MÄK RTI 2.1 Release Notes

This document provides the following release-specific information for MÄK RTI 2.1:

Document ID: RTI-2.1-2-031121

Overview

MÄK RTI 2.1 introduces partial support for the RTI 1516 specification. It continues to fully support the RTI 1.3 specification. However, since we now support multiple RTI specifications, we have dropped the "-ngc" suffix from our version numbering.

The MÄK RTI has been verified by DMSO as fully compliant with the HLA Interface Specification, version 1.3. All services are implemented.

Supported Systems

Table 1 lists the platforms supported by the MÄK RTI 2.1. Please contact MÄK if you need it for another platform. Application code must be built with the indicated compilers in order to link to the MÄK RTI libraries.

Table 1: Platforms supported

Platform	Compiler
SGI IRIX6.5	MipsPro7.3 C++ compiler (available from SGI)
Sun Sparcstation with Solaris2.7 or later	SPARCompiler C++ version 5.2 (available from Sun)
PC with Red Hat Linux 8.0	GNU C++ (g++) distributed with Red Hat distribution. gcc version 3.2
PC with Windows NT 4.0/2000/XP	Microsoft Visual C++ 6.0 Microsoft .NET

MÄK RTI 2.1 was built with VR-Link 3.9.

Qt Release Compatibility

The MÄK RTIspy GUI uses Trolltech's Qt cross-platform GUI toolkit. In particular, the GUI included with MÄK RTI 2.1 relies on Qt version 3.1.2. (The Qt run-time libraries are distributed with the MÄK RTI). If a federate also uses Qt, it must be sure to use the same version of Qt as the MÄK RTIspy GUI, in order for the RTIspy GUI to function properly for that federate. Federates built with other versions of Qt should disable the RTIspy GUI to avoid these incompatibility problems.

Backwards Compatibility

The MÄK RTI 2.1 libraries are not run-time compatible with previous versions. (In other words, you cannot have a federation execution in which some federates use MÄK RTI 2.1 and some use a previous version of the MÄK RTI.) However, the MÄK RTI 2.1 is link compatible with previous versions, so a federate does not need to be recompiled to use this version. We recommend that all federates use the same version of the RTI when operating within the same federation (or using the same rtiexec).

Network Compatibility

The MÄK RTI is not network compatible with the DMSO RTI. (In general, no two RTI implementations are network compatible, due to differences in the low-level network mechanisms that they use (which include, but are not limited to packet layout.)) Applications that want to interoperate in the same federation execution must all use the same RTI implementation. A federation can easily switch from one RTI implementation to another between runs, but during each run, all participants must agree on which RTI to use, much as they must also agree on which FOM to use.

Compatibility with the DMSO RTI

With the exception of several additional functions (such as, DtReadFileDescriptor(), described in MÄK RTI documentation), the MÄK RTI 2.1 has the exact same API as the DMSO RTI 1.3 NG (except v5). Unless you are using the RtReadFileDescriptor function, an application built against the DMSO RTI NG does not need to be recompiled in order to be used with the MÄK RTI 2.1, or vice versa (except that the Linux version is not compatible with the DMSO RTI).

MÄK RTI 2.1 uses the same FED file format as the DMSO RTI, but a different RID file format. (RID file formats depend on the RTI-implementation.)

MÄK products, including the MÄK RTI 2.1, are not run-time compatible with the DMSO RTI NG v5 due to changes in the how the DMSO RTI handles namespaces.

New Features and Updates

MÄK RTI 2.1 has the following new features and updates:

- Partial support for the RTI 1516 specification (see "Support for the RTI 1516 Specification," on page 1-10)
- Updated RTIspy Plug-in API (see "Updates to the RTIspy Plug-in API," on page 1-30)
- Updated rtiexec GUI including:
  - Time Management display
  - Display of host network address
- Updated RTIspy LRC GUI
- Support for FDD files
- Support for distributing FED or FDD files
- Support for running the rtiexec as a daemon (UNIX only)
- New configuration options.

Updated rtiexec GUI

The rtiexec GUI has been reorganized and has one new feature. The changes are as follows:

- The Quit button has been removed. To exit, choose File → Exit.
- The configuration information previously displayed at the top of the window is now displayed in its own window. To view it, choose Display → Configuration.
- The optional display of console output is no longer displayed at the bottom of the rtiexec window. To view console output, choose Display → Console Log.
- You can write the console log out to a file, *RtiExecLog.txt*. Check the Log to File check box on the Console Log window.
- The option to display synchronization points is now a check box. If you check it, the window expands to show the additional information, just like it did previously.

The Time Management Display

The rtiexec GUI displays Time Management state information for each federate in the federate list view.

➤ To enable or disable the Time Management display, check or clear the Show Time State check box.

Enabling the Time Management display adds some overhead to RTI performance. Therefore, if you do not need the information, you should not enable this feature.

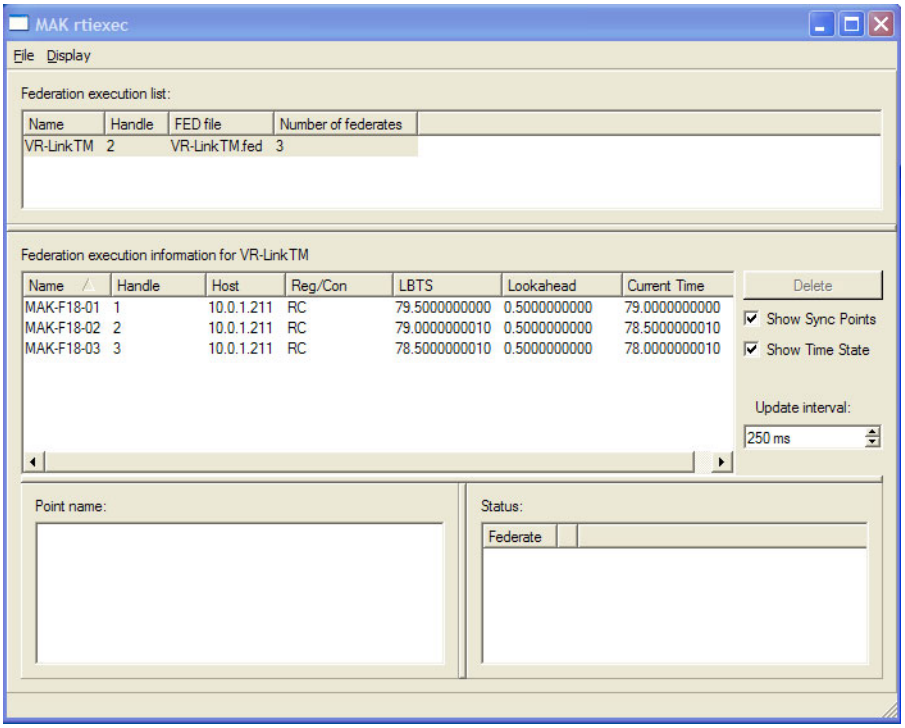
When the Time Management display is enabled, the federate list view displays the following information:

- Reg/Con – Is the federate regulating (R), constrained (C), both (RC), or neither?
- LBTS – The Lower Bound Time Stamp on messages that the federate may generate. This view of LBTS is the time to which a regulating federate constrains all other constrained federates, rather than the minimum LBTS by which the federate is itself constrained. When the federates are sorted by the LBTS value, the minimum LBTS value indicates the LBTS of messages that any federate may receive (except the smallest federate that uses the next smallest value.)
- Lookahead – The federate's current lookahead setting.
- Current Time – The federate's current view of simulation time. This is the time of the federate's last Time Advance Grant, and is unaffected when the federate enters the Time Advancing state.

Since Time Management state may progress very rapidly (often faster than is human readable), you can specify how frequently the display is updated. A reasonable update interval also helps keep the GUI refresh from slowing down the rtiexec. In general, the default update interval (250 milliseconds) should provide a clear view of how simulation time is advancing without consuming excessive resources.

