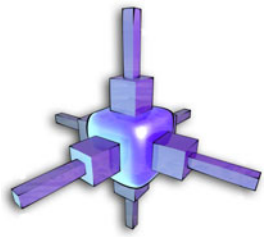

MÄK RTI 3.1 Release Notes

This document provides the following release-specific information for MÄK RTI 3.1:

Document ID: RTI-3.1-2-061115

MÄK RTI 3.0 has been verified by DMSO as fully compliant with the IEEE 1516 version of the HLA Interface Specification (SISO DLC HLA API 1516 (SISO-STD-004.1-2004).)

Supported Systems

Table 1 lists the platforms supported by MÄK RTI 3.1. 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

Operating System	Compiler
SGI IRIX6.5	MipsPro7.4.2m C++ compiler
Sun Sparcstation with Solaris 8	SPARCompiler C++ 5.2
Red Hat Enterprise Linux Workstation 3	GNU C++ (GCC) 3.2.3; glibc 2.3.2
Red Hat Enterprise Linux Workstation 4	GNU C++ (GCC) 3.4.4; glibc 2.3.4
Red Hat Fedora Core 3	GNU C++ (GCC) 3.4.4; glibc 2.3.6
Windows 2000/XP	Microsoft Visual C++ 7.1, 8.0

Qt Release Compatibility

The MÄK RTIspy GUI uses Trolltech's Qt cross-platform GUI toolkit. (The Qt run-time libraries are distributed with the MÄK RTI). If a federate also uses Qt, it must use the same version of Qt as the MÄK RTIspy GUI, in order for the RTIspy LRC GUI to function properly for that federate. Federates built with other versions of Qt should disable the RTIspy LRC GUI to avoid these incompatibility problems. If your federate has an incompatible Qt GUI, you can still use the RTIspy Web GUI to monitor your federate.

The RTIspy LRC Console GUI Plug-in is built with Qt 3.3.5.

Compiler Compatibility on Windows

MÄK provides versions of product releases that have been compiled with Microsoft Visual C++ 7.1 and 8.0. When you run MÄK products together on the same computer, for example, the MÄK RTI and the Stealth, we strongly recommend that you run versions compiled with the same compiler. Mixing products compiled with different versions of the compiler can result in program instability.

Libraries and Binaries Built with Visual Studio 2005

All MÄK products built with Microstoft Visual Studio require the C Runtime Library to function. The C runtime libraries have always been available from Microsoft for download, they are also installed on a user's machine when a Microsoft compiler is installed. The C runtime libraries are not part of the normal Windows installation. For customers who plan to use MÄK products on machines that do not have a compiler installed, MÄK has historicaly distributed a copy of the C Runtime Libraries with MÄK products. These libraries were put in the *bin* directory used by the MÄK products. MÄK products would then use the libraries in the *bin* directory and customers would not have a problem if copies of the libraries were not already installed.

Unfortunatly, with the release of the new C Runtime Libraries required by Microsoft Visual Studio 2005 (MSVC++8.0), the libraries can no longer just be copied into the *bin* directory of an application. The libraries need to be installed correctly into Windows system folders. (The process is actually a little more complicated, a manifest file needs to be created to tell Windows where to find the libraries.)

To accommodate this change, MÄK is distributing the Windows installer for the C runtime libraries with all MÄK products released for MSVC++8.0. The installer is named *vcredist_x86.exe* and is in the base directory of any installed MÄK product.

Running the installer requires Administrator privileges for the machine the installer is run on. MÄK has chosen to not integrate the MÄK installer and the Microsoft installer so as not to require users to have Administrator privileges to install MÄK products. Therefore, if you who do not have a compiler installed, or get error messages like "Software has not been installed correctly, please re-install", you must apply the patch.

For more information see this Microsoft URL:

http://msdn2.microsoft.com/en-us/library/ms235299.aspx

FLEXlm Support

MÄK RTI 3.1 uses FLEXlm 10.8 for Windows and Linux. SGI IRIX and Sun Solaris versions use FLEXlm 9.2.

