

The RTI 1.3 implementation of this method realizes the "Time Advance Request Available" Time Management service as specified in the *HLA Interface Specification* (§8.9 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
void
RTI::RTIambassador::
   timeAdvanceRequestAvailable (
      RTI::FederationTime theTime
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress,
   RTI::TimeAdvanceAlreadyInProgress
 )

// RTI 1.3 Only
void
RTI::RTIambassador::
   timeAdvanceRequestAvailable (
      const RTI::FedTime& theTime      ← Changes Types
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::EnableTimeConstrainedPending,  ← RTI 1.3 Only
   RTI::EnableTimeRegulationPending,   ← RTI 1.3 Only
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::InvalidFederationTime,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress,
   RTI::TimeAdvanceAlreadyInProgress
)
```

ARGUMENTS

theTime

the time-stamp representing the point on the federation time axis to which to advance the federate's logical time

DESCRIPTION

This service releases all time-stamp-ordered (TSO) events between the federate's current logical time and the requested time, inclusive, for delivery to the federate. Each relevant TSO event occurring in this interval will be delivered to the federate as soon as it can be guaranteed that all relevant TSO events with earlier time stamps have already been delivered. That is, an event will be delivered if and only if

- all events currently queued for TSO delivery to the federate have a time stamp that is not greater than the event's time stamp
- the federation LBTS is not less than the time stamp of the event (i.e., the LRC can guarantee that no TSO events with an earlier time stamp will arrive subsequently)

When all relevant TSO events with time stamps less than the requested time have been delivered to the federate (i.e., the *minimum next-event time* is greater than the requested time), the federate will receive a `timeAdvanceGrant()` callback indicating that the time-advancement has completed. Only after receiving such a callback may the federate proceed to initiate another time-advancement service. The *available* variant of this service does *not* necessarily deliver all TSO events occurring exactly at the requested time before making a `timeAdvanceGrant()`.

Subsequent to initiating a `timeAdvanceRequestAvailable()`, the federate may not generate TSO events whose time stamps are less than the requested time plus the federate's lookahead. That is, the logical time of the federate is immediately set equal to the requested time upon a `timeAdvanceRequestAvailable()` service invocation.

For non-time-constrained federates, time advances are trivial: by definition such federates do not receive any TSO events, so a time-advance grant is immediately scheduled for delivery by a subsequent invocation of the `tick()` service.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully initiated the time-advancement process. TSO events from the current time through the requested time (inclusive) may now be delivered to the federate, and the federate may no longer generate TSO events with a time-stamp less than the requested time plus the federate lookahead. The federate will receive notification of the successful completion of the time advance (as described previously) via its `timeAdvanceGrant()` callback.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::EnableTimeConstrainedPending **(RTI 1.3 Only)**

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::EnableTimeRegulationPending **(RTI 1.3 Only)**

A time-advancement service may not be initiated while a request to enable regulation or constraint is pending.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed

The requested time is less than the current logical time of the federate.

RTI::InvalidFederationTime

The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::TimeAdvanceAlreadyInProgress
A previous time advance request, next event request, or flush queue request has not yet been completed.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador::
timeAdvanceGrant()

RTI::RTIambassador::
flushQueueRequest()
nextEventRequest()
requestFederateTime()
requestLBTS()
setTimeConstrained()
tick()
timeAdvanceRequest()
turnRegulationOn()

RTI 1.3

RTI::FederateAmbassador::
timeAdvanceGrant()

RTI::FedTime

RTI::RTIambassador::
enableTimeConstrained()
enableTimeRegulation()
flushQueueRequest()
nextEventRequest()
queryFederateTime()
queryLBTS()
tick()
timeAdvanceRequest()

A.5.27 turnRegulationOff()

RTI 1.0

ABSTRACT

This service indicates that a federate does not wish its logical time to be considered in regulating the progress of federation logical time. **The RTI 1.3 equivalent of this service is named `disableTimeRegulation`; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method does not directly correspond to a service specified in the *HLA Interface Specification* version 1.1.

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   turnRegulationOff ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

These methods allow the federate to specify whether its logical time should be considered in the determination of the federation's lower-bound time stamp (LBTS), i.e. the greatest time-stamp such that the federation can guarantee that no time-stamp ordered messages will be delivered with an earlier time-stamp.

RTIambassador::turnRegulationOnNow sets the federate's logical time to the current federation LBTS before turning time regulation on. If *RTIambassador::turnRegulationOn* is used instead, the federate must be sufficiently advanced in time that it will not generate time-stamp ordered messages that will be in the federation's past (i.e. the federate's logical time plus its lookahead must not be less than the federation LBTS.) Note that not all updates and interactions sent by a time-regulating federate are necessarily time-stamp ordered; the ordering is determined on a per-attribute or per-interaction basis based on the definitions in the federation FED file (*$RTI_CONFIG/[federation name].fed*) or dynamically specified by the federate via *RTIambassador::changeAttributeOrderType* or *RTIambassador::changeInteractionOrderType*.

If a federate is not time-regulating, its logical time will not be considered in the determination of the federation LBTS, and all updates and interactions sent by the federate will be processed receive-order, regardless of their individual ordering policies.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully indicated its desire to participate or not participate in the regulation of federation time.

RTI::RTIambassador::turnRegulationOnNow returns the new federate logical time, i.e. the current value of the federation's lower-bound time stamp.

By default, federates are not time regulating.

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed
: The specified logical time argument lies in the federation’s past, whereas the context of the service invocation required a future logical time.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::
: changeAttributeOrderType()
changeInteractionOrderType()
requestFederateTime()
requestLBTS()
requestLookahead()
setLookahead()
setTimeConstrained()
timeAdvanceRequest()

turnRegulationOn(), turnRegulationOnNow()

A.5.28 turnRegulationOn()

RTI 1.0

ABSTRACT

This service indicates that a federate wishes its logical time to be considered in regulating the progress of federation logical time. **The RTI 1.3 equivalent of this service is named *`enableTimeRegulation`*; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method does not directly correspond to a service specified in the *HLA Interface Specification* version 1.1.

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   turnRegulationOn ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::FederationTimeAlreadyPassed,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

These methods allow the federate to specify whether its logical time should be considered in the determination of the federation's lower-bound time stamp (LBTS), i.e. the greatest time-stamp such that the federation can guarantee that no time-stamp ordered messages will be delivered with an earlier time-stamp.

RTIambassador::turnRegulationOnNow sets the federate's logical time to the current federation LBTS before turning time regulation on. If *RTIambassador::turnRegulationOn* is used instead, the federate must be sufficiently advanced in time that it will not generate time-stamp ordered messages that will be in the federation's past (i.e. the federate's logical time plus its lookahead must not be less than the federation LBTS.) Note that not all updates and interactions sent by a time-regulating federate are necessarily time-stamp ordered; the ordering is determined on a per-attribute or per-interaction basis based on the definitions in the federation FED file (*$RTI_CONFIG/[federation name].fed*) or dynamically specified by the federate via *RTIambassador::changeAttributeOrderType* or *RTIambassador::changeInteractionOrderType*.

If a federate is not time-regulating, its logical time will not be considered in the determination of the federation LBTS, and all updates and interactions sent by the federate will be processed receive-order, regardless of their individual ordering policies.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully indicated its desire to participate or not participate in the regulation of federation time.

RTI::RTIambassador::turnRegulationOnNow returns the new federate logical time, i.e. the current value of the federation's lower-bound time stamp.

By default, federates are not time regulating.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed

The specified logical time argument lies in the federation’s past, whereas the context of the service invocation required a future logical time.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

changeAttributeOrderType()
changeInteractionOrderType()
requestFederateTime()
requestLBTS()
requestLookahead()
setLookahead()
setTimeConstrained()
timeAdvanceRequest()

turnRegulationOn(), turnRegulationOnNow(), turnRegulationOff()

A.5.29 turnRegulationOnNow()

RTI 1.0

ABSTRACT

This service indicates that the federate wishes its logical time to be considered in regulating the progress of federation logical time. The federate's logical time will be set to the currently value of federation logical time. **The RTI 1.3 equivalent of this service is named *`enableTimeRegulation`*; the RTI 1.3 implementation is discussed in a separate section.**

HLA IF SPECIFICATION

This method does not directly correspond to a service specified in the *HLA Interface Specification* version 1.1.

SYNOPSIS

```
#include <RTI.hh>

RTI::FederationTime
RTI::RTIambassador::
   turnRegulationOnNow ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

These methods allow the federate to specify whether its logical time should be considered in the determination of the federation's lower-bound time stamp (LBTS), i.e. the greatest time-stamp such that the federation can guarantee that no time-stamp ordered messages will be delivered with an earlier time-stamp.

RTIambassador::turnRegulationOnNow sets the federate's logical time to the current federation LBTS before turning time regulation on. If *RTIambassador::turnRegulationOn* is used instead, the federate must be sufficiently advanced in time that it will not generate time-stamp ordered messages that will be in the federation's past (i.e. the federate's logical time plus its lookahead must not be less than the federation LBTS.) Note that not all updates and interactions sent by a time-regulating federate are necessarily time-stamp ordered; the ordering is determined on a per-attribute or per-interaction basis based on the definitions in the federation FED file (*$RTI_CONFIG/[federation name].fed*) or dynamically specified by the federate via *RTIambassador::changeAttributeOrderType* or *RTIambassador::changeInteractionOrderType*.

If a federate is not time-regulating, its logical time will not be considered in the determination of the federation LBTS, and all updates and interactions sent by the federate will be processed receive-order, regardless of their individual ordering policies.

RETURN VALUES

A non-exceptional return indicates that the federate has successfully indicated its desire to participate or not participate in the regulation of federation time.

RTI::RTIambassador::turnRegulationOnNow returns the new federate logical time, i.e. the current value of the federation's lower-bound time stamp.

By default, federates are not time regulating.

EXCEPTIONS

RTI::ConcurrentAccessAttempted
: An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
: The RTIambassador instance is not currently associated with a FedExec.

RTI::FederationTimeAlreadyPassed
: The specified logical time argument lies in the federation's past, whereas the context of the service invocation required a future logical time.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

changeAttributeOrderType()
changeInteractionOrderType()
requestFederateTime()
requestLBTS()
requestLookahead()
setLookahead()
setTimeConstrained()
timeAdvanceRequest()

A.6 Data Distribution Management

A.6.1 associateRegionForUpdates()

RTI 1.3

ABSTRACT

This service associates attributes of an object-instance with a region of a routing space. The specified region replaces the currently associated regions for the specified attribute-instances.

HLA IF SPECIFICATION

This method realizes the "Associate Region For Updates" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.6 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   associateRegionForUpdates (
       RTI::Region             &theRegion,
       RTI::ObjectHandle        theObject,
const RTI::AttributeHandleSet &theAttributes
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidRegionContext,
   RTI::ObjectNotKnown,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theRegion

a region to associate with the specified instance-attributes

theObject

handle of object whose instance-attributes are being associated with the region

theAttributes

handle-set specifying attributes of the given instance that are to be associated with the region

DESCRIPTION

This service associates attributes of an object instance with a distribution region. The federation is responsible for establishing common semantics for the region extents such that all updates of the attribute-instances are relevant only to the observers of the prescribed region.

An attribute-instance is only associated with a single region at a given point in (wallclock) time; the region subject of this service replaces the currently associated regions for the specified attribute-instances. In addition, the attribute-set replaces any previous attribute-set associated with the region for the object instance in question. If this service is invoked twice for the same region/object combination, attributes not included in the intersection of the old and new attribute-sets revert to the default region.

Invoking this service with an empty attribute-set is equivalent to calling `unassociateRegionForUpdates()` for the same region/object pair.

Note that the extents actually associated with the attributes at a given instant are the most recent set of extents for the region that have been committed to the RTI using the `notifyAboutRegionModification()` service. The effective extents of the attributes can change over time if the associated region object is modified and recommitted.

INTERFACE SPECIFICATION NOTES

The HLA Interface Specification states that attribute-instances may only be associated with regions of the space to which the attribute is bound in the FED file. The 1.3 implementation will allow attribute-instances to be registered with regions of any routing space. Federates should not rely on this behavior.

RETURN VALUES

A non-exceptional return indicates that the specified attribute-instances have been associated with the specified region, and that any attributes no longer associated with the region have been associated with the default region.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidRegionContext

This exception is not thrown by the current implementation of this service.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RegionNotKnown

The region argument is not recognized as a valid region as instantiated by the `createRegion()` service.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::Region

RTI::RTIambassador

createRegion()

notifyAboutRegionModification()

registerObjectInstanceWithRegion()

subscribeObjectClassAttributesWithRegion()

unassociateRegionForUpdates()

updateAttributeValues()

A.6.2 createRegion()

RTI 1.3

ABSTRACT

This service allocates a new region object for a specified routing space and with a specified number of extents. Upon initialization, all extents are set to span the entire routing space across all dimensions.

