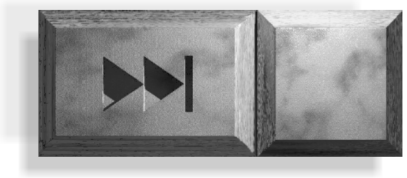

MÄK Logger User's Guide Addendum B

This is the second addendum to *MÄK Logger User's Guide* Revision LGSG/W-970611a. This addendum supports MÄK Logger Release 3.2. It covers the following:

- System requirements, on page 1
- Backward compatibility, on page 3
- New features, on page 4
- Fixed Bugs, on page 10

The major changes in this release are:

- Greater FOM flexibility
- Support for RPR FOM 0.3, through VR-Link 3.2
- Ability to control the Logger remotely via Logger Control PDUs and Interactions

System requirements

License Key

The license management scheme has changed slightly in Logger 3.2. If you are upgrading from Logger 3.1, you must contact MÄK Technologies and get a new license key.

Document ID: LOG-3.2-1B-980602

The HLA RTI

Like VR-Link 3.2, the HLA version of Logger 3.2 requires either the MAK RTI version 1.0.3, or the DMSO RTI version 1.0.3. Both RTIs conform to the same C++ API, so you can switch between the two by putting the desired RTI DSO somewhere in your path environment variable.

The MAK RTI is available from MAK Technologies. See *http://www.mak.com/rti/webpage.html* for details.

The DMSO RTI is available through the DMSO web site: *http://www.dmso.mil*. Follow the HLA link, and choose HLA Software Distribution Center to register for, and download the DMSO RTI. Remember that the DMSO RTI 1.0.3 requires the following patches on IRIX6: 1403, 1404, 1644, 1918, 2000, 2044.

See Chapter 1, "Installation" chapter in the *VR-Link Developer's Guide* for more information about obtaining and configuring the DMSO RTI.

FOM Flexibility

In HLA, during recording, Logger 3.1 would only subscribe to a hard-coded set of RPR FOM classes. Using VR-Link 3.2, Logger 3.2 reads the *.fed* file, and subscribes to all non-MOM classes in the file. Therefore, you can modify or extend the FOM and the Logger will record and play back your new data.

In Logger 3.1, you were required to use the same *.fed* file during recording of a Logger file and playback of the file. Logger 3.2 allows you to change the *.fed* file between recording and playback. As long as the *.fed* file you are using on playback contains object and interaction classes, and attributes and parameters with the same names as those that were recorded, your file can be played back successfully.

Classes, attributes and parameters can be rearranged in the *.fed* file, and new ones can be added. If any class names have been deleted from the *.fed* file between recording and playback, recorded interactions and updates for those classes will not be included in the Logger's outgoing data stream. If any attributes or parameter names have been deleted from the *.fed* file, those attributes and parameters will not appear in outgoing updates and interactions.

Support for RPR FOM 0.3

Logger 3.2 is built against VR-Link 3.2. This change makes Logger compliant with RPR-FOM 0.3. (Logger 3.1 was built against VR-Link 3.1 and complied with RPR-FOM 0.1.6.) FOM knowledge is necessary for the Logger to be able to do scaling of velocities and accelerations, and for *lgrDump* to be able to meaningfully print data from Logger files.

The *VR-Link.fed* file distributed with Logger 3.2 differs from the *.fed* file distributed with the RPR FOM 0.3 in the following ways:

- Several parameters of the derived *RadioSignal* interactions duplicate parameters in the base class, and thus are commented out.
- View Control and Logger Control interactions have been added.
- The following object classes have been added:
  - *BaseEntityOther*
  - *PhysicalEntityOther*
  - *MilitaryEntityOther*
  - *MilitaryPlatformOther*

RPR FOM 0.3 can be obtained from
http://www.sisostds.org/doclib/doclib.cfm?SISO_RID_1000019

RPR FOM 0.3 and RPR FOM 0.1.6 represent many attributes and parameters differently. They are essentially two different protocols. This means that you cannot play back a Logger file recorded with Logger 3.1 (when the participants were using RPR FOM 0.1.6) in the same federation execution as VR-Link 3.2-based applications, which expect data to be represented as specified by RPR FOM 0.3.

Backward compatibility

Logger File Format

HLA only

To support the FOM flexibility feature in the HLA version of Logger 3.2, it was necessary to modify the Logger file format to include information about the FOM being used during recording. This change means that the HLA Logger 3.2 cannot play back files recorded with Logger 3.1. Even if you were to convert an old Logger file to the new format, its use would be limited since you couldn't play it back into a RPR-FOM 0.3 based federation execution (see above).

DIS only

The DIS Logger format has not changed, and Logger 3.2 can play DIS Logger files created in releases prior to release 3.2.

New features

Logger Tools (HLA Only)

The class, attribute and parameter handles printed by the HLA version of *lgrDump* may not be the actual handles that were originally recorded, if you are not using the same *.fed* file with *lgrDump* that you used when you recorded the file with the Logger. The handle numbers that are printed will be those associated with the recorded names in the new FOM.

