VR-Link 3.10 Release Notes

This release note provides the following release-specific information for VR-Link Release 3.10:

- Systems Supported 2
  - Disk Space Requirements 2
  - Building VR-Link on UNIX 2
  - Compiler Compatibility on Windows 3
  - FLEXlm Support 3
- Backward Compatibility 3
- Network Compatibility 3
- RPR FOM Versions Supported 3
- RTI Support 4
- New Features and Changes 4
  - TENA Support 4
  - Dynamic Libraries for UNIX Platforms 7
  - Library Name Changes 8
  - Changes to the vlutil Library 8
  - Matrix Library Changes 9
  - Miscellaneous API Changes 10
- Documentation Updates 11
- Bug Fixes 11
- Known Problems 12

Document ID: VRL-3.10-3-060801

Systems Supported

Table 1 lists the platforms currently supported by MÄK VR-Link 3.10. 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
SGI IRIX 6.5	MipsPro7.3 C++ compiler (available from SGI)
Sun Sparcstation with Solaris 2.8 or later	SPARCompiler C++ version 5.2 (available from Sun)
Red Hat Enterprise Linux Workstation 3.0	GNU C++ (GCC) 3.2
Red Hat Fedora Core 3	GNU C++ (GCC) 3.4.4
Red Hat Enterprise Linux Workstation 4.0	GNU C++ (GCC) 3.4.4
PC with Windows 2000/XP	Microsoft Visual C++ 7.1 Microsoft Visual C++ 8.0

Table 2: Platforms supported for TENA

Platform	Compiler
Red Hat Fedora Core 3	GNU C++ (GCC) 3.4.4
Red Hat Enterprise Linux Workstation 4.0	GNU C++ (GCC) 3.4.4
PC with Windows 2000/XP	Microsoft Visual C++ 7.1

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.

Building VR-Link on UNIX

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.

FLEXlm Support

VR-Link 3.10 uses FLEXlm 10.8 for Windows and Linux versions. Solaris and IRIX versions continue to use FLEXlm 9.2.

Backward Compatibility

VR-Link 3.10 is not fully compatible with previous releases due to changes to *vlutil* and to the link line for UNIX. For more details, please see "Compiling and Linking," on page 5 and "Changes to the vlutil Library," on page 8.

Network Compatibility

HLA only

VR-Link 3.10 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
- DMSO RTI NG 6.0
- Pitch RTI 1.3 C++ interface.

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 3.10 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 3.10 uses RPR FOM 1.0.

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 3.10 has been tested with DMSO RTI NG v6, Pitch RTI 1.3, and MÄK RTI 3.0.

VR-Link 3.10 may work with older versions of the MÄK RTI, but we recommend using it with release 3.0.

! VR-Link 3.10 for Sun Solaris is not compatible with versions of the MÄK RTI prior to version 2.4.

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 3.10 has the following changes and new features:

- Support for TENA
- Use of dynamic libraries for UNIX
- Changes to library names
- Changes to the vlutil library
- Changes to the matrix library
- Link line change for UNIX
- Miscellaneous API changes
- Support for FLEXlm 10.8 (please see “FLEXlm Support,” on page 3).

TENA Support

TENA (Test and Training Enabling Architecture) is an interoperability architecture used to pass data in the form of object updates and messages (like HLA objects and interactions) from one application to another. It was developed by the Foundation Initiative 2010 (FI2010) program, which is managed by the Director of the Central Test and Evaluation Investment Program (CTEIP). CTEIP is part of the Office of the Undersecretary of Defense for Test and Evaluation (DOTE-OSD). TENA was developed to serve as the interoperability architecture for the Live Range community. The direction of TENA is managed by the TENA Architecture Management Team (AMT) – a group that meets about once a quarter.

TENA uses a data definition called a Logical Range Object Model (LROM) (comparable to an HLA FOM.) An LROM is a set of object/message definitions that may be used in an exercise. These object models are available via the TENA SDA (Software Development Activity) website, and there are some standard TENA object models that were developed by the TENA community. TENA also uses a process called an execution manager that each application must connect to in order to be a part of the exercise.