HLA IF SPECIFICATION

This method realizes the “Create Region” Data Distribution Management service as specified in the *HLA Interface Specification* (§9.2 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::Region*
RTI::RTIambassador::
   createRegion (
       RTI::SpaceHandle theSpace,
       RTI::ULong       numberOfExtents
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidExtents,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress,
   RTI::SpaceNotDefined
)
```

ARGUMENTS

theSpace

the routing space for which to create a new region

numberOfExtents

the number of extents that will be used to define the region

DESCRIPTION

This service serves as a factory for creating new region objects. Memory for created regions is allocated on the heap. A region instance should be deleted using the `deleteRegion()` service when no longer needed. The `delete` operator should not be used to deallocate resources associated with the region.

Note that modifications of the returned region will not take effect until the changes have been committed to the RTI. A new region initially spans the entirety of the associated routing space, so it is expected that the application will modify the extents of the region and recommit it using the `notifyAboutRegionModification()` service.

RETURN VALUES

A successful invocation of this service returns a pointer to a new region object that initially spans the entire routing space.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidExtents

This exception is not thrown by this method in the current implementation.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

RTI::SpaceNotDefined

The *SpaceHandle* argument does not correspond to a valid federation routing space. Use *RTIambassador::getRoutingSpaceHandle* to obtain valid space handles.

SEE ALSO

RTI::Region

RTI::RTIambassador

associateRegionForUpdates()

deleteRegion()

getRoutingSpaceHandle()

notifyAboutRegionModification()

registerObjectInstanceWithRegion()

sendInteractionWithRegion()

subscribeInteractionClassWithRegion()

subscribeObjectClassAttributesWithRegion()

A.6.3 deleteRegion()

RTI 1.3

ABSTRACT

This service removes an unused region object from the RTI's internal tables and deallocates the memory associated with the object.

HLA IF SPECIFICATION

This method realizes the "Delete Region" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.4 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   deleteRegion (
       RTI::Region *theRegion
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theRegion

region object to be deleted

DESCRIPTION

This service deletes the memory associated with a region object and removes references to the region from the RTI's internal tables. The memory referenced by the service's pointer argument is deallocated upon a successful completion and should not be subsequently used by the application.

A region may only be deleted if it is not currently the subject of any object or interaction subscriptions and is not associated with any locally owned instance-attributes. The federate should disassociate the region with any locally owned attribute-instances and undo all subscriptions based on a region before deleting it.

RETURN VALUES

A non-exceptional return indicates that the region object is recognized as an unused region and has been deleted from the RTI.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RegionNotKnown

The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::Region

RTI::RTIambassador

createRegion()

unassociateRegionForUpdates()

unsubscribeInteractionClassWithRegion()

unsubscribeObjectClassWithRegion()

A.6.4 notifyAboutRegionModification()

RTI 1.3

ABSTRACT

This service commits changes made to a region object. Subscriptions and associations based on the region object will be updated accordingly.

HLA IF SPECIFICATION

This method realizes the “Modify Region” Data Distribution Management service as specified in the *HLA Interface Specification* (§9.3 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   notifyAboutRegionModification (
       RTI::Region &theRegion
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidExtents,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theRegion

the region whose extent changes are to be committed to the RTI

DESCRIPTION

This service informs the federation of changes made to a region object. Any associations and subscriptions for this region will be updated in accordance with the new extents. If the update results in a

RETURN VALUES

A non-exception return indicates that the changes to the region extents have been committed to the RTI.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidExtents

One or more dimensions of one or more extents associated with the region are incorrectly specified. (A dimension is incorrect if its lower bound is greater than its upper bound or the lower or upper bound falls outside the extent ranges defined by the RTI.)

RTI::RegionNotKnown

The region argument is not recognized as a valid region as instantiated by the `createRegion()` service.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::Region

RTI::RTIambassador

associateRegionForUpdates()

createRegion()

registerObjectInstanceWithRegion()

sendIntreactionWithRegion()

subscribeInteractionClassWithRegion()

subscribeObjectClassWithRegion()

A.6.5 registerObjectInstanceWithRegion()

RTI 1.3

ABSTRACT

This service simultaneously registers a new object-instance with the federation and associates some or all of the object's attributes with DDM regions other than the default.

HLA IF SPECIFICATION

This method realizes the "Register Object Instance With Region" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.5 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::ObjectHandle
RTI::RTIambassador::
   registerObjectInstanceWithRegion (
       RTI::ObjectClassHandle theClass,
       const char             *theObject,
       RTI::AttributeHandle   theAttributes[],
       RTI::Region            *theRegions[],
       RTI::ULong             theNumberOfHandles
   )
   throw (
       RTI::ConcurrentAccessAttempted,
       RTI::FederateNotExecutionMember,
       RTI::ObjectAlreadyRegistered,
       RTI::ObjectClassNotDefined,
       RTI::ObjectClassNotPublished,
       RTI::RegionNotKnown,
       RTI::RestoreInProgress,
       RTI::RTIinternalError,
       RTI::SaveInProgress
   )

RTI::ObjectHandle
RTI::RTIambassador::
   registerObjectInstanceWithRegion (
       RTI::ObjectClassHandle theClass,
       RTI::AttributeHandle   theAttributes[],
       RTI::Region           *theRegions[],
       RTI::ULong             theNumberOfHandles
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::ObjectClassNotPublished,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object class to associate with the newly registered object

theObject

a unique symbolic name identifying the object instance

theAttributes

array of attribute handles of the specified object class that are to be initially associated with regions

theRegions

array of regions to be associated with the corresponding items in the attribute-handle array

theNumberOfHandles

the number of items contained in the attribute-handle and region arrays

DESCRIPTION

This service is essentially shorthand for registering an object using object-management services, then associating attribute-instances with regions through one or more invocations of `associateRegionForUpdates()`. The two variants of this service correspond to the named and unnamed variants of the `registerObjectInstance()` service.

The *theAttributes* and *theRegions* arguments represent parallel arrays defining a subset of the object's attributes and the regions with which they should be associated. The *n*th handle in the attribute-handle array will be associated with the *n*th region in the regions array. Positions containing attribute handles that are not valid in the context of the specified object class or handles of attributes that are not published by the local federate will be ignored. If the same attribute handle appears twice in the array, the region corresponding to the first occurrence will be used. Any locally published attributes for which a region is not specified will be associated with the default region of the routing space bound to the attributes in the FED file.

Note that the extents actually associated with the attributes at a given instant are the most recent set of extents for the region that have been committed to the RTI using the `notifyAboutRegionModification()` service. The extents associated with newly created attributes may not match the current state of the corresponding region object if the region object differs from its most recent committed state. The effective extents of the attributes can change over time if their associated region objects are modified and recommitted.

The creation of a new object instance is immediately announced to the federation, resulting in `discoverObjectInstance()` callbacks for any federates whose subscription interests include at least one class-attribute of the registered object class. The object instance may also be discovered as a result of subsequent updates and data-distribution management operations affecting instance-attributes of the object.

INTERFACE SPECIFICATION NOTES

The HLA Interface Specification states that attribute-instances may only be associated with regions of the space to which the attribute is bound in the FED file. The 1.3 implementation will allow attribute-instances to be registered with regions of any routing space. Federates should not rely on this behavior.

RETURN VALUES

Upon successful completion, this service returns a handle used to reference the newly instantiated object.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectAlreadyRegistered

The symbolic name associated with the object has already been registered within the federation.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::ObjectClassNotPublished
: The operation attempted requires that the object class be currently published by the federate.

RTI::RegionNotKnown
: The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
: An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
: The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::Region

RTI::RTIambassador

associateRegionForUpdates()

getAttributeHandle()

getObjectClassHandle()

notifyAboutRegionModification()

registerObjectInstance()

subscribeObjectClassAttributesWithRegion()

updateAttributeValues()

A.6.6 requestClassAttributeValueUpdateWithRegion()

RTI 1.3

ABSTRACT

This service is similar to `requestClassAttributeValueUpdate()`, except that updates are only solicited for attribute-instances relevant in a specified region.

HLA IF SPECIFICATION

This method realizes the "Request Attribute Value Update With Region" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.13 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   requestClassAttributeValueUpdateWithRegion (
         RTI::ObjectClassHandle   theClass,
  const RTI::AttributeHandleSet &theAttributes,
  const RTI::Region             &theRegion
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

object class for which updates are being solicited

theAttributes

subset of the attributes of the specified class for which updates are requested

theRegion

region of relevance used to narrow the logical scope of the solicitation

DESCRIPTION

This service is similar to the `requestClassAttributeValueUpdate()` Object Management service, except that the set of attribute-instances for which updates are solicited is filtered according to the specified region. Upon receipt of such a request by a remote LRC, the LRC will invoke its federate's `provideAttributeValueUpdate()` callback for exactly those attribute-instances that

- are instances of the class-attributes associated with the request (including instance-attributes of instances of subclasses of the objet class specified by the request)
- are owned by the federate
- are associated with a region that intersects the region specified by the request (attribute-instances associated with the default region will always be included in the solicitation)

It is the responsibility of the remote federates to respond to the solicitations using the `updateAttributeValues()` service. (The RTI has no "memory" of federation state, so it cannot automatically respond to requests for attribute values.) There is no direct integration of time-management with `requestClassAttributeValueUpdateWithRegion()`; remote federates will respond to the solicitation with updates at whatever logical time(s) are appropriate for a particular federate (or no logical time at all.)

Invoking this service with a region argument equal to default region (i.e., a region spanning the entire routing space to which the attributes are bound in the FED file) is equivalent to invoking the `requestClassAttributeValueUpdate()` service for the same class-attributes.

RETURN VALUES

A non-exceptional return indicates that updates for instances of the specified class-attributes relevant in the specified region will be solicited from the federation.

INTERFACE SPECIFICATION NOTES

The *HLA Interface Specification* version 1.3 describes a variant of this service that solicits updates on a per-instance instead of a per-class basis. This variant is not implemented by the RTI 1.3.

The *HLA Interface Specification* version 1.3 specifies that the arguments to this service consist of pairs of region/attribute groupings. The RTI 1.3 implementation has one region per invocation that is associated with all requested attributes.

The HLA Interface Specification states that attribute-instances may only be associated with regions of the space to which the attribute is bound in the FED file. The 1.3 implementation will allow attribute-instances to be registered with regions of any routing space. Federates should not rely on this behavior.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RegionNotKnown

The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet
RTI::FederateAmbassador::
provideAttributeValueUpdate()
RTI::Region
RTI::RTIambassador::
associateRegionForUpdates()
createRegion()
getAttributeHandle()
getObjectClassHandle()
notifyAboutRegionModification()
registerObjectInstance()
registerObjectInstanceWithRegion()
requestClassAttributeValueUpdate()

A.6.7 sendInteractionWithRegion()

RTI 1.3

ABSTRACT

This service is similar to the `sendInteraction()` service, except that the interaction is only delivered to federates that have declared a subscription interest in a specified subspace of the routing space bound to the interaction class.

HLA IF SPECIFICATION

This method realizes the "Send Interaction With Region" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.12 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::EventRetractionHandle
RTI::RTIambassador::
   sendInteractionWithRegion (
       RTI::InteractionClassHandle
          theInteraction,
const RTI::ParameterHandleValuePairSet&
          theParameters,
const RTI::FedTime&
          theTime,
const char
          *theTag
const RTI::Region
         &theRegion
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotPublished,
   RTI::InteractionParameterNotDefined,
   RTI::InvalidFederationTime,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)

void
RTI::RTIambassador::
   sendInteractionWithRegion (
        RTI::InteractionClassHandle
          theInteraction,
 const RTI::ParameterHandleValuePairSet
          &theParameters,
 const char
          *theTag,
 const RTI::Region
          &theRegion
      )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotPublished,
   RTI::InteractionParameterNotDefined,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theInteraction

the class of the interaction being sent

theParameters

handle-value pair-set containing a subset of the parameters of the interaction class and its superclasses

theTime

optional time-stamp associated with the interaction

theTag

opaque string data passed to receivers of the interaction

theRegion

region of relevance, used to limit recipients of the interaction

DESCRIPTION

This service is similar to the `sendInteraction()` Object Management service, except that the set of remote federates receiving the interaction is limited based on classes *and* regions of subscription. The interaction will be delivered to those remote federates that have subscribed to the interaction class (or a superclass) with a region that intersects the region associated with the interaction. The delivered class of the interaction for a particular remote federate will be the most-specific interaction class for which the remote federate's region of subscription intersects the region associated with the interaction. If this results in the interaction being *promoted* to a more general class for delivery to a federate, those parameters that are not present in the delivered class will be filtered from the set of values delivered to that federate.

Matching interactions to subscribers is performed by the local LRC upon sending an interaction, and by remote LRCs when the interaction is to be delivered to a remote federate. If a remote federate changes subscriptions between the time an interaction is sent out and the time at which it is considered for delivery to the federate, the interaction may not be delivered. For example, if an interaction that has been deferred by an LRC pending a logical time-advance is no longer relevant at the time the federate achieves the appropriate logical time, it will be discarded rather than delivered to the federate. The converse is not generally true: a remote federate subscribing to a relevant class and region will not receive an interaction that has already been sent, even if the interaction occurs in the federate's logical-time future.

Invoking this service with a region argument equal to the default region (i.e., a region spanning the entire routing space to which the interaction class is bound in the FED file) is equivalent to invoking `sendInteraction()` with the same arguments. A federate subscribing to an interaction class without a region will receive all instances of that class (and its subclasses) regardless of the region associated with a particular instance.

RETURN VALUES

A non-exceptional return indicates that the interaction will be delivered to remote federates that have declared a subscription interest in the interaction class (or a subclass) in the specified region.

The timed variant of this service returns a *retraction handle* that may be used to uniquely refer to the event for purposes of event retraction.

INTERFACE SPECIFICATION NOTES

The HLA Interface Specification states that interactions may only be associated with regions of the space to which the interaction class is bound in the FED file. The 1.3 implementation will allow interactions to be sent with regions of any routing space. Federates should not rely on this behavior.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined
The interaction class handle is not valid in the context of the current federation.

RTI::InteractionClassNotPublished
The operation attempted requires that the interaction class be currently published by the federate.

RTI::InteractionParameterNotDefined
One or more of the specified parameter handles is not valid in the context of the specified interaction class.

RTI::InvalidFederationTime
The specified logical time argument does not represent a valid point on the federation time axis. RTI 1.0 and 1.3 semantics define logical time as the set of non-negative numbers that may be represented as an **IEEE** `double`.

RTI::RegionNotKnown
The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::
receiveInteraction()

RTI::ParameterHandleValuePairSet

RTI::Region

RTI::RTIambassador::
createRegion()

getInteractionClassHandle()

getParameterHandle()

notifyAboutRegionModification()

sendInteraction()

subscribeInteractionClass()

subscribeInteractionClassWithRegion()

A.6.8 subscribeInteractionClassWithRegion()

RTI 1.3

ABSTRACT

This service instructs an LRC to begin delivering interactions of a specified class occurring in a specified subspace to the federate.

HLA IF SPECIFICATION

This method realizes the "Subscribe Interaction Class With Region" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.10 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   subscribeInteractionClassWithRegion (
       RTI::InteractionClassHandle theClass,
       RTI::Region                 &theRegion,
       RTI::Boolean           active = RTI::RTI_TRUE
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateLoggingServiceCalls,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

interaction class to which to subscribe in the specified region

theRegion

the region being subscribed

active

flag indicating whether the subscription should be considered by remote federates in determining relevance of interactions

DESCRIPTION

This service is similar to the `subscribeInteractionClass()` Declaration Management service, except that the subscription is limited based on a specified region of the routing space to which the interaction class is bound in the FED file. The effects of this service are cumulative; that is, a newly subscribed region is added to the set of regions already subscribed for the interaction class. An interaction of the specified class (or its subclasses) is delivered to a federate if the region associated with the interaction instance intersects any region that is subscribed by the federate for the interaction class. The union of all regions in which an interaction is subscribed may be thought of as the *effective* subscription region of the interaction class for a federate.

An interaction instance will be delivered to a federate as the most-specific interaction class for which the region associated with the interaction intersects a region subscribed by the federate. If an instance is *promoted* in this fashion for delivery to a federate, parameters not present in the delivered class will be filtered from the set of values delivered to that federate.

If the optional *active* argument is equal to *RTI::RTI_TRUE*, remote publishers of the subscribed interaction class (and its subclasses) may receive `turnInteractionsOn()` callbacks to advise them of the presence of a subscriber.

If the optional *active* argument is equal to *RTI::RTI_FALSE*, the federation will not be notified of the subscription. Thus, no `turnInteractionsOn()` callbacks will be made as a result of the subscription. This option is appropriate for a federate that should not have interactions generated solely for its benefit, but that should receive any interactions that would normally be generated (e.g. data-logging federates).

Invoking this service with a region argument equal to the default region (i.e., the region spanning the entire routing space to which the interaction class is bound in the FED file) is equivalent to invoking `subscribeInteractionClass()` for the same interaction class.

RETURN VALUES

A non-exceptional return indicates that the specified region has been added to the set of regions for which the specified interaction class is subscribed by the federate.

INTERFACE SPECIFICATION NOTES

The HLA Interface Specification states that interactions may only be associated with regions of the space to which the interaction class is bound in the FED file. The 1.3 implementation will allow interactions to be sent with regions of any routing space. Federates should not rely on this behavior.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateLoggingServiceCalls

This exception is not thrown by the current implementation.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::RegionNotKnown

The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

receiveInteraction()

turnInteractionsOn()

RTI::Region
RTI::RTIambassador::
createRegion()
getInteractionClassHandle()
notifyAboutRegionModification()
subscribeInteractionClass()
unsubscribeInteractionClassWithRegion()

A.6.9 subscribeObjectClassAttributesWithRegion()

RTI 1.3

ABSTRACT

This service instructs an LRC to begin delivering to the federate updates of instances of a specified set of class-attributes occurring in a specified subspace.

HLA IF SPECIFICATION

This method realizes the "Subscribe Object Class Attributes With Region" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.8 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   subscribeObjectClassAttributesWithRegion (
        RTI::ObjectClassHandle   theClass,
        RTI::Region             &theRegion,
 const RTI::AttributeHandleSet &attributeList,
        RTI::Boolean        active = RTI_TRUE
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the object class for which to subscribe to the specified region

theRegion

the region to which to subscribe

attributeList

the attributes of the specified object class to which to subscribe

active

flag indicating whether the subscription should be considered by remote federates in determining relevance of updates and registrations

DESCRIPTION

This service is similar to the `subscribeObjectClassAttributes()` Declaration Management service, except that the subscription is limited based on a specified region of the routing space to which the class-attributes class are bound in the FED file. The effects of this service are cumulative; that is, a newly subscribed region is added to the set of regions already subscribed for each of the specified class-attributes. The union of all regions in which a class-attribute is subscribed may be thought of as the *effective* subscription region of that class-attribute for a federate.

An invocation of this service replaces the effects of any previous invocations for the same class/region pair. That is, any class-attributes previously subscribed in the specified region will be implicitly unsubscribed in that region if they do not appear in the class-attribute set of a subsequent `subscribeObjectClassAttributeWithRegion()` service invocation whose subject is the same class/region pair.

An object instance will be discovered at the most specific object class for which there exists a class-attribute whose effective subscription region intersects the region associated with the corresponding instance-attribute of the instance at the time the discovery is computed. Subsequent reflections for an instance will include values for a non-empty subset of attributes of the discovered class that are subscribed by the federate. Only attribute-instances whose associated update region intersects the federate's effective subscription region for the corresponding class-attribute at the level of the discovered class will be included in a reflection.

If the optional *active* argument is equal to *RTI::RTI_TRUE*, remote publishers of the subscribed object class (and its subclasses) may receive `startRegistrationForObjectClass()` callbacks to advise them of the presence of a subscriber.

If the optional *active* argument to the subscription is equal to *RTI::RTI_FALSE*, the federation will not be notified of the subscription. Thus, no `startRegistrationForObjectClass()` callbacks will be made as a result of the subscription. This option is appropriate for a federate that should not have registrations and updates made solely for its benefit, but that should receive any updates that would normally be generated (e.g. data-logging federates).

Subscriptions occur independently of federation logical time. Subscription to a class-attribute generally does not cause updates occurring in the (wallclock) past of the federation to be delivered to the subscribing federates, even if such updates occur at a logical time that is in the federate's future. Unsubscription to a class-attribute (implicit or explicit) will cause updates for instances of the attribute that have been previously queued for delivery to be discarded at the point that they would have otherwise been delivered to the federate.

Invoking this service with a region argument equal to the default region (i.e., the region spanning the entire routing space to which the attributes are bound in the FED file) is equivalent to invoking the `subscribeObjectClassAttributes()` service with the same arguments. Invoking this service with an empty attribute set is equivalent to invoking the `unsubscribeObjectClassWithRegion()` service with the specified object class and region.

RETURN VALUES

A non-exceptional return indicates that the specified region has been added to the subscriptions for the specified class-attributes. In addition, the specified region has been removed (if present) from the subscriptions of any attributes of the specified class not appearing in the specified set of class-attributes.

INTERFACE SPECIFICATION NOTES

The HLA Interface Specification states that attribute-instances may only be associated with regions of the space to which the attribute is bound in the FED file. The 1.3 implementation will allow attribute-instances to be registered with regions of any routing space. Federates should not rely on this behavior.

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember
The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined
The object class handle is not valid in the context of the current FedExec.

RTI::RegionNotKnown
The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError
An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress
The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet
RTI::FederateAmbassador::
attributesInScope()
discoverObjectInstance()
reflectAttributeValues()
startRegistrationForObjectClass()
RTI::Region
RTI::RTIambassador::
associateRegionForUpdates()
getAttributeHandle()
getObjectClassHandle()
notifyAboutRegionModification()
publishObjectClass()
registerObjectInstanceWithRegion()
subscribeObjectClass()

A.6.10 unassociateRegionForUpdates()

RTI 1.3

ABSTRACT

This service removes the association between a region and any attributes of an object-instance associated with the region. The affected attributes revert to the default region.

HLA IF SPECIFICATION

This method realizes the "Unassociate Region For Updates" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.7 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador
   unassociateRegionForUpdates (
       RTI::Region       &theRegion,
       RTI::ObjectHandle  theObject
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InvalidRegionContext,
   RTI::ObjectNotKnown,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError
   RTI::SaveInProgress,
)
```

