VR-Link 5.1 Release Notes

This release note provides the following release-specific information for VR-Link Release 5.1:

Document ID: VRL-5.1-3-140805

Systems Supported

Table 1 lists the platforms currently supported by MÄK VR-Link 5.1. 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
Red Hat Enterprise Linux Workstation 4.0	default compiler
Red Hat Enterprise Linux Workstation 5.0, 6.0 (32 bit and 64 bit libraries)	default compiler
SUSE 11 Linux (32 bit and 64 bit libraries)	default compiler
PC with Windows XP/Vista/Windows 7	Microsoft Visual C++ 7.1 Microsoft Visual C++ 8.0 Microsoft Visual C++ 9.0 (32-bit and 64 bit) Microsoft Visual C++ 10.0 (32-bit and 64 bit) Microsoft Visual C++ 11.0 (32-bit and 64 bit)

VR-Link for C# supports VR-Link 5.1, Microsoft Visual C++ 10.0 and MSVC++ 11.0, 32 bit and 64 bit.

Disk Space Requirements

Linux installations require approximately 1.2 GB of disk space. On Windows, you need approximately 1.5 GB of disk space.

Compiler Compatibility on Windows

MÄK provides versions of product releases that have been compiled with different versions of Microsoft Visual C++. When you run MÄK products together, for example, the Logger and a VR-Vantage application, we strongly recommend that you run versions compiled with the same compiler. Mixing products compiled with different versions of the compiler on the same computer can result in program instability.

License Manager

To run the licensed version of VR-Link, you must install license management software. VR-Link 5.1 uses FLEXlm 11.11 for all versions except the Windows VC++ 7.1 version, which continues to use FLEXlm 11.6. If you are upgrading from a version of the VR-Link that used an older version of FLEXlm, you must upgrade your license management files. You do not need a new license. Licenses are forward compatible.

The License Manager files are not part of the VR-Link installer. You can download them at:

- Windows:
  ftp://ftp.mak.com/out/MAKLicenseManager-win-setup.exe (for 32-bit systems)
  ftp://ftp.mak.com/out/MAKLicenseManager-win64-setup.exe
- Linux:
  ftp://ftp.mak.com/out/MAKLicenseManager-linux-setup.tar.gz (for 32-bit systems)
  ftp://ftp.mak.com/out/MAKLicenseManager-linux64-setup.tar.gz

A license manager FAQ is available at:

http://www.mak.com/license-support.html

Network Compatibility

HLA only

VR-Link 5.1 is compliant with:

- RPR-FOM 0.5, 0.7, 0.8, 1.0, and a subset of 2.0 (draft 6, 14, 17, and 20)
- MÄK RTI 2.x, 3.x, 4.x (HLA 1.3 and 1516-2000 only)
- MÄK RTI 4.0.4 or later (HLA Evolved)
- Pitch RTI 1.3 C++ interface.

Other RTIs that support the HLA 1.3 specification, the SISO DLC HLA API 1516 version of the IEEE 1516 specification (SISO-STD-004.1-2004), and HLA Evolved. To use an RTI with VR-Link it must use the same operating system and be built with the same compiler.

DIS only

VR-Link 5.1 supports DIS 4, 5, 6, and 7 and can therefore interoperate with DIS applications of any of these versions.

Backward Compatibility

DtReflectedObject lists no longer use the deprecated class *DtObjectIdHashList*. Instead they use std::map which is much faster. Unfortunately this does cause a slight change to all *DtReflectedObject*. Previously they were *DtListedObj*, and now they are not. This should not affect your VR-Link code unless you have created new HLA classes (for a FOM Mapper for example).

VR-Link 5.0 renamed header files to conform to MÄK file naming conventions. If you have not already migrated to these new file names, the next two sections describe the changes and the migration script.

Using the Compatibility Directory

VR-Link 5.1 has a *compatibility* directory in its *include* directory. This directory has the same directory structure as the main *include* directory with header files that have the same names as previous versions of VR-Link. The header files in this directory do not duplicate the declarations in the renamed header files, they include the renamed header files themselves. To use the *./include/compatibility* directory, add it to your project's include directory list. You will not have to update the include statements in your project files.