The Show Time State check box has been added. If you select it, the window displays the update interval. You can change the update interval by entering a value in the text box. The following illustration shows the rtiexec window with synchronization points and the time state enabled.

The MÄK RTI 1516 does not support Time Management. Therefore, if you turn on this feature in the 1516 rtiexec GUI, nothing is displayed.



Updated RTIspy GUI

The RTIspy LRC GUI has been enhanced. It now includes the RTI Network Monitoring window.

The MÄK RTI 1516 does not support the LRC console plug-in. Therefore, you cannot run the RTIspy LRC GUI with federates built with the 1516 version of the RTI. However, since 1.3 federates and 1516 federates can interoperate in a 1516 federation execution, you can run the RTIspy LRC GUI with a 1.3 federate and observe some of the data in a 1516 federation execution.

The RTI Network Monitoring Window

To open the RTI Network Monitoring window, choose View → Display Network Statistics.

The Network Monitoring window has three tabs:

- Message Types – displays summary statistics about the messages sent and their size.
- Objects and Interactions – lists the objects and interactions. You can highlight the objects and interactions that are registered or discovered. You can display a graph of the objects and interactions over time.
- RTI API Call Statistics – displays the RTI Ambassador and RTI Federate calls being made and displays CPU use.

For all three tabs, you can change the plot range and refresh interval. You can also record the data to a log file. For complete documentation, see "The RTIspy Local RTI Component (LRC) GUI," on page 1-16.

Support for FOM Document Data (FDD) Format

The HLA 1.3 and HLA 1516 versions of the MÄK RTI each support both the 1516 FDD (FOM Document Data) and 1.3 FED (federation execution data) file formats. FDD files must have a file extension of either *.xml* or *.fdd*. FED files are expected to have an extension of *.fed* or *.mtl*; however, any other extensions are assumed to be in the FED format.

VR-Link 3.9 is distributed with an expanded version of the RPR 2 draft 14 FDD file (*VR-Link20014.xml*) as well as the FED file version (*VR-Link20014.fed*).

Distributing FDD and FED Files

You can distribute the FED file or FDD file from the federate that creates the federation to all joining federates instead of reading a local copy of the file at each federate. When the first federate comes online and creates the federation, it passes the FED file to the RTI Execution. As additional federates join, the RTI Execution passes the file to them along with their join confirmation.

This feature is enabled with the RID file parameter **RTI_distributeFedFile**. However, the federation must also enable **RTI_useRtiExec** and **RTI_internalMsgReliable**.

The primary reason to enable FED file distribution is to ensure that the FOM used by each federate is consistent while you are developing a federation. During federation development, the FOM may be in a state of flux. If different federates attempt to join the federation using different revisions of the FOM, unexpected and difficult to diagnose problems may arise.

However, distributing the FED file is a relatively expensive network operation, which may slow the join process and, in the case of late joining federates, may even impede the performance of a running federation. Therefore, we recommend that you use either FED file distribution or statically distributed files in the federation development phase, and use only statically distributed FDD or FED files when the simulation is deployed.

Any federate that attempts to create a federation must have a local copy of the FED file since, for the MÄK RTI, the createFederationExecution API routine holds the precondition that a valid FED file exists.

Running the rtiexec as A Daemon

On UNIX systems, it may be desirable to run the rtiexec in the background with no controlling console. This allows the rtiexec to be run from a script (for example at machine startup).

To enable daemon mode start the rtiexec with the `-D` startup option.

New Configuration Options

MÄK RTI 2.1 has the following new RID file parameters:

- RTI_enableNetworkMonitoring
- RTI_logNetworkMonitoringStatistics
- RTI_enableBigMessage
- RTI_distributeFedFile (see "Distributing FDD and FED Files," on page 1-7.)

Enabling Network Monitoring

The RTI_enableNetworkMonitoring parameter enables or disables the RTIspy network monitoring application. Network monitoring is enabled by default. The RTIspy LRC Console plug-in must also be loaded and enabled to enable network monitoring.

Enabling network monitoring may affect the performance of the RTI. For more information, see "The RTI Network Monitoring Window," on page 1-6.

Logging Network Monitoring Statistics

The RTI_logNetworkMonitoringStatistics parameter sets the initial state of network statistic file logging for the network monitoring application. It is disabled by default. The RTIspy network monitoring application must be enabled for this parameter to be relevant.

Logging network statistics imposes a performance penalty above and beyond basic network monitoring. Log files may grow quickly in large federations since each RTI operation is monitored. You can enable and disable logging dynamically with the Record button in the RTIspy GUI to record snapshots of critical sections of federation execution. For more information, see "The Network Statistics Log Files," on page 1-30.

Enabling Support for Big Messages

The **RTI_enableBigMessage** parameter enables or disables support for the size of attributes and parameters and resulting update and interaction messages to exceed 64 KB. The default message format uses 16-bit integers (max 64 KB) to represent the size of elements in handle value pair sets and the overall message size. By enabling this option, the message format is modified to use 32-bit integers for each value. In addition, the underlying reliable connection is modified to support the resulting larger messages (affects all messages sent reliably). This feature relies on the fact that the reliable connection is implemented on top of TCP that handles message fragmentation.

While the message format does not rely on the underlying transport type, fragmentation of messages is not supported for best effort (uses UDP). Therefore, large attributes and parameters (or a messsage where the sum of parameters is large) must be sent reliable.

Documentation Updates

The printed version of *MÄK RTIspy Reference Manual* states that the MÄK RTI comes with, or is included with, all MÄK products. These statements mean that the MÄK RTI software is on product CDs. It does not mean that purchase of a MÄK product includes a MÄK RTI license. The MÄK RTI is licensed separately from other MÄK products.

Fixed Bugs

This release fixes the following problems in previous releases:

- When a federate terminates abnormally and fault tolerance is enabled, its Manager.Federate object is now removed from other federates.
- The federation time implementation has changed the value of epsilon from 0.000001 to 0.000000001.
- When the RTI calls the Federate Ambassador function initiateFederateRestore, it now returns the Federate Handle as expected (rather than the FedEx Handle).

Known Problems

This release has the following known problems:

- The RTI can sometimes fail to throw an "ObjectAlreadyRegistered" exception if a federate tries to register an object with a name that is already being used by another object. The federate can register an object using a name that belongs to another object that is unknown to the local RTI component (LRC). The LRC only maintains the names of objects that are registered or discovered by the federate. While we plan to fix this problem, a good preventive measure is to always subscribe and tick the RTI before registering objects.
- If you are using the Linux version of the MÄK RTIspy with versions of MÄK products earlier than the most recent releases, the MÄK products may crash. If this happens, contact MÄK technical support and we will provide you with a fix for your application.
- If you are using Microsoft Visual C++ 6.0 to build with MÄK RTI 1516, you must install a patch. See "Patching the Microsoft Visual C++ 6.0 Compiler," on page 1-14.

Support for the RTI 1516 Specification

The MÄK RTI 1516 is a partial implementation of the HLA 1516 interface specification. The basic set of services for most non-time managed federations are provided, including:

- Create/destroy/join/resign federation
- (Un)publish/(un)subscribe object/interaction class (attributes)
- Register/discover/delete/remove object instance
- Update/reflect attribute, and send/receive interaction.

The services that are not implemented are:

- Federation save/restore
- Ownership management
- Time management
- DDM
- MOM.

List of supported Services

Table 2 lists all RTI 1516 services supported by MÄK RTI 2.1. The † symbol indicates a federate ambassador call (versus an RTI ambassador call).