ARGUMENTS

theRegion

a region to be unassociated with the specified instance-attributes

theObject

the instance whose attributes are being unassociated

DESCRIPTION

This service undoes the effect of any *RTIambassador::registerObjectInstanceWithRegion* or *RTIambassador::associateRegionForUpdates* invocations for the specified region/object pair. Any attributes of the object-instance that are associated with the region revert to the default region of the routing space to which the attribute is bound in the FED file.

This service must be used to break the association between a region and any locally owned attribute-instances before the region may be deleted.

RETURN VALUES

A non-exceptional return indicates that the associations (if any) between attributes of the specified object instance and the specified region have been removed.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidRegionContext

This exception is not thrown by the current implementation of this service; if the region is not associated with any attributes of the object-instance, the service equates to a no-op.

RTI::ObjectNotKnown

The specified object ID is not valid within the current FedExec or is not known to the federate.

RTI::RegionNotKnown

The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador

associateRegionForUpdates()

notifyAboutRegionModification()

registerObjectInstanceWithRegion()

deleteRegion()

A.6.11 unsubscribeInteractionClassWithRegion()

RTI 1.3

ABSTRACT

This service withdraws a federate's interest in receiving interactions of a specified class in a specified region.

HLA IF SPECIFICATION

This method realizes the "Unsubscribe Interaction Class With Region" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.11 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   unsubscribeInteractionClassWithRegion (
       RTI::InteractionClassHandle theClass,
       RTI::Region                &theRegion
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionClassNotSubscribed,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

ARGUMENTS

theClass

the interaction class to unsubscribe in the specified region

theRegion

the region being unsubscribed

DESCRIPTION

This service removes the specified region from the set of regions in which the specified interaction class is subscribed by the local federate. The arguments to this service must be a class/region pair that has been the subject of a previous `subscribeInteractionClassWithRegion()` invocation by the local federate.

Subsequent to an invocation of `unsubscribeInteractionClassWithRegion()`, interactions of the specified class (and its subclasses) will only be delivered to the federate if some other region subscribed by the local federate intersects the region associated with a particular interaction instance. In the presence of overlapping subscription regions, unsubscripion to a region does not necessarily imply the cessation of interaction delivery over the entirety of the unsubscribed region. Only those portions of the unsubscribed region that are unique among the set of currently subscribed regions will be effectively removed from the cumulative subscription region for the interaction class. That is, interactions will continue to be delivered to a federate if their associated regions intersect any of the remaining subscribed regions, even if the region of intersection is entirely subsumed by the unsubscribed region.

Subscription changes by a federate affect interactions that are queued for delivery to the federate at the time of the subscription change (e.g., time-stamp-ordered interactions deferred by the LRC pending the achievement of the appropriate logical time by the federate). This is in addition to interactions that occur subsequently in the federation (in wallclock time).

If the subscription region removed by an invocation of this service was an *active* subscription, remote publishers of the specified interaction class (and its subclasses) may receive `turnInteractionsOff()` callbacks advising them of an absence of subscribers.

Invoking this service with a region argument equal to the default region (i.e., the region spanning the entire routing space to which the interaction class is bound in the FED file) is equivalent to invoking `unsubscribeInteractionClass()` for the same interaction class.

RETURN VALUES

A non-exceptional return indicates that the specified region has been removed from the set of regions subscribed for the specified interaction class.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::InteractionClassNotSubscribed

The interaction class is not currently subscribed by the local federate, but is used in a context requiring a subscribed interaction class.