Using the Migration Script

VR-Link provides a script (*./compatibility/updateIncludes.sh* on Linux, *./compatibility/updateIncludes.exe* on Windows) that will go through all of your source files and modify the include statements to reference the correct header files.

The update script uses the mapping files located in *./compatibility/mappings* to update include statements for the renamed VR-Link header files in a given set of source directories. The script searches all input directories for header (*.h and *.hpp), inline (*.inl), and source files (*.c, *.cpp, and *.cxx), and checks the include statements in those files against the mapping files. For the script to correctly map the header file names, VR-Link include statements must specify the VR-Link library path. This is so the scripts can identify which mapping file to use to map the include statement. Therefore, the base VR-Link include directory must be set as the include path for projects, instead of adding the individual library directories (*include/vl*, *include/mtl*, and so on.).

The script does the following:

- If a header file has been renamed, the header file in the include statement is renamed.
- If a header file has been removed, the include statement is removed.

The command syntax for the script is as follows:

```
updateIncludes [options] directory1 ... directoryn
```

where:

- `directory1 ... directoryn` is a list of directories in which the script will search for source and header files for updating.
- `-d` displays debug level information about the files and include statements as they are processed
- `-h` provides information about the various flags.
- `-v` displays slightly more verbose output than `-d`.
- `-x` tells the script to perform the replacements. If the script runs without the `-x` flag, it outputs the commands that will be run if the `-x` flag is specified. This is so you can verify the changes that are going to be made before the script changes all the files.

RPR FOM Versions Supported

VR-Link 5.1 has built-in support for versions 0.5, 0.7, 0.8, 1.0, and 2.0, drafts 6, 14, 17, and 20, of the RPR FOM. By default, VR-Link 5.1 uses RPR FOM 1.0.

VR-Link does not officially support RPR FOM 2 Draft 18. 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, 2.0017, 2.0020) 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.

RPR FOM 2.0 Draft 20 and HLA Evolved

In previous versions of VR-Link, custom extensions to the RPR FOM were included in the HLA Evolved XML file. For VR-Link 5.1, we provide an XML file that only includes the standard RPR FOM 2.0 draft 20 and we use FOM modules for VR-Link extensions. This should make it easier for customers who do not need the FOM extensions to interoperate with non VR-Link federates. The files supplied are:

- *RPR_FOM_v2.0_draft20_1516-2010.xml*
- *DIGuyModule.xml*
- *VR-Link_LgrControl_evolved*.xml
- *VR-Link_ViewControl_evolved*.xml
- *VR-Link_VrlExt_evolved*.xml.

RTI Support

VR-Link 5.1 has been tested with the MÄK RTI 4.2 for HLA 1.3, HLA 1516, and HLA Evolved. It has also been tested with Pitch's pRTI 4.3.1 for HLA 1.3 and HLA Evolved.

If you use the MÄK RTI, remember to make sure that the 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.

Applications built with VR-Link 5.1 for HLA can use any RTI that conforms to the relevant dynamic link compatible HLA standard (1.3, SISO DLC 1516, HLA Evolved) and was built using the same operating system and compiler.

New Features and Changes

VR-Link 5.1 has the following new features:

- A C# API that provides the same protocol-independent support for DIS and HLA that is provided by the C++ APIs. For complete details, please see section 9 in *VR-Link Developers Guide*.
- Full support for RPR version 2.0020, which is also the first official RPR 2.0 version. You can access this version by simply selecting RPR version 2.0 in your HLA Exercise Connection and choosing an official RPR 2.0 FOM. Most of the changes in this release of RPR are fairly small, however it does include a significant fix in the silent entity structure of aggregates that used to limit appearance records to just one. If you are using an older version of the RPR FOM, this limitation continues to exist.

