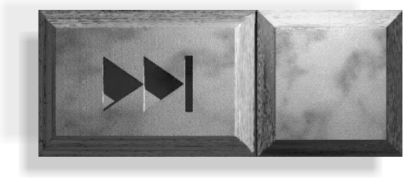

MÄK Logger User's Guide Addendum A

This document contains the following release-specific information for the MÄK Logger Release 3.4:

Copyright © 1999 MÄK Technologies, 185 Alewife Brook Parkway, Cambridge, MA 02138
All rights reserved.

Document ID: LOG-3.4-1A-990514

System Requirements

Logger 3.4 is being released initially for the following platforms.

- SGI IRIX6.2 or later (both n32 and o32 ABIs)
- Sun Sparcstation with SunOS4.1.3 or later
- Sun Sparcstation with Solaris2.5 or later
- PCs with Red Hat Linux 5.1
- PCs with Windows NT or Windows 95/98
- SGI Visual Workstation with Windows NT

You must have VR-Link 3.4 and its required compiler versions to use the Logger library, but not the executables. See VR-Link documentation for information about required compiler versions.

RAM and Memory Requirements

To install and run the Logger on a PC, we recommend the following minimum configuration:

- Pentium class PC 133 MHz or above
- 45MB disk space
- 64 MB RAM.

Installation of the Logger on a UNIX machine requires approximately 75 MB of disk space.

Network Compatibility

HLA only

The Logger 3.4 is built against VR-Link 3.4. This change makes the Logger compliant with:

- RPR-FOM 0.5, 0.7, and 0.8
- MÄK RTI 1.3.1
- DMSO RTI 1.3v4, 1.3v5, 1.3v6.

If you run the HLA Logger 3.4 with version 0.5 of the RPR FOM, it is network compatible with applications that are using VR-Link 3.3. It is not network compatible with applications that use VR-Link 3.2 or earlier and it is not compatible with applications using VR-Link 3.3 if the Logger is using RPR FOM 0.7 or 0.8.

DIS only

The DIS Logger 3.4 is network compatible with previous releases. The DIS protocol has not changed.

Changes to FLEXlm Setup and Configuration

With this release of the MÄK Logger, we have simplified the process for configuring the FLEXlm license management. It is no longer necessary to put a license file on each client machine, and you no longer have to merge license files when you purchase new products or additional licenses.

If you do not have a prior installation of MÄK Technologies products, follow the instructions in "Setting Up and Using the FLEXlm License Manager" on page 5. If you already have MÄK products installed, your upgrade procedure depends on the version of FLEXlm that you are running. Table 1 lists MÄK products and how to proceed.

Table 1: Product versions and license management

Product	FLEXlm Version	Procedure to Follow
VR-Link 3.0, 3.1 Stealth for SGI 3.1, 3.1.1 Stealth for PCs 3.2.1 Logger 3.1	5.x	Follow installation procedure in original product documentation. Merge license files as described in "Merging New License Files for Products Using FLEXlm 5.x" on page 4.
VR-Link 3.2 or later Stealth for SGI 3.2 or later Stealth for PCs 3.3 or later Logger 3.2 or later Plan View Display 1.x	6.0	Follow the instructions in "Updating Your Existing License Management Configuration" on page 3.

Updating Your Existing License Management Configuration

If you have FLEXlm license management configured for products you purchased prior to the current release, you must do the following to update your license management configuration:

1. On the license server machine, rename the *license.dat* file to *mak.lic* (or any name ending in *.lic*). (If you are unsure about the distinction between the license server machine and client machines, see "The FLEXlm Client-Server Architecture" on page 5.)
2. Open the license file in a text editor.
3. Remove the port number from the SERVER line. For example, if the first line of the license file is:

   ```
   SERVER oak 0800359de785 4567
   ```

   the port number is 4567. Delete it from the file.
4. Remove the LM_LICENSE_FILE environment variable from the server machine and all client machines unless another vendor requires it.