Table 2: Supported RTI 1516 services

Category	Service
Federation Management	◆ Create Federation Execution ◆ Destroy Federation Execution ◆ Join Federation Execution ◆ Resign Federation Execution ◆ Register Federation Synchronization Point ◆ Confirm Synchronization Point Registration† ◆ Announce Synchronization Point † ◆ Synchronization Point Achieved ◆ Federation Synchronized †
Declaration Management	◆ Publish Object Class Attributes ◆ Unpublish Object Class Attributes ◆ Publish Interaction Class ◆ Unpublish Interaction Class ◆ Subscribe Object Class Attributes ◆ Unsubscribe Object Class Attributes ◆ Subscribe Interaction Class ◆ Unsubscribe Interaction Class ◆ Start Registration for Object Class † ◆ Stop Registration for Object Class † ◆ Turn Interactions On † ◆ Turn Interactions Off †
Object Management	◆ Reserve Object Instance Name ◆ Object Instance Name Reserved † ◆ Register Object Instance ◆ Discover Object Instance † ◆ Update Attribute Values ◆ Reflect Attribute Values † ◆ Send Interaction ◆ Receive Interaction † ◆ Delete Object Instance ◆ Remove Object Instance † ◆ Local Delete Object Instance ◆ Change Attribute Transportation Type ◆ Change Interaction Transportation Type ◆ Request Attribute Value Update ◆ Provide Attribute Value Update † ◆ Turn Updates On For Object Instance † ◆ Turn Updates Off For Object Instance †

Table 2: Supported RTI 1516 services

Category	Service
Support Services	◆ Get Object Class Handle ◆ Get Object Class Name ◆ Get Attribute Handle ◆ Get Attribute Name ◆ Get Interaction Class Handle ◆ Get Interaction Class Name ◆ Get Parameter Handle ◆ Get Parameter Name ◆ Get Object Instance Handle ◆ Get Object Instance Name ◆ Get Known Object Class Handle ◆ Get Transportation Type ◆ Get Transportation Name ◆ Get Order Type ◆ Get Order Name ◆ Enable Object Class Relevance Advisory Switch ◆ Disable Object Class Relevance Advisory Switch ◆ Enable Attribute Relevance Advisory Switch ◆ Disable Attribute Relevance Advisory Switch ◆ Enable Interaction Relevance Advisory Switch ◆ Disable Interaction Relevance Advisory Switch ◆ Initialize RTI ◆ Finalize RTI ◆ Evoke Callback ◆ Evoke Multiple Callbacks ◆ Enable Callbacks ◆ Disable Callbacks

RTI 1516 and RTI 1.3 Interoperability

The MÄK RTI 1516 implementation uses the same code base as the MÄK RTI1.3 implementation. As a result, the two implementations share the same on-the-wire protocol for exchanging internal RTI information. This shared protocol allows a federate using the MÄK RTI 1516 to interoperate with a federate using the MÄK RTI 1.3. The following sections provide more details on using the MÄK RTI 1.3 and MÄK RTI 1516 together.

The 1516 API

The HLA 1516 interface specification is essentially a refinement (through the IEEE standards balloting process) of the HLA 1.3 draft interface specification. There are very few substantive differences between the two interface specifications with the exception of DDM and the MOM. Even the differences between these services still provide the same functionality, just in a different way. Listing the specific differences between the two interface specifications is beyond the scope of this document[1]. However, one significant difference between the two interface specifications is the C++ application programmer interface (API).

The 1516 interface specification introduced an API that was strongly based on the use of the C++ templates and the standard template library (STL). This API is substantially different from the 1.3 API. In its early attempts to develop a 1516 RTI, DMSO discovered that the 1516 specification could not be implemented. The SISO Dynamic Link Compatibility HLA API Product Development Group (HLA API PDG, http://www.sisostds.org/index.cfm) has been working to fix the API so that it could be implementable and support link compatibility. As part of this process, the HLA API PDG migrated the API away from the use of templates for defining the API classes and towards simpler C++ declarations. The main differences are:

- The use of the "pImple" idiom for handles (decouples the class interfaces from their private member declarations)
- The use of simple enumerations (which are strongly typed in C++)
- Passing parameters by reference instead of auto_ptr (except for factories).

The API still takes advantage of STL map, set, and auto_ptr where appropriate. You can view the work of the HLA API PDG at http://65.223.150.15/ and http://confs.itcenter.org/listprocedu/index.cfm?list=SAC-PDG-HLAAPI.

The MÄK RTI 1516 was implemented using the SISO HLA API PDG draft 1516 API. By implementing against this 1516 API, we are promoting link compatibility with other RTI vendors, which is not possible with the original IEEE 1516 API. The IEEE 1516 API only allowed compile time compatibility (especially true with the use of templates). However, link compatibility is important because it allows federates built using one RTI to execute with the local RTI component (LRC) library and central RTI component (CRC) executable of a different RTI without having to recompile the federate.

Using the MÄK RTI 1516

Since both the MÄK RTI 1.3 and MÄK RTI 1516 share the same code base, many of the procedures for using the MÄK RTI 1516 are nearly identical to the MÄK RTI 1.3. This section describes the differences in these procedures. For a complete description of the procedures, refer to *MÄK RTI Reference Manual*.

1. For details, see the 1.3 interface specification at https://www.dmso.mil/public/transition/hla/techspecs; see the 1516 interface specification http://www.ieee.org.

Compiling and Linking

The procedures for compiling and linking with the MÄK RTI 1516 are very similar to what is done with the MÄK RTI 1.3. However, the MÄK RTI 1516 library names are named differently.

For UNIX

- *librti1516.a* and *.so* and *libfedtime1516.a* and *.so.*

For Windows:

- *librti1516.lib* and *.dll* and *libfedtime1516.lib* and *.dll.*

To build an application that uses the MÄK RTI 1516:

1. Include RTI1516.h in any source files that use RTI functionality.
2. Add the MÄK RTI 1516 *./include* and *./lib* directories to the appropriate search paths.
3. Link with the MÄK RTI 1516 libraries.

For example, on UNIX, if you have installed the MÄK RTI in */usr/products/makRtix.x*, the following should appear on your compile line:

```
-I/usr/products/makRtix.x/include
```

and the following should appear on your link line:

```
-L/usr/products/makRtix.x/lib -lrti1516 -lfedtime1516
```

On Windows:

1. If the MÄK RTI is installed in *c:\MAK\makRtix.x*, add *c:\MAK\makRtix.x\include* to the list of "additional include directories" on the C/C++ tab of your project settings.
2. On the Link tab of your project settings:
   a. Add *c:\MAK\makRtix.x\lib* to the "additional library path".
   b. Add *librti1516.lib* and *libfedtime1516.lib* to the list of Object/library modules.

Patching the Microsoft Visual C++ 6.0 Compiler

If you use Microsoft Visual C++ version 6.0, you must install a patch to a standard header file, in order to successfully build applications against the IEEE 1516 header files. The patch comes from Dinkumware, Ltd (the company that developed the Standard C++ Library header files for Microsoft), but for convenience, we are distributing it with MÄK RTI releases.

The patch fixes a bug that would cause applications to crash if they pass std::sets or std::maps from an application to a DLL or vice-versa, something that IEEE 1516 federates must do. For details about the patch go to http://www.dinkumware.com/vc_fixes.html.

To install the patch:

1. In the top level MÄK RTI directory, is a file called *XTREE-msvc++patch*. Rename this file *XTREE*.
2. Make a back-up version of the original *XTREE*.
3. In the Visual Studio *include* directory (for example, *C:\Program Files\Microsoft Visual Studio\VC98\Include*), replace the original version of *XTREE* with the patched version.