TENA provides much of the same functionality as VR-Link, such as mechanisms for receiving callbacks for object updates/received messages, so the objective of VR-Link is to provide a protocol-independent API over the existing TENA API to allow existing VR-Link applications to switch to TENA with minimal effort.

Compiling and Linking

The TENA version of the *vl* library is *libvlTENA.so* (UNIX), and *vlTENA.lib* (Windows). We continue to use MD for static libraries and RT for dynamic libraries on Windows (there are 4 libraries: *vlTENAMD.lib*, *vlTENAMDd.lib*, *vlTENART.lib*, and *vlTENARTd.lib*).

When compiling a VR-Link application for TENA, the macro DtTENA=1 must be defined.

Creating an Exercise Connection for TENA

In order to create an exercise connection, the user must specify the ORBdefaulInitRef (a string that tells a TENA application where to connect to the execution manager), execution name, session name, and an LROM Mapper (this will be discussed later). This may be done through a *DtExConnInitializer* (as in HLA or DIS). There is another constructor in which these values may be passed in individually, but in order to use this constructor the user must have implemented an LROM Mapper in code and pass in an instantiated LROM Mapper. The reason no default LROM Mapper is provided in the TENA version of VR-Link, as there is in HLA is that providing an LROM Mapper within the *vlTENA* library would force users to link in those object models that the LROM Mapper uses, in order to use VR-Link with TENA, regardless of whether they want to use those objects or not.

The exercise connection loads the LROM Mapper DLL specified (if the *DtEx-ConnInitializer* is used), connects to the TENA exercise with the execution name, session name, and ORBdefaultInitRef parameter specified, initializes the LROM Mapper, and instantiates an AMO (Application Management Object) publisher.

An AMO is a standard TENA object that is supposed to be used by every TENA application. It gives other applications information about the local application. The exercise connection, by default, publishes an AMO object, updates it at a specified rate (the rate is part of the AMO's state repository and is defaulted to 30 seconds), and also updates the number of objects published and proxies (TENA reflected objects) held. You can disable creation of the AMO by setting a flag in the initializer or constructor.

The TENA DtExerciseConn will throw an exception if:

- No LROM Mapper is specified/passed in
- The application cannot join the exercise
- It is told to publish an AMO, but no mapping exists in the LROM Mapper

TENA Support

VR-Link was built against the 5.1.1 version of the TENA Middleware.

The Windows libraries were built against the Windows XP MSVC++ 7.1 versions of the TENA Middleware.

Because the *vlTENA* library provides some utility conversions between very commonly used TENA object models and VR-Link classes, the following object model libraries were linked into the *vlTENA* library:

- TENA-PlatformType-v1
- TENA-Time-v1.1
- TENA-UniqueID-v2.

The object model libraries linked into the Common Object LROM Mapper are:

- DIS-EntityType-v2
- TENA-Acceleration-v1-noConversion-v2
- TENA-Acclereation-v1
- TENA-Affiliation-v1
- TENA-AMO-v1-basic-v1
- TENA-AMO-v1
- TENA-AngularAcceleration-v1
- TENA-AngularVelocity-v1
- TENA-Engagement-v3.1-basic-v1
- TENA-Engagement-v3.1
- TENA-Orientation-v1
- TENA-ORM-v1
- TENA-Platform-v3.1-basic-v1
- TENA-Platform-v3.1
- TENA-PlatformDetails-v3.1-basic-v1
- TENA-PlatformDetails-v3.1
- TENA-Position-v1-fullConversion-v2
- TENA-Position-v1
- TENA-SRF-v1-utils-v2
- TENA-SRF-v1
- TENA-Time-v1.1-fullConversion-v2

- TENA-Time-v1.1
- TENA-TSPI-v4-positionConversion-v3
- TENA-TSPI-v2
- TENA-UniqueID-v2-userDefined-v1
- TENA-UniqueID-v2
- TENA-Velocity-v1-noConversion-v2
- TENA-Velocity-v2

The MÄK LROM includes the previously listed object model libraries, plus:

- JNTC-LROM2005S1-v2-basic-v1
- JNTC-LROM2005S1-v2
- MAK-LROM-v2-basic-v1
- MAK-LROM-v2
- TENA-EmbeddedSensor-v2
- TENA-EmbeddedSystem-v2
- TENA-EmbeddedWeapon-v2
- TENA-Munition-v2.1
- TENA-Organization-v1
- TENA-SRFserver-v1