RTI::RegionNotKnown

The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::
turnInteractionsOff()

RTI::Region

RTI::RTIambassador::
createRegion()

getInteractionClassName()

notifyAboutRegionModification()

sendInteractionWithRegion()

subscribeInteractionClass()

subscribeInteractionClassWithRegion()

A.6.12 unsubscribeObjectClassWithRegion()

RTI 1.3

ABSTRACT

This service withdraws the federate's interest in receiving updates of a specified object class in a specified region.

HLA IF SPECIFICATION

This method realizes the "Unsubscribe Object Class With Region" Data Distribution Management service as specified in the *HLA Interface Specification* (§9.9 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   unsubscribeObjectClassWithRegion (
       RTI::ObjectClassHandle theClass,
       RTI::Region           &theRegion
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RegionNotKnown,
   RTI::RestoreInProgress,
   RTI::RTIinternalError
   RTI::SaveInProgress,
)
```

ARGUMENTS

theClass

object class for which to unsubscribe the region

theRegion

the region being unsubscribed

DESCRIPTION

This service removes the specified region from any subscription sets for class-attributes at the level of the specified object class in which the region is currently present. The arguments to this service must be a class/region pair that has been the subject of a previous `subscribeObjectClassAttributesWithRegion()` invocation by the local federate.

Subsequent to an invocation of `unsubscribeObjectClassWithRegion()`, discoveries and updates of class-attributes at the level of the specified class will only be delivered to the federate if some other region subscribed by the local federate intersects the region associated with a particular attribute-instance. In the presence of overlapping subscription regions, unsubscripion to a region does not necessarily imply the cessation of class-attribute delivery over the entirety of the unsubscribed region. Only those portions of the unsubscribed region that are unique among the set of currently subscribed regions will be effectively removed from the cumulative subscription region for the class-attribute. That is, discoveries and updates will continue to be delivered to a federate if their associated regions intersect any of the remaining subscribed regions, even if the region of intersection is entirely subsumed by the unsubscribed region.

Unsubscription to a class/region pair may cause updates for object instances that have been discovered as the unsubscribed class by the federate to be no longer delivered. Such attribute-instances will not be delivered to the federate even if the federate is subscribing to the class-attributes in a relevant region at an object-class level different from the discovered class of the object instance. The federate may wish to use the `localDeleteObjectInstance()` service to allow for rediscovery of object instances at a currently subscribed object-class level. A federate will receive `attributesOutOfScope()` callbacks advising it as to which attribute-instances will no longer be updated as a result of the unsubscription.

Subscription changes by a federate affect updates that are queued for delivery to the federate at the time of the subscription change (e.g., time-stamp-ordered updates deferred by the LRC pending the achievement of the appropriate logical time by the federate). This is in addition to updates that occur subsequently in the federation (in wallclock time).

If the subscription region removed by an invocation of this service was an *active* subscription, remote publishers of the affected class-attributes (including subclasses) may receive `stopRegistrationForObjectClass()` and `turnUpdatesOffForObjectInstance()` callbacks advising them of an absence of subscribers.

Invoking this service with a region argument equal to the default region (i.e., the region spanning the entire routing space to which the class-attributes are class is bound in the FED file) is equivalent to invoking `unsubscribeObjectClass()` for the same set of class-attributes at the same object-class level.

RETURN VALUES

A non-exceptional return indicates that the specified region has been removed from the subscription sets of any class-attributes at the specified object-class in which it was present.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RegionNotKnown

The region argument is not recognized as a valid region as instantiated *by* the `createRegion()` service.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador::

attributesOutOfScope()

stopRegistrationForObjectClass()

turnUpdatesOffForObjectInstance()

RTI::Region
RTI::RTIambassador::
createRegion()
getObjectClassHandle()
notifyAboutRegionModification()
subscribeObjectClassAttributes()
subscribeObjectClassAttributesWithRegion()
unsubscribeObjectClass()

A.7 Types and Ancillary Services

A.7.1 ~RTIambassador()

RTI 1.0 RTI 1.3

ABSTRACT

This destructor frees heap-allocated memory and all other resources associated with an *RTIambassador* object instance.

HLA IF SPECIFICATION

This destructor does not correspond to a service explicitly listed in the *HLA Interface Specification* version 1.3.

SYNOPSIS

```
#include <RTI.hh>

RTI::RTIambassador::
   ~RTIambassador( )
throw (
   RTI::RTIinternalError
)
```

DESCRIPTION

This destructor is implicitly invoked when an *RTIambassador* object is destroyed, either by a stack-allocated object going out of scope or a heap-allocated object being explicitly deallocated using the `delete` operator.

The destructor deallocates memory and other resources associated with the *RTIambassador* instance. **The federate is responsible for properly resigning from any federation execution that may be associated with an *RTIambassador* before destroying the object instance.**

RETURN VALUES

A non-exceptional return indicates that the resources associated with the RTI ambassador instance have been freed.

EXCEPTIONS

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador::

resignFederationExecution()

RTIambassador()

A.7.2 dequeueFIFOasynchronously()

RTI 1.0

ABSTRACT

This service specifies whether a federate wishes to process receive-ordered events in the absence of an in-progress time-advancement service. **The RTI 1.3 equivalents of this service are named *`enableAsynchronousDelivery`* and *`disableAsynchronousDelivery`*; the RTI 1.3 implementation is discussed in separate sections.**

HLA IF SPECIFICATION

This method does not directly correspond to a service specified in the *HLA Interface Specification* version 1.1.

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   dequeueFIFOasynchronously (
       RTI::Boolean theSwitch
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::RestoreInProgress,
   RTI::RTIinternalError,
   RTI::SaveInProgress
)
```

DESCRIPTION

This service allows the federate to specify whether or not it wishes to process receive-ordered events in the absence of an outstanding time-advance service. This is only meaningful for time-constrained federates, as non-time-constrained federates always process events as soon as possible.

A true setting will result in receive-ordered events being delivered to the federate as soon as possible in response to a *RTIambassador::tick* invocation. A false setting (the default) will result in receive-ordered events being queued until the federate initiates a time-advancement service (e.g. *RTIambassador::timeAdvanceRequest*.)

RETURN VALUES

A non-exceptional return value indicates that the federate has reset its asynchronous-dequeue preference.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

setTimeConstrained()
tick()
timeAdvanceRequest()

A.7.3 disableAttributeRelevanceAdvisorySwitch()

RTI 1.3

ABSTRACT

This service instructs the LRC to stop notifying the federate of changes in attribute-instance update relevance.

HLA IF SPECIFICATION

This method realizes the “Disable Attribute Relevance Advisory Switch” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.26 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   disableAttributeRelevanceAdvisorySwitch( )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to stop notifying the federate of changing attribute-instance update relevance. The federate will no longer receive `turnUpdatesOnForObjectInstance()` or `turnUpdatesOffForObjectInstance()` callbacks.

The attribute relevance advisory is disabled by default.

Attribute-instance update relevance advising may be re-enabled using the `enableAttributeRelevanceAdvisorySwitch()` service. The federate will *not* be retroactively notified of changes in relevance that occurred while relevance advising was disabled.

Invoking this service while attribute-instance update relevance advising is already disabled results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will not inform the federate of subsequently occurring changes in attribute-instance update relevance.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador

turnUpdatesOffForObjectInstance()

turnUpdatesOnForObjectInstance()

RTI::RTIambassador

disableAttributeScopeAdvisorySwitch()

disableClassRelevanceAdvisorySwitch()

disableInteractionRelevanceAdvisorySwitch()

enableAttributeRelevanceAdvisorySwitch()

A.7.4 disableAttributeScopeAdvisorySwitch()

RTI 1.3

ABSTRACT

This service instructs the LRC to stop notifying the federate of changes in attribute-instance scope.

HLA IF SPECIFICATION

This method realizes the “Disable Attribute Scope Advisory Switch” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.28 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   disableAttributeScopeAdvisorySwitch( )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to stop notifying the federate of changing attribute-instance scope. The federate will no longer receive `attributesInScope()` or `attributesOutOfScope()` callbacks.

The attribute scope advisory is disabled by default.

Attribute-instance scope advising may be re-enabled using the `enableAttributeScopeAdvisorySwitch()` service. The federate will *not* be retroactively notified of changes in scope that occurred while relevance advising was disabled.

Invoking this service while attribute-instance scope advising is already disabled results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will not inform the federate of subsequently occurring changes in attribute-instance scope.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador

attributesInScope()

attributesOutOfScope()

RTI::RTIambassador

disableAttributeRelevanceAdvisorySwitch()

disableClassRelevanceAdvisorySwitch()

disableInteractionRelevanceAdvisorySwitch()

enableAttributeScopeAdvisorySwitch()

A.7.5 disableClassRelevanceAdvisorySwitch()

RTI 1.3

ABSTRACT

This service instructs the LRC to stop notifying the federate of changes in object-class registration relevance.

HLA IF SPECIFICATION

This method realizes the “Disable Class Relevance Advisory Switch” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.24 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   disableClassRelevanceAdvisorySwitch( )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to stop notifying the federate of changing object-class registration relevance. The federate will no longer receive `startRegistrationForObjectClass()` or `stopRegistrationForObjectClass()` callbacks. Object-class registration relevance advising may be re-enabled using the `enableClassRelevanceAdvisorySwitch()` service. The federate will *not* be retroactively notified of changes in relevance that occurred while relevance advising was disabled.

Invoking this service while object-class registration relevance advising is already disabled results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will not inform the federate of subsequently occurring changes in object-class registration relevance.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador

startRegistrationForObjectClass()

stopRegistrationForObjectClass()

RTI::RTIambassador

disableAttributeRelevanceAdvisorySwitch()

disableAttributeScopeAdvisorySwitch()

disableInteractionRelevanceAdvisorySwitch()

enableClassRelevanceAdvisorySwitch()

A.7.6 disableInteractionRelevanceAdvisorySwitch()

RTI 1.3

ABSTRACT

This service instructs the LRC to stop notifying the federate of changes in interaction-class relevance.

HLA IF SPECIFICATION

This method realizes the “Disable Interaction Relevance Advisory Switch” Federation Management service as specified in the *HLA Interface Specification* (§10.30 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   disableInteractionRelevanceAdvisorySwitch( )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to stop notifying the federate of changing interaction-class relevance. The federate will no longer receive `turnInteractionsOn()` or `turnInteractionsOff()` callbacks.

Interaction-class relevance advising may be re-enabled using the `enableInteractionRelevanceAdvisorySwitch()` service. The federate will *not* be retroactively notified of changes in relevance that occurred while relevance advising was disabled.

Invoking this service while interaction-class relevance advising is already disabled results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will not inform the federate of subsequently occurring changes in interaction-class relevance.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador

turnInteractionsOff()

turnInteractionsOn()

RTI::RTIambassador

disableAttributeRelevanceAdvisorySwitch()

disableAttributeScopeAdvisorySwitch()

disableClassRelevanceAdvisorySwitch()

enableInteractionRelevanceAdvisorySwitch()

A.7.7 enableAttributeRelevanceAdvisorySwitch()

RTI 1.3

ABSTRACT

This service instructs the LRC to begin notifying the federate of changes in attribute-instance update relevance.

HLA IF SPECIFICATION

This method realizes the “Enable Attribute Relevance Advisory Switch” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.25 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   enableAttributeRelevanceAdvisorySwitch( )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to begin notifying the federate of changing attribute-instance update relevance, using the `turnUpdatesOnForObjectInstance()` and `turnUpdatesOffForObjectInstance()` callbacks. Updates of an attribute-instance are considered to be relevant for a federate if and only if the attribute-instance is actively subscribed by at least one remote federate.

Relevance advising is disabled by default.

This service is required to enable the service and when re-enabling relevance advising after a `disableAttributeRelevanceAdvisorySwitch()` invocation. The federate will *not* be retroactively notified of changes in relevance that occurred while relevance advising was disabled.

Invoking this service while attribute-instance update relevance advising is already enabled results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will inform the federate of subsequently occurring changes in attribute-instance update relevance.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador

turnUpdatesOffForObjectInstance()

turnUpdatesOnForObjectInstance()

RTI::RTIambassador

disableAttributeRelevanceAdvisorySwitch()

enableAttributeScopeAdvisorySwitch()

enableClassRelevanceAdvisorySwitch()

enableInteractionRelevanceAdvisorySwitch()

A.7.8 enableAttributeScopeAdvisorySwitch()

RTI 1.3

ABSTRACT

This service instructs the LRC to begin notifying the federate of changes in attribute-instance scope.