Third Party Library Dependencies

In addition to the Qt dependencies mentioned previously, the MÄK RTI 3.1 has the following dependencies:

- QT - 3.3.5 - http://www.trolltech.com/
- QWT - 0.4.1 - http://qwt.sourceforge.net/
- EHS - 1.3.1 - http://xaxxon.slackworks.com/ehs/
- PME - 1.0.4 - http://xaxxon.slackworks.com/pme/
- PCRE - 6.4.1- http://www.pcre.org/
- ZLIB - 1.2.3 - http://www.zlib.net/
- ICONV - 1.8 - http://www.gnu.org/software/libiconv/
- LIBXML2 - 2.6.22 - http://www.xmlsoft.org/
- prototype.js - 1.5.0 - http://prototype.conio.net/
- script.aculo.us - 1.6.1 - http://script.aculo.us

RTIspy Web GUI Configuration

The RTIspy Web GUI has the following system considerations:

- The RTIspy Web GUI is optimized for screen resolutions of 1024x768 and larger. For smaller screen sizes, additional window scrolling will be required.
- The RTIspy Web GUI has been tested and will display correctly with Mozilla Firefox version 1.5 and higher and Internet Explorer versions 6 and higher. Other browsers may be compatible, but they have not been tested and are not specifically supported.
- The RTIspy Web GUI requires an RTI Plus license for each LRC. If licenses are available for only a subset of the federation, then those licenses will be consumed in the order that federates are started. Once all licenses are used, additional federates will not be available for monitoring (although they will still be viewable at the federation level).

MÄK RTI Licensing for the RTI Forwarder

You can run the RTI Forwarder without an RTI Plus license if it is used stand-alone (that is, centralized forwarding, in which it is not part of a distribution network and the routes file is empty.) It uses a license if it is run as part of a distributed network with other forwarders. An RTI Plus license allows you to run:

- One federate with GUI and plug-ins
- One rtiexec with GUI
- One distributed forwarder
- Plug-ins for either the rtiexec or the Distributed Forwarder (but not both with a single license).

Backwards Compatibility

The MÄK RTI 3.1 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 3.1 and some use a previous version of the MÄK RTI.) However, the MÄK RTI 3.1 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 other RTIs. (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 Other RTIs

The MÄK RTI 1.3 implements the API dictated by the 1.3 version of the HLA Interface Specification, as amended by DMSO's official HLA 1.3 Interpretations document. All RTIs that are verified as compliant with the HLA 1.3 specification are written to this exact API standard. This insures that the MÄK RTI is link-compatible with other RTIs that have been verified as HLA compliant.

The MÄK RTI 1516 implements the SISO DLC HLA API[1] 1516 (SISO-STD-004.1-2004). This API supports dynamic link compatibility, which was problematic with the original IEEE 1516 API. The IEEE 1516 API only supports compile time compatibility (uses C++ templates and implementation-specific header files). The MÄK RTI is compatible with any RTI written to the SISO DLC HLA API 1516.

Because an RTI is typically accessed by an application through a dynamic library (*.dll* on Windows, *.so* on UNIX), applications do not need to be recompiled or relinked to switch among dynamic-link-compatible RTI implementations that have been built using the same compilers. The only requirement is that the appropriate version of the library be located within the shared library search path. For more information, please see section 4.2, "Compiling and Linking with the MÄK RTI 1.3" in *MÄK RTI Reference Manual*.

1. Simulation Interoperability Standards Organization Dynamic Link Compatible High Level Architecture Application Program Interface

Because the DMSO RTI NG v6 was last built with the MSVC++ 6.0 compiler, it is not link compatible with the MÄK RTI 3.1. However, updated versions of the DMSO RTI provided by other vendors using compilers supported by MÄK RTI 3.1 would be link compatible.

New Features and Updates

MÄK RTI 3.1 has the following new features and updates:

- RTIspy Web GUI
- Shared memory for co-located federates
- Grid-based DDM
- TCP and UDP sockets support compression
- All packet forwarding is handled by one executable
- You can specify the HLA 1516 RID file and parameters dynamically
- RID consistency checking
- Changes to the RTIspy API
- New Time Management example
- New configuration parameters.

RTIspy Web GUI

RTIspy now has a web browser-based version that provides functionality similar to that provided by the rtiexec GUI and the LRC GUI. The major distinction of the Web GUI is that it allows centralized monitoring of all components of the RTI. To use the rtiexec GUI or LRC GUI, you must have physical access to the machine on which it is running. Using the Web GUI, you can monitor the rtiexec or any LRC from a single machine. A second advantage is that you can monitor multiple RTI components simultaneously for further analysis and side-by-side comparisons, which is not possible with the local GUIs.

