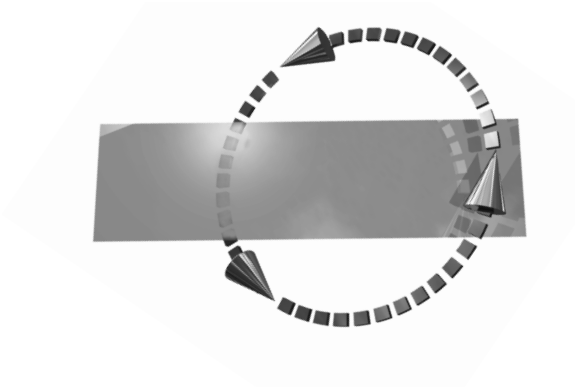

VR-Link 3.9.5 Release Notes

This release note provides the following release-specific information for VR-Link Release 3.9.5:

Document ID: VRL-3.9.5-3-050701

Systems Supported

Table 1 lists the platforms currently supported by MÄK VR-Link 3.9.5. 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, 9.0, Red Hat Enterprise 3.0 (use 9.0 build), Fedora Core 3	GNU C++ (GCC) 3.2
PC with Windows 2000/XP	Microsoft Visual C++ 6.0 SP5 Microsoft .NET

You must use *libXml2* to build any HLA application.

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.

Compiler Compatibility on Windows

MÄK provides versions of product releases that have been compiled with Microsoft Visual C++ 6.0 and 7.1. When you run MÄK products together, for example, the VR-Link 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.

Patching the Microsoft Visual C++ 6.0 Compiler

If you use Microsoft Visual C++ version 6.0, you must install a patch to a standard header file, in order to successfully build applications for HLA 1.3 or 1516. The patch comes from Dinkumware, Ltd (the company that developed the Standard C++ Library header files for Microsoft), but for convenience, we are distributing it with VR-Link releases.

The patch fixes a bug that would cause applications to crash if they pass std::sets or std::maps from an application to a DLL or vice-versa. For details about the patch go to http://www.dinkumware.com/vc_fixes.html.

To install the patch:

1. In the top level VR-Link directory, is a file called *XTREE-msvc++patch*. Rename this file *XTREE*.
2. Make a back-up version of the original *XTREE*.
3. In the Visual Studio *include* directory (for example, *C:\Program Files\Microsoft Visual Studio\VC98\Include*), replace the original version of *XTREE* with the patched version.

Backward Compatibility

VR-Link 3.9.5 is highly compatible with 3.9.4. For example, the *talk* and *listen* example programs from VR-Link 3.9.3 and previous versions compile and link unmodified against VR-Link 3.9.5.

Network Compatibility

HLA only

VR-Link 3.9.5 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 2.1
- 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 3.9.5 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 3.9.5 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-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`. 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`
- RPR FOM 2.0 draft 17: `vlRPR20017`.

RTI Support

VR-Link 3.9.5 has been tested with DMSO RTI NG v6, Pitch RTI 1.3, and MÄK RTI 2.2.

VR-Link 3.9.5 may work with older versions of the MÄK RTI, but we recommend using it with release 2.3.3.

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 3.9.5 has the following changes:

- Changes to *DtString*
- Changes to *hostStructs.h*
- New *DtExerciseConn* constructor
- New classes for parsing command line input and MTL files
- Support for HLA synchronization points
- New class: *DtAssert*
- New class: *DtSharedMemoryPoolManager*
- Updated print utility
- Changes to *DtTcpSocket*.

Changes to *DtString*

The ostream operator previously wrapped strings with spaces in quotes before printing them. This has been fixed. Given these two strings:

```
DtString str1 = "ASingleWord";
DtString str2 = "Several Words with spaces";
```

Previously:

```
std::cout << str1 << std::endl << str2 << std::endl:
```

printed:

```
aSingleWord
"Several Words with spaces"
```

It now prints:

```
aSingleWord
Several Words with spaces
```

The ostream operator has been implemented for all VR-Link state repositories. You may now use:

```
DtEntityStateRepository esr;
DtInfo << esr << std::endl;
```

which will provide the same functionality as:

```
 esr.printData();
```

Changes to *hostStructs.h*

The file, *HostStructs.h* has been deprecated. It is replaced by the following files, which contain the classes previously defined in *HostStructs.h*:

- *globalObjDes.h*
- *simulationAddress.h*
- *entityIdentifier.h*
- *entityType.h*
- *burstDescriptor.h*.