HLA IF SPECIFICATION

This method realizes the “Enable Attribute Scope Advisory Switch” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.27 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   enableAttributeScopeAdvisorySwitch( )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to begin notifying the federate of changing attribute-instance scope, using the `attributesInScope()` and `attributesOutOfScope()` callbacks. An attribute-instance is considered in-scope for a federate if and only if:

- the object instance is known to the federate through registration or discovery
- the attribute-instance is not owned by the federate
- the federate has subscribed to the class-attribute with a region overlapping the associated attribute-instance’s update region

Scope advising is disabled by default. This service is required to enable attribute scope advisory and when re-enabling scope advising after a `disableAttributeScopeAdvisorySwitch()` invocation. The federate will *not* be retroactively notified of changes in scope that occurred while scope advising was disabled.

Invoking this service while attribute-instance scope advising is already enabled results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will inform the federate of subsequently occurring changes in attribute-instance scope.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador

attributesInScope()

attributesOutOfScope()

RTI::RTIambassador

disableAttributeScopeAdvisorySwitch()

enableAttributeRelevanceAdvisorySwitch()

enableClassRelevanceAdvisorySwitch()

enableInteractionRelevanceAdvisorySwitch()

A.7.9 enableClassRelevanceAdvisorySwitch()

RTI 1.3

ABSTRACT

This service instructs the LRC to begin notifying the federate of changes in object-class registration relevance.

HLA IF SPECIFICATION

This method realizes the “Enable Class Relevance Advisory Switch” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.23 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   enableClassRelevanceAdvisorySwitch( )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to begin notifying the federate of changing object-class registration relevance, using the `startRegistrationForObjectClass()` and `stopRegistrationForObjectClass()` callbacks. Registration of an object class is considered to be relevant for a federate when there exists an attribute of the object class that is both published by the federate and actively subscribed by at least one remote federate. This includes subscriptions to more general superclasses of the object class in question.

Relevance advising is enabled by default, so this service is only required when re-enabling relevance advising after a `disableClassRelevanceAdvisorySwitch()` invocation. The federate will *not* be retroactively notified of changes in relevance that occurred while relevance advising was disabled.

Invoking this service while object-class registration relevance advising is already enabled results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will inform the federate of subsequently occurring changes in object-class registration relevance.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador

startRegistrationForObjectClass()

stopRegistrationForObjectClass()

RTI::RTIambassador

disableClassRelevanceAdvisorySwitch()

enableAttributeRelevanceAdvisorySwitch()

enableAttributeScopeAdvisorySwitch()

enableInteractionRelevanceAdvisorySwitch()

A.7.10 enableInteractionRelevanceAdvisorySwitch()

RTI 1.3

ABSTRACT

This service instructs the LRC to begin notifying the federate of changes in interaction-class relevance.

HLA IF SPECIFICATION

This method realizes the “Enable Interaction Relevance Advisory Switch” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.29 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

void
RTI::RTIambassador::
   enableInteractionRelevanceAdvisorySwitch( )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RestoreInProgress,
  RTI::RTIinternalError,
  RTI::SaveInProgress
)
```

DESCRIPTION

This service instructs the LRC to begin notifying the federate of changing interaction-class relevance, using the `turnInteractionsOn()` and `turnInteractionsOff()` callbacks. Generation of an interaction class is considered to be relevant for a federate when at least one remote federate has subscribed to the interaction class. This includes subscriptions to more general superclasses of the interaction class in question.

Relevance advising is enabled by default, so this service is only required when re-enabling relevance advising after a `disableInteractionRelevanceAdvisorySwitch()` invocation. The federate will *not* be retroactively notified of changes in relevance that occurred while relevance advising was disabled.

Invoking this service while interaction-class relevance advising is already enabled results in a no-op.

RETURN VALUES

A non-exceptional return indicates that the LRC will inform the federate of subsequently occurring changes in interaction-class relevance.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::FederateAmbassador

turnInteractionsOff()

turnInteractionsOn()

RTI::RTIambassador

disableInteractionRelevanceAdvisorySwitch()

enableAttributeRelevanceAdvisorySwitch()

enableAttributeScopeAdvisorySwitch()

enableClassRelevanceAdvisorySwitch()

A.7.11 getAttributeHandle()

RTI 1.0 RTI 1.3

ABSTRACT

This service converts a symbolic (string) attribute name and an object-class context to the RTI handle associated with the attribute. **The syntax and semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Get Attribute Handle" RTI Support & Operational Services as specified in the *HLA Interface*

Specification (not explicitly listed in version 1.1; §10.4 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::AttributeHandle
RTI::RTIambassador::
   getAttributeHandle (
       const RTI::AttributeName theName
       RTI::ObjectClassHandle   whichClass
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::NameNotFound,
   RTI::ObjectClassNotDefined,
   RTI::RestoreInProgress,             ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                 ← RTI 1.0 Only
)

// RTI 1.3 Only
RTI::AttributeHandle
RTI::RTIambassador::
   getAttributeHandle (
       const char             *theName, ← Changes Types
       RTI::ObjectClassHandle whichClass
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::NameNotFound,
  RTI::ObjectClassNotDefined,
  RTI::RTIinternalError
)
```

ARGUMENTS

theName

a string specifying the symbolic attribute class name to be converted to an RTI-defined handle

whichClass

the object-class context of the specified attribute name

DESCRIPTION

This service converts an attribute *name* (i.e., the descriptive string identifier found in the FED file) to an RTI *handle* that is used by the various RTI services to refer to the attribute. The attribute name must refer to an attribute of the specified object class, i.e. the attribute must be natively defined in the object class or defined in a superclass of the object class.

Conceptually, an attribute handle is only valid when considered in the context of an object class. Implications of this include:

the same physical value may refer to different attributes in the context of two different object classes (this is true of RTI 1.0 and RTI 1.3)

different physical values may refer to the same attribute in the context of different object classes (this is not true of RTI 1.0 and RTI 1.3)

multiple distinct attributes may share the same name provided they may never occur in the context of the same object class (i.e., no object class containing an attribute name is a superclass of another object class containing the same attribute name)

The returned handle is valid throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

RETURN VALUES

A successful invocation of this service returns the RTI handle associated with the specified attribute.

RELEASE NOTES

RTI 1.0

Names in RTI 1.0 are case-sensitive.

RTI 1.3

Names in RTI 1.3 are case-insensitive.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

This method is safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NameNotFound

The symbolic name does not correspond to a handle of the requested type.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet

RTI::AttributeHandleValuePairSet

RTI::RTIambassador::

createFederationExecution()

getAttributeName()

getObjectClassHandle()

joinFederationExecution()

A.7.12 getAttributeName()

RTI 1.0 RTI 1.3

ABSTRACT

This service converts an RTI attribute handle and its object-class context to the symbolic (string) name of the FED attribute they represent. **The syntax and semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Get Attribute Name" RTI Support & Operational service as specified in the *HLA Interface Specification* (not explicitly listed in version 1.1; §10.5 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::AttributeName
RTI::RTIambassador::
   getAttributeName (
       RTI::AttributeHandle theHandle
       RTI::ObjectClassHandle whichClass
   )
throw (
   RTI::AttributeNotDefined,
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RestoreInProgress,            ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                ← RTI 1.0 Only
)

// RTI 1.3 Only
char *                                ← Changes Types
RTI::RTIambassador::
   getAttributeName (
       RTI::AttributeHandle   theHandle,
       RTI::ObjectClassHandle whichClass
   )
throw (
  RTI::AttributeNotDefined,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectClassNotDefined,
  RTI::RTIinternalError
);
```

ARGUMENTS

theHandle

a handle representing the attribute whose name is being queried

whichClass

the object-class context of the specified attribute handle

DESCRIPTION

This service converts an RTI attribute *handle* (i.e., a unique identifier used by the RTI to refer to an attribute), and its object-class context, to the symbolic (string) name of the attribute as found in the FED file.

The returned name is valid (i.e., guaranteed to be associated with the specified handle) throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

RETURN VALUES

A successful invocation of this service returns the symbolic attribute name corresponding to the specified attribute handle and object-class context. **The caller is responsible for freeing the memory associated with the returned string using the `delete[]` operator.**

RELEASE NOTES

RTI 1.0

The name returned is the attribute name exactly as it appears in the FED file, e.g. "FederateHandle".

RTI 1.3

The name returned is the lower-case attribute name, e.g. "federatehandle".

EXCEPTIONS

RTI::AttributeNotDefined

One or more of the specified attribute handles is not valid within the context of the specified object class.

RTI::ConcurrentAccessAttempted

This method is safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::AttributeHandleSet
RTI::AttributeHandleValuePairSet
RTI::RTIambassador::

createFederationExecution()

getAttributeHandle()

getObjectClassHandle()

getObjectClassName()

joinFederationExecution()

A.7.13 getAttributeRoutingSpaceHandle()

RTI 1.3

ABSTRACT

This service queries the routing space bound to an attribute in the FED file.

HLA IF SPECIFICATION

This method realizes the "Get Attribute Routing Space Handle" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.16 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::SpaceHandle
RTI::RTIambassador::
   getAttributeRoutingSpaceHandle (
       RTI::AttributeHandle   theHandle,
       RTI::ObjectClassHandle whichClass
   )
throw (
  RTI::AttributeNotDefined,
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectClassNotDefined,
  RTI::RTIinternalError
)
```

ARGUMENTS

theHandle

a handle representing the attribute whose routing space is being queried

whichClass

the object-class context of the specified attribute handle

DESCRIPTION

This service queries the routing space bound to a particular attribute in the FED file. If no routing space is specified in the FED file, the attribute is bound to the default routing space.

Different attributes of the same object class may be bound to different routing spaces. An attribute is always bound to the same routing space in any subclasses of which it is (implicitly) an attribute as it is in the object class in which it is natively declared.

RETURN VALUES

A successful invocation of this service returns a handle to the routing space bound to the attribute in the FED file.

INTERFACE SPECIFICATION NOTES

The HLA Interface Specification states that attribute-instances may only be associated with regions of the space to which the attribute is bound in the FED file. The 1.3 implementation will allow attribute-instances to be registered with regions of any routing space. Federates should not rely on this behavior.

EXCEPTIONS

RTI::AttributeNotDefined

The attribute handle is not valid in the context of the specified object class.

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::Region

RTI::RTIambassador::

associateRegionForUpdates()

createFederationExecution()

getInteractionRoutingSpaceHandle()

getRoutingSpaceName()

joinFederationExecution()

registerObjectInstance()

registerObjectInstanceWithRegion()

A.7.14 getDimensionHandle()

RTI 1.3

ABSTRACT

This service converts a symbolic (string) dimension name and a routing-space context to the RTI handle associated with the dimension.

HLA IF SPECIFICATION

This method realizes the "Get Dimension Handle" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.14 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::DimensionHandle
RTI::RTIambassador::
   getDimensionHandle (
       const char         *theName,
       RTI::SpaceHandle   whichSpace
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::NameNotFound,
  RTI::RTIinternalError,
  RTI::SpaceNotDefined
)
```

ARGUMENTS

theName

a string specifying the symbolic dimension name to be converted to an RTI-defined handle

whichSpace

the routing-space context of the specified dimension name

DESCRIPTION

This service converts a dimension *name* (i.e., the descriptive string identifier found in the FED file) to an RTI *handle* that is used by the various RTI services to refer to the dimension. The dimension name must refer to a dimension of the specified routing space.

Conceptually, a dimension handle is only valid when considered in the context of a routing space. Implications of this include:

- the same physical value may refer to different dimensions in the context of two different routing spacees (this is true of RTI 1.0 and RTI 1.3)
- multiple distinct dimensions may share the same name provided they do not occur in the same routing space

The returned handle is valid throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

Names in RTI 1.3 are case-insensitive.

RETURN VALUES

A successful invocation of this service returns the RTI handle associated with the specified dimension.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

This method is safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NameNotFound

The symbolic name does not correspond to a handle of the requested type.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SpaceNotDefined

The *SpaceHandle* argument does not correspond to a valid federation routing space. Use *RTIambassador::getRoutingSpaceHandle* to obtain valid space handles.

SEE ALSO

RTI::Region

RTI::RTIambassador::

createFederationExecution()

getDimensionName()

getRoutingSpaceHandle()

joinFederationExecution()

A.7.15 getDimensionName()

RTI 1.3

ABSTRACT

This service converts an RTI dimension handle and its routing-space context to the symbolic (string) name of the FED dimension they represent.

HLA IF SPECIFICATION

This method realizes the “Get Dimension Name” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.15 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

char *
RTI::RTIambassador::
   getDimensionName (
       RTI::DimensionHandle theHandle,
       RTI::SpaceHandle     whichSpace
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::DimensionNotDefined,
  RTI::FederateNotExecutionMember,
  RTI::RTIinternalError,
  RTI::SpaceNotDefined
)
```

ARGUMENTS

theHandle

a handle representing the dimension whose name is being queried

whichSpace

the routing-space context of the specified dimension handle

DESCRIPTION

This service converts an RTI dimension *handle* (i.e., a unique identifier used by the RTI to refer to a dimension), and its routing-space context, to the symbolic (string) name of the dimension as found in the FED file.