The RTIspy Web GUI requires an RTI Plus license for each LRC. If licenses are available for only a subset of the federation, then those licenses will be consumed in the order that federates are started. Once all licenses are used, additional federates will not be available for monitoring (although they will still be viewable at the federation level).

For complete details about the RTIspy Web GUI, please see Chapter 10, *Using the RTIspy Web GUI* in *MÄK RTI Reference Manual*.

Shared Memory for Federates and rtiexec on a Single Computer

Federates and an rtiexec running on the same CPU or multi-processor machine can now communicate with each other over a shared message queue. The shared memory communication can be used indepedently of any network I/O or it can be used in conjunction with federates communicating over the network. Please see Chapter 7, *RTI Shared Memory*, in *MÄK RTI Reference Manual* for details.

Fixed-Grid DDM

MÄK RTI now supports a fixed-grid implementation for DDM message routing. A fixed-grid approach to DDM allows the RTI to approximate the routing that would occur through update and subscription region overlap calculations without having to exchange region information between LRCs. The fixed-grid approach relies on an a priori configuration of multicast addresses into a fixed-grid. Each axis of the fixed-grid is associated with a FOM dimension. Each cell of the fixed-grid is assigned a multicast address. The fixed-grid takes the place of the opposing region in the DDM region overlap calculations. The federate regions are overlapped with the fixed-grid and the overlapped grid cells establish the multicast addresses used for message routing. In the case of subscription regions, the LRC joins each multicast address that is found in overlapped grid cells. In the case of update regions, messages are sent using the multicast addresses found in overlapped grid cells. For complete details, please see section 12.2, "The Fixed Grid Approach" in *MÄK RTI Reference Manual*.

TCP and UDP Sockets Support Compression

New classes of socket objects have been incorporated into the RTI that support compression and decompression of RTI messages (including the header and payload). You can enable compression separately for UDP and TCP. The amount of compression applied can be controlled.

! While you can connect a socket with compression enabled to a non-compressing socket, the RTI messages will be unintelligible to each endpoint. Therefore, the **RTI_enableTcpCompression** and **RTI_enableUdpCompression** parameters cannot be verified by the RID Consistency Checker and must be consistent federation-wide at startup. Inconsistent settings for these parameters within a federation will lead to undefined results. Please see *MÄK RTI Reference Manual* for more information about RID Consistency Checking.

Merged Forwarders

The Distributed TCP Forwarder application has been combined with the UDP Forwarder and the distinction between the RTI 1.3 and RTI 1516 versions has been eliminated so that all forwarder variants are now handled by a single application executable. The Distributed Forwarder supports the following modes of UDP Forwarder operation:

- No forwarding.
- Legacy mode, which mimics the operation of the UDP Forwarder application in MÄK RTI 3.0 and earlier. Configuration of RID parameters (**RTI-addDestAddrString**) necessary to set up a Distributed UDP Forwarder network is the same as when using the UDP Forwarder application.
- Combined mode, which allows the Distributed Forwarder to collect and forward UDP broadcast and multicast messages over the same links established for the TCP Forwarder network. It does not require configuration of RID parameters to set up a distinct UDP Forwarder network.

For details, please see section 6.3, "Configuring Networking on a WAN" in *MÄK RTI Reference Manual*.

Creating the Routes File for Hosts with Multiple Network Interfaces

If a host running the RTI Forwarder application has multiple network interface adapters (NICs) or has multiple IP addresses assigned to an interface, the IP addresses of the interfaces on which it will connect to other RTI Forwarders via the WAN must be entered into the *routes.mtl* file Interface Address List section. Under these circumstances at least one such entry must correspond to the entry identifying the host in the Forwarder List section of the *routes.mtl* file. Consider the following configuration of RTI Forwarders:

```
;; Forwarding Port
;; Select a port for inbound connections to the Distributed Forwarder
(setqb RTI_distributedForwarderPort 5000)
;; Forwarder List
;; List all Distributed Forwarder endpoint IPs here
(RTI_addForwarder "A" "192.168.1.1")
(RTI_addForwarder "B" "192.168.1.2")
(RTI_addForwarder "C" "192.168.1.3")
```

