# DIS Logger Interchange Format (DLIF-95)
# Draft Standard

**Mike Garnsey**
**STRICOM**
**Orlando, FL**

**Keith Green**
**IDA**
**Alexandria, VA**

**Chris Kennedy**
**Concurrent**
**Berkshire, UK**

**Jesse Lieu**
**AcuSoft**
**Orlando, FL**

**Greg Schow**
**STRICOM**
**Orlando, FL**

**Scott Smith**
**IST**
**Orlando, FL**

**Doug Wahrenberger**
**Loral ADS**
**Orlando, FL**

## ABSTRACT

The proliferation of incompatible simulation data recording formats in the Distributed Interactive Simulation (DIS) community has consistently been a burden to implementors of DIS, to analysts, and to operational DIS sites. In response to this situation, the co-authors of this paper developed and implemented a prototype standard datalogger interchange format at the I/ITSEC 1994 DIS interoperability demonstration. Feedback from this I/ITSEC 1994 event has resulted in the revision of the prototype format into the current DIS Logger Interchange Format (DLIF-95) draft standard. DLIF-95 is intended to provide the mechanism for the entire DIS community to archive and exchange recorded simulation data efficiently and with confidence. This paper describes the DLIF-95 draft standard, the rationale behind its creation and the lessons learned in its implementation

## AUTHORS' BIOGRAPHIES

Mr. Mike Garnsey has been a project engineer at the United States Army Simulation, Training and Instrumentation COMmand (STRICOM) since June of 1991. He is involved in numerous DIS-related RDT&E activities, including DIS networking and protocol architecture research, development of distributed data-logging and session monitoring/ management techniques of DIS exercises, and DIS compliance and interoperability testing research. Mr. Garnsey holds a Masters of Science degree in Simulation Systems from the University of Central Florida, and a Bachelor of Science degree in Computer Engineering from the University of South Florida.

Keith Green is a Programmer and Analyst with the Institute for Defense Analyses (IDA) in Alexandria, VA. Keith holds a Masters of Engineering degree in Engineering Math and Computer Science from the University of Louisville in Louisville, KY. He has six years of experience with SIMNET and DIS and has been programming for almost 20 years.

Mr. Chris Kennedy has been a member of Concurrent Computer Corporations Engineering division for 11 years. He is a project manager responsible for several internal Research and Development projects involving distributed computing technology. He has worked on projects involving OSI standards, client/server application development for real time environments, and more recently on the development of DIS applications. Mr. Kennedy holds a Bachelor of Technology (Honours) degree from Brunel University, West London, United Kingdom.

Mr. Jesse Lieu, President and founder of AcuSoft, Inc. received his Bachelor of Science degree from the University of Taiwan and his MSEE from the University of Florida. He has more than 10 years experience in the design and implementation of simulation systems. In addition, to his duties as President of AcuSoft, Inc., Mr. Lieu is also the Chief Technical Officer of the company. In this capacity, Mr. Lieu directs the technical efforts of the AcuSoft system and software engineers. His simulation experience covers an extensive list of DIS related projects, the most current

being the ADST STOW/PRAIRIE WARRIOR 95 Exercise which utilized several AcuSoft Developer's Tools to include a Loral/AcuSoft ADST I project, the After-Action-Review (AAR) System STRIPES.

Mr. Greg Schow has been a project engineer at the United States Army Simulation, Training and Instrumentation COMmand (STRICOM) since March of 1994. While at STRICOM he has worked on developing DIS applications, Such as timestamping, a common database standard and DIS testing. Mr. Schow holds a Masters of Science degree in Industrial Engineering from Texas A & M University and a Bachelor of Science degree in Electrical Engineering (Minor in Physics) from the University of Portland, Portland, Oregon.

Scott H. Smith is Principal Research Associate at IST. He received an M.S. in Computer Science from the University of Central Florida in 1981. He is currently Project Manager for the TRIDIS project and Program Manager for DIS related activities at IST. His research interests lie in the areas of DIS, CGF, and Human/Computer Interfaces.

Doug Wahrenberger is currently employed on the Advanced Distributed Simulation Technology (ADST) contract for the US Army Simulation, Training, and Instrumentation Command as the delivery order manager for Architecture and Standards. Mr. Wahrenberger has over twenty two years of experience in the fields of space telemetry and command systems, C3I systems engineering, networked simulation, and simulation software engineering. He holds an M.E.E. degree in Electrical Engineering and has completed his course work for a Ph.D. (ABD) in communications engineering from the Catholic University of America.

# DIS Logger Interchange Format (DLIF-95)
# Draft Standard

## THE PROBLEM

A datalogger is a computer with associated software (called a datalogger program, or just 'datalogger' for short) which records the network packets from a simulation exercise to a file on disk or some other medium. Whether the data within a datalogger file is rigorously analyzed for research or used to replay the events of a simulation exercise for demonstration or After Action Review (AAR), it is usually an important record of a DIS activity. There are many occasions when the availability of recorded simulation traffic for data analysis and review is critical to the success of a simulation exercise or experiment. In the case of training exercises, data analysis of the network packets comprising the exercise has the potential to assist with the process of After Action Review. And when simulation is used as a tool in the evaluation of tactics or doctrine or as a guide in the materiel acquisition process, it is imperative that a complete record of the simulation experiment be maintained for analysis and archival purposes.

