Schlumberger

LIS 79 DESCRIPTION REFERENCE MANUAL

Schlumberger

LIS 79 DESCRIPTION REFERENCE MANUAL

CONFIDENTIAL

CONTENTS

FIGURES

Chapter 1.

INTRODUCTION

1.1. Definition: What Is The LIS?

The *Log Information Standard (LIS)* was developed by Schlumberger in 1974 to provide a standard method of recording well log data. This version of the LIS has come to be called the *79 Subset* or *LIS 79*.

In 1984, Schlumberger defined an extended version of the LIS in order to allow more efficient and flexible use of the present data and to permit easier changes as new logging services and their data are made available. This version of the LIS has come to be called the *Enhanced LIS* or *LIS 84*.

All well logs generated by the Schlumberger Cyber Service Units (CSU) and by Schlumberger Field Log Interpretation Centers (FLICs) are recorded on magnetic tapes in LIS format.

1.2. Aim: Why Should You Read This Manual?

This manual is intended to describe in detail the LIS 79 only. As a reference guide, it has several important features, including:

- A pictoral discussion of the relationships between parts of the LIS
- A simplified explanation of the terminology used to define each part of the LIS
- A complete description of all parts of the LIS.

1.3. Audience: Who Is This Manual For?

This manual serves as a reference source for several groups of computer professionals:

- Analysts who need to plan for and evaluate future changes to the LIS
- Programmers who are assigned to modify the LIS

1.4. Structure: How Is This Manual Organized?

The LIS 79 Description Manual has been recast. The purpose of such restructuring is to ensure that it conforms as closely as possible to the LIS 84 Description Manual. The reason for this is to simplify references between one LIS and another–i.e., the differences in their configurations, terminologies, functioning, and the like.

Put plainly, this means that major chapters and sections in one manual correspond to those in the other. This manual is divided into six (6) chapters and several appendices:

Chapter 1 :	*INTRODUCTION*
Chapter 2 :	*DATA ORGANIZATION*, a description of the physical and logical hierarchies, their organizations and relationships
Chapter 3 :	*LOGICAL RECORD SYNTAX*, a description of how logical records are represented and a summary of the logical record format types
Chapter 4 :	*LOGICAL RECORD SEMANTICS*, a description of LIS data and the logical record types that contain them
Chapter 5 :	*LIS IMPLEMENTATION*, a discussion of the software necessary to interface between the application programs and the physical device on which the data are stored
Chapter 6 :	*ENCRYPTION*, a discussion of how proprietary data are handled
Appendix A:	*ASCII CODES*, a list of ascii codes (alphabetized)
Appendix B:	*REPRESENTATION CODES*, a list of representation codes (numerically ordered)
Appendix C:	*CHECKSUM ALGORITHM*, a description of the algorithm used to compute the checksum

1.5. Notation: How Is This Manual Written?

Numerous cases of unusual capitalization appear in this manual. These words are those that have special, formal, technical connotations in the context of the LIS. The capitalization should remind you not to attach the normal English connotations to these words.

Acronyms are always fully defined before they are used.

Tables for record types include descriptions of all information listed. Parentheses enclosing an entry indicate that this entry is a component of the first entry above it that is not enclosed in parentheses. For example, in the following excerpt from the File Header Logical Record, the 6 byte (Service Name), 1 byte ("."), and 3 byte (File Number) entries are actually components of the 10 byte File Name entry. Note, too, that each table entry has a comment number associated with it; actual comments will appear immediately below the table.

File Header Logical Record (Type 128)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
File Name	10	65	2
(Service Name)	(6)	(65)	(2a)
(".")	(1)	(65)	(2b)
(File Number)	(3)	(65)	(2c)
2 blanks	2	65	3
Service Sub Level Name	6	65	4

1.6. Maintenance: Who Updates This Manual?

This manual is maintained by the Austin Systems Center/FLIC Department and updated on an as-needed basis. Questions, suggestions, or errors in the manual should be directed to the following address:

```
Schlumberger Well Services
Austin Systems Center
Attn: FLIC Department
12112 Technology Blvd.
Austin, Texas 78727
```

1.7. Related Documents: How Does This Manual Fit In?

The How to Use LIS/A Manual describes the set of FORTRAN subroutines designed to provide a set of tools for reading and writing information represented under the LIS.

The LIS 84 Description Manual describes in detail how information is represented and labelled in the 1984 Log Information Standard.

The LIS/A System's Programmer Guide gives instructions for a systems programmer on how to install and maintain LIS/A on your system.