5. Delete the *license.dat* file from all client machines.
6. Set the MAKLMGRD_LICENSE_FILE environment variable on all client machines (including the license server machine if you run MÄK Technologies products on it).

   The syntax for the environment variable is: *@Server_name*. For example, if the server machine is oak, set the environment variable to `@oak`.

Merging New License Files for Products Using FLEXlm 5.x

If you are using FLEXlm 5.x, you cannot use the new method of license management. If you receive a new license file from MÄK, you need to merge it with the old file.

1. Your new license file will look something like the following:

```
SERVER mollusc 006097752f93
DAEMON maklmgrd .
PACKAGE dir maklmgrd 1999.177 60D00041FA200D8A795D \
      COMPONENTS=dir1
INCREMENT dir maklmgrd 1999.177 1-jan-0 1 0BB480511615C175B8CF \
      PLATFORMS=i86_n
```

   Your existing *license.dat* file is similar to the following:

```
SERVER mollusc 006097752f93 4567
DAEMON maklmgrd "c:\Program Files\MÄK Technologies\stealth\flexlm\
      maklmgrd"
PACKAGE makproduct maklmgrd 1999.177 40C020D12D609AB04D55 \
      COMPONENTS="hprod1 dprod1 prod1"
INCREMENT makproduct maklmgrd 1999.177 1-jan-0 1 AB4450916DD1944435D2 \
      PLATFORMS=i86_n
```

Notes

- The server name and host id in the two license files should be identical. The new file will not have a port number.
- The line continuation marks (\) in the examples are for clarity of presentation. They may not be present in the files you receive.

2. Open up the two files in a text editor. Append everything except the Server and Daemon lines (the Server lines will be identical except for the port) from the new *license.dat* file to the end of the old *license.dat* file. For example:

```
SERVER mollusc 006097752f93 4567
DAEMON maklmgrd "c:\Program Files\MÄK Technologies\stealth\flexlm\
      maklmgrd"
PACKAGE makproduct maklmgrd 1999.177 40C020D12D609AB04D55 \
      COMPONENTS="hprod1 dprod1 prod1"
INCREMENT makproduct maklmgrd 1999.177 1-jan-0 1 AB4450916DD1944435D2 \
      PLATFORMS=i86_n
PACKAGE dir maklmgrd 1999.177 60D00041FA200D8A795D \
      COMPONENTS=dir1
INCREMENT dir maklmgrd 1999.177 1-jan-0 1 0BB480511615C175B8CF \
      PLATFORMS=i86_n
```

3. Put the updated *license.dat* file in the *flexlm* directory on the license server and on all machines on which you have the Logger installed.

Setting Up and Using the FLEXlm License Manager

The Logger, like other MÄK Technologies products, uses the FLEXlm license manager. Before you can use the Logger, you must obtain a valid license file and configure the license server and client machines. (For information about FLEXlm, view the FAQ page at www.globetrotter.com/flmfaq.htm.) Contact the MÄK Technologies sales department for information about special licensing agreements.

The FLEXlm Client-Server Architecture

This section explains the FLEXlm architecture to provide a context for the setup instructions that follow. FLEXlm uses a client-server architecture. The FLEXlm executable, *maklmgrd* runs on one computer, known as the license server. This machine services every machine (including possibly itself) on which you want to run a MÄK Technologies product. The machines on which the MÄK Technologies products are running are called clients.

The license server has a license file (supplied by MÄK Technologies). This file identifies the MÄK Technologies products for which you have licenses. On each client machine, the MAKLMGRD_LICENSE_FILE environment variable (which you must set) points to the server.

Licenses "float", so you can install product software on more machines than you have licenses for, but at any given time, you can run only as many instances of a product as you have licensed.

! The license key in the license file is tied to the host id of the server machine. Therefore, if you decide to change the server, you need to get a new license file. Then, you must update the MAKLMGRD_LICENSE_FILE environment variable on every client. This is not a trivial process, and is not recommended.

Figure 1 illustrates the client-server architecture for FLEXlm and MÄK products.

