VR-Link 3.12 Release Notes

This release note provides the following release-specific information for VR-Link Release 3.12:

Document ID: VRL-3.12-3-080617

Systems Supported

Table 1 lists the platforms currently supported by MÄK VR-Link 3.12. Table 2 lists the platforms supported by the TENA version of VR-Link. 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 for HLA and DIS

Platform	Compiler
Red Hat Enterprise Linux Workstation 4.0, 5.0 Fedora 7	default compiler
PC with Windows XP/Vista	Microsoft Visual C++ 7.1 Microsoft Visual C++ 8.0

Table 2: Platforms supported for TENA

Platform	Compiler
Red Hat Enterprise Linux Workstation 4.0, 5.0	default compiler
PC with Windows XP/Vista	Microsoft Visual C++ 7.1 Microsoft Visual C++ 8.0

Disk Space Requirements

Linux installations require approximately 80 MB for VR-Link files and an additional 120 MB for documentation files.

On PCs, you need 320 MB for VR-Link files and 120 MB for all documentation files.

Building VR-Link on Linux

VR-Link does not require you to use the MÄK build system, but if you choose to, you must use gmake 3.81 or later. You can download it from the following URL:

http://savannah.gnu.org/projects/make/

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, for example, the Logger 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.

TENA Compatibility

VR-Link 3.12 was built against TENA 5.2.2. The MÄK LROM version is 2.1.

FLEXlm Support

VR-Link 3.12 uses FLEXlm 11.4-2.

Libraries and Binaries Built with Visual Studio 2005

All MÄK products built with Microsoft 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 historically 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.

Unfortunately, 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

Patch Required for AMD Dual-processor Windows PCs

VR-Link-based products use a high resolution counter for time calculations on Windows PCs. Customers who are running Windows on PCs with multiple AMD Athlon 64-bit processors may notice clock jitter, which may cause time in MÄK products to run backwards. This occurs when the Windows scheduler changes the CPU the MÄK process is using. If the high resolution counters on each processor are not synchronized, the application may witness a decrease in the high resolution counter value stored in the processor causing an incorrect time calculation. To fix this problem customers, apply the AMD Dual-Core Optimizer patch provided by AMD. You can get the patch at:

http://www.amd.com/us-en/Processors/TechnicalResources/0,,30_182_871_9706,00.html

If you get an error when you try to access this URL, reload the page.

Backward Compatibility

Though MÄK tries to maintain a high degree of source compatibility between VR-Link releases, changes to the articulated and attached parts API and the organization of header files means that you may have to change code written with previous versions of VR-Link. You will also have to change project files and makefiles.

Network Compatibility

HLA only

VR-Link 3.12 is compliant with:

- RPR-FOM 0.5, 0.7, 0.8, 1.0, and a subset of 2.0 (draft 6, 14, and 17)
- MÄK RTI 2.x, 3.x
- Pitch RTI 1.3 C++ interface.

Other RTIs that support the HLA 1.3 specification or the SISO DLC HLA API 1516 (SISO-STD-004.1-2004 version of the IEEE 1516 specification.)

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 3.12 has built-in support for versions 0.5, 0.7, 0.8, 1.0, and 2.0, drafts 6, 14, and 17, of the RPR FOM. By default, VR-Link 3.12 uses RPR FOM 1.0.

VR-Link does not officially support RPR FOM 2 Draft 18 at this time. However, the draft appears to be similar enough to RPR FOM 2 Draft 17 that the 2.0017 FOM Mapper may be used for federates wishing to interoperate with RPR FOM 2 Draft 18.