Mapping the 79 Subset and the Enhanced LIS explains those ways in which the LIS 84 differs from the currently used LIS 79 and by which data written under one version of the LIS may be rewritten under corresponding data of the other version of the LIS.

Chapter 2.

DATA ORGANIZATION

2.1. Hierarchy

The LIS format was designed primarily for use with magnetic tape, although it may also be used with other types of storage media. A hierarchy is used for storing and retrieving data. This hierarchy separates the logical representation of the data on the tape from the tape's physical makeup.

The LIS format is made up, then, of two structures:

- the *Logical Structure*, referring to the type and organization of the data
- the *Physical Structure*, referring to the physical dimensions of the data.

Each structure is discussed separately below.

2.2. Logical Structure

The LIS logical structure consists of three (3) elements:

- the *Logical Record*, a group of bytes or data characters
- the *Logical File*, a group of related logical records
- the *Logical Tape*, a group of logical files.

The relationship of these elements is illustrated in the following diagram.

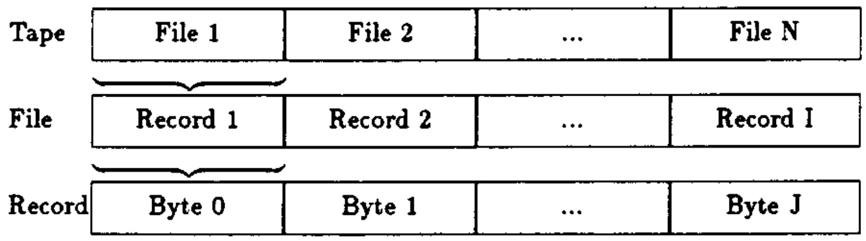


Figure 2.1: Logical Hierarchy

2.2.1. Logical Records (LR)

Logical Records form the basic coherent bodies of information in the LIS format. A Logical record consists of

- a *Logical Record Header (LRH)*
- a *Logical Record Body (LRB).*

2.2.1.1. Logical Record Header (LRH)

The Logical Record Header has the following format:

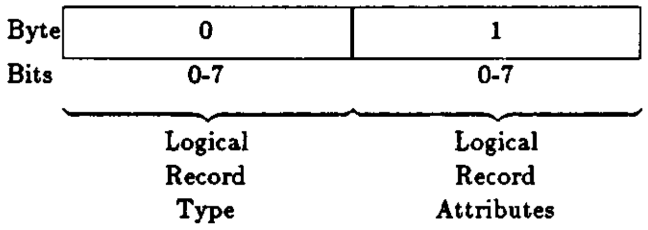


Figure 2.2: Logical Record Header

Where:

- *Logical Record Type (LRT)* is an 8-bit, unsigned, binary integer quantity (Representation Code 66) specifying the type of the logical record. This record type is used to tell the program how to interpret the record.
- *Logical Record Attributes (LRA)* is an 8-bit bit-string, unused and reserved.

2.2.1.2. Logical Record Body (LRB)

The Logical Record Body is an ordered set of 8-bit bytes.

2.2.2. Logical Files (LF)

A Logical File is composed of Logical Records. The first logical record in a file must be the *File Header Logical Record (FHLR)*. A Logical File is terminated by the *File Trailer Logical Record (FTLR)*.

2.2.2.1. File Header Logical Record (FHLR)

The File Header Record is 58 bytes in length. It contains general information identifying the file.

File Header Logical Record (Type 128)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
File Name	10	65	2
(Service Name)	(6)	(65)	(2a)
(".")	(1)	(65)	(2b)
(File Number)	(3)	(65)	(2c)
2 blanks	2	65	3
Service Sub Level Name	6	65	4
Version Number	8	65	5
Date of Generation	8	65	6
(Year)	(2)	(65)	(6)
("/")	(1)	(65)	(6)
(Month)	(2)	(65)	(6)
("/")	(1)	(65)	(6)
(Day)	(2)	(65)	(6)
1 blank	1	65	3
Maximum Physical Record Length	5	65	7
2 blanks	2	65	3
File Type	2	65	8
2 blanks	2	65	3
Optional Previous File Name	10	65	9

Figure 2.3: File Header Logical Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *File Name* is a unique name for a file within a logical tape and consists of following parts:
   (a) *Service Name* is the name of the service or program that created the tape.
   (b) *"."* is simply a separator.
   (c) *File Number* is a 3-character counter (001, 002, etc, 999) that counts the files in a logical tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Service Sub Level Name* is a subdivision of the Service ID that is used to further classify the source of data.
5. *Version Number* is the version number for the software that wrote the original data.