Dynamic Libraries for UNIX Platforms

VR-Link is now distributed with dynamic libraries instead of static libraries on UNIX (Linux, IRIX, Solaris) platforms. Dynamic libraries were required for the TENA LROM Mapper libraries, as well as HLA FOM Mapper libraries. Typical LROM Mapper and FOM Mapper libraries had many symbol dependencies on other VR-Link libraries too, this led to a number of problems, including memory curruption errors when a Mapper library was closed and duplicate global symbols. Using dynamic libraries solves these problems.

These changes should be semi-transparent to customers. It does not effect the link line.

Library Name Changes

- The *vl5* library has been renamed *vlDIS*. The number 5 was misleading as the library was not just for DIS version 5, but also for DIS version 4 and 6. You must change your link line to compile for DIS.

  The link line has changed to use the renamed *vlDIS* library. The old link line is as follows:

  ```
  -lvl5 -lmtl -lmatrix -lvlutil
  ```

  The new link line is:

  ```
  -lvlDIS -lvl -mtl -lmatrix -lvlutil
  ```

- A new protocol-independent Simulation Library has been created. Classes such as *DtEntityType*, *DtEntityIdentifier*, *DtSimulationAddress*, and so on, have been extracted and added to a protocol neutral simulation library. The library is named *vl*, without the protocol suffix. Each protocol-specific library (*vlDIS*, *vlHLA13*, and so on) will require *vl* to link. This allows common data types to be used in simulators inside libraries which would otherwise be protocol independent.

Changes to the *vlutil* Library

The VR-Link utility library *vlutil* has been updated. The *vlutil* library contains two types of classes: OS-independent classes for OS-dependent functionality, and common data structures used by VR-Link and other MÄK products.

- All file names have been changed to use the "vl" prefix and use consistent capitalization. For example, files like *hashlist.h* have been renamed *vlHashlist.h*. This will allow developers to easily identifiy MÄK data structures without confusing them with system includes. Because we recognize that legacy systems may not be able to change, all old headers have been copied to *./include/compatibility*. Projects which need to maintain source compatability should add *$VR-Link/include/compatibility* to the search path on their compile line.
- The class *DtDynamicLibrary* has been modified and is not backwards compatible. In past releases it was difficult to modify how the dynamic library was opened, something that is fairly common to do. To fix this we needed to break source compatability. The constructor is now protected. This class now implements the virtual constructor design pattern. A new protected virtual function called open() has been provided, as well as a static create() method which needs to be called to create a *DtDynamicLibrary* instance. The only change customers who use this class will need to make is:

  ```
  DtDynamicLibrary *dl = new DtDynamicLibrary("foo"); // Old code.
  DtDynamicLibrary *dl = DtDynamicLibrary::create("foo"); // New Code
  ```

- The following classes and files were removed from the VR-Link libraries. The declarations as well as the implementations can be found in the *./examples/obsolete* directory. These files are available for backwards compatability with no restrictions.
  - *DtOldHashlist* and *DtHashedItem* were removed. These classes were in *oldHashList.h* They were deprecated in VR-Link 3.3.
  - *declSmartPtr.h*, which contains the macro DtDECLARE_DEFINE_SMART_PTR has been removed. VR-Link now contains Boost Smart Pointers. For details, please see *vlSmartPointers.h*.
- The following classes are deprecated in this release of VR-Link. These classes are provided for backwards compatability only.
  - *DtList*. *DtList* should be replaced by std::list. MÄK products will continue to use *DtList* in existing code for some time in order to provide source code compatability. New code will use std::lists.
  - *DtHashlist*. *DtHashlist* should be replaced with std::map. Though the C++ Standard Map is not a hashlist, a std::map has been shown to have significant performance gains on *DtHashlist*. MÄK products will continue to use *DtHashlist* in existing code for some time in order to provide source code compatability. New code will use std::maps.
  - *DtHashItem* and *DtBaseHashKey*. These classes were required to support *DtHashlist*.
- The following classes were condensed into single files:
  - The class declaration for *DtHashItem* has been moved to *vlHashlist.h* as this is a helper class for *DtHashlists*.
  - The classes *DtIntHashKey*, *DtBaseHashKey*, *DtPtrHashKey*, *DtstrHashKey*, *DtWStringHashKey* have been moved to *vlHashKeys*, and deprecated.

