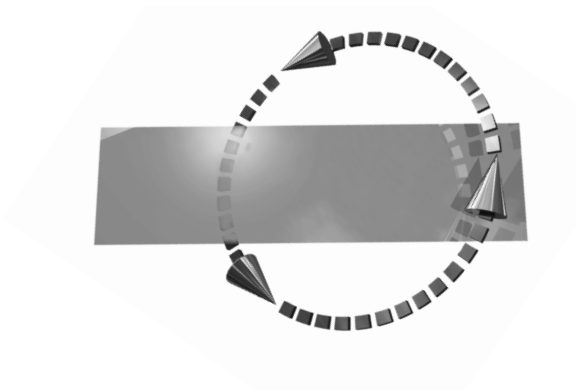

MÄK VR-Link 3.8-ngc Release Notes

This release note provides the following release-specific information for MÄK VR-Link Release 3.8-ngc:

Document ID: VRL-3.7.3-3-030131

Systems Supported

Table 1 lists the platforms currently supported by MÄK VR-Link 3.8-ngc. Please contact MÄK if you are interested in purchasing VR-Link for other platforms. Application code must be built with the indicated compilers in order to link to VR-Link 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++ (GCC) 3.0.2
PC with Windows NT/2000/XP	Microsoft Visual C++ 6.0 SP5

Disk Space Requirements

On SGI IRIX, you need approximately 270 MB for VR-Link files and 60 MB for all documentation files. Other UNIX installations require approximately 50 MB for VR-Link files and an additional 60 MB for documentation files.

On PCs, you need 50 MB for VR-Link files and 60 MB for all documentation files.

Online Help Browser Requirements

The online help system for the FOM Mapper Builder is an HTML-based system. To view online help, you must have a web browser installed that supports Javascript. Javascript must be enabled.

Backward Compatibility

VR-Link 3.8-ngc is highly compatible with VR-Link 3.7.3-ngc. For example, the *talk* and *listen* example programs from VR-Link 3.7.3-ngc and previous versions compile and link unmodified against VR-Link 3.8-ngc.

Network Compatibility

HLA only

VR-Link 3.8-ngc is compliant with:

- RPR-FOM 0.5, 0.7, 0.8, 1.0, and a subset of 2.0 (draft 6 and 14)
- MÄK RTI 2.0-ngc
- DMSO RTI NG v3.0, v3.1, v3.2, 4.0, and 6.0 (but not 5.0).

The Linux 7.2 verison of VR-Link is compatible with the DMSO RTI 6.0.

DIS only

This release supports DIS 2.0.4, IEEE 1278.1, 2.1.4, and IEEE 1278.1a, and can therefore interoperate with DIS applications of any of these versions.

RPR FOM Versions Supported

VR-Link 3.8-ngc has built-in support for versions 0.5, 0.7, 0.8, 1.0, and 2.0 of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, VR-Link 3.8-ngc uses RPR FOM 1.0.

The keywords for specifying different versions of the RPR FOM are:

- RPR FOM 0.5: `vlRPR05`
- RPR FOM 0.7: `vlRPR07`
- RPR FOM 0.8: `vlRPR08`
- RPR FOM 1.0: `vlRPR10`
- RPR FOM 2.0 darft 6: `vlRPR20006`
- RPR FOM 2.0 draft 14: `vlRPR20014`.

If you want to use a version of the RPR FOM other than 1.0, pass the version number (0.5, 0.7, 0.8, or 2.0) to the *DtRprFomMapper* constructor and pass the resulting object to the *DtExerciseConn* constructor. Also, make sure you are using a federation execution name that corresponds to the right FED file. For example:

```
DtExerciseConn conn("VR-Link20006", "MyApp", new DtRprFomMapper(2.0006));
```

VR-Link examples like *f18* and *hlaNetdump* have a command line option, `-f`, that you can use to choose an RPR FOM version by passing a RPR FOM keyword as the argument to `-f`.

RTI Support

VR-Link 3.8-ngc comes with the MÄK RTI 2.0-ngc, which is fully compliant and is link-compatible with DMSO RTI NG. VR-Link 3.8-ngc has been tested with DMSO RTI NG v6.

The Linux 7.2 version of VR-Link 3.8-ngc (which was compiled with gcc 3.0.2) runs only with DMSO RTI NG v6 and MÄK RTI 2.0-ngc for gcc 3.0.2.