6. *Date of Generation* is the date of generation for the software that wrote the original data. The format is:

   Year/Month/Day

   For example, 84/12/25.
7. *Maximum Physical Record Length* is the representation in alphanumeric digits of the maximum physical record length.
8. *File Type* is a 2-character indicator of the kind of information in the file. For example, LL for Label, LO for Log Data, CA for Calibration.
9. *Optional Previous File Name* is intended for disk-based implementations of LIS in which there may be no obvious predecessor or successor file. When used, it has the same format as the File Name (comment 2). When unused, it consists of 10 alphanumeric space characters. File Headers, File Trailers, Tape Trailers, and Reel Trailers are identical except for the record type and the definition of these 10 bytes.

2.2.2.2. File Trailer Logical Record (FTLR)

The File Trailer Record is 58 bytes in length. If present, it must be the last logical record in a file. It may be followed by a Tape Trailer but are always followed by End-of-File marks (EOF's).

The File Trailer Record is used primarily to check the data without having to backspace to the beginning of the last file.

File Trailer Logical Record (Type 129)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
File Name	10	65	2
(Service Name)	(6)	(65)	(2a)
(".")	(1)	(65)	(2b)
(File Number)	(3)	(65)	(2c)
2 blanks	2	65	3
Service Sub Level Name	6	65	4
Version Number	8	65	5
Date of Generation	8	65	6
(Year)	(2)	(65)	(6)
("/")	(1)	(65)	(6)
(Month)	(2)	(65)	(6)
("/")	(1)	(65)	(6)
(Day)	(2)	(65)	(6)
1 blank	1	65	3
Maximum Physical Record Length	5	65	7
2 blanks	2	65	3
File Type	2	65	8
2 blanks	2	65	3
Optional Next File Name	10	65	9

Figure 2.4: File Trailer Logical Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *File Name* is a unique name for a file within a logical tape and consists of following parts:
   (a) *Service Name* is the name of the service or program that created the tape.
   (b) *"."* is simply a separator.
   (c) *File Number* is a 3-character counter (001, 002, etc, 999) that counts the files in a logical tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Service Sub Level Name* is a subdivision of the Service ID that is used to further classify the source of data.

5. *Version Number* is the version number for the software that wrote the original data.
6. *Date of Generation* is the date of generation for the software that wrote the original data. The format is:

   Year/Month/Day

   For example, 84/12/25.
7. *Maximum Physical Record Length* is the representation in alphanumeric digits of the maximum physical record length.
8. *File Type* is a 2-character indicator of the kind of information in the file. For example, LL for Label, LO for Log Data, CA for Calibration.
9. *Optional Next File Name* is intended for disk-based implementations of LIS in which there may be no obvious predecessor or successor file. When used, it has the same format as the File Name (comment 2). When unused, it consists of 10 alphanumeric space characters. File Headers, File Trailers, Tape Trailers, and Reel Trailers are identical except for the record type and the definition of these 10 bytes.

2.2.3. Logical Tape (LT)

A Logical Tape is composed of Logical Files. A Logical Tape is delimited by a *Tape Header Logical File (THLF)* at its beginning and by a *Tape Trailer Logical File (TTLF)* at its end.

These files are special LIS logical files in that they don't contain File Header or File Trailer Logical Records. More specifically:

- Tape Header Logical File consists of a *Tape Header Logical Record (THLR)*, preceded, when this file is the first on a Reel, by a *Reel Header Logical Record (RHLR)*.
- Tape Trailer Logical File consists of a *Tape Trailer Logical Record (TTLR)*, optionally followed, when this file is the last on a Reel, by a *Reel Trailer Logical Record (RTLR)*.

2.2.3.1. Tape Header Logical Record (THLR)

The Tape Header Logical Record is 128 bytes in length. It is used

- to identify the beginning of a set of Logical Files that constitute an LIS Logical Tape
- to provide the consumer with some information about a specific Logical Tape.

Tape Header Logical Record (Type 130)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Service Name	6	65	2
6 blanks	6	65	3
Date	8	65	4
(Year)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Month)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Day)	(2)	(65)	(4)
2 blanks	2	65	3
Origin of Data	4	65	5
2 blanks	2	65	3
Tape Name	8	65	6
2 blanks	2	65	3
Tape Continuation Number	2	65	7
2 blanks	2	65	3
Previous Tape Name	8	65	8
2 blanks	2	65	3
Comments	74	65	9

Figure 2.5: Tape Header Logical Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Service Name* is the name of the service or program that created the tape. The first six characters of this name are used in all File Header and File Trailer Records. In this fashion, all of the file names within a Logical Tape will be unique. The construction is to use the six-character service name and a ".", followed by a three-character counter (001, 002, etc, 999), which counts the files in a Logical Tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Date* is the date when the data was originally acquired. The format is:

   Year/Month/Day

   For example, 84/12/25.
5. *Origin of Data* is the system that originally acquired or created the data.
6. *Tape Name* is an ID that can be used to identify the Logical Tape, where applicable.