Matrix Library Changes

The following classes have been added to the VR-Link matrix library.

- *Dt3dPoint* - A class to represent a 3D point in space.
- *Dt4dPoint* - A class to represent a 4D point in space. This is useful for homogeneous coordinates.
- *Dt3dExtent* – A class to represent a 3D extent in space.
- *Dt3dBoundingVolume* – A class to represent a 3D bounding volume.
- *Dt3dChord* – A class to represent a 3D chord in space.

Miscellaneous API Changes

- The command line parser has been changed so that the application initializer and the *cmdLine* class can specify "Press any key to continue".

  This defaults to true in Windows (the application will wait until the user presses a key after syntax help or the version is displayed); false on other platforms.

- The maximum size read from a socket is now set at socket construction by DtMaxPacketSize() for the *DtNetSocket* and *DtAsynchSocket* classes. Previously, this could be changed after the socket was created, and could cause a buffer overrun.
- The *DtFOM* classes now handle attribute handle/name, and parameter handle/name lookup. Specifically, a federate can now look up an attributeName from an attribute handle using the *DtObjClassDescriptor* classes. By forcing this abstraction none of the encoder or decoder functions depend on the rtiAmbassador, rather they depend on a *DtFOM* implementation. You can now override *DtExerciseConn* with a *DtFOM* subclass and use the decoders and encoders without a network connection.
- *DtEntityIdentifier* now has an ostream operator.

New Functions

The following new functions have been added to the API:

- Added DtYield() to *vlProcessControl.h* which yields the current thread. DtSleep() does not necessarily yield processor control.
- Added robust platform independent thread (and thread safety) classes and added an example of their use to the *./examples* directory.
- Added a masked/cloaked bit in disEnums, and entitySR (bit-31 of appearance).
- Added DtBaseEntityStateRepository::clearHostGlobalId() to be used to unattach an entity from a host. The old mechanism of doing this, by passing an empty globalId to setHostGlobalId() did not work because in DIS 0:0:0 is a valid attachment.
- Added function const char *DtCommentPdu::comment() const to *commentPdu.h*. This function was missing in the DIS interface, but was present in the HLA interface.
- Added setDestinationAddress(DtInetAddr) to *DtVrlApplicationInitializer* in DIS (*dExConnInit.h*), because it was hiding the base class function.

Documentation Updates

A new chapter has been added to *VR-Link Developer's Guide* that describes support for TENA.

Section 6.10 in *VR-Link Developer's Guide* describes interoperability issues for HLA 1.3 federates and IEEE 1516 federates running in the same federation execution.

In section 5.10.2, "Setting Articulated Parts Data" in *VR-Link Developer's Guide*, there are errors in two code samples.

The sample that starts:

```
DtArtPartSpec* specList =
```

should read:

```
DtArtPartSpec* specList [] =
```

DtApAzimuth is misspelled in the following code sample:

```
artParts->setArtParamVal(DtPrimaryTurret1, DtApAzimuth,DtDeg2Rad(90.0));
```

Bug Fixes

VR-Link 3.10 fixes the following bugs:

- In *DtReflectedEntityList*, lookup(DtEntityIdentifier) would not return a valid ID if the EntityID arrived in the HLA discovery message. This class has been fixed to correctly register the EntityID when it arrives in either the Discovery or later update messages.
- The verbose warning of duplicate joins of multicast addresses now outputs the address and device instead of outputting the device twice.
- In *DtHlaObject*, clearRequestedAttributes() now clears the attributes in HLA 1516.
- A file descriptor is no longer leaked for every Lisp (MTL) file that is opened.
- The checker for Variable Axis Records now catches changes in the data within the variable records.
- WSAStartup is now called only once when *DtSockets* are created.
- Command line strings with spaces, such as --stringArgument "This string has spaces" no longer causes a usage error.

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 Environmental-Records. 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.