Unfortunately, there are numerous datalogger file formats currently being implemented at various organizations active in DIS. The incompatibility between the different formats, however, prevents the convenient exchange of both the recorded simulation data and the associated data analysis tools between organizations. Instead of focusing DIS software development activities on enhanced data review and analysis tools, many DIS sites expend considerable time writing software routines to convert datalogger files into a format they are familiar with. Additionally, a site does not always know what format a particular file was originally logged in, which makes the problem more difficult. As more organizations become active in DIS and develop their own datalogger formats, this file exchange problem will only get worse.

## ORIGINS

In September of 1994, STRICOM formed a "tiger team" composed of a diverse group of government, industry and academic DIS logger file format activists. Applying the Internet community's philosophy of "rough consensus and running code", the immediate goal of this logger file format team was to baseline a logger file format for implementation at the 1994 I/ITSEC DIS interoperability demonstration — with the ultimate goal being to evaluate and promote this I/ITSEC-94 logger file format as an interchange format for adoption by the DIS community. A substantial amount of DIS network traffic at I/ITSEC-94 had been archived in the prototype 1994 DIS Logger Interchange Format (DLIF-94). Lessons learned from implementation and usage of this format have resulted in the revision of the prototype format into the current DLIF-95 format. The DLIF-95 format has been successfully presented as a proposed DIS standard datalogger interchange format to the Fidelity Management and Usability (FMU) Working group at the 12th DIS Workshop on the Standards for the Interoperability of Distributed Simulation. Currently, DLIF-95 is in the process of being forwarded as a draft standard for incorporation into the IEEE 1278 DIS standard. DLIF-95 is being implemented at major DIS events such as the Prairie Warrior '95 exercise, and is planned for use at the 1995 I/ITSEC DIS demonstration and, with modifications for multicast, in the STOW exercises.

## DLIF OVERVIEW

The general requirements of the DLIF-95 file format are as follows:

- All fields are in network byte order (most significant byte first --otherwise known as "big endian").
- All numerical fields are set to 0xFFFF(FFFF), signed or unsigned, if they are not supported, or have no real value assigned to them.
- ASCII string fields use the "C" convention of terminating with a null character (so that "no value available" is specified by having a null character in the first byte).
- All Major Data Components (File Header, Each Segment, File Trailer) must be 64-bit aligned.

The DLIF-95 file format (Figure 1) consists of three major components: the file header, one or more segments, and the file trailer consisting of an index of the segments. As figure (1) illustrates, all segments begin with a header, but the frame portion of a segment may be located either before or after the index portion of the segment. The format is designed to accommodate both low-level data link layer network packets, as well as high-level-only DIS types of simulation PDU data.

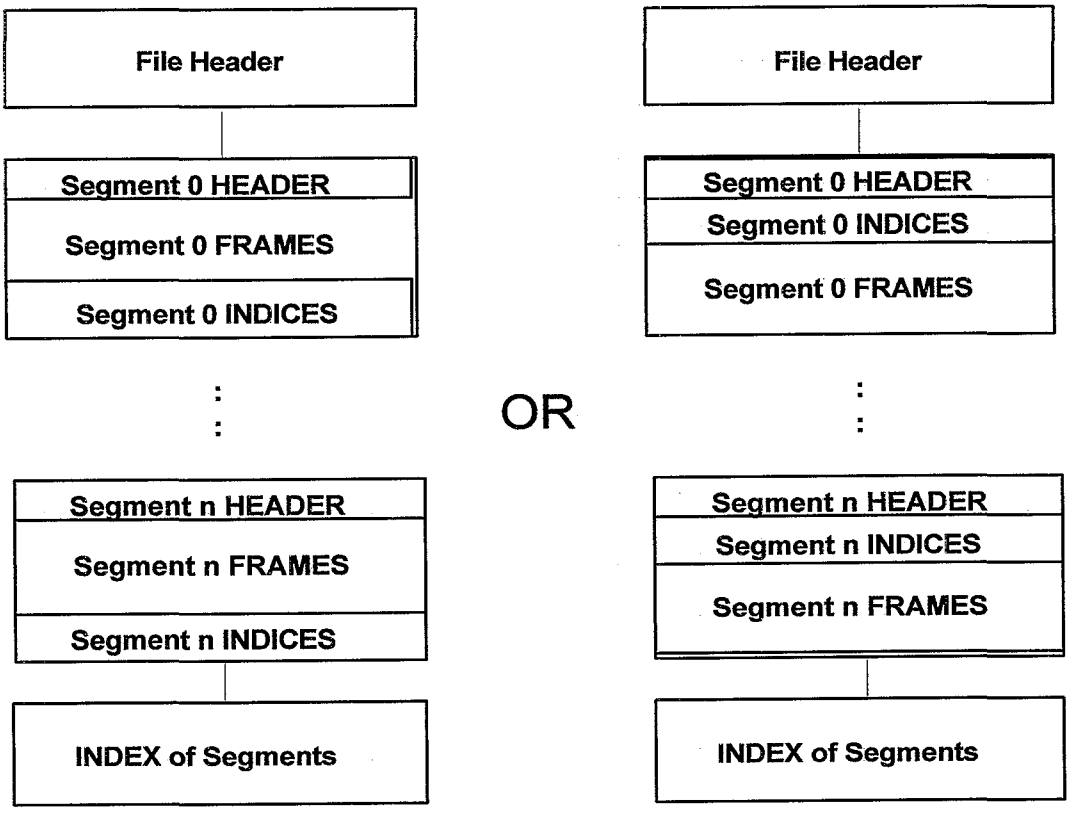


Figure 1: DLIF-95 Overview

The following is a more detailed description of these three major DLIF-95 components:

1. The file header (Figure 2) contains fixed length text fields to document various aspects of the recorded DIS data file, such as: the simulation exercise name, data recording site, and terrain database name. The file header component also includes a variable length text field at the end of the file header to capture any data deemed pertinent to describing the recorded DIS data file.