VR-Link 3.8-ngc may work with older versions of the MÄK RTI, but we recommend using it with release 2.0-ngc.

If you use the MÄK RTI, remember to make sure that VR-Link can find your FED file, and optional *rid.mtl*. Put the FED file in the directory from which you are running, or set the environment variable RTI_CONFIG to the directory that contains it. See *MÄK RTI Reference Manual* for a list of the options for specifying the location of the *rid.mtl* file.

New Features and Changes

This section describes the following features and changes to VR-Link:

- Support for asynchronous I/O
- Support for Time Management
- Support for RPR FOM 2.0 draft 14.

Asynchronous IO for DIS

VR-Link 3.8-ngc supports asynchronous IO for DIS using the new *DtAsyncNetSocket* class. (Asynchronous IO for HLA is handled by the RTI.) *DtAsyncNetSocket* has all the functionality of *DtNetSocket*; however, it creates two threads that read and write to the network. The read thread reads packets from the network and places them on a queue. When an application reads a packet from *DtAsyncNetSocket* it reads it from the queue. When an application writes to the network using *DtAsyncNetSocket*, it writes to a queue. The send thread reads the queue and sends packets to the network as soon as possible.

You create *DtAsyncNetSocket* just like you create *DtNetSocket*. You can enable and disable either thread (send or receive).

The DIS version of *DtExerciseConn* can use the *DtAsyncNetSocket* in either of the following ways:

- Pass a *DtAsyncNetSocket* pointer to the constructor, as follows:

```
// Pass a pointer to the constructor
DtAsyncNetSocket *netSocket = new DtAsyncNetSocket(port);
DtExerciseConn  *exConn = new DtExerciseConn(netSocket, exerciseId,
  siteId, applicationNum );
. . .
delete exConn;
delete netSocket;
```

- Set the *DtExerciseConn* **use_async** parameter to DtTrue. The default is DtFalse.

```
// Set use_async to DtTrue
DtExerciseConn exConn(port, exerciseId, siteId, applicationNum, status,
  DtTrue);
```

Time Management

VR-Link 3.8-ngc supports HLA Time Management. The use of Time Management in an exercise allows federates to synchronize and order events with a timeline maintained by the RTI. Typically, federates use time management when they are interacting with other federates that have a different rate of simulation. However, federates do not have to use Time Management, even if other federates in a federation use it. This description of Time Management in VR-Link assumes that you are familiar with Time Management terms and concepts. Please see your HLA and RTI documentation for more information.

Federation Time

Federation Time (FedTime) is the managed simulation time for an exercise. In a time-managed federation execution, the progression of time is accomplished by federates making time requests and receiving time grants from the RTI. Regulating federates control the advancement of FedTime. Constrained federates can advance only as quickly as FedTime allows. At any point in the simulation, FedTime may be different for different federates.

In VR-Link most calls to the RTI are made using the RTI Ambassador directly and are not wrapped by VR-Link. Time Regulation can be turned on and off at any time in an exercise. Many other aspects of time management can be configured such as the LookAhead. For more information about these methods please consult your RTI documentation.

Some example calls are as follows:

```
DtExerciseConn exConn(. . .);
. . .
exConn.rtiAmb()->enableTimeRegulation(mySuggestedFedTime, myLookAhead);
exConn.rtiAmb()->enableTimeConstrained();
exConn.rtiAmb()->disableTimeConstrained();
exConn.rtiAmb()->timeAdvanceRequest(RequestedFedTime);
exConn.rtiAmb()->disableTimeRegulation();
. . .
```

Sending Time Stamp Order (TSO) Messages

To send time stamp order (TSO) messages in VR-Link, tell VR-Link to attach a FedTime to outgoing messages, as follows:

```
exConn.setSendFedTime(DtTrue);
```

When sendFedTime() is DtTrue, all messages are sent TSO. When sendFedTime() is DtFalse (the default), messages are sent read order (RO). This flag can be toggled at any time, but it must be toggled before the message is sent. VR-Link uses VR-Link simTime when it sends FedTime:

```
ExConn.clock()->simTime();
```

You need to maintain simTime as FedTime + LookAhead for the message to be delivered. If simTime is less than this, the RTI throws an exception. You can run simTime ahead of FedTime + Lookahead, because a Regulator can send messages with any time in the future, as long as the time is at least FedTime + Lookahead.