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, 2.0006, 2.0014, or 2.0017) 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-Link20017", "MyApp", new DtRprFomMapper(2.0017));
```

VR-Link examples like *f18* and *hlaNetdump* have a command line option, `--rprFomVersion`, that you can use to choose a RPR FOM version, using one of the version numbers listed in the previous paragraph.

RTI Support

VR-Link 3.12 has been tested with the Pitch RTI 1.3 and MÄK RTI 3.2.

If you use the MÄK RTI, remember to make sure that VR-Link can find your FED file or FDD file, and optional *rid.mtl*. Put the FED file or FDD 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

VR-Link 3.12 has the following new features and changes:

- The articulated and attached parts API has been rewritten and significantly improved.
- Header files have been reorganized.
- *DtVrlApplicationInitializer* now parses XML files. The public and protected APIs have been updated.
- Entity State Update PDU support. VR-Link now processes incoming Entity State Update PDUs and updates reflected entities with the data supplied by the PDU.
- The TENA Middleware has been upgraded to 5.2.2.
- Improved support for the SISO Link 16 standard (SISO-STD-002-2006), which is available from www.sisostds.org.
- Built-in support for DI-GUY data updates.
- Dead reckoner and smoother improvements.
- Improved class documentation.
- Improved FOM Mapper API.
- Improved MOM support. *DtReflectedFederateList* is a reflected list for MOM federate data.
- Significant performance improvements in Network Types and Environmental Process.
- Miscellaneous API updates.

Changes to the Articulated Parts API

The articulated and attached parts API has changed significantly in VR-Link 3.12. The following class names have changed to better describe the classes:

- *DtArtPartList* is now *DtArticulatedPartCollection*
- *DtArtPart* is now *DtArticulatedPart*
- *DtAttachedPartList* is now *DtAttachedPartCollection*

A new class, *DtAttachedPart*, has been added.

In previous releases, articulated and attached parts were stored in an ordered list. They are now stored in a hash that maps part type to articulated part and station to attached part, so they can be thought of as a collection, rather than a list. The *DtEntityStateRepository* and *DtDetonationInteraction* previously contained a *DtArtPartList*; they now contain a *DtArticulatedPartCollection*, which can be accessed through the classes' accessor methods: artPartList() (for the *DtEntityStateRepository*) and articulatedParts() (for the *DtDetonationInteraction*).

Setting Articulated Parts

Previously, an articulated part list had to be initialized with a *DtArtPartSpec*. The *DtArtPartSpec* has been removed from VR-Link. Now, no initialization is needed to use an articulated part list. The list is simply empty on creation. To create articulated or attached parts, call the getPart() method on the collection.

This creates a part of the specified type and returns it to the caller. *DtArtPartSpec* specified which part an articulated part attached to. This is now specified by calling attachPart() on the collection, and passing in a pointer to the child part and a pointer to the parent part. By default, parts are attached to the base.

Previously, a turret and a gun would be added to an articulated part list (in an entity state repository), with the gun attached to the turret, as follows:

```
DtArtPartSpec specList[3] =
{
  { DtPrimaryTurret1, DtAttachedToBase,
    { DtApAzimuth, DtApAzimuthRate, 0 } },
  { DtPrimaryGun1,    DtPrimaryTurret1,
    { DtApElevation, 0 } },
  { DtNullArtPartSpec }
};
myESR->initArtParts( specList );
```

Now, it is done as follows:

```
DtArticulatedPartCollection* artPartCollection = myESR->artPartList();
DtArticulatedPart& turret = artPartCollection->getPart(DtPrimaryTurret1);
DtArticulatedPart& gun = artPartCollection->getPart(DtPrimaryGun1);
artPartCollection->attachPart(&gun, &turret);
```

This creates two parts: the turret and the gun, and attaches the gun to the turret.

Formerly, the parameters for each part needed to be specified in the *DtArtPartSpec*. Only after they were initialized, could they be set. Now, when an articulated part is created, it has no parameters, and nothing will be sent in an update on the network if none have been set. Previously, to set a value for a parameter, the part type, parameter type, and value had to be passed into the setArtParamVal() method of the *DtArtPartList*. Now, setParameter() is called on the *DtArticulatedPart* (which could be retrieved by getPart().

getPart() will create a part if it does not exist, or will simply return the part if it does).

Previously, the values for azimuth and azimuth rate would be set for the turret, and the elevation would be set for the gun as follows:

```
myESR->artPartList()->setArtParamVal( DtPrimaryTurret1, DtApAzimuth,
  (DtFloat32)DtDeg2Rad( myTurretAz ) );
myESR->artPartList()->setArtParamVal( DtPrimaryTurret1, DtApAzimuthRate,
  (DtFloat32)DtDeg2Rad( myTurretAzRate ) );
myESR->artPartList()->setArtParamVal( DtPrimaryGun1, DtApElevation,
  (DtFloat32)DtDeg2Rad( myGunElev ) );
```

Now, it is done as follows:

```
artPartCollection->getPart(DtPrimaryTurret1).setParameter(DtApAzimuth,
  DtDeg2Rad(myTurretAz));
artPartCollection->getPart(DtPrimaryTurret1).setParameter(
  DtApAzimuthRate, DtDeg2Rad(myTurretAzRate));
artPartCollection->getPart(DtPrimaryGun1).setParameter(DtApElevation,
  DtDeg2Rad(myGunElev));
```

Retrieving Articulated Parts

In previous versions of VR-Link, articulated parts were typically read by iterating through the articulated part list, and then iterating through each parameter type, getting the values from the part retrieved from the list.

Now, you iterate through the articulated part collection using STL iterators, as the *DtArticulatedPartCollection* provides a begin() and end() iterator. Then, each part returns an STL vector of parameters that were set for that part.

Previously, parts were iterated over as follows:

```
DtArtPartList* artPartList = myESR->artPartList();
for (int partNum = 1; partNum <= artPartList->numArtParts();
  ++partNum)
{
  DtArtPart *ap = artPartList->artPartFromPartNum(partNum);
  for (int param = ap->firstValidParamType(); param;
       param = ap->nextValidParamType(param))
  {
    DtFloat32 val = ap->val(param);
    DtInfo << "Part Type: " << ap->partType()
           << "Param Type: " << param
           << "Value:      " << value << std::endl;
  }
}
```

Now, the following would be done:

```
DtArticulatedPartCollection* artPartList = myESR->artPartList();
for(DtArticulatedPartCollection::const_iterator
  artPartIter = artPartList->begin();
  artPartIter != artPartList->end();
  ++artPartIter)
{
  int currentPartType = artPartIter->first;
  DtArticulatedPart* currentPart = artPartIter->second;

  // Once we have a chosen a DtArticulatedPart, we can retrieve a
  // vector of the parameter metrics set, and iterate through it.
  std::vector<DtArticulatedPart::ParameterMetric> parameterMetrics;
  currentPart->getParameterMetrics(parameterMetrics);

  std::vector<DtArticulatedPart::ParameterMetric>::const_iterator
    paramMetricIter = parameterMetrics.begin();
  std::vector<DtArticulatedPart::ParameterMetric>::const_iterator
    paramMetricEnd = parameterMetrics.end();

  for(;paramMetricIter != paramMetricEnd; ++paramMetricIter)
  {
```

```
      int paramMetric = *paramMetricIter;
      DtArticulatedPart::Parameter parameter;

      currentPart->getParameter(paramMetric, parameter);

      DtInfo << "Part Type: " << currentPartType
             << "Param Type: " << paramMetric
             << "Value: " << parameter.value << std::endl;
    }
  }
```

Attached Parts

Attached parts are managed similarly to articulate parts. The *DtAttachedPartCollection* map maps station numbers to attached parts, rather than part types to articulated parts. Therefore,

```
DtAttachedPart& attachedPart = attPartList->getPart(5);
```

would retrieve the attached part at station number 5.

Then, instead of containing a vector of parameters, each attached part simply has an entity type. To set the entity type of the above part, call:

```
attachedPart.setEntityType(...);
```

Iterate over an attached part collection and print out the entity types, as follows:

```
DtAttachedPartCollection* attPartList = myESR->attPartList();
for(DtAttachedPartCollection::const_iterator
  attPartIter = attPartList->begin();
  attPartIter != attPartList->end();
  ++attPartIter)
{
  int currentStation = attPartIter->first;
  DtAttachedPart* currentPart = attPartIter->second;
  DtInfo << "Station: " << currentStation
         << "Entity Type: " << currentPart->entityType().string()
         << std::endl;
}
```

Header Files have been Reorganized

The header directory structure has been reorganized. Each header file is now in a subdirectory of the *./include* directory. The subdirectory describes which VR-Link library the header file applies to. The following subdirectories have been added:

- *vlutil*
- *matrix*
- *mtl*
- *vlpi*
- *vl*
- *cmdLine*
- *coLromMapper*
- *makLromMapper*.

The *vlutil*, *matrix*, *mtl*, *vl*, *vlpi*, *coLromMapper*, and *makLromMapper* directories contain the header files for the protocol dependent libraries of the same name. The *cmdLine* directory contains the command line utility header files. It does not correspond to a VR-Link library.

To migrate your existing applications to this new directory structure, you can change the include path to include all of the subdirectories, or update all include statements.

To change the include path on Linux, set the paths as:

```
-I{VLHOME}/include
-I{VLHOME}/include/vlutil
-I{VLHOME}/include/mtl
-I{VLHOME}/include/matrix
-I{VLHOME}/include/vlpi
-I{VLHOME}/include/vl
```

To change the include path on Windows, set the paths as:

```
$(MAK_VRLDIR)\include
$(MAK_VRLDIR)\include\vlutil
$(MAK_VRLDIR)\include\mtl
$(MAK_VRLDIR)\include\matrix
$(MAK_VRLDIR)\include\vlpi
$(MAK_VRLDIR)\include\vl
```

To edit all VR-Link include lines to contain these subdirectories, edit them as follows:

Old format:

```
#include <exerciseConn.h>
```

New format:

```
#include <vl/exerciseConn.h>
```

Using the fixHeaders Script

VR-Link 3.12 includes a script to help you with this process. On Windows, the script is *./fixHeaders/fixHeaders.exe*. On Linux, the script is *./fixHeaders.py*. (You must install Python to run the script.

It is strongly recommended that you back up all source before running the script. All source files are parsed regardless of whether any changes are required.

When you run the script, you are prompted to enter a directory in which to begin parsing source files. The script will iterate through the entire directory structure below the chosen directory. It adds the correct directory path to each VR-Link include statement in each source file.

For examples, includes like either of the following:

```
#include <exerciseConn.h>
#include "exerciseConn.h"
```

are changed to:

```
#include <vl/exerciseConn.h>
```

By default, the script edits all files ending in *.h*, *.inl*, *.cxx*, or *.cpp*. You can specify which file extensions to parse by passing in a file listing them as a command line argument.

For example, if you only want to parse *.cpp* and *.hh* files, create a file called *ext.txt* that contains the following:

```
.cpp, .hh
```

Then run the script as:

```
fixHeaders.exe ext.txt
```

Changes to *DtVrlApplicationInitializer*

The *DtVrlApplicationInitializer* class has two new member functions in its public API, loadFile() and saveFile(). The loadFile() member function can load MTL and XML configuration files. The saveFile() member functions saves the state of the *DtVrlApplicationInitializer* class to the specified XML file.

The protected API has also been updated. These changes only affect VR-Link users who have overridden the *DtVrlApplicationInitializer* class. The member variables are now of type DtConfigVariable, which is a templated type that can parse command lines and XML files, and can be registered with the MTL environment to parse MTL files. For an example of using these configuration variables, please see the *DtF18Initializer* class in the F-18 example.

The *DtVrlApplicationInitializer* class now contains a *DtApplicationSettings* object, with which all configuration variables are registered. Also, you can now organize configuration variables into configuration groups. You can register DtConfigVariables with the *DtApplicationSettings* object or a *DtVariableGroup*, which is registered with the *DtApplicationSettings* object. This results in a hierarchy of configuration variables in an XML configuration file.

DtConfigVariables are constructed by specifying:

- The group or application setting to which they belong
- The configuration file variable name (which doubles as the command line argument)
- The initial value it is set to
- A simple description of the variable
- Optionally, a short (one letter) command-line argument
- The scope of the variable (command line only, configuration file only, or both).

You can register DtConfigVariables with a *DtMtlEnvironment* through the registerConfigVar() member function. Once they are registered, values can be filled in as follows:

- From the command line by calling parseCommandLine() on the *DtApplicationSettings* object
- From an XML file by calling loadConfigFile() on the *DtApplicationSettings* object
- From an MTL file by calling loadLispFile() on the *DtMtlEnvironment* object.

Most changes should be invisible for applications using the object. Please see the DtF18Initializer class in the F18 example, for an example of how to extend the *DtVrlApplicationInitializer* class.

Support for DI-Guy Data Updates

VR-Link entity state repositories for lifeforms can now store and transmit the parameters sent in entity state updates by DI-Guy. DI-Guy, from Boston Dynamics, Inc. provides advanced entity models for 3D simulation applications, such as MÄK Stealth or the VR-Forces 3D Front End. This functionality must be turned on in the API and in HLA is only available when DI-GUY extensions have been added to your FOM. For details about how to use these extra parameters please see the class documentation.

Dead Reckoner and Smoother Changes

The following changes have been made to *DtDeadReckoner* and *DtSmoother*:

- *DtDeadReckoner* and *DtSmoother* have been moved to the *vlpi* library. The changes should be source compatible for all VR-Link programs.
- The performance of *DtDeadReckoner* has been improved.
- *DtSmoother* now snaps for its first real update and smooths normally afterwards to prevent smoothing from 0,0,0 on new entities.
- In TENA, you can now turn off dead-reckoning for entities. Previously, it would always dead-reckon. Now you can turn off dead reckoning by calling DtReflectedEntity::setUseDeadReckoning(false).

Improved FOM Mapper API

The FOM Mapper API now fully supports FOMs that do not follow the RPR FOM convention of placing a time stamp in the HLA TAG field, such as the MATREX FOM, via the *DtObjectTagModifier* class. For more information, please see the TAG Modification example in the HLA class documentation.

Subscribing to MOM Federate Classes

You can now easily subscribe to the *Federate* class in HLA's Management Object Model (MOM). The new classes *DtReflectedFederateList*, *DtReflectedFederate*, and *DtFederateStateRepository* allow applications to subscribe to the MOM Federate class, just as they would subscribe to any other HLA object, as long as the MOM is enabled in the RTI's RID file. For about how this class is used, please refer to the VR-Link Class Documentation.

Improved Handling of Environment Process Objects

The stability and performance of Environment Process objects have been improved. VR-Link will now check for consistency in the size of Environment Process PDU's. Previously, if the specification of length in the PDU was incorrect, VR-Link applications might crash. A cache has been added to the Environment Process PDU in VR-Link. This significantly speeds up environment record lookup.

Miscellaneous API Changes

- In the enumeration DtActionId (*disEnums.h*), the value DtActionOther2 was removed. It is not in the specification.
- The *pduWithData.h* header file was changed so that all the signatures that took an int to specify whether it uses a fixed or variable datum were changed to the enumeration, DtDatumRecordType. This makes it more protocol-independent, because the HLA implementation uses the enumeration.
- The *DtTcpSocket* creator can now be changed.
- DtOutputStream::okToPrint() was made const.
- DtOutputStream::threshold() was made const.
- *DtOutputStream* copy and assignment operators were re-scoped to private from protected.
- If an error occurs in DtTcpSocket::sendTo(), the status is now set to be not ok prior to outputting the error message to *DtWarn*.
- *DtSocket* now allows the size of the length field to be 4 bytes