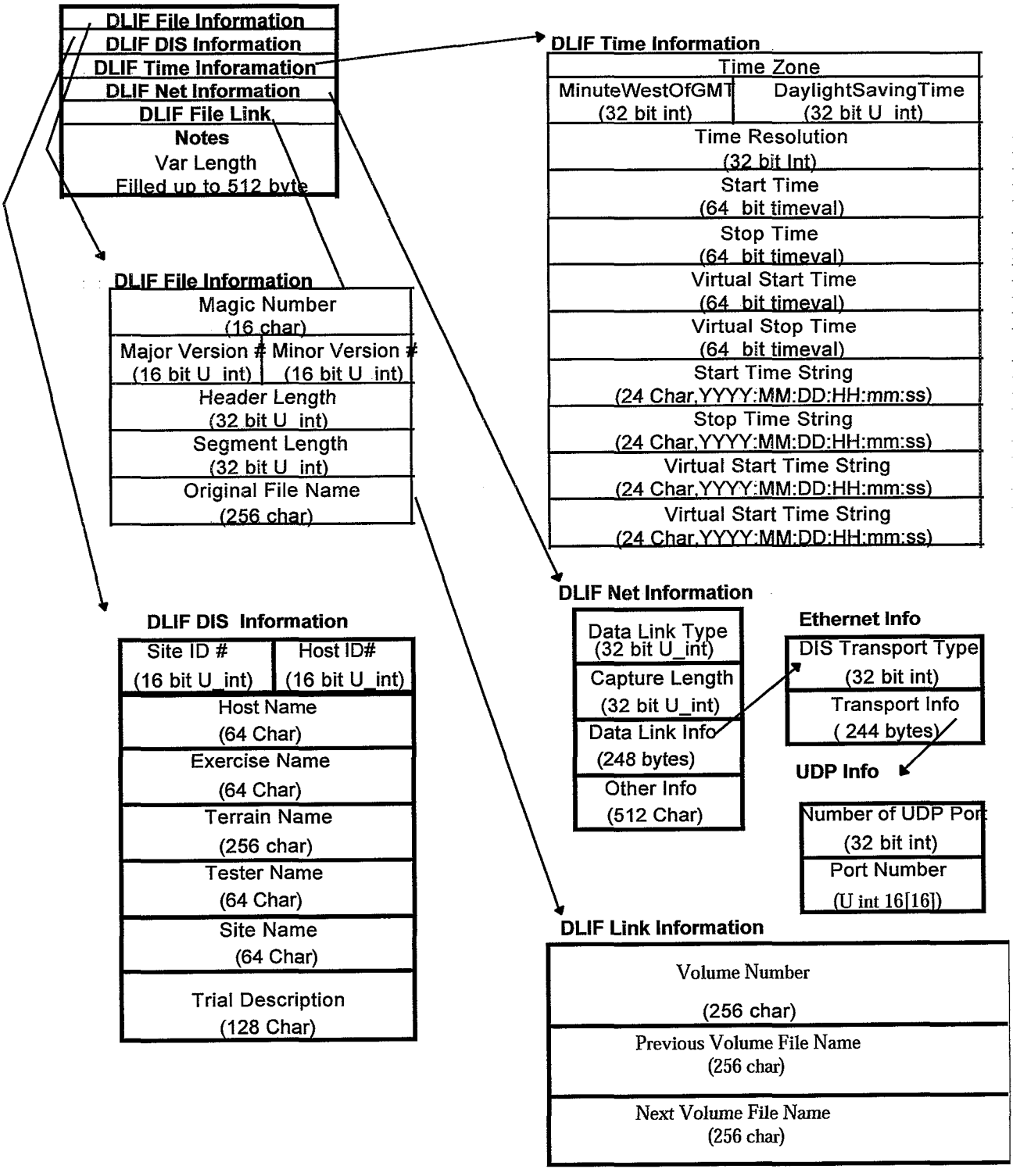


Figure 2: DLIF-95 File Header Structure

2. File segments (Figure 3), which may be fixed or variable length, consist of three components: the segment header, the segment frames and the segment indices. All segments start with a header, but the order of the segment frames and the segment indices is left to the implementer. The general segment requirements are as follows:

- Segments must be 64-bit aligned
- In a segment, the INDEX may appear anywhere in the segment.
- INDEX portion has to be contiguous.
- INDEX portion has to be 64-bit aligned.
- The DATA portion has to be contiguous,
- The DATA portion has to be 64-bit aligned
- Actual DATA frames may not straddle segments.
- DATA record shall be packed: no padding, if the data record contains the network frame at the Data Link Level.
- DATA record shall be packed and 32 bit aligned if the data record contains only Application Layer Level data.

