MÄK RTI 1.3.7-ngc Release Notes

This document provides the following release-specific information for MÄK RTI 1.3.7-ngc:

Document ID: RTI-1.3.7-2-020731

Overview

MÄK RTI 1.3.7-ngc is an implementation of a subset of the HLA Interface Specification, version 1.3.

Licensing Requirements

Although the MÄK RTI is provided to customers without charge, it requires a license. This license is not necessarily included with the license we provide for products you purchase from us. You can generate a license for the MÄK RTI on our web site at: http://www.mak.com/rti_license.htm.

MÄK RTI 1.3.7-ngc includes the new RTI Spy GUI and plug-in API. These features are licensed separately. For information about licensing the RTI Spy, contact info@mak.com.

Supported Systems

Table 1 lists the platforms supported by the MÄK RTI 1.3.7-ngc. 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	MipsPro7.2.1 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 7.2	GNU C++ (g++) distributed with Red Hat distribution. gcc version 3.0.2
PC with Windows NT 4.0, Windows 95/98/2000/XP	Microsoft Visual C++ 6.0.

MÄK RTI 1.3.7-ngc is built against VR-Link 3.7.2-ngc.

Backwards Compatibility

The MÄK RTI 1.3.7-ngc 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 1.3.7-ngc and some use a previous version of the MÄK RTI.) However, the MÄK RTI 1.3.7-ngc 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 one additional function (RtReadFileDescriptor(), described in MÄK RTI documentation), the MÄK RTI 1.3.7-ngc has the exact same API as the DMSO RTI 1.3 NG v1, v2, v3.0, v3.2, and v4.0. Unless you are using this optional function, an application built against the DMSO RTI NG does not need to be recompiled in order to be used with the MÄK RTI 1.3.7-ngc, or vice versa.

MÄK RTI 1.3.7-ngc 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 1.3.7-ngc, are not run-time compatible with the DMSO RTI NG v5 due to changes in the how the DMSO RTI handles namespaces. An upcoming release of the DMSO RTI NG (v6) will revert to the previous API. We expect MÄK products to be run-time compatible with DMSO RTI NG v6.

New Features

MÄK RTI 1.3.7-ngc has the following new features:

- Support for asynchronous input/output
- Ability to configure the size of the TCP and UDP network receive buffers.

Asynchronous I/O

You can configure the MÄK RTI to use asynchronous I/O. The federate local RTI component (LRC) creates a separate I/O thread that processes network I/O operations separately from other internal RTI processing. The I/O thread only performs network I/O operations. The federate must still invoke the tick() method to allocate CPU cycles to the RTI for its internal processing and to generate federate ambassador callbacks.

Asynchronous I/O allows the federate to better schedule its time and to better respond to peaks in I/O processing. Many of the RTI services require data to be transmitted over the network. The asynchronous I/O thread separates the network transmission portion of this processing from the federate's thread of execution. Since the network I/O is not performed during the processing of RTI Ambassador service calls, the RTI returns the thread of execution to the federate significantly faster. The federate is better able to respond to time critical processing that might in turn generate an increased output load (for example, aircraft maneuvering). The I/O thread can also use spare CPU cycles when the federate is waiting for other system operations (for example, disk access). Similarly, the asynchronous I/O thread receives data from the network separately from the federate's thread of execution. The RTI's tick() method must still be called to process the received data and generate the federate callbacks. However, the federate is less likely to affect data transmission (for example, dropped messages or blocked TCP connections) if it fails to invoke the tick() method frequently enough (for example, during initialization phases).

A disadvantage to asynchronous I/O is that it can increase message transmission latency. The timing of when messages are sent and received becomes less determined since the federate LRC thread is no longer performing the actual I/O operations. However, the MÄK RTI uses a signaling strategy between the threads that allows the latency to be small while the RTI is not heavily loaded, and for it to take advantage of queuing when loads increase. The threads signal each other when messages are available for sending and receiving.

Another disadvantage to asynchronous I/O is that the federate thread can fail to yield enough processing cycles to the I/O thread. If the I/O thread becomes starved (that is, it cannot get enough processing cycles), it will not process network I/O. The RTI messages will not be sent or received, resulting in high latency and buffer overflows. In addition, operating system mechanisms can take effect that can be detrimental to federate performance (for example, Windows gives starved threads control for some number of cycles). New parameters are provided to control the interaction between the federate LRC thread and the I/O thread. One of these parameters controls whether the LRC yields to the I/O thread during the tick() call (on by default). Having the federate LRC thread yield during the tick() call ensures that the I/O thread will be executed. We recommend that this parameter value should not be changed unless the federate has been designed to manage multi-threading issues so that it sufficiently yields execution to other threads.