Running Federates with the MÄK RTI 1516

Execution with the MÄK RTI 1516 follows the same procedures as the MÄK RTI 1.3. The MÄK RTI 1516 uses different file names. The LRC shared library is *librti1516* (loaded automatically by the federate if compiled with 1516). The CRC is named rtiexec1516; this executable should be used in place of the 1.3 rtiexec. The license requirements are the same and the RID file configuration is the same. The FDD or FED file must be accessible to the federate unless the centralized distribution is enabled in the RID. A federate that creates the federation must have access to the FDD or FED regardless of the centralized distribution feature.

The HLA 1.3 and HLA 1516 versions of the MÄK RTI each support both the 1516 FDD (FOM Document Data) and 1.3 FED (federation execution data) file formats. That is, you can use a 1516-style XML-based FDD file even if your federate was built using the 1.3 API, and you can use the 1.3-style FED file, even if your federate uses the 1516 API.

Running Federates with MÄK RTI 1.3 and MÄK RTI 1516

The MÄK RTI 1516 shares the same on-the-wire protocol as the MÄK RTI 1.3. As a result, a federate that uses the MÄK RTI 1516 can interopeate with a federate that uses the MÄK RTI 1.3. Both the libRTI-NG and the librti1516 shared libraries are in the MÄK RTI distribution and a federate should load the appropriate one based on how it was compiled and linked. However, you must use the rtiexec1516 if one or more federates is built for 1516.

1516 Implementation Issues

The MÄK RTI 1516 has the following known issues:

- While the 1516 API uses the std:wstring to exchange string information, internally, the MÄK RTI 1516 operates on narrow strings only (that is, representable by std::string). Therefore, the contents of any wide string passed through the RTI API must be convertible to a narrow string.
- The federate ambassador synchronizationPointRegistrationFailed service call always returns SYNCHRONIZATION_POINT_LABEL_NOT_UNIQUE. This default reason is returned because the current implementation does not exchange the reason between the federation execution and the LRC.
- The publish and subscribe services in the 1516 interface specification are additive when successive calls are made, versus replacement for the 1.3 interface specification. The MÄK RTI 1516 implementation follows the additive policy; however, the MÄK RTI 1.3 follows the replacement policy. As a result, the advisories and discoveries may conflict when a federation is composed of both 1.3 and 1516 federates. The RTI_processUnknownUpdatesForDiscovery RID parameter should be enabled as a work around for discoveries. This issue will be resolved in subsequent releases to apply the appropriate policy to publish and subscribe operations.

The RTIspy Local RTI Component (LRC) GUI

The following sections replace and augment the RTIspy LRC GUI documentation starting with section 4.3.4 in *MÄK RTI Reference Manual*.

Viewing Objects and Interactions

The RTIspy main window (Figure 1) opens automatically for each federate. It provides a view of the objects and interactions sent and received for that federate. The title bar displays the name of the LRC, so that you can distinguish the windows for different federates.

The initial view of the Objects tab does not show any objects selected. As soon as you select an object, the Object Summary is displayed.

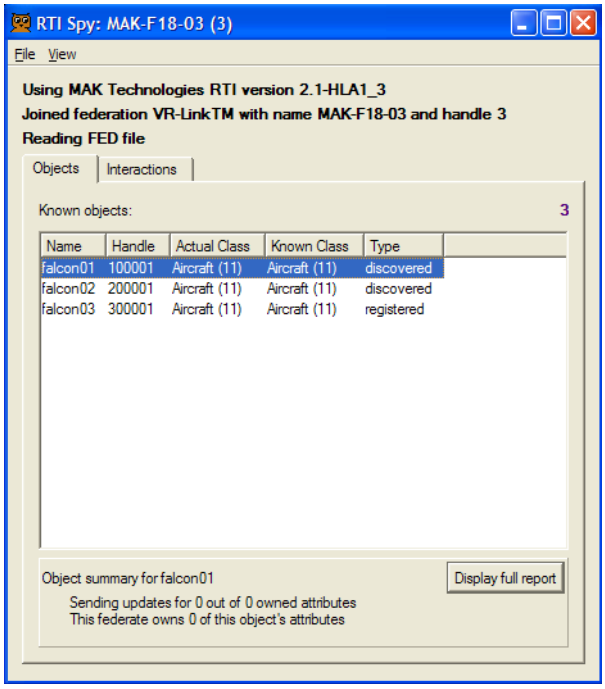


Figure 1. RTIspy main window, Object tab

The Interactions tab (Figure 2) displays a list of the interactions the federate has sent and the interactions it has received.

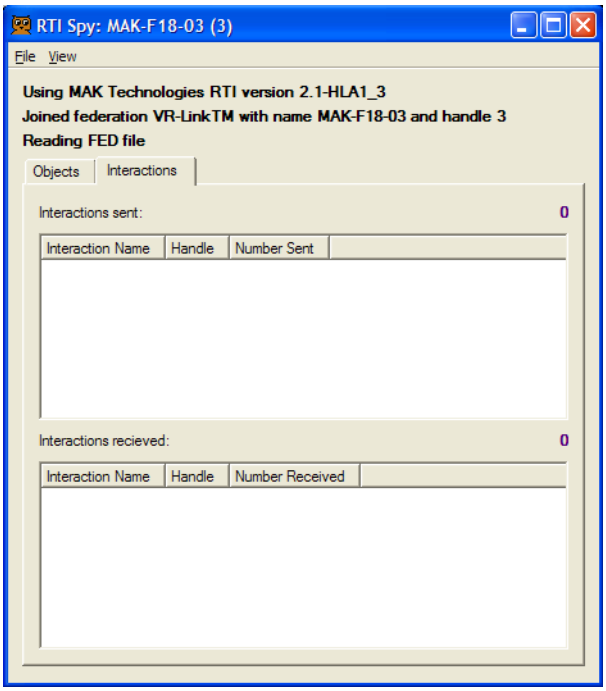


Figure 2. Main window, Interactions tab

Viewing Object Attributes

The Object Attributes window (Figure 3) shows you which attributes are being updated for the selected object. This is particularly useful when you are working with ownership management. The window also shows ownership information.

To open the Object Attributes window:

1. In the main window, select an object in the Objects list. The window expands to show summary information for the object (Figure 1).
2. Click the Display Full Report button. The Object Attributes window is displayed (Figure 3).

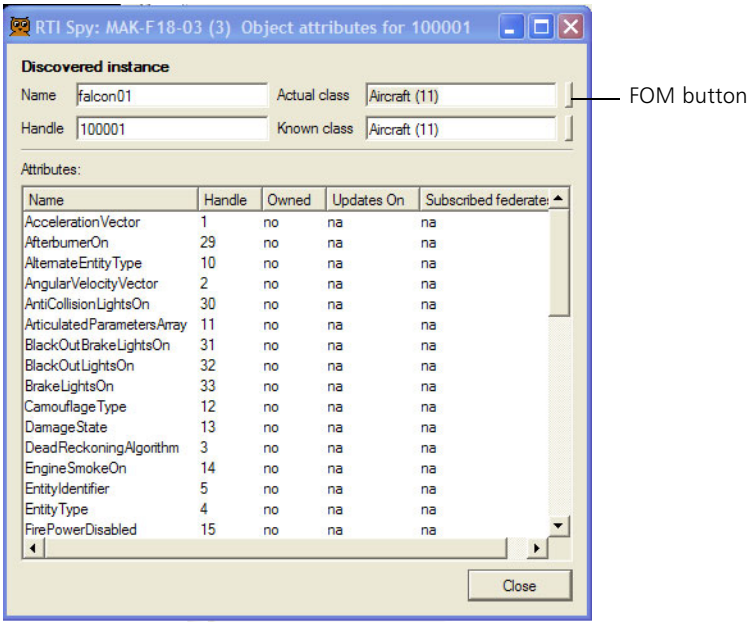


Figure 3. Object attributes window

You can open as many Object Attribute windows as you want. Keep selecting objects in the main window and clicking the Display Full Report button. (Or, just double-click the object name in the list.)