New *DtExerciseConn* Constructor

DtExerciseConn has a new contructor, which takes a *DtExerciseConnInitializer* class. This is now the preferred method for creating a *DtExerciseConn*. The *DtExerciseConn-Initializer* can set and inspect data that can be used to initialize the exercise connection, for example, execution name and federate name for HLA or port and exercise ID for DIS.

New Class for Parsing Command Lines

The *DtVrlApplicationInitializer* is a subclass of *DtExerciseConnInitializer* that can parse command line input and MTL files. Using this class, you can now specify command line arguments for VR-Link-based applications that can override the default settings in the *DtExerciseConnInitializer*. You can also read in data from MTL files. See f18.mtl for an example of MTL file use.

DtExerciseConnInitializer and *DtVrlApplicationInitializer* together allow you to replace the following code (from previous versions of the listen example):

```
int main()
{
  // Create a connection to the exercise or federation execution
#if DtHLA
  DtString execName("VR-Link");
  DtString fedName("VR-Link listen");
  DtExerciseConn exConn(execName, fedName, new DtSimpleRprFomMapper);
#elif DtDIS
  int port           = 3000;
  int exerciseId     = 1;
  int siteId         = 1;
  int applicationNum = 15;
  DtExerciseConn exConn(port, exerciseId, siteId, applicationNum,
                        0,       // No Status, any error will be fatal
                        true);   // use Async IO
#endif
```

with the following code, from the updated listen example:

```
int main(int argc, char** argv)
{
   // Create a connection to the exercise or federation execution.
   DtApplicationInitializer appInit(argc, argv, "VR-Link listen");

#if DtDIS
appInit.setUseAsynchIO(true);
#endif

   appInit.parseCmdLine();

   DtExerciseConn exConn(appInit);
```

For more information, see the comments in the header files and "Connecting to Exercises," on page 5-3 in *VR-Link Developer's Guide*.

Support for Synchronization Points

The HLA *DtExerciseConn* class can now register synchronization points and register user-defined callbacks for synchronization points. To register a synchronization point named, "point1," with the RTI, call:

```
exConn.registerSynchronizationPoint("point1");
```

You can register user-defined callbacks that will be called upon receiving a callback from the RTI. The available callbacks are:

- synchronizationPointRegistrationSucceeded()
- synchronizationPointRegistrationFailed()
- federationSynchronized()
- announceSynchronizationPoint()

To add or remove user-defined callbacks, use the following member functions:

- addSynchPointRegistrationSucceededCb()
- removeSynchPointRegistrationSucceededCb()
- addSynchPointRegistrationFailedCb()
- removeSynchPointRegistrationFailedCb()
- addFederationSynchedCb()
- removeFederationSynchedCb()
- addAnnounceSynchPointCb()
- removeAnnounceSynchPointCb()

Each member function takes a string (optional), function, and a pointer to user data that may be passed to the callback when called. If the string is provided, the callback is called for that particular label. If it is not provided, that callback is called for all labels.

For example:

```
exConn.addFederationSynchedCb("point1", userFunction, NULL);
```

causes userFunction to be called when the federation is synchronized at "point1", and:

```
exConn.addFederationSynchedCb(userFunction, NULL);
```

causes userFunction to be called when the federation is synchronized at any point.

VR-Link applications, by default, automatically respond to announced synchronization points by immediately indicating that the point has been achieved. This means that VR-Link-based applications will not hold up a federation execution when the application developer has not done anything special to respond to synchronization points. You can enable or disable this automatic response by calling:

```
exConn.setReplyingToSynchronizationPoints(false);
```

Registering an announceSynchronizationPoint() callback disables VR-Link's automatic response to announcements for the specified synchronization points.

