MÄK RTI 2.0.1-ngc Release Notes

This document provides the following release-specific information for MÄK RTI 2.0.1-ngc:

MÄK RTI 2.0-ngc has been verified by DMSO as fully compliant with the HLA Interface Specification, version 1.3.

Document ID: RTI-2.0.1-2-030320

Overview

MÄK RTI 2.0-ngc 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 2.0.1-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	MipsPro7.3 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 7.2	GNU C++ (g++) distributed with Red Hat distribution. gcc version 3.0.2
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, Windows 98/2000/XP	Microsoft Visual C++ 6.0.

MÄK RTI 2.0.1-ngc is built with VR-Link 3.8-ngc.

Backwards Compatibility

The MÄK RTI 2.0.1-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 2.0.1-ngc and some use a previous version of the MÄK RTI.) However, the MÄK RTI 2.0.1-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 several additional functions (such as, DtReadFileDescriptor(), described in MÄK RTI documentation), the MÄK RTI 2.0.1-ngc 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 2.0.1-ngc, or vice versa.

MÄK RTI 2.0.1-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 2.0.1-ngc, 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 2.0.1-ngc has the following new features:

- New RID parameter: **RTI_useBusyWaitForTickMinMax**
- Update to the **RTI_singleCallbackPerTick** parameter
- Improved compatibility with DMSO RTI NG v6
- Support for specifying a default FED file
- Ability to tolerate abnormally terminated federates and orphaned objects.

The RTI_useBusyWaitForTickMinMax Parameter

The RID parameter **RTI_useBusyWaitForTickMinMax** specifies that the RTI will use busy wait in tick(min,max) when no messages are pending.

By default, the implementation of Tick(min,max) uses select (or windows events) to wait for additional input when there are no pending messages. The granularity of the select mechanism can make the tick(min,max) exceed the maximum time by too much (for example, select takes on the order of 10msec under Linux). To have finer control of when tick(min,max) returns, replace the select mechanism with a busy wait that polls the input source.

We do not recommend enabling the busy wait mechanism if you use asynchronous IO. The busy wait will not yield to the IO thread and the IO thread must execute in order to generate input.

The syntax for **RTI_useBusyWaitForTickMinMax** is as follows:

```
RTI_useBusyWaitForTickMinMax mode
```

where *mode* is

- 0 = Disabled (use select)
- 1 = Enabled.

Default: 0

Compatibility with DMSO RTI NG v6

An ostream operator for a const exception was added for compatibility with the DMSO RTI NG v6.

The RTI_singleCallbackPerTick Parameter

The **RTI_singleCallbackPerTick** parameter now also applies to tick(min,max). When enabled, tick(min,max) returns after processing a single message or when the minimum time has been exceeded (effectively max is ignored).

A message may only cause internal processing and not necessarily generate a callback. The return value of tick() or tick(min,max) will be true if a message has been processed; otherwise, it will be false.

The RTI_defaultFedFile Parameter

You can specify a default FED file in the RID file to support joining without calling create when operating in lightweight mode (that is, without an rtiexec). When in lightweight mode and using the default FED file, the join federation service will always associate the federation name with the default FED file. Previously, when operating in lightweight mode, a federate was required to first call create federation in order to set up a mapping between a federation name and a FED file if the FED file was not federation name ".fed".

```
RTI_defaultFedFile fed_file
```

In lightweight mode, the FED file used for initializing federation data. If the value is an empty string, then the FED file will either be established by a previous create federation service call or federation name ".fed". Default: ""

Fault Detection

Fault detection was added to the federation executive (fedex) and the RTI executive (ritexec) to support automatic removal of abnormally terminated federates and their orphaned objects. Two mechanisms have been added for detecting a terminated federate.

- One mechanism detects broken TCP connections between the LRC and the rtiexec (that is, by its TCP forwarder component).
- The other mechanism relies on a heartbeat message sent by each LRC to the fedex.