RTIspy closes Object Attribute windows automatically if the object is removed from the simulation.

The FOM button lets you toggle between display of an object's leaf class name and its full FOM name.

Viewing RTI Messages

The RTIspy Service Log (Figure 4) displays a running log of HLA messages.

- To open the Service Log window, on the RTIspy main window, choose View → Display Log.
- To write the service log to a file, check the Log To File check box. The log is written to *LrcConsoleServiceLog.txt*.

The Invoked By column indicates whether a message was invoked by a federate or the RTI. RTI-invoked messages are printed in purple to help distinguish them without the need to view the Invoked By column.

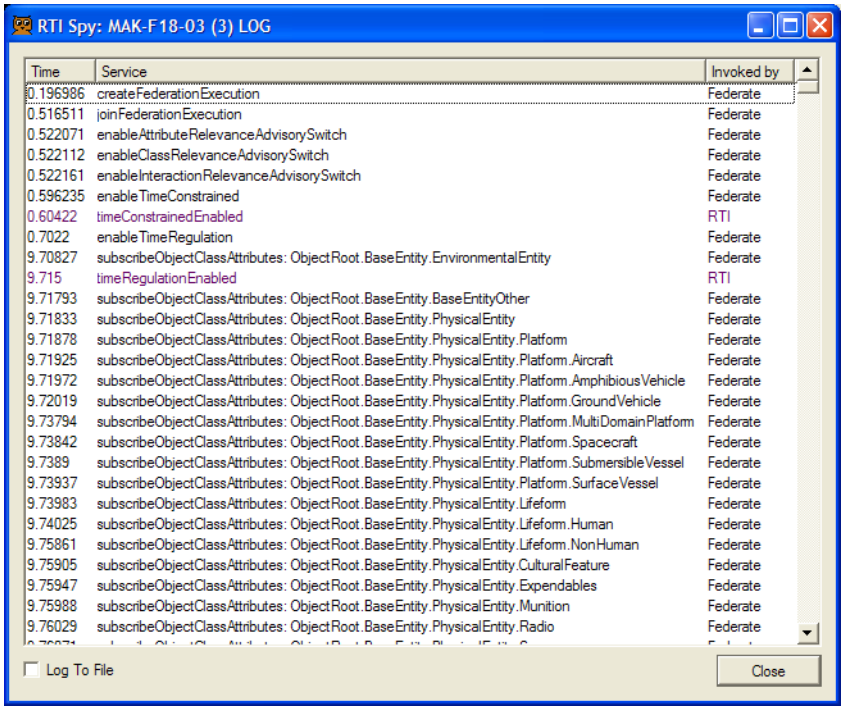


Figure 4. Service Log window

Viewing the RTI Configuration

The RTIspy Configuration window (Figure 5) allows you to view MÄK RTI configuration parameters. You cannot edit them.

➤ To open the Configuration window, on the RTIspy main window, choose View → Display Configuration.

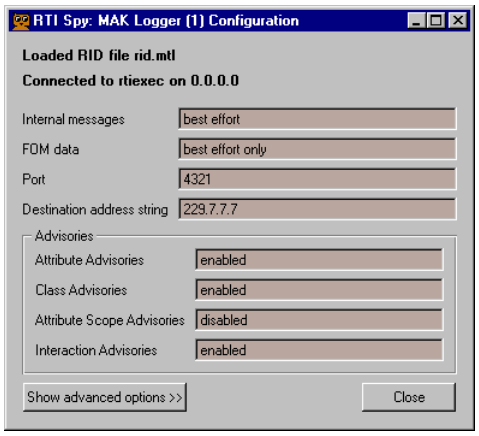


Figure 5. Configuration window

You can expand the Configuration window to see more configuration information.

➤ To expand the Configuration window, click Show Advanced Options.

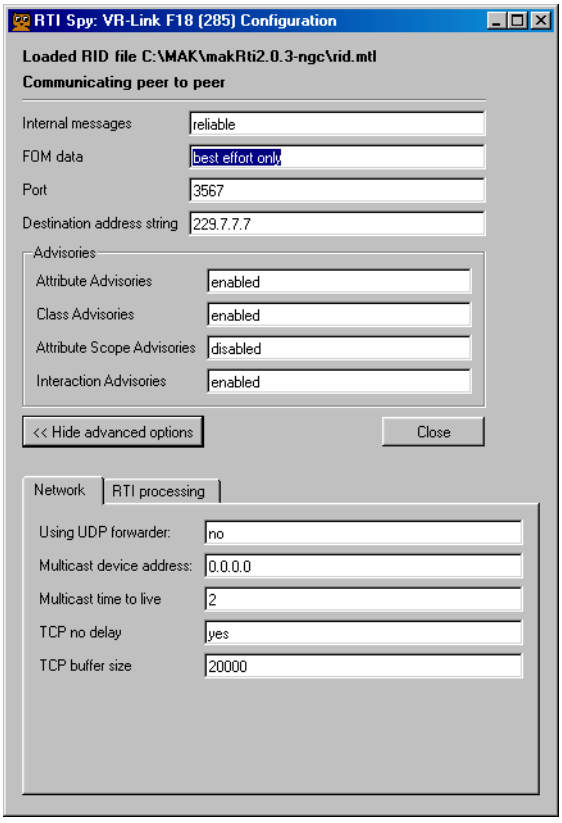


Figure 6. Configuration window, Network tab

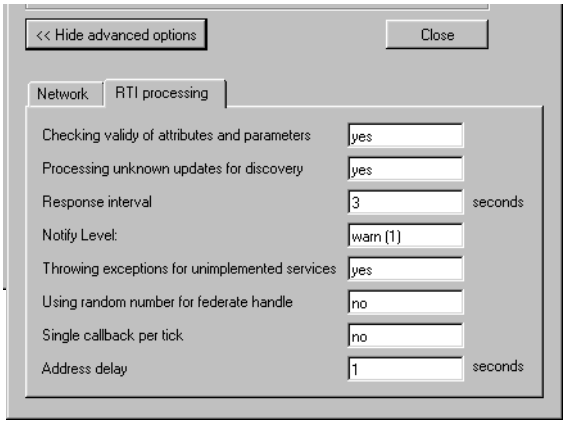


Figure 7. Configuration window, RTI processing tab

Viewing FOM Information

The FOM window (Figure 8) displays the object and interaction classes in the FOM.

➤ To open the Show FOM window, in the RTIspy main window, choose View → Display FOM.