- *DtSocket* flush has been changed to return a status (error, incomplete, complete) and the number of bytes flushed through a parameter. Significantly, *DtNetSocket* and *DtTcpSocket* now behave similarly with respect to flush. Previously the meaning of the return type was different between them.
- Added an optional revision to the RPR FOM Mapper. There was an encoding and decoding size issue with object ID lists, which has been fixed to match RPR 2 draft 17. This was fixed in the RPR FOM Mapper revision 2 for version 2.0017. The command line option for HLA **--rprFomRevision** was added to specify this, as well. The encoding and decoding issues affected the Aggregates, JammerBeams, and RadarBeams objects and the RemoveObjectRequest, AttributeChangeRequest, and ActionRequestToObject interactions.
- New appearance attributes, DtAppLifeformConcealedStationary and DtAppLifeformConcealedMovement, were added for lifeforms. The attributes are bits 30 and 31 in the appearance attribute, respectively. It used to be just DtAppLifeformConcealed, which was bit 30. DtAppLifeformConcealed was kept for backwards compatibility.
- The member function clearDiscoveryCondition() was added to all *DtReflectedObjectLists* (in all protocols) to remove a discovery condition if one had been set.

- The DIS *DtReflectedEntityList* now updates a reflected entity from entity state update PDU's, in addition to the usual entity state PDU's.
- The global functions DtGetUserHomeDirectory() and DtGetMakUserConfigurationDirectory() were added to *vlDir.h*. These functions will be used by MÄK products in the future to store user level configuration data.
- A new member function, isInitialized(), was added to the *DtUtmReferencePoint* class to determine whether the reference point has been initialized or not.
- A new member function, uninitialize(), was added to the *DtUtmReferencePoint* class to set the initialized flag to false and return all attributes to default values.
- Added enumerators to DtLifeFormLand enumeration for entity type, according to the 2006 Enumerations document. These are DtLifeformOther (0), DtDismountedInfantryNonVisible (2), DtCivilianUnarmed (3), DtCivilianWithWeapons (4), and DtLifeFormAnimal (5).
- The peerPort() member function was added to *DtTcpSocket*. It returns the port on the remote host that the socket is connected to.

Documentation Updates

VR-Link Developer's Guide has been updated for this release. The chapter "FOM Agility Examples" has been removed from the manual. The information that was in that chapter is now part of the class documentation. Documentation of API examples has been added to the class documentation for each supported simulation standard.

Bug Fixes

VR-Link 3.12 fixes the following bugs:

- In TENA, the setUseDRThresholding(false) call on the *DtEntityPublisher* did nothing. It now turns off dead-reckoned thresholding when determining whether to send out an entity update in TENA.
- Stop() would not properly stop a *DtThread* that had just been started.
- The HLA Encoded Audio Signal Interaction's data was not being byte-swapped when being sent or received.
- The RPR FOM Mapper was publishing all lifeforms other than dismounted infantry as non-human. Now, land lifeforms are published as human.
- In DIS, VR-Link applications crashed on receipt of an Environmental Process PDU that specified incorrect sizes of environmental records. Now, the PDU is checked for consistency and is dropped if the size is incorrectly specified.
- In DIS, the changing the force of an aggregate did not cause an update to be sent out.
- In TENA, passing a -1 to drainInput() caused the application to hang. It now throws an exception if -1 is passed in.

Known Problems

This release of VR-Link has the following known problems:

- VR-Link correctly encodes and decodes EnvironmentalProcess objects for RPR FOM 2, Draft 14. However, the RPR FOM specifies a way of encoding which prevents VR-Link from correctly decoding modified or custom EnvironmentalRecords. Unfortunately, many MÄK products built with VR-Link use customized EnvironmentalRecords. If you use these products or if you use customized records in your applications, please do not use the RPR FOM 2 Draft 14 FOM Mapper. This is not a problem in other FOM versions, and it has been resolved in future versions of the FOM.
- The RTI 1516 API specifies that all strings passed to and from the RTI are handled as wide strings. VR-Link (and the MÄK RTI) store strings internally as narrow strings. This means that true multibyte wide strings used by RTI Federates may not be handled correctly when received by VR-Link. VR-Link handles name reservation wide strings correctly, however, as this is a 1516-only function. VR-Link provides conversion functions from Wide To Narrow/Narrow to Wide string conversion in *vlStringUtil.h*.
- On non-windows platforms, the classes *DtSharedMemoryPoolManager* and *DtSharedMemoryPoolClient* (*vlShmPool.h*) sometimes have difficulty creating a large memory pool (greater than 1 MB) if the pool name is longer than two or three characters. The classes refuse to create the memory pool even if the requested pool is smaller than the system resource SHMMAX. Setting the pool name to fewer than three characters will work around this problem.
- Section 6.10 in *VR-Link Developer's Guide* describes interoperability issues that arise when you try to run an HLA 1.3 federate with an IEEE 1516 federate. In particular it describes the requirement to use an identical FED or FDD file for all federates. In future releases, we intend to rectify this situation, and allow run-time interoperability between federates that have loaded a 1.3-style FED file and federates that have loaded a 1516-style XML file that otherwise includes the same set of classes, attributes and parameters.