The removal of a terminated federate from a federation is treated approximately as if the federate resigned either with an action of RTI::RELEASE_ATTRIBUTES or RTI::DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES.

The detection of broken TCP connection requires that the LRC establish a TCP connection to the rtiexec's TCP forwarder (**RTI_tcpForwarderAddr**). When the rtiexec detects that this TCP connection has been lost, the federate is forcibly removed from its federation, if joined.

The heartbeat mechanism has each LRC send a heartbeat message to its federation executive once it has joined. When the federation executive has not received the heartbeat message within a time-out interval, the federate is forcibly removed from the federation. You can configure the heartbeat interval and the time-out interval in the RID file.

When a federate is forcibly removed from a federation, the object attributes that it owns become orphaned. The current fault tolerance does not support any ownership transfer of these attributes (that is, the RTI does not take ownership of the attributes). However, you can configure the RTI to delete those objects for which the federate owns the **privilegeToDelete** attribute. For any objects for which the federate owns attributes, but does not own the **privilegeToDelete**, the owned attributes become orphaned and cannot be recovered. If desired, even objects for which the federate owns the **privilegeToDelete** can be left orphaned (to support federates that own other attributes). The syntax of the new RID file parameters is as follows:

RTI_federateHeartbeatInterval

```
RTI_federateHeartbeatInterval heartbeat
```

where *heartbeat* specifies the frequency with which each federate LRC sends a heartbeat message.

If you set the heartbeat to a value <= 0 the federate does not send heartbeats. The timeout is also disabled. The default heartbeat is 10.

RTI_federateTimeoutInterval

```
RTI_federateTimeoutInterval timeout
```

where timeout specifies the time period in which the fedex must receive a heartbeat or it will forcibly remove the federate.

- <= 0 = Time-outs disabled. The federate cannot be timed out. There is no heartbeat.
- >0 = The time-out interval in seconds.

Default: 25

RTI_deleteOrphans

```
RTI_deleteOrphans mode
```

where mode specifies whether or not the objects for which a terminated federate owns the **privilegeToDelete** will be automatically deleted.

- 0 = disabled (all owned attributes are orphaned)
- 1 = enabled.

Default: 1

Documentation Updates

This section documents the following features, which are not adequately documented in MÄK RTI Reference Manual:

- Lightweight mode
- UDP forwarding.

Running the MÄK RTI in Lightweight Mode

You can run the RTI in Lightweight Mode, that is, without the rtiexec. Lightweight mode is primarily useful for developing and debugging federates and federations or for executing very simple test federations. Without an rtiexec, the process of testing new features and debugging problems is easier because there is not an additional process to manage and federates can be brought up and down quickly.

Lightweight mode is not an option for federations that require functionality that is only supported by the rtiexec process. The rtiexec is required to support the following:

- Reliable transport (TCP)
- Federation Management Synchronization Points
- Federation Management Save and Restore
- Time Management.

Even for the services that can operate without an rtiexec, running in Lightweight mode removes guarantees of strict compliance with the HLA Interface Specification. For example, without an rtiexec more than one federate can successfully invoke the create federation service without causing an RTI exception. For the most part, the absence of these type of exceptions is not a problem. However, a more critical concern is that the rtiexec guarantees each federate will be assigned a unique federate handle. Since object handles are based on the federate handle, running with the rtiexec also guarantees unique object handles. Without the rtiexec, the default behavior of the RTI is to use each federate's process ID (or a random number) as its federate handle. While the federate and object handles will be unique almost every time, this is not guaranteed.

All things considered, you should use the rtiexec when you conduct your formal experiments and training. The rtiexec's affect on performance is insignificant except in support of the centralized functionality listed above. Not counting those services, the only time the rtiexec comes into play is during create, destroy, join, and resign. Since best effort traffic is exchanged directly between federates, the rtiexec neither sends nor processes any messages during the bulk of the federation execution. So the only difference between using the rtiexec and not, is trading convenience for better guarantees on the RTI's operation.