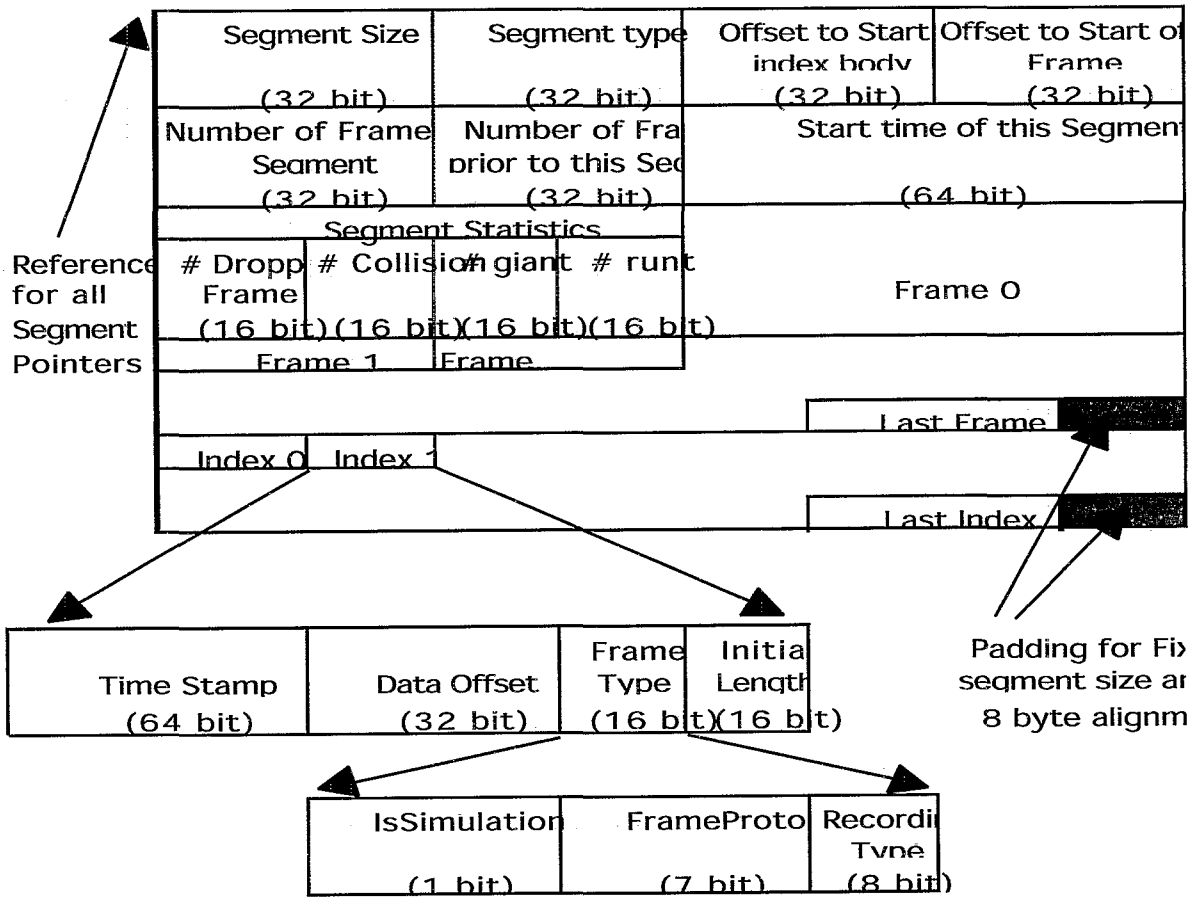


Figure 3: DLIF-95 Segment Structure

3. The file trailer (Figure 4), which contains the segment index list, is of fixed length. The fixed length limits the number of segments to 1024, but this limitation is outweighed by the ease of verifying the ends of files afforded by a fixed length field.

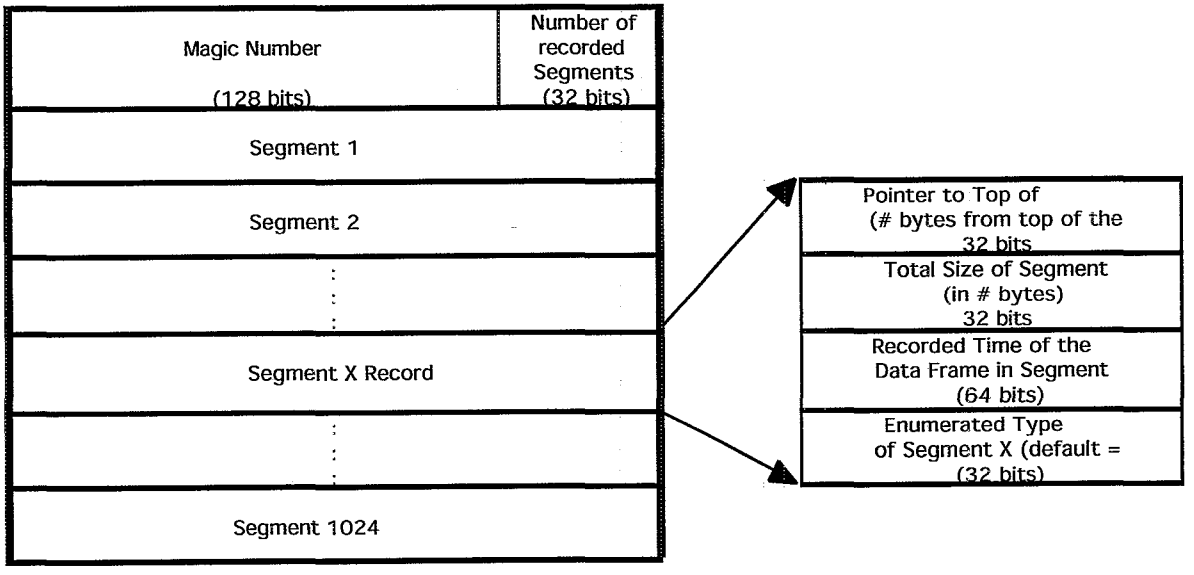


Figure 4: DLIF-95 File Trailer Structure

## RATIONALE FOR THE DLIF-95 APPROACH

When the DLIF Tiger team first came together there was a division over whether to use fixed length or variable length segments in the files. Variable length was favored, because of its flexibility, while fixed length was favored for its robustness. Other issues were: What size should segments be? In what order should the data be organized? How are unused fields dealt with? DLIF-95 has taken these issues into account and attempted to address these and other issues to create a flexible and robust interchange format.