  The 2.0020 XML file does not include VR-Link extensions. They are not provided as FOM modules. For details, please see "RPR FOM 2.0 Draft 20 and HLA Evolved," on page 6.
- Support for generic attributes and parameters. For more information, please see "Generic Attributes and Parameters," on page 9.
- HLANetdump now has the ability to subscribe passively using the `-p` or `--passive` command line arguments. When subscribed passively, netdump prints everything it sees in the network, but does not actively request any data. Therefore, your network traffic will remain exactly what it was if netdump was not active.
- VR-Link now includes callbacks for Save and Restore in HLA. You can add your own callbacks when a Save or a Restore gets requested by calling addFederate-SaveCb and addFederateRestoreCb in your DtExerciseConnection. If you do not have any callbacks for save and restore, VR-Link immediately sends a Federate-Save/RestoreComplete message, thus allowing the exercise to continue even if nothing locally gets saved.
- The use of getValuePointer has replaced getValue as the method for extracting data from the RTI attribute/parameter handle value pair sets. The getValuePointer method is more efficient, because it uses the buffer in the handle value set directly instead of copying the data into a separate buffer. By default, this change to use getValuePointer is enabled when the MAK RTI 4.3 or newer is used. For other RTIs or older MAK RTI versions, the getValue method will be used.

  There are 2 new exercise connection initializer (DtExerciseConnInitializer) parameters to control the use of getValue versus getValuePointer

  – setForcedUseGetValuePointer - Forces the use of getValuePointer without regard to the RTI version.

  – setDisabledUseGetValuePointer - Disables the use of getValuePointer without regard to the RTI version.

- There is a new example, setData, that shows how to use the setData and data interactions to send and receive generic data without creating a new intraction or PDU type.
- The Intel Thread Building Blocks library has been added to the VR Link product.

  If you use these new, threaded features, you might need to make your own implementations thread safe. For example, be careful of using shared singleton or global variables in your tick() implementations.
- HLA Update Requests are now disabled if the autoProvide flag is enabled in your FOM. There is no need to request updates if you are guaranteed to receive them anyway.
- Added a means of getting the sim time of the last update for TSPI data. The *DtApproximator* and *DtDeadReckoner* classes now include the following functions to get that information:

```
//! Get the last update time of the absolute location.
virtual const DtTime& locationUpdateTime() const;

//! Get the last update time of the absolute velocity.
virtual const DtTime& velocityUpdateTime() const;

//! Get the last update time of the absolute orientation.
virtual const DtTime& orientationUpdateTime() const;
```

- DtTaitBryan's equality checks now check for equal angles, not equal values. So, for example, 0 and 2PI would be the same value.
- The method of handling default dead reckoners and smoothers has changed. You can now set the default dead reckoner and the default smoother using the following static functions in *DtBaseEntityStateRepository*: setDefaultDeadReckoner, setDefaultSmoother, setDefaultRelativeDeadReckoner, setDefaultRelativeSmoother. This default will apply to all entities that do not have a specified dead reckoner or smoother. In addition, entities now manage their own memory when it comes to setting individual approximators. If you pass an approximator to an entity to be used it will be cloned and you no longer need to worry about making sure you delete it only after the entity gets deleted.
- DtMutex now supports a non-blocking lock: tryLock. It returns false if it cannot lock a mutex instead of deadlocking until released.
- There is a new, simpler way to do ownership transfer using the classes *DtDefaultOwnershipHandler* and *DtPartialOwnershipHandler*. A new example, *objectOwnership.cxx* is included, which shows how the new way of handling ownership works.
- Upgrade to Boost 1.55. This upgrade has an extra dependency in Linux. If you are upgrading Linux makefiles from a previous version of VR-Link, you must add a link to the *librt* library. An undefined reference to a symbol "clock_gettime" indicates that you have not linked to this library. *librt* is a standard system library, so you do not need to download any extra packages.

Generic Attributes and Parameters

Generic attributes and parameters are a way of accessing extended information in your FOM that is not normally supported. For example, suppose your FOM, based on RPR contains an extra attribute on entity objects called "RadarSignature". Once generics are enabled in VR-Link, all you have to do is ask for your data:

```
int size;
const char* val = entity->stateRep()->getAttributeByName(
  "RadarSignature", size, entity->hlaObject());
```

Interactions work in a similar manner:

```
val = fireInteraction->getParameterByName("RadiationEmitted", size);
```

There is no longer a need to write a code generator, or to hand model anything. Getting (and setting) your new attributes is simple and effective.