If you are using a different *.fed* file with *lgrDump*, but would like to see the actual handles that were used at the time of recording, you can use *lgrDump's* `-x` argument with an *execName* of "orig". This is only allowed when using *lgrDump* in raw mode. For example:

```
lgrDump -r -x orig myFile.lgr
```

In non-raw mode, we need to actually interpret the data, so we need to use the handles associated with the current *.fed* file.

When using the HLA version of *lgrMerge* and *lgrCat*, all of the input Logger files must have been recorded using the same FOM. Files recorded using FOMs that differ from that used in the first file on the command line are skipped.

Remote Control of the Logger

Logger 3.2 introduces programmatic control of the Logger. You do not need to run the Logger interface to record exercises, nor do you need someone present at the machine on which the Logger is running. This feature is described in the next section.

Controlling the Logger Remotely

You can control the Logger from a remote application via DIS or HLA. Every command available on the Logger graphic interface is available for remote control.

Logger Control PDUs work similarly to the way View Control PDUs work to control the MAK Stealth. There are 17 new Logger Control PDUs - one for each feature that can be activated remotely. All are derived from *DtLgrControlPdu*, which is defined in *lcPdu.h*, and all are defined in header files that begin with "lc". You can view the header files in the on-line version of the *VR-Link Developer's Guide* for VR-Link version 3.2.

The new classes are:

- *DtLgrEndPdu*
- *DtLgrEofActionPdu*
- *DtLgrHeartbeatPdu*
- *DtLgrPausePdu*
- *DtLgrPlayPdu*
- *DtLgrPlayActionPdu*
- *DtLgrPbFilePdu*
- *DtLgrQuitPdu*
- *DtLgrRecordActionPdu*
- *DtLgrRecFilePdu*
- *DtLgrRecordPdu*
- *DtLgrSetTimePdu*
- *DtLgrSkipTimePdu*
- *DtLgrStartPdu*
- *DtLgrStopPdu*
- *DtLgrTimeScalePdu*
- *DtLgrTimeoutPdu*

HLA only

For HLA, *DtLgrControlInteraction*, defined in *hLcInter.h*, is a wrapper around *DtLgrControlPdu*.

New Logger Commands

There are new command-line arguments for controlling the following behavior:

- Run Logger without the graphic interface

  Since the Logger can be operated remotely, the graphic interface is not a requirement, so Logger 3.2 lets you run the Logger without the interface using the `-v` argument.

- Specify which remote control PDUs the Logger will accept

  You can specify whether or not the Logger will respond to all PDUs, specific PDUs, or no PDUs, with the `-L` argument.

- Record remote control PDUs

  You can keep a record of the control PDUs sent to the Logger. Use the `-R` argument.

DIS only

- Specify an entity ID for Logger

  The Logger will be listening for instructions. It must be addressable so an entity ID is used. You specify the ID with the `-a` argument.

Table 1 describes each new argument to the Logger command and its possible values.

Table 1: Logger Command Arguments for Remote Control

Logger command arguments	MTL Syntax	Description
`-a "0:0:0"`	`entityIdStr`	Specify the site:host:entity for Logger
`-v {0|1}`	`LgrApp::visible`	1 = Logger interface visible (default) 0 = Logger interface not visible
`-R {0|1}`	`LgrAIF::recordLc`	0 = do not record Logger Control PDUs (default) 1 = record Logger Control PDUs
`-L {0|1|2}`	`LgrAIF::processLc`	0 = Do note respond to Logger Control PDUs 1 = interpret Logger Control PDUs only if they are addressed to the entity ID specified by the -a argument (default) 2 = interpret all Logger Control PDUs