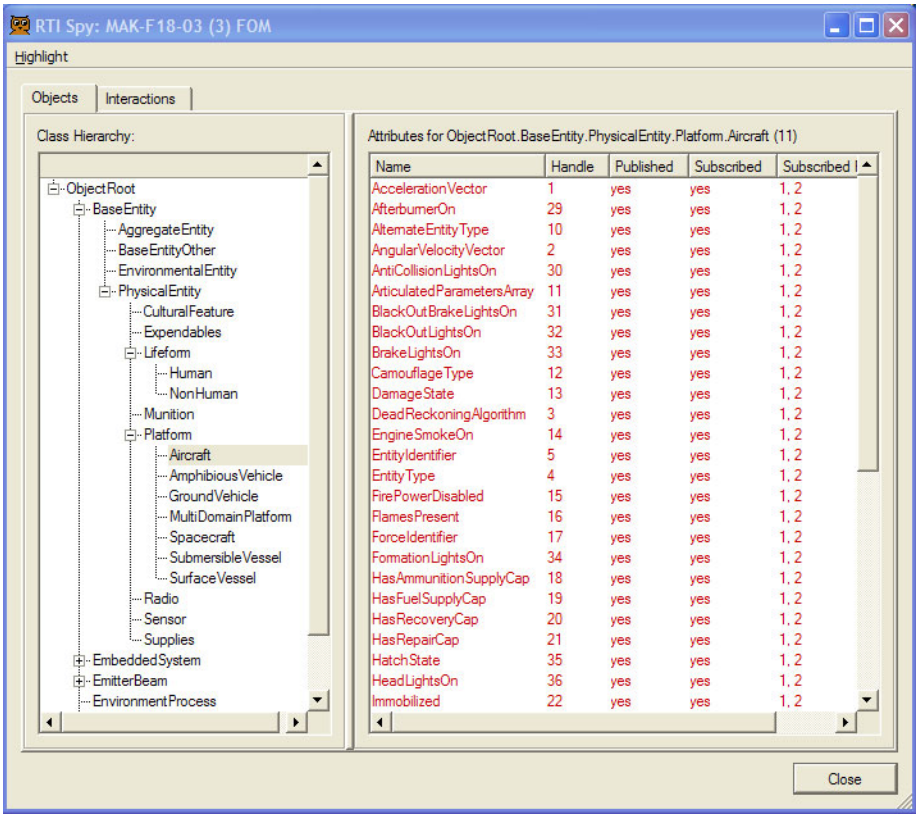


Figure 8. FOM window, Objects tab

Highlighting Published and Subscribed Classes

You can highlight the classes to which the LRC is published, subscribed, or both.

➤ To highlight classes to which the LRC is subscribed, choose Highlight → Subscribed.

➤ To highlight classes the LRC is publishing, choose Highlight → Published.

➤ To highlight classes that are both published and subscribed, choose Highlight → Published and Subscribed.

➤ To highlight all classes that are published, subscribed, or both, choose Highlight → All.

➤ To remove highlighting, choose Highlight → None.

You can tear off the Highlight menu and move it around the screen.

➤ To tear off the menu, Choose Highlight and click on the dotted line at the very top of the menu.

Viewing Object and Interaction Information

In the Objects tab, you can view attribute information. In the Interactions tab you can view parameter and subscription information.

➤ To view information about an interaction or object class, select the class in the hierarchy pane.

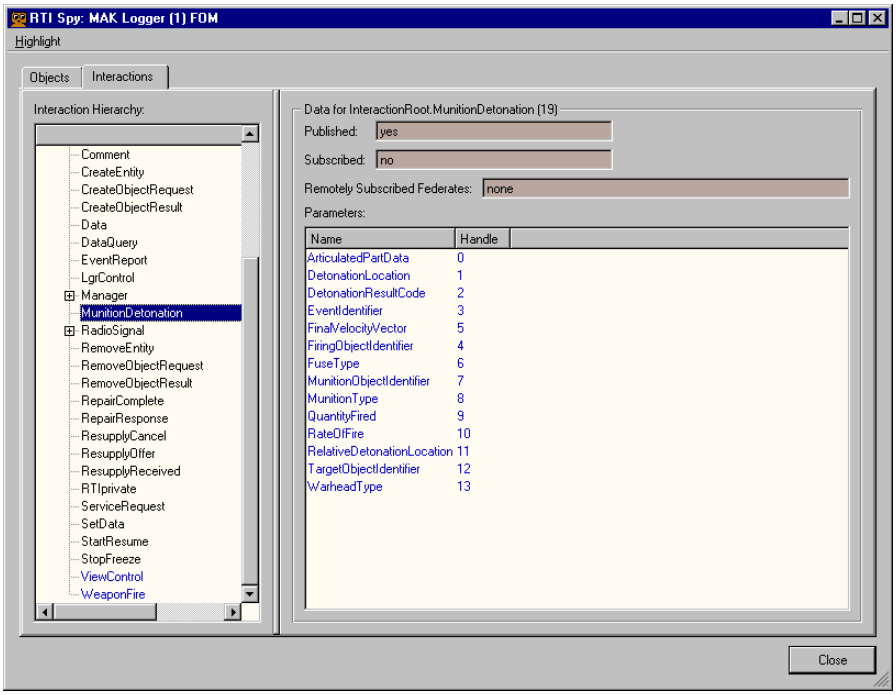


Figure 9. FOM window, Interactions tab

Monitoring and Optimizing Network Performance

The HLA API provides you with many options for optimizing performance. Since each of the RTI options offers trade-offs between CPU usage, network overhead, and functionality, it can be very difficult to predict the final effect of the various options on overall network and simulation performance. When you design a simulation, you must weigh the costs and benefits of each feature. For example, Time Management provides synchronization between federates, but imposes additional overhead in terms of message processing and increased complexity at the RTI level. Choosing data distribution management (DDM) over declaration management can greatly reduce the number of updates and interactions sent by the RTI, but adds processing overhead to each RTI call, which may result in an update or interaction message being sent, and in fact, may increase the number of messages sent under certain conditions.

The RTIspy network monitoring tool provides you with a window into the "black box" of the RTI by recording each message sent and each API call made. This provides more information to help you make the trade-offs mentioned previously. It can also help you locate the bottlenecks in your simulations, whether they are due to the underlying network, processing in the RTI, processing in Federate Ambassador callbacks, or the simulation execution itself.

The network monitoring tool exposes the internal processing of the RTI in the following ways:

- The process utilization meters allow you to see at a glance whether the RTI is truly a bottleneck for the federate, either in RTI Ambassador calls or Federate Ambassador callbacks. You can then look at the lists of API calls and messages to see which specific calls are dominating the execution.
- You can observe how much data is sent across the network to represent the simulation's objects and interactions. This can help identify possible problems and optimizations.
- You can log the gathered statistics to a file, allowing deeper analysis after a simulation run.

The use cases in this section provide examples of how you can use the RTIspy network monitoring tool to help resolve common problems.

Use Case 1 – Processing Callbacks

If you suspect that your application is spending too much time processing callbacks, the network monitoring tool can help you find the misbehaving code. To locate the source of the bottleneck, you look at the RTISpy Network Monitoring tool.

1. In the LRC window, choose View → Display Network Statistics.
2. Select the RTI API Call Statistics tab ("Viewing RTI API Call Statistics," on page 1-29.)
3. You see that the Federate Ambassador meter shows that a significant percentage of process time is spent in callbacks.

4. You want to locate the specific callback that is consuming resources, so you look at the Federate Ambassador callback list view and sort it by Total Time.
5. Go back to your callback code and look for optimizations. You might even consider moving the processing of the callback into a separate thread to isolate the network I/O and simulation processing portions of your code (at the cost of greatly increased complexity).

Use Case 2 – Large Amounts of Data Being Sent

Suppose your federate consistently sends large chunks of data via updates and interactions. The network monitoring tool can help measure the cost in terms of processing and network I/O.

1. In the LRC window, choose View → Display Network Statistics.
2. On the RTI API Call Statistics tab, look at the Network Read and Network Write fields.
3. If the time spent in network I/O is approaching the total collection period, then you should expect the network to be your bottleneck.
4. Now look at the Objects and Interactions tab to find the simulation elements that are sending the most data.
5. Consider either redesigning your update strategy for those elements (this may well be impossible, but in certain cases redesigning the FOM, or other strategies, can improve the situation quite a bit), or upgrading your physical network topology (perhaps moving to gigabit ethernet, or even a shared memory solution using the RTISpy plug-in API).

Use Case 3 – Slow-Downs under Time Management

Suppose your simulation uses the RTI's Time Management capabilities to synchronize its federates. If the federates advance through time at different rates, there may be an apparent slow-down in the federates that take larger time steps. For example, if one set of federates requests time advances of 10 seconds per step and a second set requests time steps of 1 millisecond per step, then the 10 second per step federates, which on their own would advance at a rate far greater than real-time, will appear to be slowing down. It may not be obvious what has caused the slowdown. The network monitoring tool and rtiexec GUI can help you to identify the problem.