There are two ways to enable generics. The first is the easiest. If you enable generics in your initializer, all unknown attributes will be automatically handled:

```
appInit.setGenericAttributes(true);
```

Alternatively, you can choose to only decode generics on single classes, which might be faster than decoding every entity:

```
DtInterClassDesc* interClass =
     exConn.fom()->interClassByName("WeaponFire");
interClass->enableUnknownParameters();
```

Previous versions of VR-Link already supported generic handling of new classes using DtUnknownInteraction and DtUnknownHlaObject. With VR-Link 5.1, you can have hybrid objects, where some are known and some are generic.

So why would you not just use generics? Generic attributes do not have type safety. They are only sent and received as byte arrays and it is up to you to encode and decode them correctly. For attributes that VR-Link knows, this is handled for you already, letting you deal with the data in a more natural form. The same situation applies to code generated objects. However, there are many situations where this might be more work than necessary, and generics let you work with your data as fast as you can change your FOM.

Documentation Updates

VR-Link Developers Guide has been updated to cover VR-Link for C#.

Fixed Bugs

This release fixes the following issues that were problems in previous releases:

- The Designator object now correctly encodes DIS 7 Attribute records. There was an error in VR-Link that made objects of those type skip the encoding process. 50865
- You can now specify FOM Modules in HLA Evolved through a configuration file in addition to doing it through code or through the command line. The *F18.xml* file included in the F18 examples directory shows proper usage:

```
<!--Fom Modules (HLA Evolved)-->
<fomModules>
<module value="DiGuyModule.xml"/>
</fomModules>
```

  48089
- VR-Link did not load *HLAstandardMIM.xml* if it was not in the executable's directory. Now the HLA Standard MIM (*HLAstandardMIM.xml*) is read from memory if the file cannot be found or read. 52417
- The default setting for the dead reckoning algorithm has been switched from DtDrOther to DtDrStatic due to the fact that some simulations have problems understanding how to define 'other'. If you actually want dead reckoning to be calculated when moving, change the algorithm to something other than static. 51962
- Added ability to set use a custom threshold value aside from the class default. 52813
- Object classes that are subsets of AggregateEntity are now treated as aggregates and thus correctly dead reckon. 52371
- Many Code Generator bugs were fixed. 49926
- Improved ownership management. 47695
- VR-Link failed to load the MIM when full paths were specified. 51567
- VR-Link now allows FOMs that define a transport type other than best effort or reliable. It will interpret unknown transport types as best effort. Previously it just exited, claiming it could not process the FOM. 50059
- In HLA 1.3 and 1516, you can now set the federate name by calling either federateName or federateType. They will both do the same thing. If you are using HLA Evolved, the federate name must be unique, and the federate type does not need to be. 45373
- An issue assigning the device address to the loopback device in Linux has been fixed. Previously it was determining that loopback meant you wanted a broadcast address. 41334
- The F18 example emissions have been named, to allow repeatability in names. Previous versions just had the name assigned by the RTI. 52263
- Fixed a multicast problem in Linux. 51296

- Improved error handling for RTI exceptions. 52293
- Turning on and off smoothers could get VR-Link into a corrupt state.43480
- Memory leak was eliminated in the use of DIS PDU encoder/decoders. 51612
- The reflected attributes for the DI-Guy representation as a separate object are now correctly accessed. 51545
- Underwater Acoustics reflected object lists reflected objects of other types. 51439
- Under some circumstances involving the MÄK RTI Assistant, VR-Link would return the wrong license count, even though it would correctly check out the right number of licenses used. The correct license count is now returned. 50864

Known Problems

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

- The default configuration for all VR-Link for C# examples is 64-bit debug. If your target is 32-bit or release, you must change the configuration.
- The VR-Link Code Generator may have problems with complex FOMs. We have made every effort to verify that it works with a broad range of FOMS and FED files. However, we are unable to verify that it works for all FOMs. In addition, small errors in the FOM may prevent the Code Generator from correctly processing the FOM at all. If the Code Generator is not able to process your FOM, we would be happy to help. Please send a copy of your FOM to support@mak.com and we will work with you to correctly generate your FOM.
- 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 5.10, “Interoperability Between HLA 1.3 and IEEE 1516 Federates”, in *VR-Link Developers 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.