7. *Tape Continuation Number* is a number sequentially ordering multiple Logical Tapes stored on the same reel.
8. *Previous Tape Name* is an ID that can be used to identify the previous Logical Tape, where applicable. If this is the first Logical Tape, then this entry should be all blanks.
9. *Comments* are any relevant remarks concerning the Logical Tape or information contained within the Logical Tape.

2.2.3.2. Reel Header Logical Record (RHLR)

The Reel Header Logical Record is 128 bytes in length and is the first record on any physical reel. It is intended to identify the reel of tape.

Reel Header Logical Record (Type 132)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Service Name	6	65	2
6 blanks	6	65	3
Date	8	65	4
(Year)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Month)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Day)	(2)	(65)	(4)
2 blanks	2	65	3
Origin of Data	4	65	5
2 blanks	2	65	3
Reel Name	8	65	6
2 blanks	2	65	3
Reel Continuation Number	2	65	7
2 blanks	2	65	3
Previous Reel Name	8	65	8
2 blanks	2	65	3
Comments	74	65	9

Figure 2.6: Reel Header Logical Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Service Name* is the name of the service or program that created the tape. The first six characters of this name are used in all File Header and File Trailer Records. In this fashion, all of the file names within a Logical Tape will be unique. The construction is to use the six-character service name and a ".", followed by a three-character counter (001, 002, etc, 999), which counts the files in a Logical Tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Date* is the date when the physical reel was created. The format is:

   Year/Month/Day

   For example, 84/12/25.
5. *Origin of Data* is the system that originally acquired or created the data.
6. *Reel Name* is an eight-character name used to physically identify a specific reel of tape. This name matches the visual identification written on the tape canister.

7. *Reel Continuation Number* is a number sequentially ordering multiple Physical Reels and is an alphanumeric from 1 to 99.
8. *Previous Reel Name* is an ID that can be used to identify the previous Physical Reel, where applicable.
9. *Comments* are any relevant remarks describing the Physical Reel of tape.

2.2.3.3. Tape Trailer Logical Record (TTLR)

The Tape Trailer Logical Record is 128 bytes in length. If several Logical Tapes are stored on one Physical Reel, then this type indicates the end of a Logical Tape. This record is optional; in its absence a Logical Tape is assumed to be terminated when a new Tape Header Logical Record is encountered.

Tape Trailer Logical Record (Type 131)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Service Name	6	65	2
6 blanks	6	65	3
Date	8	65	4
(Year)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Month)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Day)	(2)	(65)	(4)
2 blanks	2	65	3
Origin of Data	4	65	5
2 blanks	2	65	3
Tape Name	8	65	6
2 blanks	2	65	3
Tape Continuation Number	2	65	7
2 blanks	2	65	3
Next Tape Name	8	65	8
2 blanks	2	65	3
Comments	74	65	9

Figure 2.7: Tape Trailer Logical Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Service Name* is the name of the service or program that created the tape. The first six characters of this name are used in all File Header and File Trailer Records. In this fashion, all of the file names within a Logical Tape will be unique. The construction is to use the six-character service name and a ".", followed by a three-character counter (001, 002, etc, 999), which counts the files in a Logical Tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Date* is the date when the data was originally acquired. The format is:

   Year/Month/Day

   For example, 84/12/25.
5. *Origin of Data* is the system that originally acquired or created the data.
6. *Tape Name* is an ID that can be used to identify the Logical Tape, where applicable.

7. *Tape Continuation Number* is a number sequentially ordering multiple Logical Tapes stored on the same reel.
8. *Next Tape Name* is an ID that can be used to identify the next Logical Tape, where applicable.
9. *Comments* are any relevant remarks concerning the Logical Tape or information contained within the Logical Tape.

2.2.3.4. Reel Trailer Logical Record (RTLR)

The Reel Trailer Logical Record is 128 bytes in length and may optionally be used as the last record on a Physical Reel of tape.