### The Requirement to Segment the File

Network packets are of variable length, and because of the way DIS uses networks, there are a lot of them. Unless packets are concatenated, many gigabytes of available storage will be wasted in buffering smaller packets to the largest packet size. This concatenation makes the packets difficult to find, especially in a large file. There are several ways in which log files are read. The most common is to locate a particular packet (by time, packet number, etc.), and from there, read sequentially until some other point (usually when the area of interest has passed, or the end of the file has been reached). To facilitate searching, segmentation was introduced, breaking the file up into self contained segments such that the segment boundaries were easily locatable. Coarse grained searching can then quickly yield the segment in which a particular packet is stored, and then a fine grain search within the segment will yield the packet itself. A segment would have a header, which contains structure information about the segment (where the packets are in the segment etc.) and index information (packet numbers, timestamp information etc.) about the packets in the segment.

### The Fixed vs. Variable Segment Debate

The main purpose of a DLIF file is for interchange, i.e. moving files from one system to another, which involves some sort of transport mechanism (tape, floppy, network, etc.). All forms of transport introduce a risk of corruption, some less than others. It was thought that most forms of corruption would be localized to particular parts of a file (i.e. most of the file would be intact, but there may be a few small areas where there were corruptions). If the corruption occurred at a segment header most of that segment would be lost. If the segments are

of variable length, and the length field had been corrupted, the remainder of the file would be lost. If the segments were of fixed length, then only the single segment would be lost.

There are also processing advantages to fixed length segments. During replay, if records are of fixed length, the replay program knows in advance the size of the segment, and can therefore ensure that a buffer of the correct size is available to receive the data. Also, when it comes to read a segment, it does not have to read it in two hits (the first to read the segment header, which will contain the size of the segment, and the second to read the rest of the segment). When writing a file (while logging), a datalogger can select a segment size that best matches the buffering within his system to achieve the optimum memory to file data transfer performance.

Fixed length segments do, however, waste some storage space (as a packet cannot straddle segments), and it can impose implementation constraints that are not required for variable length segments. An implementation may fix the number of packets per segment, and without waisting any storage or having to further manipulate the data, write it to the storage medium. Thus, by using a buffer big enough to hold the data frames, even if they were all the maximum size (Maximum-Transfer-Unit-size x number-of-data-frames + segment-header-size + segment-index-size), packets can be streamed into the buffer, until the packets per segment limit is reached, and then the used portion of the buffer is written to the file. As memory is now fairly inexpensive, sacrificing 1 or 2 megabytes for programming simplicity and the subsequent performance advantage can be well worth the investment.

Most of the advantages of fixed length segments has been addressed for variable length segments by adding a trailer to the file. The trailer contains indexes to all the segments in the file. There is still a problem though, if the file is on a streaming device (such as magnetic tape), because the trailer is at the end of the file, the program has to stream the tape to the end to read the trailer before starting the normal processing of the file. However, since the purpose for the trailer is to speed random searches, this may not be an issue, since tapes are not random access devices.

In order to allow maximum flexibility to the implementors, a segmentation strategy was adopted that supported both fixed and variable length segments. Thus the implementor may choose to log to a file with either fixed segment sizes or variable segment sizes. For the most part, it does not matter to an analysis tool or a replay tool what segment strategy is used in the file.

There is a disadvantage with using the trailer system to identify the segment structure of the file. Should the datalogger crash, (hardware failure, power fail etc.) the trailer will not have been written, so the data already captured will not be easily recoverable. Also, a common file corruption is truncation. As the trailer is at the end of the file, it is more vulnerable. The trailer information can, however, be regenerated by reading the segment headers sequentially (for variable length segments), or randomly (for fixed length segments).

### The 64 Bit Alignment Requirement

Analysis, as noted above, is a common use of logfiles and logfiles are probably read more often for analysis than replay. The reality of current computer technology is that numerical values need to be properly aligned — i.e. a 16 bit integer must begin on a 16 bit boundary, a 32 bit integer on a 32 bit boundary, etc. As an aid to efficiency, the DLIF has attempted to ensure all numerical values are properly aligned within a segment, by forcing vital data areas onto 64 bit boundaries. Without this, data from the file would have to be copied from the read area, into another (properly aligned) area before it is accessed. In some cases, even the data in captured packets can be relied upon to be properly aligned.

The main reason 64 bit boundaries were chosen (rather than 32 bit boundaries) is that although current computers generally read information in 32 bit increments, there are already some that use 64 bit words. Without impairing current machines, the DLIF-95 format allows for a seemless jump to the next generation of computers.