Assume Forwarder 'B' multiple NICs, but is connected to the WAN by a particular interface. You would add its address to the Interface Address List, as follows:

```
;; Interface Address List
(RTI_addInterfaceAddr "192.168.1.2")
```

The following example illustrates the case in which a single interface has multiple IP addresses assigned to it, as might be the case if the RTI Forwarder is connected to the WAN through a VPN tunnel. In this case it is safe to list all IP addresses associated with the single interface. This example illustrates the entries necessary when Forwarder C is connected to the WAN via a VPN tunnel and has been assigned a 10.0.x.x address by the VPN concentrator.

```
;; Interface Address List
(RTI_addInterfaceAddr "192.168.1.3")
(RTI_addInterfaceAddr "10.0.1.28")
```

This additional configuration entry is only necessary when the RTI Forwarder host has multiple NICs or multiple IP addresses. It is not necessary to add the Interface Address List to the *routes.mtl* file used by RTI Forwarders that do not meet those criteria. For instance, the additional configuration entry shown in the first example would only be necessary in the *routes.mtl* file used by Forwarder B, not by Forwarder A or C. Likewise the configuration entry shown in the second example would only be necessary in the *routes.mtl* file use by Forwarder C.

Specifying HLA 1516 RID File Parameters Programmatically

The HLA 1516 createRTIambassador function has a std::vector<std::wstring> as a parameter. The caller can use this parameter to pass the RTI a list of strings containing a RID file name, individual RID settings, or both, as follows:

- If a RID file name is passed to createRTIambassador, it overrides any other method of specifying the RID file, such as the RTI_RID_FILE environment variable.
- If a list of RID settings is passed to createRTIambassador, they are applied after reading the RID file (in effect overriding the values of the RID file). Each string must be a complete lisp expression conforming to the syntax found in the RID file. Only valid lisp expressions will be applied to the RTI environment.

This feature allows individual federates to override individual RID parameters while still allowing the use of a global baseline RID file via the RTI_RID_FILE or RTI_CONFIG environment variables.

The exact syntax is:

```
RTI_RID_FILE pathToRIDFile lisp_expression1 ... lisp_expressionn
```

You can specify additional wstrings that contain lisp expressions that will override the settings in the RID file.

If you specify RTI_RID_FILE programmatically using rti1516::createRTIambassador(std::vector<std::wstring> >), the sort order is:

1. Try to open the file specified by the argument.
2. If the specified file is not found, the MÄK RTI stops looking for the RID file and fails.

RID File Consistency Checking

The MÄK RTI now provides optional runtime verification of RID file consistency across a federation execution. The RID file consistency checker allows you to guarantee that certain RID parameters are identical across all federates and the rtiexec. Some parameters may vary among federates within a federation while maintaining RID consistency. For example, socket buffer sizes can be set independently at all federates with no adverse affects to the federation. For details about RID file consistency checking, please see section 5.2.11, "The RID File Consistency Checker" in *MÄK RTI Reference Manual*.

Changes to the RTIspy API

The following changes have been made in the RTIspy API:

- RTI objects are now created via a factory method. The use of a factory allows for the extension of the object class through the plug-in API.
- The DDM implementation has been further abstracted to support both a distributed region and a fixed grid implementation. The existing *DtDataDistMgr* is mostly abstracted now, with the bulk of the implementation in the derived *DtDistributedDataDistMgr* and *DtFixedGridDataDistMgr* classes. Outside of the specific DDM implementation, the DDM Manager is accessed through this abstract class. Other basic DDM classes have similarly been abstracted, for example, *DtRegion*.
- The Federation Manager is now the only manager created before the join federation service (previously the MOM Manager was created early.) The Federation Manager creates all the other managers during the join and now only supplies a reference to itself. The other managers use the federation manager to access the other RTI components (that is, FOM, RID, and each other.)
- A multi-threaded exec class, *DtRtiExecThread*, has been added that supports embedding a multhreaded rtiexec into a custom application (for example, federation manager). It supports registering to receive an event whenever a specified RTI message kind is received.
- A new metaData class, *DtRtiMetadata*, was added that supports the addition of metaData to RTI messages.
- A setInternalCreatorFunction() member function has been added to several managers. These creator functions should not be used by a plug-in, because the internal RTI will override them. Instead use the setCreatorFunction() member function.
- A new Shared Memory Connection Manager has been added, *DtSmConnectionMgr*.
- A new compression socket has been added, *DtRtiTcpCompSocket*.