Reel Trailer Logical Record (Type 133)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Service Name	6	65	2
6 blanks	6	65	3
Date	8	65	4
(Year)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Month)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Day)	(2)	(65)	(4)
2 blanks	2	65	3
Origin of Data	4	65	5
2 blanks	2	65	3
Reel Name	8	65	6
2 blanks	2	65	3
Reel Continuation Number	2	65	7
2 blanks	2	65	3
Next Reel Name	8	65	8
2 blanks	2	65	3
Comments	74	65	9

Figure 2.8: Reel Trailer Logical Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Service Name* is the name of the service or program that created the tape. The first six characters of this name are used in all File Header and File Trailer Records. In this fashion, all of the file names within a Logical Tape will be unique. The construction is to use the six-character service name and a ".", followed by a three-character counter (001, 002, etc, 999), which counts the files in a Logical Tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Date* is the date when the physical reel was created. The format is:

   Year/Month/Day

   For example, 84/12/25.
5. *Origin of Data* is the system that originally acquired or created the data.
6. *Reel Name* is an eight-character name used to physically identify a specific reel of tape. This name matches the visual identification written on the tape canister.

7. *Reel Continuation Number* is a number sequentially ordering multiple Physical Reels and is an alphanumeric from 1 to 99.
8. *Next Reel Name* is an ID that can be used to identify the next Physical Reel, where applicable.
9. *Comments* are any relevant remarks describing the Physical Reel of tape.

2.3. Physical Structure

The LIS physical structure describes the way that Logical Tapes, Logical Files, and Logical Records are actually represented on a Physical Reel of tape.

The LIS physical structure consists of two (2) elements:

- the *Physical Reel*, a group of physical records arranged in a specific order
- the *Physical Record*, a group of eight-bit bytes or data characters arranged in a designated order.

There are no Physical Files in the LIS format. The relationship of Physical Reels and Physical Records is illustrated in the following diagram:

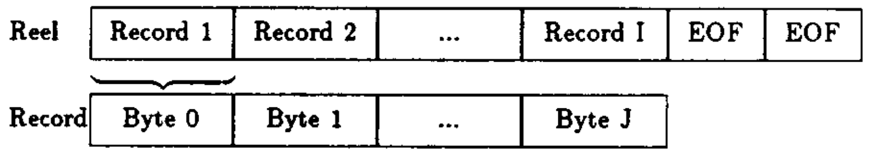


Figure 2.9: Physical Hierarchy

A special type of Physical Record, called an End-of-File mark or EOF, is used to identify the end of a Logical File on a tape. Its purpose is to allow special EOF recognition hardware to be used for high-speed tape searches.

Two consecutive EOF's indicate the end of a Physical Reel of tape.

2.3.1. Physical Records (PR)

A Physical Record consists of

- a *Physical Record Header (PRH)*
- a *Physical Record Body (PRB)*
- an optional *Physical Record Trailer (PRT)*

2.3.1.1. Physical Record Header (PRH)

The Physical Record Header has the following format:

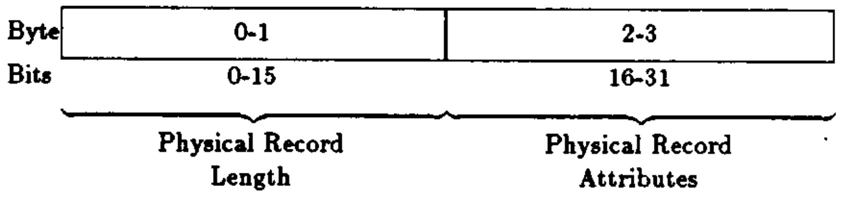


Figure 2.10: Physical Record Header

Where:

- *Physical Record Length (PRL)* is a 16-bit, unsigned, binary integer that specifies the length, in bytes, of information in the Physical Record. This length, which includes the Physical Record Header and Trailer (if present), may be smaller than the actual number of bytes written–that is, a Physical Record may be padded with null characters to guarantee a minimum record size.
- *Physical Record Attributes* is a 16-bit bit-string with the following structure:

Bit(s)	*Description*	*Comments*
16	Unused, reserved	
17	Physical Record Type	1
18-19	Checksum Type	2
	00 = no checksum present	
	01 = 16-bit checksum	
	10 = undefined	
	11 = undefined	
20	Unused, reserved	
21	File Number Presence	3
	0 = no file number entry present	
	1 = file number entry present	
22	Record Number Presence	4
	0 = no record number entry present	
	1 = record number entry present	
23	Unused	
24	Unused, reserved	
25	Parity Error	5
	0 = no parity error	
	1= a parity error occurred previously	
26	Checksum Error	6
	0 = no checksum error	
	1 = checksum error occurred previously	
27	Unused	
28	Unused, reserved	
29	Unused	
30	Predecessor Continuation Attribute	7
	0 = PR not associated with previous PR	
	1 = PR associated with previous PR	
31	Successor Continuation Attribute	7
	0 = PR not associated with next PR	
	1 = PR associated with next PR	