The returned name is valid (i.e., guaranteed to be associated with the specified handle) throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

The name returned is the lower-case dimension name.

RETURN VALUES

A successful invocation of this service returns the symbolic dimension name corresponding to the specified dimension handle and routing-space context. **The caller is responsible for freeing the memory associated with the returned string using the `delete[]` operator.**

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RestoreInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::Region

RTI::RTIambassador::

createFederationExecution()

getDimensionHandle()

getRoutingSpaceHandle()

getRoutingSpaceName()

joinFederationExecution()

A.7.16 getInteractionClassHandle()

RTI 1.0 RTI 1.3

ABSTRACT

This service converts a symbolic (string) interaction class name to the RTI handle associated with the interaction class. **The syntax and semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Get Interaction Class Handle" RTI Support & Operational service as specified in the *HLA Interface Specification* (not explicitly listed in version 1.1; §10.6 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::InteractionClassHandle
RTI::RTIambassador::
   getInteractionClassHandle (
       const RTI::InteractionClassName theName
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::NameNotFound,
   RTI::RestoreInProgress,              ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                  ← RTI 1.0 Only
)

// RTI 1.3 Only
RTI::InteractionClassHandle
RTI::RTIambassador::
   getInteractionClassHandle (
       const char *theName              ← Changes Types
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::NameNotFound,
  RTI::RTIinternalError
);
```

ARGUMENTS

theName

a string specifying the symbolic interaction class name to be converted to an RTI-defined handle

DESCRIPTION

This service converts an interaction class *name* (i.e., the descriptive string identifier found in the FED file) to an RTI *handle* that is used by the various RTI services to refer to the interaction class.

The returned handle is valid throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

RETURN VALUES

A successful invocation of this service returns the RTI handle associated with the specified interaction class.

RELEASE NOTES

RTI 1.0

- RTI 1.0 names are case-sensitive.
- The RTI 1.0 implementation expects an argument that is the unqualified class name. For example, the *Federate* subclass of the *Manager* interaction class would be referred to as simply "Federate". An implication of this is that no two interaction classes may have the same name regardless of their relative locations in the FED.

RTI 1.3

- RTI 1.3 names are case-insensitive.
- The RTI 1.3 implementation expects an argument that is the fully-qualified class name, with a period used to delimit classes. The *InteractionRoot* class that is required to be the root of the interaction class hierarchy may optionally be omitted from the fully-qualified class name. For example, the *Federate* subclass of the *Manager* interaction class might be referred to as "interactionroot.manager.federate" or simply "Manager.Federate". An implication of this is that multiple interaction classes may share the same name provided they are distinguishable based on their position in the class hierarchy (i.e., they do not have the same direct superclass.)

EXCEPTIONS

RTI::ConcurrentAccessAttempted

This method is safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NameNotFound

The symbolic name does not correspond to a handle of the requested type.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

createFederationExecution()

getParameterHandle()

getInteractionClassName()

joinFederationExecution()

A.7.17 getInteractionClassName()

RTI 1.0 RTI 1.3

ABSTRACT

This service converts an RTI interaction class handle to the symbolic (string) name of the FED interaction class it represents. **The syntax and semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Get Interaction Class Name" RTI Support & Operational service as specified in the *HLA Interface Specification* (not explicitly listed in version 1.1; §10.7 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::InteractionClassName
RTI::RTIambassador::
   getInteractionClassName (
       RTI::InteractionClassHandle theHandle
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::RestoreInProgress,            ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                ← RTI 1.0 Only
)

// RTI 1.3 Only
char *                                ← Changes Types
RTI::RTIambassador::
   getInteractionClassName (
       RTI::InteractionClassHandle theHandle
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InteractionClassNotDefined,
  RTI::RTIinternalError
)
```

ARGUMENTS

theHandle

a handle representing the interaction class whose name is being queried

DESCRIPTION

This service converts an RTI interaction class *handle* (i.e., a unique identifier used by the RTI to refer to an interaction class) to the symbolic (string) name of the interaction class as found in the FED file.

The returned name is valid (i.e., guaranteed to be associated with the specified handle) throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

RETURN VALUES

A successful invocation of this service returns the symbolic class name corresponding to the specified class handle. **The caller is responsible for freeing the memory associated with the returned string using the `delete[]` operator.**

RELEASE NOTES

RTI 1.0

The name returned is the unqualified class name exactly as it appears in the FED file, e.g. "Federate".

RTI 1.3

The name returned is the lower-case fully-qualified interaction class name, e.g. "interactionroot.manager.federate".

EXCEPTIONS

RTI::ConcurrentAccessAttempted

This method is safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

createFederationExecution()

getAttributeName()

getInteractionClassHandle()

joinFederationExecution()

A.7.18 getInteractionRoutingSpaceHandle()

RTI 1.3

ABSTRACT

This service queries the routing space bound to an interaction in the FED file.

HLA IF SPECIFICATION

This method realizes the "Get Interaction Routing Space Handle" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.18 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::SpaceHandle
RTI::RTIambassador::
   getInteractionRoutingSpaceHandle (
      RTI::InteractionClassHandle   theHandle
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InteractionClassNotDefined,
  RTI::RTIinternalError
)
```

ARGUMENTS

theHandle

a handle representing the interaction class whose routing space is being queried

DESCRIPTION

This service queries the routing space bound to a particular interaction class in the FED file. If no routing space is specified in the FED file, the interaction class is bound to the default routing space.

RETURN VALUES

A successful invocation of this service returns a handle to the routing space bound to the interaction in the FED file.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The interaction class handle is not valid in the context of the current federation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::Region

RTI::RTIambassador::

createFederationExecution()

getAttributeRoutingSpaceHandle()

getRoutingSpaceName()

joinFederationExecution()

sendInteraction()

sendInteractionWithRegion()

A.7.19 getObjectClass()

RTI 1.3

ABSTRACT

This service queries the discovered object class of an object instance.

HLA IF SPECIFICATION

This method realizes the "Get Object Class" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.17 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::ObjectClassHandle
RTI::RTIambassador::
   getObjectClass (
       RTI::ObjectHandle theObject
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectNotKnown,
  RTI::RTIinternalError
)
```

ARGUMENTS

theObject

the handle of the object instance whose class is being queried

DESCRIPTION

This service obtains the discovered object class of an object instance.

This service may be used independently of any other RTI services, i.e. the federate need not have discovered the object instance or even have subscribed to the appropriate classes to query the object's class. The only limitation is that some objects may not be known to some LRCs due to DDM filtering. Information about an instance may never be distributed to some LRCs if its attributes are confined to regions outside the actively subscribed regions of a particular federate.

If the object instance has been discovered by the local federate, the returned object class is the discovered class of the object instance. If the object instance has not been discovered by the local federate, the returned object class is the registered class of the object instance.

RETURN VALUES

A successful invocation of this service returns the handle of the specified object's object class.

INTERFACE SPECIFICATION NOTES

The HLA Interface Specification states that interactions may only be associated with regions of the space to which the interaction class is bound in the FED file. The 1.3 implementation will allow interactions to be sent with regions of any routing space. Federates should not rely on this behavior.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The object handle does not correspond to an object instance known to the LRC.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::FederateAmbassador

discoverObjectInstance()

RTI::RTIambassador

getObjectClassHandle()

getObjectClassName()

A.7.20 getObjectClassHandle()

RTI 1.0 RTI 1.3

ABSTRACT

This service converts a symbolic (string) object class name to the RTI handle associated with the object class. **The syntax and semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Get Object Class Handle" RTI Support & Operational service as specified in the *HLA Interface Specification* (not explicitly listed in version 1.1; §10.2 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::ObjectClassHandle
RTI::RTIambassador::
   getObjectClassHandle (
     const RTI::ObjectClassName theName
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::NameNotFound,
   RTI::RestoreInProgress,            ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                ← RTI 1.0 Only
 )

// RTI 1.3 Only
RTI::ObjectClassHandle
RTI::RTIambassador::
   getObjectClassHandle (
      const char *theName             ← Changes Types
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::NameNotFound,
  RTI::RTIinternalError
)
```

ARGUMENTS

theName

a string specifying the symbolic object class name to be converted to an RTI-defined handle

DESCRIPTION

This service converts an object class *name* (i.e., the descriptive string identifier found in the FED file) to an RTI *handle* that is used by the various RTI services to refer to the object class.

The returned handle is valid throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

RETURN VALUES

A successful invocation of this service returns the RTI handle associated with the specified object class.

RELEASE NOTES

RTI 1.0

- RTI 1.0 names are case-sensitive.
- The RTI 1.0 implementation expects an argument that is the unqualified class name. For example, the *Federate* subclass of the *Manager* object class would be referred to as simply "Federate". An implication of this is that no two object classes may have the same name regardless of their relative locations in the FED.

RTI 1.3

- RTI 1.3 names are case-insensitive.
- The RTI 1.3 implementation expects an argument that is the fully-qualified class name, with a period used to delimit classes. The *ObjectRoot* class that is required to be the root of the object class hierarchy may optionally be omitted from the fully-qualified class name. For example, the *Federate* subclass of the *Manager* object class might be referred to as "objectroot.manager.federate" or simply "Manager.Federate". An implication of this is that multiple object classes may share the same name provided they are distinguishable based on their position in the class hierarchy (i.e., they do not have the same direct superclass.)

EXCEPTIONS

RTI::ConcurrentAccessAttempted

This method is safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NameNotFound

The symbolic name does not correspond to a handle of the requested type.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

createFederationExecution()

getAttributeHandle()

getObjectClassName()

joinFederationExecution()

A.7.21 getObjectClassName()

RTI 1.0 RTI 1.3

ABSTRACT

This service converts an RTI object class handle to the symbolic (string) name of the FED object class it represents. **The syntax and semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Get Object Class Name" RTI Support & Operational service as specified in the *HLA Interface Specification* (not explicitly listed in version 1.1; §10.3 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::ObjectClassName
RTI::RTIambassador::
   getObjectClassName (
       RTI::ObjectClassHandle theHandle
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::ObjectClassNotDefined,
   RTI::RestoreInProgress,            ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                ← RTI 1.0 Only
)

// RTI 1.3 Only
char *                                ← Changes Types
RTI::RTIambassador::
   getObjectClassName (
       RTI::ObjectClassHandle theHandle
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectClassNotDefined,
  RTI::RTIinternalError
)
```

ARGUMENTS

theHandle

a handle representing the object class whose name is being queried

DESCRIPTION

This service converts an RTI object class *handle* (i.e., a unique identifier used by the RTI to refer to an object class) to the symbolic (string) name of the object class as found in the FED file.

The returned name is valid (i.e., guaranteed to be associated with the specified handle) throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

RETURN VALUES

A successful invocation of this service returns the symbolic class name corresponding to the specified class handle. **The caller is responsible for freeing the memory associated with the returned string using the `delete[]` operator.**

RELEASE NOTES

RTI 1.0

The name returned is the unqualified class name exactly as it appears in the FED file, e.g. "Federate".

RTI 1.3

The name returned is the lower-case fully-qualified object class name, e.g. "objectroot.manager.federate".

EXCEPTIONS

RTI::ConcurrentAccessAttempted

This method is safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectClassNotDefined

The object class handle is not valid in the context of the current FedExec.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::RTIambassador::

createFederationExecution()

getAttributeName()

getObjectClassHandle()

joinFederationExecution()

A.7.22 getObjectInstanceHandle()

RTI 1.3

ABSTRACT

This service converts a symbolic (string) name to the RTI object handle uniquely associated with the name in the current federation state.

HLA IF SPECIFICATION

This method realizes the “Get Object Instance Handle” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.10 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::ObjectHandle
RTI::RTIambassador::
   getObjectInstanceHandle (
      const char *theName
   )
throw (
    RTI::ConcurrentAccessAttempted,
    RTI::FederateNotExecutionMember,
    RTI::ObjectNotKnown,
    RTI::RTIinternalError
)
```

ARGUMENTS

theName

symbolic name associated with a global federation object

DESCRIPTION

This service converts a symbolic object name to an object handle. Every object in the federation at a given instant is associated with a unique symbolic name, either explicitly specified to the `registerObjectInstance()` service or automatically supplied by the LRC.

This service may be used independently of any other RTI services, i.e. the federate need not have discovered the object instance or even have subscribed to the appropriate classes to obtain an object handle. The only limitation is that some objects may not be known to some LRCs due to DDM filtering. Information about an instance may never be distributed to some LRCs if its attributes are confined to regions outside the actively subscribed regions of a particular federate.

Named object instances will generally be used to implement “global” objects in the federation.

RETURN VALUES

A successful invocation of this service returns the object handle uniquely associated with the symbolic name.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The specified symbolic name does not correspond to an active object known to the LRC.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador

getObjectInstanceName()

registerObjectInstance()

A.7.23 getObjectInstanceName()

RTI 1.3

ABSTRACT

This service converts an RTI object handle to the symbolic (string) name associated with the object in the current federation state.

HLA IF SPECIFICATION

This method realizes the "Get Object Instance Name" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.11 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

char *
RTI::RTIambassador::
   getObjectInstanceName (
       RTI::ObjectHandle theHandle
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::ObjectNotKnown,
  RTI::RTIinternalError
)
```

ARGUMENTS

theHandle

the RTI object handle whose unique instance name is being queried

DESCRIPTION

This service converts a symbolic object handle to an object name. Every object in the federation at a given instant is associated with a unique symbolic name, either explicitly specified to the `registerObjectInstance()` service or automatically supplied by the LRC.