The simpletime Example

MÄK RTI 3.1 has a new example, simpletime. This example shows how to create a time managed and time constrained federate. For details, please see section B.4, "The simpletime Example" in *MÄK RTI Reference Manual*.

New Configuration Parameters

The RTI has the following new configuration parameters:

- **RTI_connectionTestInterval** specifies the time to wait between connection tests for connections that may be idle for long periods, for example, the http server connection and distributed forwarder connection.
- **RTI_ridConsistencyChecking** specifies the mode of RID file consistency checking to use, as follows:
  - 0 = Disable consistency checking.
  - 1 = Local parameter values are overridden by the rtiexec if there is a mismatch. Print a message to the console indicating which parameters have been overridden.
  - 2 = Do not override local parameters. Throw an exception if there is a mismatch. Print a message to the console indicating which parameters do not match.

RTIspy Web GUI Parameters

The RTIspy Web GUI uses the following new parameters. It also is affected by parameters present in previous releases. For a complete list of parameters and more details about their use, please see section 10.2.1, "Configuring The RTIspy Web GUI" in *MÄK RTI Reference Manual*.

- **RTI_enableRtiexecWebservice** enables (1) or disables (0) the RTIspy Web GUI at the rtiexec (requires an RTIspy license per rtiexec).
- **RTI_enableLrcWebservice** enables (1) or disables (0) the RTIspy Web GUI at this federate (requires an RTIspy license per federate).
- **RTI_enableLrcWebserviceEventLog** enables the LRC Event Log in the RTIspy Web GUI.
- **RTI_enableLrcQtApp** enables (1) or disables (0) the local LRC GUI. The RTI_enableLrcGUI parameter must be enabled. This is set per federate. Default: 1.
- **RTI_webserviceHttpPort** specifies the HTTP Service port for web client connections to the HTTP Server.
- **RTI_webserviceRtiPort** specifies the port used by RTI components to connect to the HTTP Server back-end.
- **RTI_webserviceAddr** specifies the IP address of the HTTP server for this RTI component to connect to.
- **RTI_webserviceEnableServerAutoStart** attempts to start an HTTP server locally if one does not already exist (and **RTI_webserviceAddr** does not equal localhost : 127.0.0.1).

- **RTI_webserviceEnableForwarding** allows the rtiexec to forward web client requests to federates running the web-service plug-in. This allows the rtiexec to act as a proxy for the entire RTI network.
- **RTI_webserviceNotifyLevel** specifies the debug output level at the HTTP server.

Compression Parameters

- **RTI_enableTcpCompression** enables compression of RTI message payloads transmitted over TCP connections. Consistency checking is enforced on this parameter.
- **RTI_tcpCompressionLevel** specifies the amount of compression applied to RTI message payloads over TCP connections as follows:
  - 1: minimal compression.
  - 2 - 8: increasing compression.
  - 9: maximal compression.
  - 10: maximal compression of packet bundles (super bundling.) This is only valid when packet bundling is enabled.
- **RTI_enableUdpCompression** enables compression of RTI message payloads transmitted in UDP datagrams. Consistency checking is enforced on this parameter.
- **RTI_udpCompressionLevel** specifies the amount of compression applied to RTI message payloads in UDP datagrams, using the same options as **RTI_tcpCompressionLevel**.

Shared Memory Parameters

- **RTI_sharedMemoryMode** enables the RTI to use shared memory for communication between co-located federates and other RTI components (for example, the rtiexec). Valid settings are:
  - 0: disables shared memory (default)
  - 1: enables shared memory only (no network I/O)
  - 2: enable shared memory manager with network I/O.
- **RTI_sharedMemoryName** specifies the name of a shared memory file. Default: "RtiSharedMemory". All co-located federates must use the same shared memory file name in order to communicate through the same shared memory region.
- **RTI_sharedMemoryQueueLength** specifies the number of message queue segments (that is, 200 byte buckets) that can be queued in shared memory. Default: 5000.
- **RTI_sharedMemoryMgrMaxWait** specifies how long (in seconds) the Shared Memory Queue Manager process waits for messages to arrive in the message queue before it polls the network interface for incoming messages. Default: 0.25 seconds.