Figure 2.11: Physical Record Attributes

Comments:

1. The *Physical Record Type Bit* defines the type code for a Physical Record. At present, only type 0 is defined by LIS.
2. The *Checksum Type Bit* designates the type of checksum associated with a Physical Record. If a checksum is present, it is the last entity of the Physical Record Trailer.
3. The *File Number Presence Bit* defines whether or not a file number entry is present in the Physical Record Trailer. If a file number is present, it will precede the checksum in the Physical Record Trailer.
4. The *Record Number Presence Bit* defines whether or not a record number entry is present in the Physical Record Trailer.
5. The *Parity Error Bit* indicates that a parity error has occurred while copying a record sometime in the past. Since data may be transferred from one medium to another several times, it is necessary to preserve the status of a record during each transfer.
6. The *Checksum Error Bit* indicates that a checksum error has occurred while a record was being copied sometime in the past.
7. The *Predecessor and Successor Continuation Bits* provide a mechanism by which a logical record may span several physical records.

   For example, if a logical record were to span four physical records, the values of the Predecessor and Successor Continuation Bits for the four records would be:

Record(s)	*Predecessor*	*Successor*
1	0	1
2	1	1
3	1	1
4	1	0

If the Predecessor Continuation Bit in the Physical Record Header is 0, the Physical Record contains a Logical Record Header. If its value is 1, the Physical Record does not contain a Logical Record Header.

2.3.1.2. Physical Record Body (PRB)

The Physical Record Body consists of either a complete Logical Record or a fragment of a Logical Record spanning multiple Physical Records.

2.3.1.3. Physical Record Trailer (PRT)

The presence, length, and contents of the Physical Record Trailer are determined by three Physical Record Attribute bits defined in the Physical Record Header: the Record Number Presence Bit, the File Number Presence Bit, and the Checksum Type Bit (bits 22, 21, 19 respectively).

The sequence of information in the trailer is shown below. Any or all of these elements may be missing.

Physical Record Trailer			
Entry	*Size*	*Repr Code*	*Related Header Bit*
Record Number	2	79	22
File Number	2	79	21
Checksum	2	79	19

Figure 2.12: Physical Record Trailer

2.3.2. Physical Reel (PT)

A Physical Reel is composed of Physical Records, is initiated by a Reel Header Logical Record, and terminated by a Reel Trailer Logical Record and two hardware EOF records.

2.4. Mapping Logical Into Physical Hierarchy

The hierarchies of logical and physical structures are simple enough. LIS defines a set of logical elements (Logical Tape, File, and Record), while a typical recording device (Magnetic Tape or Disk) recognizes another set of physical elements (Physical Reel and Record).

Since an LIS Logical Tape consists of Logical Files, which in turn consist of Logical Records, mapping LIS logical elements into a physical structure can be viewed

- on a large scale, as mapping Logical Tapes into Physical Reels
- on a small scale, as mapping Logical Records into Physical Records.

2.4.1. Mapping Logical Tapes into Physical Reels

A Logical Tape may span multiple Physical Reels, or a Physical Reel may contain multiple Logical Tapes.

If multiple Logical Tapes are required, then they are stacked sequentially on the Physical Reel. The Tape Header Logical Record and Tape Trailer Logical Record contain identifiers to the previous and next Logical Tape (Tape Continuation Number and Previous/Next Tape Name).

If multiple Physical Reels are required, then the Reel Header Logical Record and Reel Trailer Logical Record contain identifiers to the previous and next Physical Reel (Reel Continuation Number and Previous/Next Reel Name).

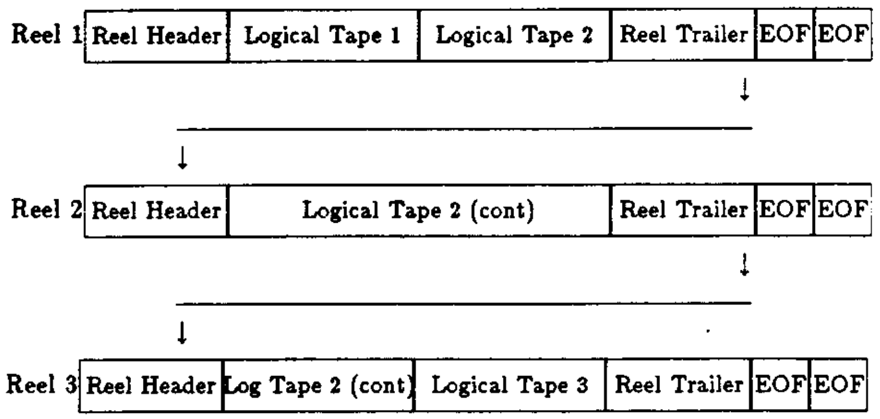


Figure 2.13: Relationship between Logical Tapes and Physical Reels

2.4.2. Mapping Logical Records into Physical Records

A Logical Record may be contained within a single Physical Record, or it may span several Physical Records.