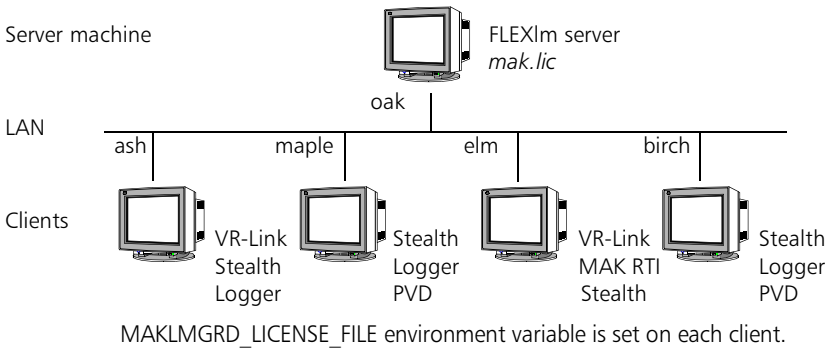

Figure 1. FLEXlm example architecture

Installing and Configuring the License Manager

The general procedure for installing and configuring license management is as follows (each step is explained in the rest of this section):

1. Install license manager software.
2. Request a license file.
3. Put the license file in the *flexlm* directory on the license server.
4. Set the MAKLMGRD_LICENSE_FILE environment variable on all client machines.

If you have MÄK Technologies products installed and you purchase additional licenses or products, you must get a license file for the new products or licenses. For this procedure, see "Adding Additional License Files" on page 9.

Installing the License Manager Software

As part of the installation of all MÄK products, the FLEXlm software is installed in the *flexlm* directory under the root product directory.

1. Install your MÄK products on the machine(s) on which you want to run them.
2. If this is a networked system, and you did not install MÄK software on the license server, copy the *flexlm* directory from a client machine to the license server machine.

Requesting A License File

License files are keyed to the host id of the license server. You need to supply MÄK with some information so that we can generate a license file for you.

To get a license file:

1. On the license server, in a command window, change to the *flexlm* directory.
2. Execute *lmhostid.* This program generates a unique number that MÄK uses to generate your license activation codes. Write down the number.
3. Go to http://www.mak.com/support/keyintro.htm and complete the license key request form. MÄK will e-mail you a file named *mak.lic*.

To get a license file, you need to know the name of the license server machine. To get the name of your machine follow the procedure for your operating system:

UNIX

At a command prompt, type `hostname`.

Windows 95/98/NT

1. On the Start menu, choose Settings → Control Panel.
2. Open the Network applet.
3. Select the Identification tab. The machine name is listed in the Computer Name field.

Putting the License File in the *flexlm* Directory

After you receive the license file, put it in the *flexlm* directory on the license server.

! If you already have a *mak.lic* file in the *flexlm* directory, do not overwrite the file. See the instructions in "Adding Additional License Files" on page 9.

Setting the MAKLMGRD_LICENSE_FILE Environment Variable

The MAKLMGRD_LICENSE_FILE environment variable identifies the server machine so that the FLEXlm software on a client can check out a license when you run a MÄK Technologies product. You must set MAKLMGRD_LICENSE_FILE on every client machine.

If you are running MÄK Technologies products on the license server machine, it is also a client, so you must set MAKLMGRD_LICENSE_FILE.

The syntax for the environment variable is: *@Server_name*. For example, if the server machine is oak, set the environment variable to `@oak`.

The following sections explain how to set environment variables on the different platforms that MÄK Technologies products run on.

Windows 95/98

On Windows 95/98, you set environment variables by adding or editing lines in the *Autoexec.bat* file. To set the MAKLMGRD_LICENSE_FILE variable, add a line similar to the following:

```
set MAKLMGRD_LICENSE_FILE=@oak
```


Do not put spaces around the equal (=) sign.