Receiving TSO Messages

VR-Link handles incoming TSO messages transparently. The RTI makes sure the messages are delivered at the correct FedTime. VR-Link uses the FedTime, when available, to perform dead reckoning computations.

Using Callbacks

When the RTI needs to update FedTime for a federate by granting a request, or enable some type of time regulation, it invokes a method in the Federate Ambassador. VR-Link provides callbacks for these methods. You can write functions that handle these RTI events. Then you can register them for callbacks.

The following examples demonstrate callbacks as well as the functions that register them.

```
void timeAdvanceRequestCb(const RTI::FedTime &theFedTime, void *userData)
{

  // The time we request might not be the time we get back!
  GlobalFedTime = theFedTime;
  // do something
  cout << "Time Advance Grant to: " << GlobalFedTime.getTime() << endl;
  cout.flush();
}

void timeConstrainedEnabledCb(const RTI::FedTime &theFedTime,
  void *userData)
{
  GlobalFedTime = theFedTime;
  // do something
  cout << "Time Constrained Enabled at time: "
    << GlobalFedTime.getTime() << endl;
  cout.flush();
}

// Call this when our time request is granted. timeAdvanceRequestCb is
// the callback. The 0 is data.
exConn.fedAmb()->addTimeAdvanceGrantCb(timeAdvanceRequestCb, 0);

// This callback lets me know I have become constrained. I must
// remember that until I receive it, I am not time constrained
// and should not ask for a time advance. timeConstrainedEnabledCb is
// the callback. The 0 is data.
exConn.fedAmb()->addTimeConstrainedEnabledCb(timeConstrainedEnabledCb,
  0);

// This callback lets me know I have become time regulating. It also
// lets me know what FedTime it is, I Must use that fed time.
// timeRegulationEnabledCb is the callback. The 0 is data.
exConn.fedAmb()->addTimeRegulationEnabledCb(timeRegulationEnabledCb, 0);
```

Converting FedTime to DtTime

Occassionally, VR-Link needs to convert from RTI::FedTime to *DtTime*. RTI::FedTime is an abstract base class and the federate needs to provide an implementation of this class. Most RTIs, including the MÄK RTI, provide a subclass for federates to use. It is typically in the library *libFedTime.a*. The base class provides no method for converting RTI::FedTime to a double (*DtTime*), but the sub-classed method provided by many RTI's (including the MÄK RTI) does . Because it is possible, and even likely, that you will use a different subclass of RTI::FedTime, VR-Link will not always know how to do this conversion. Therefore, VR-Link provides member functions in *DtExerciseConn* that allow you to register conversion functions for VR-Link to use.

```
static void DtExerciseConn::setFedTimeToVrlTimeConverter(
  DtFedTimeToVrlTimeConverter func);
static void DtExerciseConn::setVrlTimeToFedTimeConverter(
  DtVrlTimeToFedTimeConverter func);
```

DtExerciseConn has default conversion functions for the MÄK RTI. If you need to set the conversion functions back to the default, pass a NULL value to either DtExerciseConn::setFedTimeToVrlTimeConverter() or DtExerciseConn::setVrlTimeToFedTimeConverter().

When a federate needs to convert between RTI::FedTime and *DtTime*, it can use the member functions:

- static DtFedTimeToVrlTimeConverter DtExerciseConn::fedTimeToVrlTimeConverter();
- static DtVrlTimeToFedTimeConverter DtExerciseConn::vrlTimeToFedTimeConverter();

Use them as in the following example:

```
RTI::FedTime *fedTimePtr =(*DtExerciseConn::vrlTimeToFedTimeConverter())
  (myDtTimeVariable);
```

Support for RPR FOM 2.0 Draft 14

VR-Link now supports RPR FOM draft 14. VR-Link has also added support for the following HLA interactions:

- TransferControl interaction
- RecordR interaction
- SetRecord interaction.

Documentation Updates

The Packet Server no longer supports HLA.

Bug Fixes

VR-Link 3.8-ngc fixes the following bugs:

- The implementation of IFF now matches the latest RPR FOM specification.
- On UNIX, RPR FOM 2.0 spatial attributes no longer have four additional bytes padding at the end of the network structure.