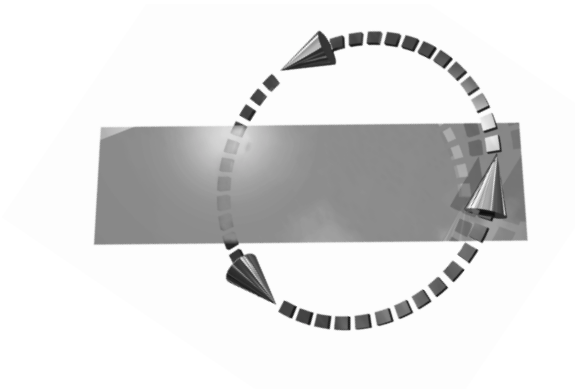

MÄK VR-Link 3.9 Release Notes

This release note provides the following release-specific information for MÄK VR-Link Release 3.9:

Document ID: VRL-3.9-3-031119

Systems Supported

Table 1 lists the platforms currently supported by MÄK VR-Link 3.9. 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 8.0	GNU C++ (GCC) 3.2
PC with Windows NT/2000/XP	Microsoft Visual C++ 6.0 SP5 Microsoft .NET

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 3.9 is highly compatible with VR-Link 3.8.x-ngc. For example, the *talk* and *listen* example programs from VR-Link 3.8-ngc and previous versions compile and link unmodified against VR-Link 3.9.

Network Compatibility

HLA only

VR-Link 3.9 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 2.1
- DMSO RTI NG 4.0, and 6.0 (but not 5.0). The Linux version is not compatible with DMSO RTI NG.

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 3.9 has built-in support for versions 0.5, 0.7, 0.8, 1.0, and 2.0, drafts 6 and 14, of the RPR FOM. By default, VR-Link 3.9 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, 20006, or 20014) 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(20006));
```

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`. The keywords for specifying different versions of the RPR FOM with the `-f` startup option 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 draft 6: `vlRPR20006`
- RPR FOM 2.0 draft 14: `vlRPR20014`.

RTI Support

VR-Link 3.9 has been tested with DMSO RTI NG v6 and MÄK RTI 2.1.

The Linux version of VR-Link 3.9 is not compatible with the DMSO RTI (because the DMSO RTI does not support Red Hat Linux 8.0.) If you need a Linux version of VR-Link that is compatible with the DMSO RTI, contact us at support@mak.com.

VR-Link 3.9 may work with older versions of the MÄK RTI, but we recommend using it with release 2.1.

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

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

- Support for the IEEE 1516 RTI specification (RTI 1516)
- Support for the FED and FDD (XML) file formats.
- The print utilities have been reorganized.
- Miscellaneous changes to the API
- New make system.

Support for the IEEE 1516 RTI

VR-Link 3.9 provides partial support for the RTI 1516 specification. As it did with previous versions of the RTI specification, VR-Link's protocol independent interface allows you to create applications for use with RTI 1516 largely without regard to the details of the RTI. Any code that you have written for use with the RTI 1.3 specification should be usable with RTI 1516 with very few, if any, changes.

VR-Link supports the current draft of the IEEE 1516 C++ API maintained by the SISO Dynamic Link Compatible RTI API Product Development Group. This API deviates from the original C++ API that was distributed with the IEEE 1516 standard. The original IEEE 1516 API does not support dynamic link compatibility among RTIs. This means that under the 1516 standard, you must link an application to the RTI version that you want to run with. Because VR-Link uses the SISO draft of the 1516 API, an application built with VR-Link and linked to an RTI that supports the SISO draft, such as the MÄK RTI 2.1, can use any other RTI that was linked against the SISO draft without recompiling the application.

As part of support for RTI 1516, the following changes have been made:

- The library *libvlRPR* is replaced by *libvlHLA13* and *libvlHLA156*
- The *./binRPR* directory is replaced with *./binHLA13* and *./binHLA1516*
- An FDD file for RPR FOM 2.0 draft 14 is provided (*VR-Link20014.xml*).

To update existing code to work with this release of VR-Link, do the following:

- Code that communicates directly with the RTI must be updated
- If you are building for HLA, you must still set #define DtHLA=1. If you are building for RTI 1516, you must also use #define DtHLA_1516.

Building Applications for use with the RTI 1516 Specification

When you build applications for use with the RTI 1516 specification consider the following issues and requirements:

- We have tried to maintain source compatibility between the 1.3 and 1516 versions of VR-Link. Most types in the 1516 RTI API have the same name as in the 13 RTI API. The exceptions are as follows:
  - In previous versions of VR-Link, RTI classes were prefaced with RTI::. This convention effectively created an RTI namespace. The new RTI API has a true namespace called "rti1516". In *rtiCompatability.h* we rename the namespace to RTI to provide backwards compatibility. This was intended by the RTI API standards committee.
  - The RTI 1516 API eliminated the *AttributeHandleValuePairSet* class. It was replaced by a std::map<AttributeHandle, VariableLengthData> type. To preserve source compatibility with VR-Link encoders and decoders that were written to use the *AttributeHandleValuePairSet*, we added a wrapper class, called *AttributeHandleValuePairSet*, which is passed to all encoders and decoders. This preserves source compatibility. The definition of this class is in the file *rtiCompatability.h*.
- When you create an object publisher, you can provide a name for the object or let the RTI name it. In the RTI 1.3. API, you could choose a name for an object at any time. In the RTI 1516 API, if you want to provide a name for an object, you must reserve the name before you create the object. If you use the VR-Link protocol independent API, VR-Link does this work for you. However, the way that VR-Link handles name reservation is not as efficient as if you do it yourself.

  To reserve a name you must make a call to the RTIambassador, as follows:

  ```
  ExConn->rtiAmb()->reserveObjectInstanceName(theName);
  ```

  Then tick the RTI. The RTI then calls the Federate Ambassador to let the federate know if the name reservation succeeded or failed.

  VR-Link keeps a list of all names that have been successfully reserved. When you create a publisher for an object and specify a name, VR-Link checks the list of reserved names, if the name is not found, it requests the reservation and ticks the RTI until it gets a response or times out. This can be time consuming, but it provides source code compatibility with previous versions of VR-Link.

  You can set the number of times the publisher ticks the RTI, and the duration of the ticks using static methods in *DtHlaObjectManager*, as follows:

  ```
  static double requestNameTickTime();
  static void setRequestNameTickTime(double time);

  static int numberOfRequestNameTries();
  static void setNumberOfRequestNameTries(int tries);
  ```

If the publisher does not get a successful name reservation after several tries, it allows the RTI to pick the name.

VR-Link applications can make RTI calls to reserve names before they create publishers. If you want to create multiple objects it would be best to batch these requests. This could greatly improve performance. VR-Link automatically registers to get all name reservation success notifications.

Compatibility of RTI 1.3 and RTI 1516 Applications

If you use the MÄK RTI 2.1, applications built using the VR-Link 1516 API and the VR-Link 1.3 API are link compatible. For example, if you build the listen example with the 1516 API and the talk example with the 1.3 API, they can interoperate.

Support for FED and FDD (XML) Files

VR-Link 3.9 supports HLA configuration files in both the FED and FDD (XML) formats. When you create an exercise connection, you can specify a *.fed* file, *.fdd* file, or *.xml* file. You can use either format with both RTI 1516 and RTI 1.3.

If you do not pass a filename to the exercise connection, VR-Link uses the federation execution name as the filename. If you are building for RTI 1516, VR-Link looks for a file name *federation_execution.xml*. If it cannot find a file with this name, it looks for a file named *federation_execution.fdd*, then for *federation_execution.fed*. If you are building for RTI 1.3, VR-Link looks for a *.fed* file first, then a *.xml* or *.fdd* file.

Changes to Print Utilities

A new class, *DtOutputStream* has been added. This class works very much like any std::ostream class. It is a buffered, re-directable way for sending debug and informational messages in VR-Link. Previously VR-Link used functions like DtInfo().

DtInfo, *DtWarn*, *DtVerbose*, *DtFatal*, and *DtDebug* are all now *DtOutputStream* objects. As such, you can write to them just like any standard C++ output stream, for example:

```
DtInfo << "something bad just happened, the error is " << error << "\n";
```

Old style usage is still valid:

```
DtInfo ("something bad just happened, the error is %d\n", error);
```

All old code that uses any of these former functions will still work.

By using *DtOutputStream* we gain flexibility. *DtOutputStream* can attach to one or more *DtPrinters*. A *DtPrinter* can print to an output device. VR-Link includes the following *DtPrinter* subclasses:

- *DtFilePrinter* --Prints to the specified file.
- *DtStdoutPrinter* -- prints to stdout
- *DtStderrPrinter* -- prints to stderr
- *DtWindowsConsolePrinter* -- prints to a windows console (windows only).

By default the *DtStdoutPrinter* is attached to *DtInfo*, *DtWarn*, *DtDebug*, *DtVerbose* and *DtFatal*. You can add any of these subclasses, or your own subclass, to all, or just one of these streams to make *DtInfo* print to a file, for example:

```
DtFilePrinter info_log("myInfo.log");
DtInfo.attachPrinter(&info_log);
```

To print to a console, do the following:

```
DtInfo.attachPrinter(&DtStandardWindowsConsole)
```

The following functions are deprecated and may be removed in the future:

```
void DtSetVrLinkPrinter()
void setFileOutput()
```

Miscellaneous Changes

- The default value of RequestClassUpdateFlag is now False. We now just request object updates. This change prevents receipt of a redundant set of attribute updates.
- The MS Windows version of VR-Link no longer includes the MT libraries. They are not compatible with the latest Microsoft Visual C++ compiler.
- The following files and their associated classes and symbols have been moved from the *libvl5* and *libvlHLA* libraries to the *libmatrix* library:
  - *refEllipsoid.h*
  - *mapDatum.h*
  - *topoCoord.h*
  - *utmCoord.h*
  - *utmRefPt.h*
  - *utmView.h*.

  If you use classes from these header files and do not use any symbols from *libvl5* or *libvlHLA*, you do not need to include these two libraries on your link line.
- The header file *radioXmitPub.h* has been added. This header file allows protocol independent use of the *DtRadoTransmitterPublisher* class. The include line:

  ```
  #include <radioXmitPub.h>
  ```

  satisfies both HLA and DIS.
- VR-Link now uses a third-party library – *libXml2*. This library has been added to support the IEEE1516 RTI FED file format. If you are building a VR-Link application for use with the 1516 RTI, you must include *libXml2* on your link line. This library is part of the Red Hat Linux distribution. We distribute it for all other platforms.
- The header file *str.h* has been renamed to *vlString.h*. The old header file is still provided for backwards compatibility, but it may be removed in the future.
- The *DtObjectId* class has been moved from *hostStructs.h* to its own file, *objectId.h*.

- The functions previously defined in *mapDatum.h* as:

```
DT_DLL_VL void DtUseMapDatum(const DtMapDatum* md);
DT_DLL_VL const DtMapDatum* DtCurrentMapDatum();
```

  are now defined as:

```
DT_DLL_VL void DtUseMapDatum(const DtMapDatum& md);
DT_DLL_VL const DtMapDatum& DtCurrentMapDatum();
```

  They now take a const reference instead of a const pointer. Previously, if you created a new mapDatum you needed to maintain the pointer until the application ended so that you could delete it properly. Now, the mapDatum is copied by VR-Link and maintenance of this pointer is not required after you set it.

- The names of some header files for encoders and decoders have changed. All HLA files start with "h" and all DIS files start with "d". If you included these files directly rather than using protocol independent header files, you must update your include instructions.

New Make System

VR-Link 3.9 provides a new make system for UNIX. The format of makefiles and their supporting scripts has been completely reworked. However, makefiles written based on former releases of VR-Link will continue to work. The examples provided with this release use the new format, and should be consulted for details.

Documentation Updates

- The example code in section 5.10.1, page 5-41, has an error. The line that reads:

```
DtArtPartList* artParts = esr->artPartsList();
```

  should be:

```
DtArtPartList* artParts = esr->artPartList();
```

- The text in section 10.1.8, Copying PDUs, is incorrect. The following is correct:

  DtPdu has the copy constructor and assignment operator overloaded. You can copy data as follows:

```
DtPdu pdu1, pdu2;
pdu1 = pdu2;
```

  pdu1 now contains the same data as *pdu2*. The two objects do not, however, share any data objects.

  Classes derived from *DtPdu* do not usually have copy constructors or assignment operators. To ensure that data is copied correctly from one PDU to another, you must use the network representation of the PDU, for example:

```
DtFirePdu pdu1;
DtFirePdu pdu2((const DtNetFirePdu*)pdu1.netRep());
```

As with the standard PDU class constructors, the *DtPdu* constructor has an optional buffer argument. (See "Using External Buffers in a PDU Class," on page 10-11.)

The following code is correct in all cases:

```
DtPdu pdu1;
DtPdu pdu2(pdu1, buffer);
```

To copy a PDU derived from *DtPdu*, using an external buffer:

```
DtFirePdu pdu1;
int sz = pdu1.length();
char* bfr = new char[sz];
DtFirePdu pdu2((const DtNetFirePdu*)pdu1.netRep(), (DtBufferPtr)bfr);
```

- In section 11.3.2, "Choosing a Reference Ellipsoid", the sentence that reads:

  "VR-Link has the following *DtMapDatums* pre-defined in *geodCoord.h*

  should read:

  "VR-Link has the following *DtMapDatums* pre-defined in *mapDatum.h*."

Bug Fixes

VR-Link 3.9 fixes the following bugs:

- A bug in the class *DtFedReader* caused a crash on Solaris if *DtNotifyLevel* was set to verbose. The member function processObjClass() tried to print a null value to sprintf.
- VR-Link on Sun Solaris no longer crashes if you are using the DMSO RTI and you delete an attribute from the RPR FOM.
- The FED file reader no longer crashes if you pass it the name of a file that does not exist.
- Fixed a long-standing bug in *DtArtPart*, that could cause crashes when receiving Detonation PDUs or interactions that included articulated part information.
- The encoding and decoding for data interactions for RPR FOM 2 Draft 14 was incorrect. It now works.

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*.