## LESSONS LEARNED

When the DLIF Tiger Team first came together, the overriding design requirements for a standard datalogger file format were flexibility and ease of implementation. To that end, the experimental I/ITSEC 1994 datalogger file format (DLIF-94) had a variable length file trailer. However, software implementation experience of this flexible trailer format structure resulted in the revised DLIF-95 format adopting a less flexible but more easily implemented

fixed length file trailer. In general, since DLIF-95 has evolved from rough consensus and compromise between this paper's co-authors, the following flexibility vs. implementability issues are being debated:

1. Should the timestamp in the data frame index be changed from 64-bit absolute time to 32-bit relative time (relative to the beginning of the appropriate segment).
2. Should the file trailer be changed from a fixed number (1024) to a variable number of segment records
3. Is the 32-bit pointer limitation of DLIF-95 (restricting individual DLIF-95 file sizes to a maximum of 4 gigabytes) acceptable.
4. Should the ordering of the major DLIF structures to accommodate the reading to and/or writing from streaming tape media be changed?
5. Should a DLIF compression scheme be implemented and, if so, how?
6. How should multicast network protocol information be recorded (currently being addressed by STOW)?
7. Can the archival information be improved to better support DLIF file merging from multiple sources?
8. Who is the cognizant authority for issuing enumerations for fields in the DLIF file (version number, etc)?

## CONCLUSION

The DLIF-95 standard, while it is neither perfect nor complete, is sufficient to meet the needs of most datalogger users: analysts, developers, operational, and site personnel. There are unresolved issues, such as the variable or fixed length segments argument. The general concensus, however, is that, while this is an important issue, it should not prevent the DIS community from adopting a standard that works and offers many tangible benefits. An interchange format will facilitate sharing of information, services, and analytic capabilities among disparate sites; it may assist in file archival for historical analysis; it opens the door for standard analytic capabilities at all sites; it allows sites to share demos easily; and it facilitates interoperability testing and with other testing.

References

Garnsey, Mike; Green, Keith; Kennedy, Chris; Lieu, Jesse; Schow, Greg; Smith Scott; Wahrenberger, Doug, "DIS Interchange Format Proposed Standard", 12th DIS Workshop Proceedings, March 13-17 1995, pp. 249-254.

Juliano, Mark, "Standard DIS Data Logger Format", 11th DIS Workshop Proceedings, September 26-30 1994, pp. 127-131.

Green, Keith L., "A position Paper for Standard Datalogger Formats", 10th DIS Workshop Proceedings, February 12 1994, pp. 545-555.

Appendix A

| Field Size (bytes) | DLIF-95 File Header Structure | |
|---|---|---|
| 284 | File Info | Magic Number - 16-character array |
| | | Major Version # - 16-bit unsigned integer |
| | | Minor Version # - 16-bit unsigned integer |
| | | File Header Length - 32-bit unsigned integer |
| | | Maximum Segment Size (in bytes) - 32-bit unsigned integer |
| | | Original DLIF-95 File Name - 256-char array |
| 644 | DIS Info | Site ID # - 16-bit unsigned integer |
| | | Host ID # - 16-bit unsigned integer |
| | | Host Name - 64-char array |
| | | Exercise Name - 64-char array |
| | | Terrain Name - 256-char array |
| | | Tester Name - 64-char array |
| | | Site Name - 64-char array |
| | | Trial Description - 128-char array |
| 140 | Time Info | # Minutes West of GMT - 32-bit integer |
| | | Daylight Savings Time - 32-bit integer |
| | | Time Resolution - 32-bit integer |
| | | Start Time - 64-bit timeval structure |
| | | StopTime - 64-bit timeval structure |
| | | Virtual Start Time - 64-bit timeval structure |
| | | Virtual StopTime - 64-bit timeval structure |
| | | Start Time Text - 24-char array |
| | | Stop Time Text - 24-char array |
| | | Virtual Start Time Text - 24-char array |
| | | Virtual StopTime Text - 24-char array |
| 768 | Network Info | Data Link Type - 32-bit enumeration |
| | | Max (non-sim) packet capture size - 32-bit unsigned integer |
| | | Data Link Info Record - 248 byte data structure |
| | | Other Network Info - 512-char array |
| 516 | Volume Link Info | Volume # of this DLIF-95 file - 32-bit unsigned integer |
| | | File Name of Previous DLIF-95 Volume - 256-char array |
| | | File Name of Next DLIF-95 Volume - 256-char array |
| T | Notes | variable length character array |

Total DLIF-95 File Header Structure Size = 2352 + T bytes

T = 1232 + (N•512) bytes

where N = number of additional 512-byte Notes space blocks desired

| Field Size (bytes) | DLIF-95 File Segment Structure (Option #1: Frame Data followed by Index Data) | |
|---|---|---|
| 40 | Segment Header | Segment Size - 32-bit unsigned integer |
| | | Segment Type - 32-bit enumeration |
| | | Byte offset to top of Index Data from top of segment 32-bit unsigned integer |
| | | Byte offset to top of Frame Data from top of segment 32-bit unsigned integer |
| | | Number of Frames in this Segment 32-bit unsigned integer |
| | | Total Number of Frames before this Segment 32-bit unsigned integer |
| | | Start time of this Segment - 64-bit timeval structure |
| | | # dropped packets - 16-bit unsigned integer |
| | | # collisions - 16-bit unsigned integer |
| | | # giant packets - 16-bit unsigned integer |
| | | # runt packets - 16-bit unsigned integer |
| $\sum_{i=1}^{n} K_i + P_F + P_S$ | Frame Data | Frame #1 Variable Size Record |
| | | ⋮ |
| | | Frame #n Variable Size Record |
| | | Padding |
| n x 16 | Index Data | Index Record #1 for Frame #1 |
| | | ⋮ |
| | | Index Record #n for Frame #n |

$$\text{Total Segment Size} = 40 + \sum_{i=1}^{n} K_i + P_F + P_S + (n \bullet 16) \quad \text{bytes}$$

n = number of frames in this segment

$P_F$ = Frame Data padding to the 64-bit alignment (0-7 bytes)