Windows NT

On Windows NT, you set environment variables in the Environment tab of the Properties dialog box for the machine:

1. Right-click the My Computer icon on the Desktop.
2. On the pop-up menu, select Properties. The System Properties dialog box opens (Figure 2).
3. Select the Environment tab.
4. In the Variable text field, type MAKLMGRD_LICENSE_FILE.
5. In the Value field, type the value, for example:

```
@oak
```

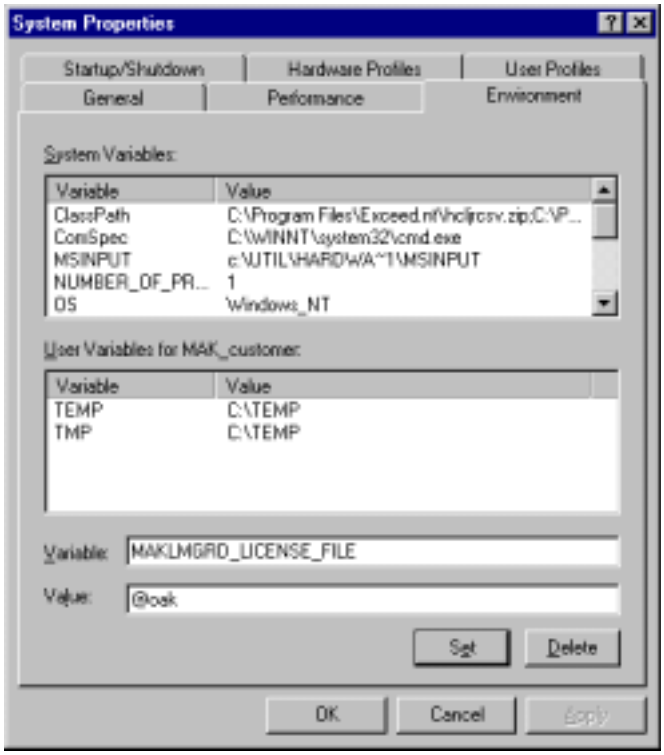

Figure 2. Setting an environment variable in Windows NT

UNIX

On UNIX, you set environment variables in your *.cshrc* (or equivalent startup file). Set the variable similarly to the following example:

```
setenv MAKLMGRD_LICENSE_FILE @oak
```

You are ready to run the license server and use your new licenses or MÄK products.

Adding Additional License Files

If you buy additional MÄK products, or additional licenses for products you already have, you must get an additional license file.

> To request an additional license file, follow the same procedure as for your first license file. See "Requesting A License File" on page 7.

To add a new license file to your license server:

1. When you get the new license file, rename it from *mak.lic* to something like *mak2.lic*, or some other name ending in *.lic*, so that it has a different name from the license file(s) currently in the *flexlm* directory on the license server.
2. Put the renamed file in the *flexlm* directory on the license server, with the other license files.

New Features and Other Changes

- Logger 3.4 supports VR-Link's ability to work with different FOMs through the FOM Mapper.
- On PCs, the default installation directory structure has changed.

Selecting A FOM Mapper

Logger 3.4 has built-in support for versions 0.5, 0.7, and 0.8 of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper.

You can specify which FOM Mapping code to use with the `-f` startup option, in one of two ways:

```
Logger -f { vlRPR05 | vlRPR07 | vlRPR08 }
Logger -f library:init_data
```

If the argument to the `-f` option is one of the vlRPR0x keywords, the Logger uses VR-Link's built-in RPR FOM mapper, configured for RPR FOM version 0.5, 0.7 or 0.8 as specified by the keyword.

If the argument to the `-f` option is not one of the three keywords, it is interpreted as the name of a shared library (DSO or DLL) containing code for an alternate FOM Mapper that you would like the Logger to use, optionally followed by any initialization data the library requires. The library name and initialization data must be separated by a colon.

Examples

```
Logger -f vrlRPR05
```

This command instructs the Logger to use the built-in RPR FOM Mapper, configured for RPR FOM version 0.5.

```
Logger -f myFomMapLib:ABC
```

This command instructs the Logger to use the FOM Mapper contained in a shared library called *myFomMapLib.so* (or *myFomMap.dll*). The string "ABC" is passed to the FOM Mapper creation function in the shared library as an initialization parameter.