Example:

```
#include <exerciseConn.h>

void fedsSynched(const char* label, void* /*usr*/)
{
  std::cout << "Federates synched at: " << label << std::endl;
}

void announcePoint(const char* label, const char* tag, void* usr)
{
  DtExerciseConn* exConn = (DtExerciseConn*)usr;
  std::cout << "Announce point: " << label << std::endl;
  std::cout << "Tag: " << tag << std::endl;
  usr->replyToSynchronizationPoint(label);
}

int main(int argc, char** argv)
{
  DtExerciseConn exConn(...);

  ...
  // fedsSynched is called whenever the federation is synched
  exConn.addFederationSynchedCb(fedsSynched, NULL);

  // announcePoint is called when the label "point2" is announced.
  // The exercise connection automatically replies to all synch points
  // excluding "point2"
  exConn.addAnnounceSynchPointCb("point2", announcePoint, &exConn);

  ...
}
```

DtAssert Class

DtAssert (in *vlAssert.h*) has been added to VR-Link. *DtAssert* is a replacement for std::assert(). For more information please see the class documentation.

DtSharedMemoryPoolManager Class

DtSharedMemoryPoolManager (*vlShmPool.h*) lets you create and manage a shared memory pool. See the header file for details.

Updated Print Utility

The file *printUtil.h* has been updated. New functions have been added to print out formatted state repository attributes using different streams. Previously, code to print out a boolean and an integer would look like:

```
int val1;
bool val2;

DtPrintInteger("an Integer: ", val1);
DtPrintBoolean("a Boolean: ", val2);
```

Now, that code is replaced by:

```
DtPrintAttribute(DtInfo, "an Integer", val1);
DtPrintAttribute(DtInfo, "a Boolean", val1);
```

DtPrintAttribute() has also been overloaded for VR-Link-defined enumerations (*parseEnum.h*). Previously code to print out an enumeration would look like:

```
DtRepairType rt;

DtDumpEnum("RepairType:", DtEnumString(rt), rt);
```

now looks like:

```
DtPrintAttribute(std::cout, "RepairType", rt);
```

Changes to DtTcpSocket

In DtTcpSocket, `int connectionEstabilished()` has changed to `bool isConnectionEstabilished()` to give it a more consistant name. The functionality is the same.

Documentation Updates

This section lists changes and updates to the printed and online versions of *VR-Link Developer's Guide*.

- Class reference documentation has been improved (by improving comments in header files.)
- The second paragraph and example in section 5.2.3 of the PDF version of *VR-Link Developer's Guide* should read as follows:

  When you use this constructor, all of the initialization values, which are protocol-dependent, are in the *DtExerciseConnInitializer*, the configuration file, or the command line. Therefore, the initialization code is completely protocol independent. The last parameter passed into the constructor is the application name. It is printed to the screen when the command line usage is printed out, and it also specifies the default federate name in HLA.

  ```
  // Create a connection to the exercise or federation execution.
  DtVrlApplicationInitializer appInit(argc, argv, "VR-Link application");

  appInit.parseCmdLine();

  DtExerciseConn exConn(appInit);
  ```

- The initialization code for the talk and listen examples in the PDF version of *VR-Link Developer's Guide* does not match the examples provided in this release. The difference is inclusion of the application name in the appInit() member function, as described in the previous bullet.
- Section 6.8.2, Responding to Synchronization Points is now a subsection of the synchronization point information provided in this release note.
- Section 4.1.3 in *VR-Link Developer's Guide* did not include link flags for HLA. The link flags are as follows:

  Link flags for HLA 1.3:

  ```
  -lvrhome/lib -lvlHLA13 -lmatrix -lmtl -lvlutil -lxml2
  ```

  Link flags for HLA 1516:

  ```
  -Lvrhome/lib -lvlHLA1516 -lmatrix -lmtl -lvlutil -lxml2
  ```

Bug Fixes

VR-Link 3.9.5 fixes the following bugs:

- The snipBack() member function of *DtString* has been modified to handle searches for the character '\0' in a manner consistant with the other *DtString* methods such as snipFront, size, and count. The null terminator is not considered to be an element of the string, and so searches for it should fail. For example, count('\0') will return 0, and snipFront('\0') and snipBack('\0') will return false and leave the string unmodified.
- Previously, the Change Indicator field of articulated parts data was sent as 0 on each update. Now, the Change Indicator is incremented with each update of an articulated parts parameter.
- In HLA, an emitter system was updated before an emitter beam, resulting in a reflection of the emitter system with beam data from the previous frame. Now, the beam is updated before the system, so that the reflected system will have the most recent beam data.

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*.