Distributed Forwarder Parameters

- RTI_distributedUdpForwarderMode determines whether or not Distributed Forwarders handle UDP forwarding, and if so, the mode to use. Consistency checking is enforced on this parameter.
  - 0: Distributed Forwarders do not handle UDP forwarding
  - 1: Distributed Forwarders imitate UDP Forwarders (Legacy Mode)
  - 2: Distributed Forwarders handle combined UDP/TCP forwarding duties.

Fixed Grid DDM Parameters

- RTI_ddmFixedGrid - Enables fixed grid DDM versus distributed region DDM when DDM is enabled. Default: disabled.
- RTI_ddmFixedGridFilename - Specifies the fixed grid configuration file name.

Changed and Deleted Parameters

- The RTI_deliverTransientMessagesDuringSave RID parameter is no longer supported. The RTI will deliver only those messages that are appropriate during a save or restore. The RTI_saveTransientMessages RID parameter is still supported. However, it should be unnecessary and it is not compliant. It is retained for backwards compatibility.
- The RTI_federateReconnectEnabled parameter has been renamed to RTI_reconnectEnabled to reflect the new capability of the rtiexec to reconnect to a federation.

Miscellaneous Changes

- Fedex message routing is now enabled by default.

Documentation Updates

MÄK RTI Reference Manual has been updated for this release.

- In section 7.2.1 of the PDF version of the manual, the note at the end of the section should be deleted and the final paragraph should read as follows:

  When the shared memory feature is enabled, a Shared Memory Queue Manager process is required. It creates the shared memory region, sets up and manage the message queue within that shared memory regaion, and relays messages between shared memory and the network. You can start the Shread Memory Queue Manager manually or automatically. For details, please see “Starting the Shared Memory Manager,” on page 7-5.

- In the Java doc that comes with the Java class bindings, the initial page of the class documentation only refers to HLA 1.3. However, the documentation covers both the HLA 1.3. API and the HLA 1516 API.

Bug Fixes

The following bugs have been fixed in this release:

- The spelling of the RID parameter **RTI_enableFomBackwardsCompatibility** was corrected (from **RTI_enableFomBackwardsCompatability**).
- The MÄK RTI now correctly handles large messages (greater than 64 KB).
- The asynchronous modes of the RTI now throw an internal error exception when the reliable connection to the RTI forwarder is lost. This makes asynchronous mode act the same way the synchronous mode works. Previously, there was no way for a federate to detect the failure.
- Significant performance improvements have been made to the distributed forwarder, which will be useful for all users.
- An incompatibility between versions of static libraries has been resolved for federates using VR-Link and the MÄK RTI 3.0 and 2.4.2 under Windows. In certain configurations, this incompatibility was causing federates to crash on exit when the federate dynamically loaded the RTI (using the LoadLibrary and FreeLibrary commands). A federate can now dynamically load the RTI and exit safely.
- Problems with the following Java bindings components have been corrected:
  - Handle set iterators
  - Announce synchronization point callback
  - Region set range bounds.
- The use of multicast discovery no longer causes the TCP Forwarder to consume excessive CPU. This issue only affected some UNIX platforms due to platform specific handling of multicast filtering.
- The RTIspy LRC GUI now works with Qt applications that run their QApplication in a secondary thread.
- When **RTI_bigMessage** is enabled, an interaction sent with time will be appropriately processed by receiving federates. Previously, these messages would be erroneously dropped by the RTI.
- Asynchronous callbacks are not implemented for HLA 1.3, and cannot be enabled via the RID file or otherwise. Previously a 1.3 federate could enable asynchronous callbacks, resulting in a federate that would throw an exception in createFederationExecution.
- On some platforms, a race condition involving asynchronous IO was eliminated. Previously it was possible that the asynchronous IO thread could exit prematurely, resulting in a federate that would not successfully join a federation.
- The rtiDump utility only generates output at an output level higher than 2. If you try to run the utility at a notify level of less than 2, the application warns you that no output will be generated.
- The rtiexec, when running in non-GUI mode, now accepts input from the console without blocking. Previously the rtiexec would not process incoming messages if characters were typed to the console without pressing the return key.