Specifying A FED File that is Appropriate for the FOM Mapper

! The `-f` option only tells the application which FOM Mapping code to use. It does not ensure that you are using the right FED file for the FOM Mapper. Make sure that you are using a FED file that is consistent with the FOM Mapper you have chosen.

By default the application uses a FED file called *VR-Link.fed*, but you can specify an alternate FED file using the `-x` option. The `-x` option actually indicates the name of the federation execution name, but we append *.fed*, and use that as the FED file name as well.

MÄK products come with three FED files:

- *VR-Link.fed*, based on RPR FOM 0.8
- *VR-Link07.fed*, based on RPR FOM 0.7
- *VR-Link05.fed*, based on RPR FOM 0.5

The following example instructs the application to use the RPR FOM 0.5 FOM Mapping code (`-f vlRPR05`), and also instructs it to use the *VR-Link05.fed* file (`-x VR-Link05`).

```
Logger -f vrlRPR05 -x VR-Link05
```

As an alternative to using the `-x` option, you could rename the *VR-Link05.fed* file to *VR-Link.fed*.

New Default Directory Structure on PCs

On PCs, the default installation directory is now *C:\Program Files\MAK\Logger.* Formerly it was *C:\Program Files\MAK Technologies\Logger.*

Documentation Updates

- Section 7.1 of the *MÄK Logger User's Guide* mentions the buffgrep utility. This utility is available only on UNIX platforms.
- The status message displayed in section 5.1 is incorrect. Only the NetSocket line is displayed.
- Section 5.1.3 states that in record mode the default behavior of the TLogger is to record immediately. This is incorrect. The default behavior is to come up in a stopped state. To record immediately, use the following command:

  `TLogger -br -r filename`
- Table 5-1 states that the default state for record mode is r (record). This is incorrect. The default state is s (stopped).
- The status message displayed in Section 5.2 is incorrect. Only the NetSocket and Logger lines are displayed.
- Section 5.2.3 states that in playback mode the default behavior of the TLogger is to begin playing immediately. This is incorrect. The TLogger comes up in stopped mode. To start playback immediately, use the following command:

  `TLogger -op -p filename`
- The Logger has an API and a library that implements the API. It is described in the following sections.

MÄK Logger File API

In addition to the standalone tools that help you view or modify MÄK Logger files, we provide an API (application programmer's interface) to the MÄK Logger file format, and a library that implements the API. The API allows you to read and write MÄK Logger files from your own programs.

Because the Logger file library uses MÄK VR-Link, you must have a licensed copy of VR-Link to use the Logger file library.

The Logger file API is in *include/loggerFile.h*. In UNIX, there are three implementations of the library, all of which are in the *lib* directory:

- *liblgrFileRPR.a* - supports HLA
- *liblgrFile5.a* - supports DIS 2.0.4, 2.1.4, IEEE 1278, and IEEE 1278.1a
- *liblgrFile3.a* - supports DIS 2.0.3

For PCs, the following libraries are provided:

- *lgrFile3MD.lib*
- *lgrFile3MT.lib*
- *lgrFile5MD.lib*
- *lgrFile5MT.lib*
- *lgrFileRPRMD.lib*

See "Compiling VR-Link Applications" in the *VR-Link Developer's Guide* for information about the different libraries.

The Logger file API has two main classes: *LtLoggerRecordFile* and *LtLoggerPlaybackFile*. Use *LtLoggerRecordFile* when you want to write a Logger file, and *LtLoggerPlaybackFile* when you want to read a Logger file. Both of these classes are derived from *LtLoggerFile*.

Reading A Logger File

To read a Logger file, create an instance of *LtLoggerPlaybackFile*, and use its member functions to navigate though a file and read packets. When we use the term "packet", we are referring to a packet of data in the Logger file: either a DIS PDU, or an HLA update, interaction, or object removal.