This service may be used independently of any other RTI services, i.e. the federate need not have discovered the object instance or even have subscribed to the appropriate classes to obtain an object name. The only limitation is that some objects may not be known to some LRCs due to DDM filtering. Information about an instance may never be distributed to some LRCs if its attributes are confined to regions outside the actively subscribed regions of a particular federate.

Named object instances will generally be used to implement "global" objects in the federation.

RETURN VALUES

A successful invocation of this service returns the string name associated with the object handle. **The caller is responsible for freeing the memory associated with this string using the `delete[]` operator.**

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::ObjectNotKnown

The object handle does not correspond to an active object known to the LRC.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador

getObjectInstanceHandle()

registerObjectInstance()

A.7.24 getOrderingHandle()

RTI 1.3

ABSTRACT

This service converts a symbolic (string) name of an ordering service category to an RTI handle.

HLA IF SPECIFICATION

This method realizes the “Get Ordering Handle” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.21 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::OrderingHandle
RTI::RTIambassador::
   getOrderingHandle (
      const char *theName
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::NameNotFound,
  RTI::RTIinternalError
)
```

ARGUMENTS

theName

a symbolic name of an ordering category

DESCRIPTION

This service converts a symbolic (string) identifier to an RTI handle associated with an ordering service category defined in the *ordering_map* section of the RID file. These handles are used by various RTI services when referring to ordering categories.

The federation may define its own named ordering service categories as aliases for timestamp- or receive-order service. This technique may be used to decouple quality-of-service decisions for various types of federation data from the code of the constituent federates.

The RTI defines two ordering categories: *receive* and *timestamp.* These identifiers should always appear in the *ordering_map* section of the RID file.

Identifiers in RTI 1.3 are case-insensitive. For example, “receive” or “Receive” may be used to obtain the handle of the *receive* service category.

RETURN VALUES

A successful invocation of this service returns the handle associated with the named ordering service category.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NameNotFound

The symbolic name does not correspond to a handle of the requested type.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador

changeAttributeOrderType()

changeInteractionOrderType()

getOrderingName()

A.7.25 getOrderingName()

RTI 1.3

ABSTRACT

This service converts an RTI ordering service category handle to the associated symbolic (string) name defined in the RID file.

HLA IF SPECIFICATION

This method realizes the "Get Ordering Name" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.22 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

char *
RTI::RTIambassador::
   getOrderingName (
      OrderingHandle theHandle
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InvalidOrderingHandle,
  RTI::RTIinternalError
)
```

ARGUMENTS

theHandle

the handle of the ordering service category whose name is being queried

DESCRIPTION

This service converts an RTI handle associated with a ordering service category to its symbolic (string) identifier defined in the *transport_map* section of the RID file. These handles are used by various RTI services when referring to ordering categories.

The federation may define its own named ordering service categories as aliases for timestamp- or receive-order service. This technique may be used to decouple quality-of-service decisions for various types of federation data from the code of the constituent federates.

The RTI defines two ordering categories: *receive* and *timestamp.* These identifiers should always appear in the *ordering_map* section of the RID file.

The returned string is all lower-case.

RETURN VALUES

A successful invocation of this service returns the symbolic ordering service-category name corresponding to the specified ordering service-category handle. **The caller is responsible for freeing the memory associated with the returned string using the `delete[]` operator.**

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidTransportationHandle

The ordering handle does not correspond to a ordering service category defined in the RID file.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador

changeAttributeTransportType()

changeInteractionTransportType()

getOrderingHandle()

A.7.26 getParameterHandle()

RTI 1.0 RTI 1.3

ABSTRACT

This service converts a symbolic (string) parameter name and an interaction-class context to the RTI handle associated with the parameter. **The syntax and semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Get Parameter Handle" RTI Support & Operational Services as specified in the *HLA Interface Specification* (not explicitly listed in version 1.1; §10.8 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::ParameterHandle
RTI::RTIambassador::
   getParameterHandle (
       const RTI::ParameterName theName
       RTI::InteractionClassHandle whichClass
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::NameNotFound,
   RTI::RestoreInProgress,            ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                ← RTI 1.0 Only
)

// RTI 1.3 Only
RTI::ParameterHandle
RTI::RTIambassador::
   getParameterHandle (
       const char *theName,           ← Changes Types
       RTI::InteractionClassHandle whichClass
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InteractionClassNotDefined,
  RTI::NameNotFound,
  RTI::RTIinternalError
)
```

ARGUMENTS

theName

a string specifying the symbolic parameter name to be converted to an RTI-defined handle

whichClass

the interaction-class context of the specified parameter name

DESCRIPTION

This service converts a parameter *name* (i.e., the descriptive string identifier found in the FED file) to an RTI *handle* that is used by the various RTI services to refer to the parameter. The parameter name must refer to a parameter of the specified interaction class, i.e. the parameter must be natively defined in the interaction class or defined in a superclass of the interaction class.

Conceptually, a parameter handle is only valid when considered in the context of an interaction class. Implications of this include:

- the same physical value may refer to different parameters in the context of two different interaction classes (this is true of RTI 1.0 and RTI 1.3)
- different physical values may refer to the same parameter in the context of different interaction classes (this is not true of RTI 1.0 and RTI 1.3)

multiple distinct parameters may share the same name provided they may never occur in the context of the same interaction class (i.e., no interaction class containing an parameter name is a superclass of another interaction class containing the same parameter name)

The returned handle is valid throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

RETURN VALUES

A successful invocation of this service returns the RTI handle associated with the specified parameter.

RELEASE NOTES

RTI 1.0

Names in RTI 1.0 are case-sensitive.

RTI 1.3

Names in RTI 1.3 are case-insensitive.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

This method is safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The specified interaction class handle is not valid in the context of the current federation execution.

RTI::NameNotFound

The symbolic name does not correspond to a handle of the requested type.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::ParameterHandleSet
RTI::ParameterHandleValuePairSet
RTI::RTIambassador::

createFederationExecution()

getParameterName()

getInteractionClassHandle()

joinFederationExecution()

A.7.27 getParameterName()

RTI 1.0 RTI 1.3

ABSTRACT

This service converts an RTI parameter handle and its interaction-class context to the symbolic (string) name of the FED parameter they represent. **The syntax and semantics of this service have changed from RTI 1.0 to RTI 1.3.**

HLA IF SPECIFICATION

This method realizes the "Get Parameter Name" RTI Support & Operational service as specified in the *HLA Interface Specification* (not explicitly listed in version 1.1; §10.9 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

// RTI 1.0 Only
RTI::ParameterName
RTI::RTIambassador::
   getParameterName (
       RTI::ParameterHandle        theHandle
       RTI::InteractionClassHandle whichClass
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::FederateNotExecutionMember,
   RTI::InteractionClassNotDefined,
   RTI::InteractionParameterNotDefined,
   RTI::RestoreInProgress,             ← RTI 1.0 Only
   RTI::RTIinternalError,
   RTI::SaveInProgress                 ← RTI 1.0 Only
)

// RTI 1.3 Only
char *                                 ← Changes Types
RTI::RTIambassador::
   getParameterName (
       RTI::ParameterHandle        theHandle,
       RTI::InteractionClassHandle whichClass
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InteractionClassNotDefined,
  RTI::InteractionParameterNotDefined,
  RTI::RTIinternalError
);
```

ARGUMENTS

theHandle

a handle representing the parameter whose name is being queried

whichClass

the interaction-class context of the specified parameter handle

DESCRIPTION

This service converts an RTI parameter *handle* (i.e., a unique identifier used by the RTI to refer to an parameter), and its interaction-class context, to the symbolic (string) name of the parameter as found in the FED file.

The returned name is valid (i.e., guaranteed to be associated with the specified handle) throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

RETURN VALUES

A successful invocation of this service returns the symbolic parameter name corresponding to the specified parameter handle and interaction-class context. **The caller is responsible for freeing the memory associated with the returned string using the `delete[]` operator.**

RELEASE NOTES

RTI 1.0

The name returned is the parameter name exactly as it appears in the FED file, e.g. "ReportPeriod".

RTI 1.3

The name returned is the lower-case parameter name, e.g. "reportperiod".

EXCEPTIONS

RTI::ConcurrentAccessAttempted

These methods are safe for reentrance into the RTI ambassador; this exception is not thrown.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InteractionClassNotDefined

The specified interaction class handle is not valid in the context of the current Federation Execution.

RTI::InteractionParameterNotDefined

The specified parameter handle is not valid in the context of the specified interaction class.

RTI::RestoreInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "restore" operation.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SaveInProgress **(RTI 1.0 Only)**

The attempted action would result in a change in the internal state of the RTI, which is not permitted during a "save" operation.

SEE ALSO

RTI::ParameterHandleSet
RTI::ParameterHandleValuePairSet
RTI::RTIambassador::

createFederationExecution()

getParameterHandle()

getInteractionClassHandle()

getInteractionClassName()

joinFederationExecution()

A.7.28 getRegion()

RTI 1.3

ABSTRACT

This service converts an abstract region identifier (“token”) to the physical address of the corresponding region.

HLA IF SPECIFICATION

This method does not correspond to a service explicitly listed in the *HLA Interface Specification* version 1.3.

SYNOPSIS

```
#include <RTI.hh>

RTI::Region *
RTI::RTIambassador::
   getRegion(
       RTI::RegionToken token
   )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RegionNotKnown,
  RTI::RTIinternalError
)
```

ARGUMENTS

token

an abstract region identifier returned by the `getRegionToken()` service

DESCRIPTION

This service converts an abstract region identifier obtained from the `getRegionToken()` service to the physical address of the corresponding region object. This facility exists so that restoring federates may reconstitute references to region objects even though the physical location of the objects will differ from their locations in the saved LRC.

Note that a region token is no longer valid if the region is subsequently deleted using the `deleteRegion()` service.

RETURN VALUES

A successful invocation of this service returns the physical address of the region referenced by the abstract region identifier.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RegionNotKnown

The region token is not valid in the context of the current LRC state.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador

getRegionToken()

initiateFederateRestore()

initiateFederateSave()

A.7.29 getRegionToken()

RTI 1.3

ABSTRACT

This service converts a physical address associated with a region object to an abstract region identifier (“token”).

HLA IF SPECIFICATION

This method does not correspond to a service explicitly listed in the *HLA Interface Specification* version 1.3.

SYNOPSIS

```
#include <RTI.hh>

RTI::RegionToken
RTI::RTIambassador::
   getRegionToken(
       RTI::Region *region
   )
throw(
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RegionNotKnown,
  RTI::RTIinternalError
)
```

ARGUMENTS

region

the physical address of the region object whose token is being queried

DESCRIPTION

This service converts a physical address of a region object to an abstract region identifier (“token”). This token may be reconverted to a physical address using the `getRegion()` service. This facility exists to allow saving federates to marshal references to region objects that can be reconstituted by restoring federates even though the physical locations of the region objects will differ.

Note that a region token is no longer valid if the region is subsequently deleted using the `deleteRegion()` service.

RETURN VALUES

A successful invocation of this service returns a unique abstract region identifier (“token”) corresponding to the specified region.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador

getRegion ()

initiateFederateRestore()

initiateFederateSave()

A.7.30 getRoutingSpaceHandle()

RTI 1.3

ABSTRACT

This service converts a symbolic (string) routing space name to the RTI handle associated with the routing space.

HLA IF SPECIFICATION

This method realizes the “Get Routing Space Handle” RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.12 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::SpaceHandle
RTI::RTIambassador::
   getRoutingSpaceHandle (
       const char *theName
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::NameNotFound,
  RTI::RTIinternalError
)
```

ARGUMENTS

theName

a string specifying the symbolic routing space name to be converted to an RTI-defined handle

DESCRIPTION

This service converts a routing space *name* (i.e., the descriptive string identifier found in the FED file) to an RTI *handle* that is used by the various RTI services to refer to the routing space.

The returned handle is valid throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

Names in RTI 1.3 are case-insensitive.

RETURN VALUES

A successful invocation of this service returns the RTI handle associated with the specified routing space.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the Programmer’s Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NameNotFound

The symbolic name does not correspond to a handle of the requested type.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::Region

RTI::RTIambassador::

createFederationExecution()

getDimensionHandle()

getRoutingSpaceName()

joinFederationExecution()

A.7.31 getRoutingSpaceName()

RTI 1.3

ABSTRACT

This service converts an RTI routing space handle to the symbolic (string) name of the FED routing space it represents.

HLA IF SPECIFICATION

This method realizes the "Get Routing Space Name" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.13 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

char *
RTI::RTIambassador::
   getRoutingSpaceName (
       const RTI::SpaceHandle theHandle
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::RTIinternalError,
  RTI::SpaceNotDefined
)
```

ARGUMENTS

theHandle

a handle representing the routing space whose name is being queried

DESCRIPTION

This service converts an RTI routing space *handle* (i.e., a unique identifier used by the RTI to refer to an routing space) to the symbolic (string) name of the routing space as found in the FED file.