UDP Forwarding

UDP forwarding is available as part of the RTI Plus product.

Section 3.9 in *MÄK RTI Reference Manual* documents how the **RTI-addDestAddrString** parameter can be used to support communications across a WAN. The MÄK RTI inherently supports directing its best effort messages to multiple destinations. The primary destination is designated using the **RTI_destAddrString** RID parameter. You can designate additional destinations with the **RTI-addDestAddrString** parameter. When a best effort message is sent, it is delivered to each of the destination addresses. Typically, the primary destination is a broadcast or multicast address that supports distributing the message to all of the local federates. Federates on a given LAN then use **RTI-addDestAddrString** to designate additional destinations to each of the federates on remote LANs. This configuration requires transmitting a separate message over the WAN for each remote federate. The UDP forwarder can reduce these remote communications across the WAN by replacing multiple destinations to a remote LAN with a single destination.

The UDP forwarder allows you to send UDP traffic to just one destination per remote LAN, instead of sending a separate copy to each remote federate. The UDP forwarder takes messages from over the WAN, and forwards them on a local multicast or broadcast address. Instead of sending a message to each remote federate, a single message can be delivered to the remote UDP Forwarder. The UDP Forwarder will then distribute the messages on its LAN and thus to each of the remote federates. The UDP Forwarder reduces the bandwidth transmitted over the WAN and reduces the number of messages that the LRC must transmit.

The UDP Forwarder should be run at each LAN which has multiple federates. Consider the case where there are three LANs A, B, and C. On LAN A, there is a single federate. It would not make sense to run a UDP Forwarder at this LAN since any RTI messages distributed to this LAN would only go to that single federate. On LAN B and C, there are multiple federates. On a LAN with just two federates, there is a trade off in adding the UDP Forwarder versus having two copies of messages sent to the LAN. You should consider that the UDP Forwarder only reduces the messages transmitted to its LAN. It has no effect on the transmission of messages from federates on its LAN to remote LANs. Assume that a UDP Forwarder is required on LAN B and C.

Each federate's RID file must be configured to add the appropriate destination addresses. For the federate on LAN A, there are no other local federates; its only destinations are the federates on LANs B and C. Even so, there are some messages that can be sent by an LRC that are processed by the sending LRC (for example, provideAttributeValue). For this reason, the primary destination address must still be either a broadcast or multicast address (which will loopback the message). Then the addresses of the two UDP Forwarders at LAN B and C would be added as additional destinations. The RID files for federates on LAN B should be configured with a multicast address as the primary destination for the other federates on the LAN. Then the address of the federate on LAN A and the UDP Forwarder on LAN C should be added as additional destinations. The RID files for federates on LAN C should be configured similarly except with the LAN Bs UDP Forwarder address.

The UDP Forwarder takes two command line parameters as follows:

```
udpForwarder [-f fedex] [-A bindAddr]
```

The `-f` parameter specifies the name of the federation execution. The `-A` parameter designates the IP address of the network device that the UDP Forwarder will listen to for remote messages. The UDP forwarder uses an RTI RID file to configure the primary destination address and RTI port. The UDP Forwarder also supports the class-based multicast filtering described in section 3.16. However, it does not support the DDM multicast filtering.

Fixed Bugs

This release fixes the following problems in previous releases:

- Tick(min,max) now continues to process pending messages up to max time. Previously, it would return after min time had expired regardless of whether or not there were pending messages. Checking for the presence of pending messages (when using asynchronous IO) has also been corrected. Previously an internal buffer that may have contained messages was not being checked.
- The MOM federation object is now discovered before being reflected. Previously, the federation object would be reflected without being discovered (if the federate subscribed to the class).
- Any number of DDM spaces is now permitted in the federation. It is no longer fixed at 20.
- The RID parameter **RTI_useExceptForUnimplServ** was replaced with **RTI_throwExceptionForDisabledService** and affected warning messages were changed appropriately.

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.