Creating An LtLoggerPlaybackFile

DIS only

In DIS, the *LtLoggerPlaybackFile* constructor takes only the name of the file you want to read:

```
LtLoggerPlaybackFile inFile("myFile.lgr");
```

HLA only

In HLA, a Logger file contains information about the FOM being used in the recorded exercise. The reason for this is that during playback, we may have to convert class, attribute, and parameter handles stored in the Logger file to their corresponding handles in a different federation execution. (There is no guarantee that the same classes, attributes and parameters are assigned the same handles in different federation executions, especially if the FED file has changed at all).

When you pass a valid *DtExerciseConn* (a VR-Link object that represents a connection to an HLA exercise) pointer as the second argument to the *LtLoggerPlaybackFile* constructor, *LtLoggerPlaybackFile's* packet reading functions perform conversions to the handles assigned by the *DtExerciseConn's* RTI connection. The following example shows the *LtLoggerPlaybackFile* constructor for HLA.

```
LtLoggerPlaybackFile("myFile.lgr", myExConn);
```

If the optional *DtExerciseConn* argument is omitted, the update and interaction information returned by its packet reading calls contain exactly the same class, attribute, and parameter handles that were recorded from the original exercise.

For both HLA and DIS, to read data from standard input, rather than from an actual file, pass the string "stdin" as the filename argument to the *LtLoggerPlaybackFile* constructor.

Reading Data

When you first create an *LtLoggerPlaybackFile*, you are at the beginning of the file. The functions described in Table 2 either read and return a packet, or change the location of a file pointer that determines where we are in the file:

Table 2: LtLoggerPlaybackFile functions for reading or returning packets

Function	Description
readPacket()	Reads and returns next packet, NULL if none. In HLA, packets might be skipped if the file was created with a non-NULL exConn, and the packet cannot be converted to the exConn's FOM.
readPacketBackwards()	Reads and returns previous packet, NULL if none. In HLA, packets might be skipped if the file was created with a non-NULL exConn, and the packet cannot be converted to the exConn's FOM.
skipPacketForwards() skipPacketBackwards()	Skip over next packet either forwards or backwards direction. Return 1 if successful, 0 if no packets in that direction.
jumpToStart() jumpToEnd()	Go to the beginning or end of the file.
seekTime(DtTime t)	Jump to time t (in seconds) in the file.
seekForward(DtTime t) seekBackward(DtTime t)	Jump forward or backwards in the file by t seconds.

The functions described in Table 3 provide information about the file and about your current location in the file:

Table 3: LtLoggerPlaybackFile functions for reporting time and location

Function	Description
atStart() atEnd()	Returns 1 if at beginning or end of file, otherwise returns 0.
fileTimeBackwards(DtTime *t) fileTimeForwards(DtTime *t)	Fills t with time of next or previous packet. Returns 1 on success, 0 if no packet in that direction.
firstPacketTime(DtTime *t) lastPacketTime(DtTime *t)	Fills t with time of first or last packet in the file.

The functions readPacket() and readPacketBackwards() return their data in the form of a pointer to an instance of the *LtPacketInfo* class. See "Interpreting LtPacketInfo" on page 16 for information about how to interpret this object.

The *LtPacketInfo* instance returned is valid only until the next call to one of *LtLoggerPlaybackFile's* reading or jumping member functions. Do not save this pointer, or attempt to delete it.

Writing a Logger File

To write a Logger file, create an instance of *LtLoggerRecordFile*, and use its writePacket() member function to write each DIS PDU or HLA update, interaction, or removal.

Creating an LtLoggerRecordFile

DIS only

In DIS, the *LtLoggerRecordFile* constructor takes only a file name:

```
LtLoggerRecordFile outFile("myFile.lgr");
```

HLA only

In HLA, a Logger file contains information about the FOM being used in the recorded exercise. The reason for this is that during playback, we may have to convert class, attribute and parameter handles stored in the Logger file to their corresponding handles in a different federation execution. (There is no guarantee that the same classes, attributes and parameters are assigned the same handles in different federation executions, especially if the FED file has changed at all).