Each Logical Record starts at the beginning of a new Physical Record.

If a Logical Record spans several Physical Records, then the Physical Record Header contains identifiers to the previous and/or next Logical Record (Successor/Predecessor Continuation Bits).

If a Logical Record does not span several Physical Records, then the Successor/Predecessor Continuation Bits of the Physical Record Header are clear.

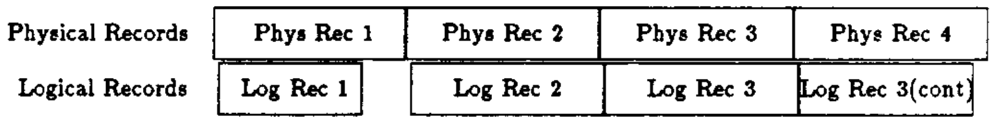


Figure 2.14: Relationship between Logical and Physical Records

Chapter 3.

LOGICAL RECORD SYNTAX

3.1. Record Format Categories

From a syntactical point of view, we could speak of several different kinds of LIS Logical Record structures for storing information–namely:

- Fixed Format Logical Record
- Explicitly Formatted Logical Record
- Indirectly Formatted Logical Record

3.2. Fixed Format Logical Record

A Fixed Format Logical Record has a fixed-length Logical Record Body consisting of a fixed number of data fields. Each data field contains a predefined item of information whose size and representation are also predetermined.

The delimiter Logical Records–i.e., Tape/File Header/Trailer Logical Records–are Fixed Format Logical Records.

3.3. Explicitly Formatted Logical Record

An Explicitly Formatted Logical Record has a variable-length Logical Record Body whose format is derived from an analysis of the record itself.

3.3.1. Data Information Record

Information Records can contain

- information identifying company name, well name, and the like
- information about parameters used in computation

- information about data environments, such as how a curve is presented on a graphic display.

Information Records are constructed depending upon whether they are used

- to represent table-like data structures (that is, information made up of rows and columns)
- to represent only single value parameters.

3.3.1.1. Tables in Information Records: General Layout

A logical unit of information–for example, how a single curve is presented on a graphic display–may consist of several pieces of information–for example, track position, trace coding, scale information, and so on.

You can visualize this collection of information for a single curve as one row of a table. A table which consists of several rows would, then, describe information for a collection of curves.

Each logical unit of information is contained in that part of the Information Record called a *Data Block* (one row of information).

Each individual piece of information within a logical unit is contained in that part of the Data Block called a *Component Block* (one element or column entry of a row of information).

Each Component Block contains a set of *Components*–that is, information needed to define and represent each individual piece of information (defining one element or column entry of a row of information).

The general layout for this relationship is illustrated below.

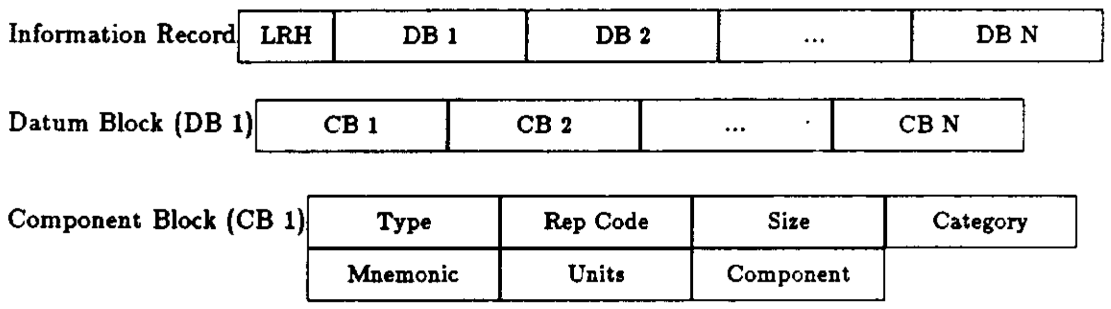


Figure 3.1: General Layout of an Information Record

3.3.1.2. Tables in Information Records: Specific Layout

How is one Datum Block distinguished from another? When an Information Record contains a table, the First Datum Block of the record is a single Type 73 Component Block. The quantity contained in this Datum Block is the name of the specific table.