Example of Remote Control for a DIS Exercise

The following example demonstrates programmatic remote control of the Logger:

```
/*********************************************************
```

```
** Copyright (c) 1997 MaK Technologies, Inc.
** All rights reserved.
************************************************************/
#include "stdio.h"
#include "exerciseConn.h"
#include "ProcControl.h"
#include "EntityTypes.h"

// Include each LoggerControl header
#include "lcEnd.h"
#include "lcEofAction.h"
#include "lcHeartbeat.h"
#include "lcInter.h"
#include "lcPause.h"
#include "lcPdu.h"
#include "lcPlay.h"
#include "lcPlayAction.h"
#include "lcPlayFile.h"
#include "lcQuit.h"
#include "lcRecAction.h"
#include "lcRecFile.h"
#include "lcRecord.h"
#include "lcSetTime.h"
#include "lcSkipTime.h"
#include "lcStart.h"
#include "lcStop.h"
#include "lcTimeScale.h"
#include "lcTimeout.h"


char * helpstr =
"    PDUTYPE       OPT_ARG\n"
"0   PbFile        filename\n"
"1   RecFile       filename\n"
"2   Quit          \n"
"3   Rewind        \n"
"4   End           \n"
"5   Play          \n"
"6   Pause         \n"
"7   Stop          \n"
"8   Record        \n"
"9   SetTime       float\n"
"10  SkipTime      float\n"
"11  TimeScale     float\n"
"12  PlayAction    {PLAY=0, PAUSE=1, STOP=3}\n"
"13  RecordAction  {RECORD=2, STOP=3}\n"
"14  EofAction     {PAUSE=1, STOP=3, LOOP=5}\n"
"15  HeartBeat     float\n"
"16  Timeout       float\n"
"-1  Quit this application\n"
;

//
// Application
//
```

```
main()
{
   // Create a connection to the exercise or federation execution
   int port          = 3000;
   int exerciseId    = 1;
   int siteId        = 39;
   int applicationNum = 16;

   // Identify the receiving (remote) Logger
   DtObjectId target("9:8:7");

   // Optionally identify yourself
   DtObjectId myId("3:4:5");

   // Create Exercise Connection
   DtExerciseConn exConn(port, exerciseId, siteId, applicationNum);

   // Initialize simulation time
   DtTimeInit();

   // Init work variables
   char  cmdline[128], optionalArg[128];
   int cmd = 0;

   while (cmd >= 0)
   {
      // Print usage instructions and prompt for input
      printf("%s\n> ",helpstr);
      gets (cmdline);

      // Scan for input
      cmd = 0 ; optionalArg[0] = NULL;
      sscanf(cmdline, "%d%s",&cmd,optionalArg);

      // Construct and send desired PDU
      switch(cmd)
      {
      case DtLgrPbFile:
         // Set remote Logger playback-file
         exConn.sendStamped(DtLgrPbFilePdu(myId, target,
            optionalArg));
         break;

      case DtLgrRecFile:
         // Set remote Logger record-file
         exConn.sendStamped(DtLgrRecFilePdu(myId, target,
            optionalArg));
         break;

      case DtLgrQuit:
         // Terminate remote Logger
         exConn.sendStamped(DtLgrQuitPdu(myId,target));
         break;

      case DtLgrStart:
```

```
      // Rewind remote Logger (during playback)
      exConn.sendStamped(DtLgrStartPdu(myId,target));
      break;

   case DtLgrEnd:
      // Fast-forward remote Logger to end (during playback)
      exConn.sendStamped(DtLgrEndPdu(myId,target));
      break;

   case DtLgrPlay:
      // Start sending recorded traffic onto network
      // (during playback)
      exConn.sendStamped(DtLgrPlayPdu(myId,target));
      break;

   case DtLgrPause:
      // Pause sending recorded traffic onto network
      // (during playback)
      exConn.sendStamped(DtLgrPausePdu(myId,target));
      break;

   case DtLgrStop:
      // Stop sending recorded traffic onto network
      // (during playback)
      exConn.sendStamped(DtLgrStopPdu(myId,target));
      break;

   case DtLgrRecord:
      // Start recorded traffic from network (during record mode)
      exConn.sendStamped(DtLgrRecordPdu(myId,target));
      break;

   case DtLgrSetTime:
      // Jump to specified time during playback
      exConn.sendStamped(DtLgrSetTimePdu(myId,target,atof
         (optionalArg)));
      break;

   case DtLgrSkipTime:
      // Move forward or backwards by specified amount of time
      // during playback
      exConn.sendStamped(DtLgrSkipTimePdu(myId,target,
         atof(optionalArg)));
      break;

   case DtLgrTimeScale:
      // Control the speed of playback
      exConn.sendStamped(DtLgrTimeScalePdu
         (myId,target,atof(optionalArg)));
      break;

   case DtLgrPlayAction:
      // Action to take when entering play mode
```

```
            exConn.sendStamped(DtLgrPlayActionPdu(myId,target,
               (DtLgrMode)atoi(optionalArg)));
            break;

         case DtLgrRecordAction:
            // Action to take when entering record mode

            exConn.sendStamped(DtLgrRecordActionPdu(myId,target,
               (DtLgrMode)atoi(optionalArg)));
            break;

         case DtLgrEofAction:
            // Action to take when play mode reaches end of tape

            exConn.sendStamped(DtLgrEofActionPdu(myId,target,
               (DtLgrMode)atoi(optionalArg)));
            break;

         case DtLgrHeartbeat:
            // set heartbeat on remote Logger
            exConn.sendStamped(DtLgrHeartbeatPdu
               (myId,target,atof(optionalArg)));
            break;

         case DtLgrTimeout:
            // Set DIS timeout value for remote Logger
            exConn.sendStamped(DtLgrTimeoutPdu(myId,target,atof
               (optionalArg)));
            break;
         }
      }
}
```

Note: A program can use wildcards to identify the Loggers it sends commands to. For example:

```
DtObjectId target("255:255:7");
```

Fixed Bugs

- TLogger did not drain input when in record-stop mode. This means that when you gave the "record" command, packets that were received before the record command was given could be saved in the Logger file. This has been fixed in Logger 3.2.
- A bug in the dialog box that asks whether you want to overwrite an old Logger file has been fixed. This used to cause the Logger to crash on some platforms if you asked it to overwrite an existing file.