When you create an *LtLoggerRecordFile* in HLA, you must provide it with information about the FOM being used, so that it can store it in the Logger file. You can do this in one of two ways: either by passing a *DtExerciseConn*, or by passing a FOM description object obtained from another Logger file that you have opened with *LtLoggerPlaybackFile*. That class's fomDesc() member function will return the FOM description in its file.

For example:

```
LtLoggerPlaybackFile inFile("inFile.lgr");
LtLoggerRecordFile outFile("outFile.lgr", inFile.fomDesc());
```

For both HLA and DIS, to write data to standard output, rather than to a file, pass the string "stdout" as the filename argument to the *LtLoggerRecordFile* constructor.

Writing Data

LtLoggerRecordFile's writePacket() member function looks like this:

```
virtual int writePacket(LtPacketInfo *packetInfo);
```

Its argument is a pointer to an instance of the class *LtPacketInfo*. See "Interpreting LtPacketInfo" on page 16 for information about filling out this object with the data you would like written.

Interpreting LtPacketInfo

DIS only

In DIS, *LtPacketInfo* class looks like this:

```
class LtPacketInfo
{
public:
   int size;                  // bytes
   double arrivalTime;        // seconds
   char packet[LtMAX_PACKET];
};
```

The packet field contains the packet itself - the actual bytes of data that make up the PDU. The PDU's size, in bytes, is indicated by the size field. **arrivalTime** indicates the time associated with this PDU. This is not the PDU's timestamp, but rather a time offset into the file, written by the Logger or other application writing the file. If the file was written by the MÄK Logger, **arrivalTime** is the time since the Logger started recording that the PDU was received (Figure 3).

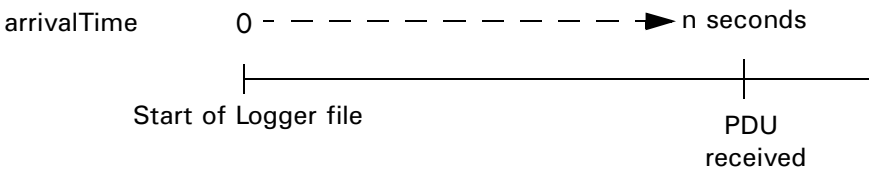

Figure 3. arrivalTime

HLA only

In HLA, *LtPacketInfo* is more complicated. The class looks like this:

```
class LtPacketInfo
{
public:
   LtPacketInfo();
   ~LtPacketInfo();

   LtPacketKind packetKind;
   DtTime arrivalTime; // seconds
   DtU32 objectId;     // zero if interaction
   DtU32 classHandle;  // object or interaction
   DtString name;      // "" if interaction
   RTI::AttributeHandleValuePairSet *attrs;
   RTI::ParameterHandleValuePairSet *params;
};
```

The *packetKind* field indicates whether this packet represents an attribute update, an interaction, or an object removal. Its values are as follows:

```
enum LtPacketKind
{
   LtPacketInteraction,
   LtPacketUpdate,
   LtPacketRemoval
};
```

The arrival time has the same meaning as in DIS - a time offset into the file that represents when the data was received by the Logger.

The interpretation of the other fields in *LtPacketInfo* depends on what kind of packet you are dealing with:

- If the packet kind is *LtPacketInteraction*, then **classHandle** contains the interaction class handle associated with the interaction. The **attrs** field is a pointer to an RTI::ParameterHandleValuePairSet object that contains the actual interaction data. In files recorded by the Logger, this is the object passed to Logger by the RTI through the receiveInteraction service. The other fields are not valid.
- If the packet kind is *LtPacketUpdate*, then **classHandle** contains the object class handle of the object being described. The **objectId** field contains the object's object instance handle, as assigned by the RTI. The **name** field contains the name by which the object is known to the RTI. The **attrs** field contains a pointer to an RTI::AttributeHandleValuePairSet object that contains the actual update data. In files recorded by the Logger, this is the object passed to Logger by the RTI through the reflectAttributeValues service. Other fields are not valid.
- If the packet kind is *LtPacketRemoval*, then the packet represents the removal of an object from the exercise. In files written by the Logger, this type of packet is written in response to a removeObjectInstance RTI service invocation. Only the **objectId**, **name**, and **classHandle** fields are valid, and they have the same meanings as in *LtPacketUpdate* packets described above.