1. In the LRC GUI for one of the 10 second per step federates, open the network monitoring tool.
2. On the RTI API Call Statistics tab look at the RTI process meter. It will probably show that the RTI is using a significant amount of process time, and the Tick / Sec. bar shows a higher than expected tick rate.

3. Sort the RTI Ambassador list view by Total Time. This will show that almost all of the federate's time is spent processing tick() calls. This is a sign that your simulation is blocked and waiting for an RTI event to proceed – in this case a Time Advance Grant.
4. In the rtiexec GUI, check Show Time State. You should see the 1 millisecond per-step federates advancing through time while the 10 second per step federates step only about every 10 seconds. This is because they are constrained by the 1 millisecond federates and can only advance when the 1 millisecond federates have stepped 10,000 times.

Use Case 4 – Comparing Benefits of RTI Features

The network monitoring tool can help you to weigh the costs and benefits of various RTI features. Some of the more advanced RTI features, such as the MOM or DDM, offer a great deal of functionality, but for some simulations, the costs in terms of additional message passing and RTI processing will outweigh the benefits. For example, the use of DDM requires the RTI to exchange additional messages for default region information even if simple DM calls are being used. To see this try the following:

1. Run the rtiexec and a simple federate without DDM enabled.
2. Open the network monitoring tool for the federate.
3. Select the RTI Message tab and observe the messages sent and received.
4. Run the federate with DDM enabled.
5. Even though DDM has no effect on a simple simulation, you will still see additional messages passed such as AssocPubRegMsgKind which are required by the DDM implementation.

The RTI Network Monitoring Window

The RTI Network Monitoring window has three tabs that allow you to monitor various aspects of the RTIs network performance.

➤ To open the RTI Network Monitoring window, in the LRC window, choose View → Display Network Statistics.

Network Monitoring Common Controls

The tabs share a set of common controls for refreshing the screen and logging performance.

Table 3 describes these controls.

Table 3: Common controls on the RTI Network Monitoring dialog box

Controls	Description
Reset	Sets all values in the list windows back to zero.
Pause Display	Pauses updating of the display. The display may change quickly. The Pause button lets you stop updates so that you can examine the values at a particular point.
Plot Range	Specifies the number of seconds to display for the time axis of the object and interaction graphs.
Refresh Interval	Specifies the frequency at which the display is updated.
Record	Starts logging performance statistics.
Sample Interval	The frequency that the RTI uses to log data.

Viewing Message Types

The Message Types tab lists the RTI message types and show how often each message type has been sent and received and how large the messages are. It also provides summary statistics on the number of messages sent and received and their sizes.

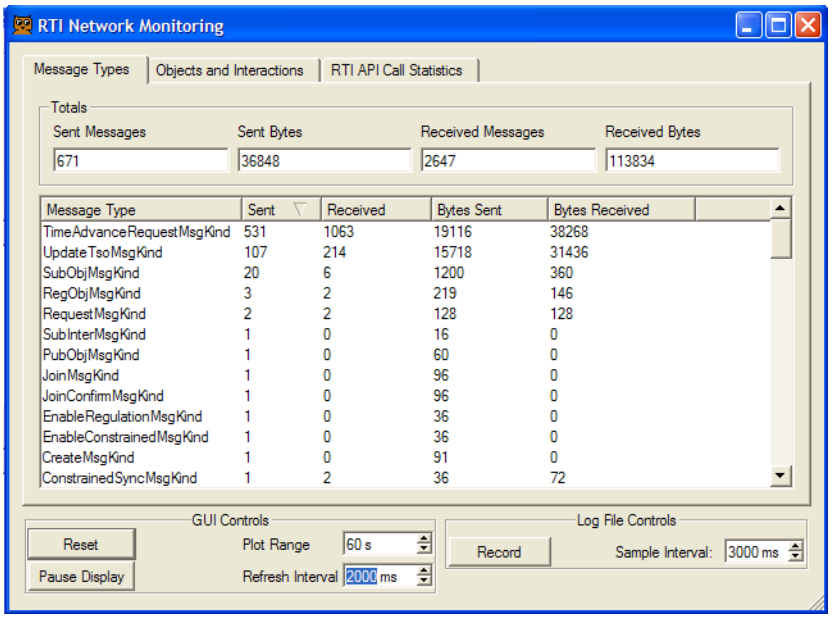


Figure 10. RTI Network Monitoring, Message Types tab

Viewing Objects and Interactions

The Objects and Interactions tab lists the objects and interactions sent and received by this federate. You can highlight registered messages and discovered messages to make it easier to distinguish them.

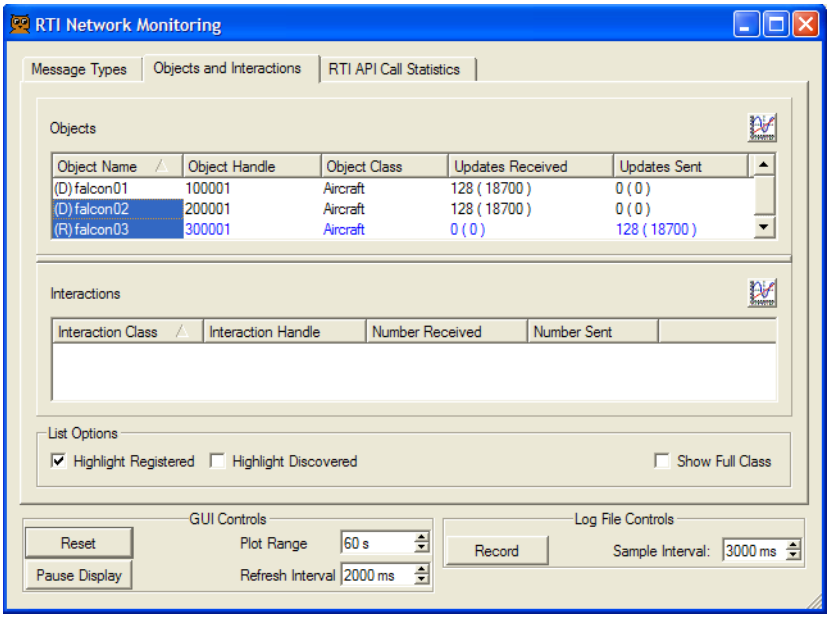


Figure 11. RTI Network Monitoring, Objects and Interactions tab

You can also display graphs showing objects and interactions over time.

To display a graph of objects or interactions:

1. In the Objects list or Interactions list, select the objects or interactions that you want to plot over time.
2. Click the plot button for the list window. A graph is displayed (Figure 12).

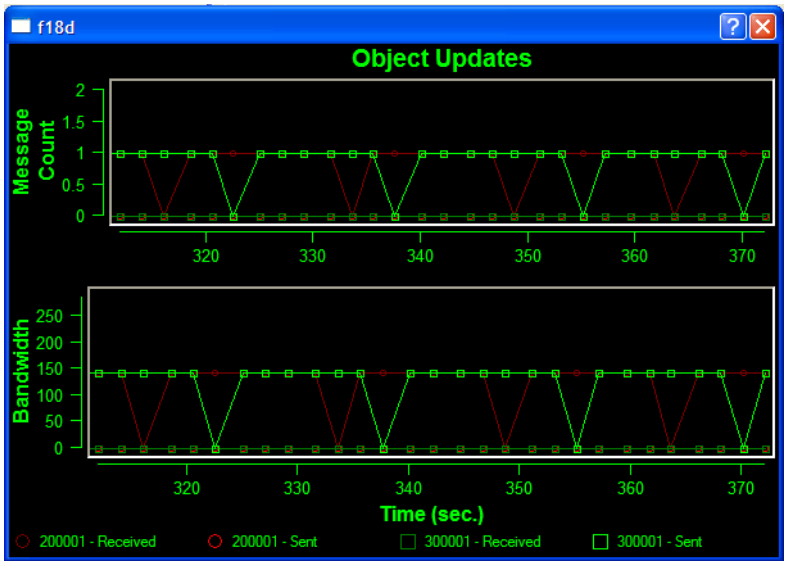


Figure 12. RTI Network Monitoring, Object Updates graph

Viewing RTI API Call Statistics

The RTI API Call Statistics displays a list of all the calls made to the RTI Ambassador and the calls that the federate has made to the Federate Ambassador. Observing the amount of time spent on calls that the federate makes, can help you locate bottlenecks in your application's use of the RTI and in your callbacks.