$P_S$ = padding for fixed-segment-size DLIF-95 implementations (64-bit aligned)

$$P_F = \lceil \sum_{i=1}^{n} K_i \ / 8 \rceil \bullet 8 \ - \sum_{i=1}^{n} K_i$$

$$P_S = \text{Fixed Segment Size} - ( 40 + \sum_{i=1}^{n} K_i + P_F + (n \bullet 16) ) \quad \text{bytes}$$

$\lceil x \rceil = 1 +$ largest integer $< x$

| Field Size (bytes) | DLIF-95 File Segment Structure (Option #2: Index Data followed by Frame Data) | |
|---|---|---|
| 40 | Segment Header | Segment Size - 32-bit unsigned integer |
| | | Segment Type - 32-bit enumeration |
| | | Byte offset to top of Index Data from top of segment 32-bit unsigned integer |
| | | Byte offset to top of Frame Data from top of segment 32-bit unsigned integer |
| | | Number of Frames in this Segment 32-bit unsigned integer |
| | | Total Number of Frames before this Segment 32-bit unsigned integer |
| | | Start time of this Segment - 64-bit timeval structure |
| | | # dropped packets - 16-bit unsigned integer |
| | | # collisions - 16-bit unsigned integer |
| | | # giant packets - 16-bit unsigned integer |
| | | # runt packets - 16-bit unsigned integer |
| n x 16 | Index Data | Index Record #1 for Frame #1 |
| | | ⋮ |
| | | Index Record #n for Frame #n |
| $\sum_{i=1}^{n} K_i + P_F + P_S$ | Frame Data | Frame #1 Variable Size Record |
| | | ⋮ |
| | | Frame #n Variable Size Record |
| | | Padding |

$$\text{Total Segment Size} = 40 + \sum_{i=1}^{n} K_i + P_F + P_S + (n \bullet 16) \quad \text{bytes}$$

n = number of frames in this segment

$P_F$ = Frame Data padding to the 64-bit alignment (0-7 bytes)

$P_S$ = padding for fixed-segment-size DLIF-95 implementations (64-bit aligned)

$$P_F = \lceil \sum_{i=1}^{n} K_i \ / 8 \rceil \bullet 8 \ - \sum_{i=1}^{n} K_i$$

$$P_S = \text{Fixed Segment Size} - ( 40 + \sum_{i=1}^{n} K_i + P_F + (n \bullet 16) ) \quad \text{bytes}$$

$\lceil x \rceil$ = 1 + largest integer < x

| Field Size (bytes) | DLIF-95 Segment Frame Record Variable Length Structure (Data-Link Level Network data) |
|---|---|
| $K_i$ | Network Frame Data at the Data-Link Level (packed data without padding) |

$K_i$ = Total Frame Record Size (bytes)

| Field Size (bytes) | DLIF-95 Segment Frame Record Variable Length Structure (Application-Layer Only data) |
|---|---|
| $D_i$ | Application-Layer only portion of Network Frame Data (packed data without padding) |
| $P_i$ | Padding to 32-bit alignment |

$K_i$ = Total Frame Record Size = $D_i + P_i$ bytes

$D_i$ = length of packet data in bytes

$P_i$ = padding to the 4-byte alignment in bytes

$P_i = \lceil D_i / 4 \rceil \bullet 4 - D_i$

$\lceil x \rceil = 1 +$ largest integer $< x$

| Field Size (bytes) | DLIF-95 Segment Index Record Structure (Data-Link Level Network data) |
|---|---|
| 8 | Frame Data Time Stamp - 64-bit timeval structure |
| 4 | Byte Offset from top of Segment to start of Frame Record 32-bit unsigned integer |
| 2 | Frame Type - 16-bit enumeration |
| 2 | Original Length of Frame Data - 16-bit unsigned integer |

Total Segment Index Record Structure Size = 16 bytes

| Field Size (bytes) | DLIF-95 File Trailer Structure | |
|---|---|---|
| 20 | Trailer Header | Magic Number - 16-character array |
| | | Total number of Segments Used - 32-bit unsigned integer |
| 20 | Segment #1 Index Record | Byte Offset from Top of File to Top of Segment #1 32-bit unsigned integer |
| | | Length (in bytes) of Segment #1 - 32-bit unsigned integer |
| | | Segment #1 Start Time - 64-bit timeval structure |
| | | Segment #1 Type - 32-bit enumeration |
| | ⋮ | |
| 20 | Segment #1024 Index Record | Byte Offset from Top of File to Top of Segment #1024 32-bit unsigned integer |
| | | Length (in bytes) of Segment #1024 - 32-bit unsigned integer |
| | | Segment #1024 Start Time - 64-bit timeval structure |
| | | Segment #1024 Type - 32-bit enumeration |

Total File Trailer Structure Size = 20,500 bytes

# Appendix B

## DLIF-95 Terminology

**Application-Layer Only Data** - The portion of transmitted data unit which excludes the transport (UDP) and lower layer header information.