Editing Logger Files

One of the most common tasks users want to do with *LtLoggerFiles* is to read data from one Logger file, edit it, and write it to another Logger file. Below are two examples of this. Note that they use several classes from VR-Link:

Here is a DIS example of filtering out all Entity State PDUs from the entity with an ID of 3:4:5.

```
int main()
{
   // Open the input file
   LtLoggerPlaybackFile infile("inFile.lgr");

   // Open the output file.
   LtLoggerPlaybackFile outfile("outFile.lgr");

   // Create an PDU factory that can create instances of the right
   // kind of DtPdu suclass depending on the PDU kind in the packet.
   DtDisPduFactory factory;

   LtPacketInfo *info;
   while (info = infile.readPacket())
   {
      // Use the factory to create an instance of a DtPdu subclass
      DtPdu* pdu = factory.createPdu((DtNetPacket*) info->packet);

      int writeIt = 1;
      // If it's an entity state PDU from 3:4:5, don't write it.
      if (pdu->kind() == DtEntityStatePduKind)
      {
         // Cast to DtEntityStatePdu - should be safe
         DtEntityStatePdu* esPdu = (DtEntityStatePdu*) pdu;
```

```
            if (esPdu->entityType() == DtEntityType(3,4,5));
            {
               writeIt = 0;
            }
         }
         if (writeIt)
         {
            outfile.writePacket(info);
         }
      }
   }
```

Here is an HLA example of filtering out all interactions:

```
int main()
{
   // Open the input file
   LtLoggerPlaybackFile infile("inFile.lgr");

   // Open the output file, which should have the same FOM description
   // as the input file.
   LtLoggerPlaybackFile outfile("outFile.lgr", infile.fomDesc());

   LtPacketInfo *info;
   while (info = infile.readPacket())
   {
      if (info->packetKind != LtPacketInteraction)
      {
         outfile.writePacket(info);
      }
   }
}
```

Compiling and Linking with the Logger Library

In order to use the Logger file API, you must add the following include directive to your compile line:

`-I` *`loggerDir`*`/include`

where *`loggerDir`* is the path to the root of your Logger installation.

You must link with *libLgrFile*, as follows:

`-L` *`loggerDir`*`/lib -llgrFile`*`x`*

The exact name of *lgrFilex* depends on the simulation standard you are using (DIS3, DIS5, RPR).

In addition, you should follow the directions for compiling and linking with VR-Link, in your VR-Link documentation. The Logger library link flag above must come before the VR-Link linking flags on your link line.

The preceding instructions are oriented towards UNIX. On a PC, make sure you are using the correct libraries in your project files, as listed in "MÄK Logger File API" on page 12.

Known Problems

- On Linux, when you run the HLA Logger or TLogger using the MÄK RTI, they hang when you try to exit.VR-Link-based applications that use libmtl functions and the MÄK RTI may hang when exiting on the Irix6o32 and Linux platforms. This is due to limitations in the linker on these platforms that do not allow us to keep dual copies of statics that are used by both application code and by the MÄK RTI.
- Piping (|) between Logger utilities yields unpredictable results. If you are having a problem with this, redirect (>) the output to a file, then take the input from a file (<).

Fixed Bugs

The following problems, which existed in previous versions of the Logger, have been fixed:

- The `-x` option to the Logger, TLogger, and *lgrDump* did not work.
- On SGI and Solaris, pressing TLogger's the pause command stopped the Logger, rather than pausing it.
- On PCs, the Pause button on the graphical user interface did not work.
- The `-s` option for the *lgrCat* utility did not work.
- On Windows 95/NT, the logger utilities did not always work properly when using standard input or standard output.