This tab displays the time that the federate has spent writing to the network and receiving from the network. It also has three CPU meters that provide a graphical view of the application's execution cycle. The meters display tick rate, percentage of the collection period spent executing RTI code, and the percentage of the collection period spent executing Federate Ambassador callbacks. These meters provide a running view of process utilization to help identify bottlenecks.

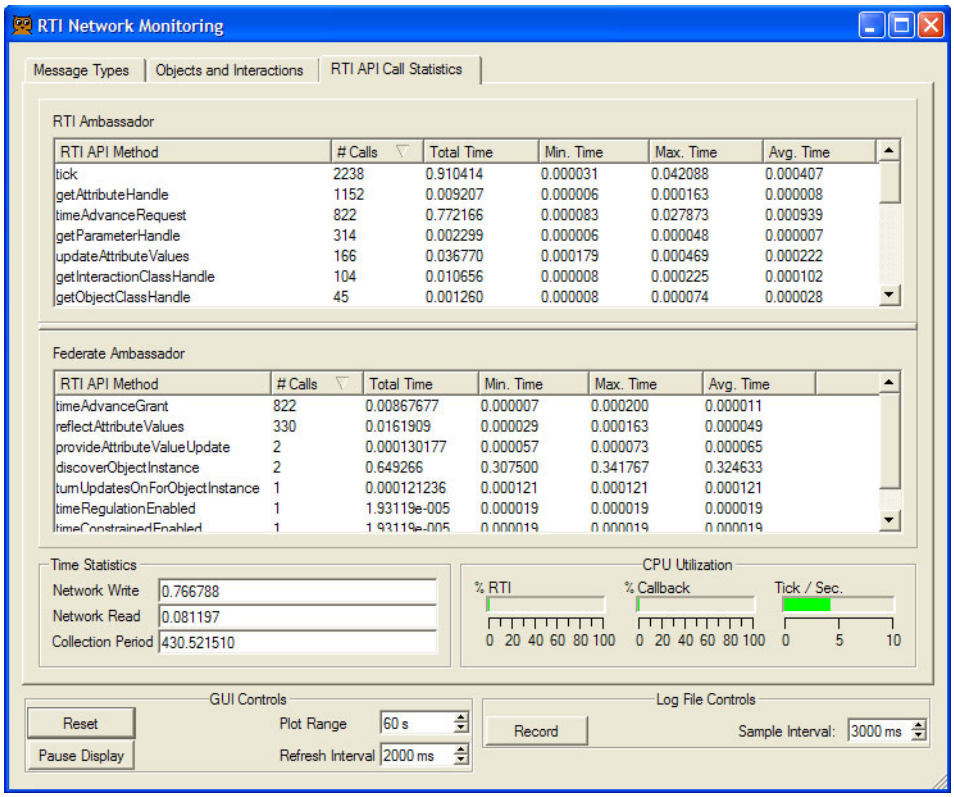


Figure 13. RTI Network Monitoring, Message Types tab

The Network Statistics Log Files

If you enable logging, the RTIspy creates three log files, one for each tab. The log files are named using the federate name and handle. The file extensions indicate the tab from which the file was generated.

Updates to the RTIspy Plug-in API

This section provides section-by-section updates to Chapter 5, "The MÄK RTI Plug-in API," in *MÄK RTI Reference Manual*. If a section is not included, there are no changes.

5.1 Introduction

The RTIspy API has undergone numerous changes to support the implementation of the HLA 1516 API. However, since the HLA 1516 interface specification is essentially a refinement of the 1.3 interface specification, most of the changes are superficial in nature. For example, the most extensive change is the replacement of RTI API types (for example, RTI::FederateHandle, RTI::ObjectClassHandle, and so on.) by equivalent internal RTI types (for example, *DtFederateHandle*, *DtObjectClassHandle*). In some cases, the method for exchanging data between components has been changed but not the data itself (for example, attribute handle value pair sets are now filled in by update and interaction messages instead of being returned as a reference). A slightly more obtrusive change is the requirement that the RTI create a federation time factory instance in order to generate federation time instances versus being able to call a static class method.

NOTE: The 1516 API uses wide strings, but these are converted to narrow strings (and vice-versa) within the RTI implementation. MÄK VR-Link provides portable utilities for converting between wide and narrow strings.

5.2 The RTI Managers

The 1516 RTIspy API has the same internal components and managers as the 1.3 RTIspy API. The classes may have been modified to some degree (for example, RTI type replacement), but they essentially perform the same functions.

5.3 Initializing a Plug-in

The 1516 RTIspy API plug-in initialization function names have a 1516 postfix so that the three functions are SetPluginCreators1516, InitLRCPlugin1516, and InitRtiexecPlugin1516. A 1.3 plug-in must use the unadorned names and a 1516 plug-in must have names with the 1516 postfix.

5.4 Monitoring RTI Service Invocations

The 1516 RTIspy API does not yet fully support monitoring RTI ambassador (*DtRtiAmbassadorObserver*) and federation ambassador (*DtFedAmbWrapper*) service calls. This omission will be corrected in the next release.

Service call monitoring in the 1.3 RTIspy API remains unchanged.

5.5 Creating Custom RTI Managers

Creating custom RTI managers is supported in both the 1.3 and 1516 RTIspy API. There are no changes to these procedures.

5.6 Subclassing DtRtiambassadorImplementor

Subclassing *DtRtiambassadorImplementor* is supported in both the 1.3 and 1516 RTIspy API. There are no changes to these procedures.

5.7 Communicating with Remote LRCs and the rtiexec

The connection manager, connections, and messages are largely unchanged between 1.3 and 1516. Aside from replacing RTI types with internal types, the only significant change is that the connection manager must now have access to an instance of a logical time factory, *DtLogicalTimeFactory*. The *DtLogicalTimeFactory* class is actually a wrapper around either the 1.3 or 1516 logical time factory classes. Many operations that could simply use the 1.3 RTI::FedTimeFactory static methods must now use the logical time factory (for example, accessing time values, including printing messages, requires the time factory for decoding the time).

5.9 rtiexec Plugins

Other than replacing the RTI types with internal types, there are no significant changes to the support for rtiexec plugins. The rtiexec plugins are supported in both the 1.3 and 1516; however, the Time Management, DDM, save/restore, and MOM functionality is not supported in 1516.

5.11 Compiling and Linking

All the changes to the RTIspy API were made in line by using the precompile directive, DtIFSPEC1516. As a result, there is a single RTIspy API for both the 1.3 and 1516 RTI implementations. The precompile directive controls the compilation of code as either 1.3 (DtIFSPEC1516 is undefined) or 1516 (DtIFSPEC1516 is defined). When a plug-in is developed and compiled, it should adhere to the types and components of the respective implementation. The HTML class documents were generated from preprocessed files to present the classes as they are compiled for the respective implementations.

The MÄK RTI 1516 does not yet support Time Management, DDM, save/restore, or MOM. The precompile directives for these services, DtTIME, DDM, DtRTISave, and DtMOM, should not be defined. However, these precompile directives must be defined for the compiling with the 1.3 RTIspy API.

For windows, the RTIspy 1516 plugins must be linked against *librti1516.lib*.

5.12 Loading Plug-Ins

The procedure is the same for loading either 1.3 or 1516 plugins; however, the plugins must match the implementation or they will not load.