**Data Frame** - Data on a network appear as discrete units called Protocol Data Units (PDUs) or Data Frames. A Data Frame may contain Application Layer Only data or Data-Link Level Network Data.

**Data-Link Level Data** - Transmitted data unit including the Transport and lower layer headers (UDP header, IP header and MAC header).

**Data-Link Type** - This field in the file header defines the data link type used during capture. The only currently defined values for this field is 1, which indicates that the data link layer is Ethernet, 2 for FDDI and 3 for ATM.

**Datalogger** - A computer (with associated software) which records the data units, transmitted on a network by systems from a distributed simulation exercise, and stores them in a file.

**Datalogger Program** - The software portion of a Datalogger. (Sometimes the datalogger program itself is called the 'datalogger'.)

**DIS Transport Type** - This field indicates the transport type of the DIS packets in the file. The currently defined value for this field is 1 for UDP/IP.

**DLIF-95** - DIS Logger Interchange Format vintage 1995.

**Dropped Packet** - A data unit that has not been captured, or has been discarded for reasons other than filtering.

**Exercise Name** - Name, in human readable text, given to an exercise.

**File Header** - Initial portion of a DLIF-95 logfile which contains general information about the contents of the Logfile.

**Giant Packet** - A data unit that is bigger than the biggest permitted, i.e. a number of bits (or bytes) greater than the maximum transfer unit of the medium.

**Host ID #** - A unique number that identifies a particular host system at a particular site.

**Host Name** - Name , in human readable text, that uniquely identifies a particular host system on the network.

**Index Data** - Portion of segment which contains index, information about the transmitted data units in the Frame Data portion of the segment.

**Logfile** - A file containing data captured from the network during a distributed simulation exercise.

**Magic Number** - The initial bytes of a file which uniquely identifies the files "type". A DLIF file has the first 16 bytes set to the ASCII string "#!DIS-Datalogger".

**Major Version Number** - A number which uniquely identifies the type of the DIS logger file. For the DLIF-95 format, Major Version Number is 5.

**Maximum Transfer Unit (MTU)** - The largest unit you can transfer over a given segment. (For example, Ethernet MTU is approximately 1500 bytes. FDDI is approximately 4096 bytes, I think.) If you have a 'packet' of data bigger than that to transfer, then you have to rely that your packet driver (or your application or SOMETHING) has to do segmentation of the packet at the sending end and reassembly at the receiving end.

**Minor Version Number** -A number which uniquely qualifies which variation of the logger type identified by the Major Version Number. The MinorVersion Number for DLIF-95 file is 0.

**Multi-volume logfile** - A sequence of files (volumes) which together form the complete record of the data captured by a single Datalogger.

**Network Order** - The order in which data appears on the network (i.e. most significant byte and the most significant bit at the smallest address).

**Packet Capture Size** - Largest packet that may be captured from the network. This depends on the medium from which the data was captured.

**Packet Collisions, Number of Packet Collisions** - A characteristic of the Ethernet protocol, which occurs when more than one host on the network attempts to transmit data at the same time.

**Protocol Data Unit(PDU)** - See Data Frame.

**Runt packet** - A data unit that is smaller than the smallest permitted.

**Segment** - A discrete portion of a DLIF-95 logfile in which contains the captured data.

**Segment Header** - Initial portion of a Segment which contains general information about the data held in the segment.

**Site ID #** - A unique number that identifies a particular site connected to the network.

**Site Name** - The name, in human readable text that uniquely identifies a particular site.

**Start Time (File, Segment, Frame)** - The real world (wall clock) time of the start of the exercise.

**Stop Time Stop Time** - Real world (wall clock) time of the end of the exercise.

**Terrain Name Terrain Name** - The name, in human readable text, given to the terrain used in the exercise.

**Tester Name** - The name, in human readable text, of the operator of the Datalogger.

**Time Resolution** - The structure used to timestamp packets contains two 4 byte integers. The first indicates number of seconds since the start of January 1, 1970. The second is fractions of a second. The resolution of the fractions is determined by the Time Resolution field in the file header. The value of the fraction can be determined by multiplying the fraction field by 10 to the power of the Time Resolution value (i.e. if the value of the fraction is 234, and the Time Resolution is -6, the value of the fraction is 0.000234).

**Trial Description** - The description, in human readable text, of the purpose/disposition of the exercise captured in the logfile.

**UDP Port Number** - These fields in the UDP Information structure in the file header indicate which UDP ports contain DIS data frames.

**Virtual Start Time** - Virtual world time of the start of the exercise.

**Virtual Stop Time** - Virtual world time of the end of the exercise.

**Volume Number** - Number which identifies the position of an individual file within a sequence of files that make up a Multi-volume logfile.

## Appendix C

### Subscribing to the Mailing List

To subscribe to the mailing list, send an email message to listserv@proto.ida.org with no subject line and a single line in the body of the message which says: subscribe loggers <First Name> <Last Name>, where <First Name> and <Last Name> are your real name. For example, if George Washington wanted to subscribe to the group, he would send an email with the message body saying:

subscribe loggers George Washington

Thereafter, if George wished to send email to all participants in this group, he would send his messages to loggers@proto.ida.org. Note: It is polite to include your name in messages to the group, since the list server will remail the message to the participants with its own name, and no would otherwise know who sent the original message.

We are currently discussing collecting sample datalogger files against which to test datalogger implementations. For information regarding these developments, please subscribe to the aforementioned mailing list.