The following parameters adjust the behavior involving the asynchronous IO thread:

Table 2: Asynchronous I/O parameters

Parameter	Description
RTI_asynchronousIO	Enables or disables multithreading. Activates all other multithreading options. ◆ 0 = Off ◆ 1 = On. Default: 0.
RTI_maxIOQueue	Specifies how many messages can be queued to send before blocking. Default 20000
RTI_tickFavorsNetwork	Enables or disables a thread yield in tick(). ◆ 0 = Off ◆ 1 = On. Default: 1.
RTI_IOPeriod	Specifies the amount of time polling operations take in seconds. Not used in Windows which is event driven. Must be > = 0. Default: 0.01.
RTI_maxIOCount	Specifies the number of times the thread can cycle before it must yield. Must be > 0. Default: 1000.
RTI_IOLockQueue	Specifies the number of messages processed per locking of the queues. Adjustment of this value allows for fine/coarse locking. Must be > 0. Default: 500.

Socket Receive Buffer Size

A new RID parameter configures the size of the TCP and UDP network receive buffers. Increasing the size can increase the RTI's tolerance of peak network loading. The RTI initializes the buffer size to 32K.

```
(setqb RTI_socketReceiveBufferSize buffer_size)
```

Documentation Updates

The following sections of *MÄK RTI Reference Manual* have changed:

- Resolution of the FED file, as described in section 3.2.1
- The functioning of the Connection Manager in the MÄK RTI API, as described in section 5.7.1.

Update to Section 3.2.1: Reading the FED File

The method for resolving the FED file that the RTI will read when a federate creates or joins a federation has changed. The local RTI component in each federate must read a valid FED file from disk in order to successfully create or join a federation. The FED file information itself is not distributed across the network; however, the FED file name is distributed to each joining federate when using an rtiexec.

When you do not use the rtiexec, the association between a federation execution and its FED file is not distributed by the RTI. Therefore, in order for a federate to specify what FED file to use, it should call createFederationExecution() before calling joinFederationExecution() even if the federation execution has already been created by another federate. When you do not use the rtiexec, the createFederationExecution() service call does nothing except validate the FED file and cache the mapping between the federation execution name and the FED file name. Alternatively, the federate can omit calling createFederationExecution() and the RTI will default to using the federation execution name with a *.fed* suffix. In either case, the RTI looks for the file relative to the working directory (unless an absolute path is specified).

When you use the rtiexec, only one federate will successfully create the federation execution. The FED file name that it uses to create the federation is distributed to each federate as it joins. This FED file name, stripped of any path, will be used by each local RTI component to read the FED file. The RTI looks for this file name in its working directory.

Update to Section 5.7.1: The Connection Manager

The *DtConnectionMgr* has increased its encapsulation of the *DtRtiConnections* used for sending and receiving messages. The previous *DtConnectionMgr* provided access to three connections (internalConn(), reliableConn() and udpConn()) that were used by managers to send and receive messages. While access to the three connections is still provided, for most cases, their use is now deprecated. Instead, the *DtConnectionMgr* provides wrapper methods to perform most of the send and receive processing. The send wrapper methods take a transport type to select the connection to use for sending. Most of the receive processing was and still is performed within the *DtConnectionMgr* except that it has directly taken over some of the message processing previously performed by the *DtRtiConnections*.

DtAsyncConnectionMgr is a new subclass that supports asynchronous I/O. While the interface to the connections is still through the *DtConnectionMgr*, it is the *DtAsyncConnectionMgr* class that is instantiated by the RTI. The *DtAsyncConnectionMgr* class supports asynchronous or polling I/O based on RID file parameters. The connection manager can still be overridden (from either *DtConnectionMgr* or *DtAsyncConnectionMgr*) using the DtConnectionMgr::setCreatorFunction() method.

Fixed Bugs

This release fixes the following problems in previous releases:

- A problem with reliable transport (TCP) has been fixed. The effect was that federates and the rtiexec would abnormally terminate or their I/O buffers would overflow.
- A problem with discovery of existing objects by late joining federates has been fixed. This problem was introduced in the 1.3.6 release. The effect was that late joining federates would fail to discover objects that already existed in the federation.

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 RTI Spy 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.