The returned name is valid (i.e., guaranteed to be associated with the specified handle) throughout the remainder of the current federation execution. RTI 1.0 and RTI 1.3 assign handles in a predictable fashion based on the layout of the FED file, but federates are discouraged from relying on this behavior.

The name returned is the lower-case routing space name, e.g. "geographicspace".

RETURN VALUES

A successful invocation of this service returns the symbolic class name corresponding to the specified class handle. **The caller is responsible for freeing the memory associated with the returned string using the `delete[]` operator.**

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SpaceNotDefined

The *SpaceHandle* argument does not correspond to a valid federation routing space. Use *RTIambassador::getRoutingSpaceHandle* to obtain valid space handles.

SEE ALSO

RTI::Region

RTI::RTIambassador::

createFederationExecution()

getDimensionName()

getRoutingSpaceHandle()

joinFederationExecution()

A.7.32 getTransportationHandle()

RTI 1.3

ABSTRACT

This service converts a symbolic (string) name of a transportation service category to an RTI handle.

HLA IF SPECIFICATION

This method realizes the "Get Transportation Handle" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.19 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

RTI::TransportationHandle
RTI::RTIambassador::
   getTransportationHandle (
      const char *theName
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::NameNotFound,
  RTI::RTIinternalError
)
```

ARGUMENTS

theName

symbolic name of a transportation category

DESCRIPTION

This service converts a symbolic (string) identifier to an RTI handle associated with a transportation service category defined in the *transport_map* section of the RID file. These handles are used by various RTI services when referring to transportation categories.

The federation may define its own named transportation service categories as aliases for reliable or best-effort service. This technique may be used to decouple quality-of-service decisions for various types of federation data from the code of the constituent federates.

The RTI defines two categories of reliable service, *reliable* and *state_consistent*, and one category of best-effort service, *best_effort*. These identifiers should always appear in the *transport_map* section of the RID file.

Identifiers in RTI 1.3 are case-insensitive. For example, "state_consistent" or "State_Consisent" may be used to obtain the handle of the *state_consistent* service category.

RETURN VALUES

A successful invocation of this service returns the handle associated with the named transportation service category.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::NameNotFound

The symbolic name does not correspond to a handle of the requested type.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador

changeAttributeTransportType()

changeInteractionTransportType()

getTransportationName()

A.7.33 getTransportationName()

RTI 1.3

ABSTRACT

This service converts an RTI transportation service category handle to the associated symbolic (string) name defined in the RID file.

HLA IF SPECIFICATION

This method realizes the "Get Transportation Name" RTI Support & Operational service as specified in the *HLA Interface Specification* (§10.20 in version 1.3).

SYNOPSIS

```
#include <RTI.hh>

char *
RTI::RTIambassador::
   getTransportationName (
       TransportationHandle theHandle
   )
throw (
  RTI::ConcurrentAccessAttempted,
  RTI::FederateNotExecutionMember,
  RTI::InvalidTransportationHandle,
  RTI::RTIinternalError
)
```

ARGUMENTS

theHandle

the handle of the transportation service category whose name is being queried

DESCRIPTION

This service converts an RTI handle associated with a transportation service category to its symbolic (string) identifier defined in the *transport_map* section of the RID file. These handles are used by various RTI services when referring to transportation categories.

The federation may define its own named transportation service categories as aliases for reliable or best-effort service. This technique may be used to decouple quality-of-service decisions for various types of federation data from the code of the constituent federates.

The RTI defines two categories of reliable service, *reliable* and *state_consistent*, and one category of best-effort service, *best_effort.* These identifiers should always appear in the *transport_map* section of the RID file.

The returned string is all lower-case.

RETURN VALUES

A successful invocation of this service returns the symbolic transportation service-category name corresponding to the specified transportation service-category handle. **The caller is responsible for freeing the memory associated with the returned string using the `delete[]` operator.**

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the *RTIambassador* has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See "Concurrent Access" in the Programmer's Guide.

RTI::FederateNotExecutionMember

The RTIambassador instance is not currently associated with a FedExec.

RTI::InvalidTransportationHandle

The transportation handle does not correspond to a transportation service category defined in the RID file.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador

changeAttributeTransportType()

changeInteractionTransportType()

getTransportationHandle()

A.7.34 RTIambassador()

RTI 1.0 RTI 1.3

ABSTRACT

This constructor initializes internal data structures of an RTI ambassador instance.

HLA IF SPECIFICATION

This constructor does not correspond to a service explicitly listed in the *HLA Interface Specification* version 1.3.

SYNOPSIS

```
#include <RTI.hh>

RTI::RTIambassador::
   RTIambassador( )
throw (
  RTI::MemoryExhausted,
  RTI::RTIinternalError
);
```

DESCRIPTION

This constructor is implicitly invoked when an RTI ambassador instance is declared on the stack or allocated on the heap using the `new` operator. The constructor prepares the RTI ambassador instance for use by initializing any data structures and network connections that are independent of a particular federation execution. To access the majority of RTI ambassador services, a newly constructed instance should be associated with a particular federation using `joinFederationExecution()`.

For an RTI ambassador to be successfully constructed, RTI environment variables such as *RTI_CONFIG* must be set correctly and global RTI processes such as the RtiExec should be reachable from the machine on which the federate is running.

RETURN VALUES

A non-exceptional return indicates that the RTI ambassador instance has been successfully initialized.

EXCEPTIONS

RTI::MemoryExhausted

Not enough system resources are available to construct a *RTIambassador* instance.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

SEE ALSO

RTI::RTIambassador::

~RTIambassador()

joinFederationExecution()

A.7.35 tick()

RTI 1.0 RTI 1.3

ABSTRACT

This service is invoked by the federate to yield processor time to the LRC. During a `tick()` invocation, the LRC will process incoming traffic, deliver callbacks to the federate, and perform various internal RTI maintenance essential to the operation of the federation. **The semantics of `tick()` have changed substantially between RTI 1.0 and RTI 1.3.**

HLA IF SPECIFICATION

This service does not correspond directly to a service specified in the *HLA Interface Specification* version 1.1 or version 1.3.

SYNOPSIS

```
#include <RTI.hh>

RTI::Boolean
RTI::RTIambassador::
   tick ( )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::RTIinternalError,
   RTI::SpecifiedSaveLabelDoesNotExist
)

RTI::Boolean
RTI::RTIambassador::
   tick (
       RTI::TickTime minimum
       RTI::TickTime maximum
   )
throw (
   RTI::ConcurrentAccessAttempted,
   RTI::RTIinternalError,
   RTI::SpecifiedSaveLabelDoesNotExist
)
```

ARGUMENTS

minimum

the minimum time interval (in wall clock seconds) to spend in `tick()`

maximum

the maximum time interval (in wall clock seconds) after which `tick()` will not begin execution of any additional processing

DESCRIPTION

The `tick()` service temporarily passes execution control from the federate to the LRC. The LRC will perform periodic federation maintenance (e.g., sending federate "heartbeats" and updating and servicing MOM objects and interactions) and process incoming traffic from the network.

It is essential that all federates invoke `tick()` frequently so that internal RTI communications may be serviced in a timely fashion. Federates should use a timed tick in lieu of a `sleep` or other system call designed to suspend execution for a specified wall clock time interval.

If a federate is time-constrained, time-stamp ordered events (i.e., updates, interactions, deletions, and saves) will only be delivered by `tick()` in accordance with an in-progress time-advancement service. If asynchronous delivery of receive-ordered events is not enabled, receive-ordered events will only be delivered when a time-advancement service is in progress. If asynchronous delivery of receive-ordered events is enabled, receive-ordered events may be delivered during any invocation of `tick()`.

Any callbacks that are not subject to time-stamp ordering (i.e., anything except for updates, interactions, deletions, and saves) may be delivered to a federate during any invocation of `tick()`, regardless of whether a time-advancement service is in progress.

In general, no federate-ambassador callbacks will be invoked except during an invocation of `tick()`. Exceptions to this rule are noted in the individual `FederateAmbassador` service descriptions.

RTI ambassador functions, with the exception of a handful of reentrant operational and support functions, may not be invoked from within callbacks made during a `tick()` invocation; such an attempt will result in a `ConcurrentAccessAttempted` exception.

LRC processing is broken into a number of atomic operations that must be completely executed within a single invocation of `tick()`. For this reason, it is not always possible to interrupt `tick()` execution after precisely the specified maximum time interval. In these cases, `tick()` will return immediately after the completion of the atomic processing unit that was in progress at the end of the specified maximum time interval. As such, the maximum-time argument to `tick()` should not be used to enforce extremely precise timing constraints.

RELEASE NOTES

RTI 1.0

The zero-argument variant, known as an *instantaneous tick*, generally results in one federate-ambassador callback being invoked for the federate. If there are no callbacks ready to be delivered, an instantaneous tick invocation returns immediately. Some sets of closely related callbacks, such as an object-discovery callback and an initial reflection for the discovered object, may all occur during a single invocation of an instantaneous tick.

The two-argument variant, known as a *timed tick*, may result in a number of federate-ambassador callbacks being invoked for the federate. A timed-tick invocation will continue to process traffic and invoke callbacks until the amount of wall clock time specified by the *maximum* argument has elapsed. If a timed-tick invocation runs out of processing to do, it will block waiting for incoming traffic until the amount of wall clock time specified by the *minimum* argument has elapsed, as measured from the start of the `tick()` invocation. If there is no pending processing to be done and the wall clock interval specified by *minimum* has elapsed, the timed tick invocation will return.

The `dequeueFIFOasynchronously()` service is used to toggle the delivery of receive-ordered events in the absence of an in-progress time-advancement service.

If a federate is not time-constrained, receive-ordered events will not be delivered in the absence of time-advancement services unless asynchronous delivery is enabled. It is recommended that non-time-constrained federates enable asynchronous delivery of receive-ordered events.

The `tick()` service should only be invoked from the same thread in which the RTI ambassador was instantiated.

RTI 1.3

The zero-argument variant of tick reads all available network traffic, then does as much processing as possible without blocking for additional network communications. This may result in many federate-ambassador notifications being delivered to the federate. The federate should not rely on the zero-argument variant of `tick()` completing execution in any particular interval of time.

The two-argument variant of `tick()` reads all available

network traffic and does as much processing as possible without blocking or exceeding the specified maximum time interval. If the specified minimum time interval has not elapsed after all available processing has been done, the LRC will pause until the minimum time interval has elapsed, then return immediately. While paused, the LRC will continue to perform further processing if additional network traffic subsequently arrives at the LRC within the minimum time interval.

The `enableAsynchronousDelivery()` and `disableAsynchronousDelivery()` services are used to toggle the delivery of receive-ordered events in the absence of an in-progress time-advancement service on and off, respectively.

If a federate is not time-constrained, any events may be delivered during any invocation of `tick()`, regardless of whether or not asynchronous delivery is enabled.

RETURN VALUES

RTI 1.0

A return value of RTI::RTI_TRUE indicates that the LRC still has processing that is pending execution. Immediately invoking `tick()` again may be in order. A return value of RTI::RTI_FALSE indicates that the LRC has performed all processing that may be done based on traffic that has currently been received from the network.

A return from a timed tick indicates that the wall clock time interval specified by *maximum* has elapsed, or that the wall clock time interval specified by *minimum* has elapsed and the LRC has no further processing that may be done immediately.

RTI 1.3

The zero-argument always returns *RTI::RTI_FALSE*.

The two-argument variant returns *RTI::RTI_TRUE* if processing was interrupted because of the maximum time interval being exceeded; otherwise, it returns *RTI::RTI_FALSE*.

EXCEPTIONS

RTI::ConcurrentAccessAttempted

An illegal attempt to reenter the RTIambassador has been detected. Typically: A *FederateAmbassador* callback method has called an *RTIambassador* method. See “Concurrent Access” in the programmer’s Guide.

RTI::RTIinternalError

An RTI internal error has occurred. Consult the Federate log file for more details.

RTI::SpecifiedSaveLabelDoesNotExist

This exception is not thrown by the current implementations.

SEE ALSO

RTI 1.0

RTI::FederateAmbassador

RTI::RTIambassador::

dequeueFIFOasynchronously()

enableAsynchronousDelivery()

enableTimeConstrained()

joinFederationExecution()

timeAdvanceRequest

RTI 1.3

RTI::FederateAmbassador

RTI::RTIambassador::