- The **RTI_enableNetworkMonitoring** parameter is automatically set if **enableLRCGUI** is set to 1. Previously this caused an crash.
- When a federate instantiates an RTI Ambassador, the LRC outputs a message indicating the version of the RTI that was loaded. This version message now contains an indication of whether the LRC was built in debug mode.
- When **RTI_bigMessage** is enabled, the stand alone Forwarder will forward them correctly. Previously, these messages would be corrupted.
- Previously the distributed forwarder was incompatible with the sender-side filtering (smart forwarding) option in the RID file. This was a serious flaw since smart forwarding is most useful for the distributed forwarders. This issue has been resolved.
- The FDD DTD attribute can specify a path for the DTD file. This correction was actually made in MÄK RTI 3.0, but not documented. However, please see "Known Problems," on page 16, regarding another issue with the DTD.
- **RTI_federateHeartbeatInterval** and **RTI_federateTimeoutInterval**, must both be either 0 or non zero. If one parameter is 0, the other is forced to 0. Having inconsistent values for these parameters caused federates to be forcefully resigned.

Known Problems

This release has the following known problems:

- The HLA 1.3 version of 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 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.
- The rtiexec does not connect the output streams to a console until after reading the RID file in Windows. As a result, errors opening or parsing the RID file are not displayed. If **RTI_enablePopUpErrorMsgs** is enabled then these errors are displayed in a popup box, however this solution works only under Windows.
- The asynchronous callback feature is not available for HLA 1.3 federates. It is only available for 1516 federates. Enabling this option will have no effect on an HLA 1.3 federate.

- Distributed forwarder multicast subscription does not work for late-joining forwarders. With the introduction of the unified Distributed Forwarder a need arose to communicate between federates on a LAN and forwarders on remote LANs when a DM or DDM multicast group is joined or dropped, such that the forwarders can monitor or drop those groups on its LAN and forward messages back to the federates joined to such multicast groups. However the scheme to accomplish this does not work if the forwarder is not connected to the forwarder network when the federate joins the multicast group. The workaround is to ensure that the forwarder network is completely established prior to launching federates, or at the least, prior to allowing federates to attempt to join DM or DDM multicast groups.
- The RID consistency feature relies on the ability to establish a connection between a federate and the RID tester (presumably the rtiexec), however if the federate's **RTI_enableTcpCompression** parameter and the RID tester's parameter are different, the two cannot communicate, and consequently the RID consistency test request and response messages cannot be exchanged. The workaround is to ensure that the **RTI_enableTcpCompression** parameter is consistent among all participants prior to launch.
- Bundling (**RTI_enablePacketBundling**) cannot be used when also using big messages (**RTI_enableBigMessage**). The format of messages when big messages is enabled conflicts with how messages are bundled.
- The MÄK Technologies Lisp (MTL) files used by MÄK products to specify configuration settings are parsed at startup by the RTI. When a configuration file has certain errors, such as unmatched parentheses, the parser will not necessarily return an error code. This can cause the RTI to reach a partially initialized state or cause the RTI to initialize itself with default RID values, causing unexpected behavior. This applies to lisp expressions in the RID file as well as the expressions passed to the 1516 createRTIambassador function.
- The XML processing of the FDD DTD attribute differs between Linux and Windows. In Linux, the DTD is found relative to the federate's execution directory. In Windows, the file is found relative to the FDD location.
- In 1516, the HLA DTD is not distributed with the FDD file. In order for the LRC to process the FDD file, the HLA DTD must be accessible as specified in the FDD DTD attribute.
- The help text for the rtiexec console commands incorrectly indicates that the character **`d`** can be used in place of the text `delete` to delete a federate from the federation. This is not correct. Use `delete`.
- To use shared memory on operating systems other than Windows, you must start the Shared Memory Queue Manager manually. Automatic launching of the shared memory queue manager has been disabled because the current implementation resulted in a conflict for console I/O.

- The RTI can be configured so that when a federate terminates abnormally, the RTI deletes the objects for which the federate owns the privilegeToDelete attribute. If a federate has divested this attribute, the RTI does not recognize the ownership transfer with regard to the delete orphan behavior. As a result, when a federate terminates, abnormally or through normal resignation, the delete orphan behavior is enforced for any object it had created, whether it owns the privilegeToDelete or not. The exception is when the federate still owns the privilegeToDelete attribute and resigns normally with an action of UNCONDITIONALLY_DIVEST_ATTRIBUTES.

RTI 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 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.