The first Component Block in all subsequent Datum Blocks is a Type 0 Component Block. All Component Blocks following a Component Block of Type 0 belong to the same Datum Block until the next Type 0 Component Block is encountered. All intermediate Components between two Type 0 Component Blocks are given Type 69. In this way, one Datum Block can be separated quite easily from another.

The specific layout for this relationship is illustrated below. Note that it is not necessary that each Datum Block have the same number of Component Blocks.

Data Block	Component Block Type 73
Data Block	Component Block Type 0
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69
Data Block	Component Block Type 0
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69
Data Block	Component Block Type 0
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69
	Component Block Type 69

Figure 3.2: Specific Layout of an Information Record

3.3.1.3. Tables in Information Records: Specific Example

Let's take a specific example and see how it translates into an Information Record. Suppose a Film table is used to define the depth scale for the graphic display of a set of log data. It may also define the grid pattern on which the log data is presented. Shown in tabular form, the Film table might look like this:

MNEM	GCOD	GDEC	DEST	DSCA
1	E2E	2	PF1	S5
2	BBB	-	PF2	S5

Figure 3.3: Specific Example of an Information Record

The important point of this illustration is how we map the individual table entries into Component Blocks, not what the individual entries mean. The diagram below illustrates the construction of an Information Record that contains the table shown above.

Logical Record Header		
Component	Type	73
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	TYPE
Component	Units	
Component		FILM
Component	Type	0
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	MNEM
Component	Units	
Component		1
Component	Type	69
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	GCOD
Component	Units	
Component		E2E
Component	Type	69
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	GDEC
Component	Units	
Component		2
Component	Type	69
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	DEST
Component	Units	
Component		PF1
Component	Type	69
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	DSCA
Component	Units	
Component		S5

Figure 3.4: Logical Representation of Specific Information Record

Logical Record Header		
Component	Type	0
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	MNEM
Component	Units	
Component		2
Component	Type	69
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	GCOD
Component	Units	
Component		BBB
Component	Type	69
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	GDEC
Component	Units	
Component		—
Component	Type	69
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	DEST
Component	Units	
Component		PF2
Component	Type	69
Component	Rep Code	65
Component	Size	4
Component	Category	0
Component	Mnemonic	DSCA
Component	Units	
Component		S5

Figure 3.5: Logical Representation of Specific Information Record (cont)

You will note that

- the Category and Units fields of the Component Block are not used in the table
- the mnemonics used in the Mnemonic entry of the Component Block are dictionary items
- the quantity entered as the component itself is not a dictionary item but defined within

the context of the system that understands the table.

3.3.1.4. Single Parameters in Information Records

In some cases you may want to encode one or several single parameter values in an Information Record. You can visualize this as a table with only one column. In this case, you can use a simplified form for the Information Record, omitting the Type 73 Datum Block. Instead, you will have one or more Type 0 Component Blocks. The following table illustrates the Single Parameter Information Record.

Single Parameter Information Record
Logical Record Header
Component Block Type 0
Component Block Type 0
Component Block Type 0
Component Block Type 0

Figure 3.6: Specific Layout of a Single Parameter Information Record

This type of Information Record might be used to store identifying information such as Company Name (CN), Well Name (WN), and the like. For example, the well name is given in this single Component Block.

Component Block		
Component	Type	0
Component	Rep Code	65
Component	Size	8
Component	Category	(undefined)
Component	Mnemonic	WN
Component	Units	(none)
Component		Smith N1

Figure 3.7: Specific Example of a Single Parameter Information Record

3.3.2. Data Format Specification Record (DFSR)

In order to understand what a Data Format Specification Record is and what it does, we need the following definitions:

- An *Entry* is a single value–i.e., a single number for a single sensor response at a single depth.
- A *Block* is simply a grouping of Entries–i.e., a number of sensor responses at a single depth.
- A *Frame* is a Block composed of a number of Entries.
- A *Data Record* is an integral number of Frames.

The Data Format Specification Record is the key to the LIS format as it defines the format of data in a data record that follows. This logical record has two sections:

- The first section, called an *Entry Block*, is a sequence of miscellaneous entries, each of which is tagged with Type, Size, and Representation Code. These are used to record various properties of a data record–i.e., a group of Frames–as a whole.
- The second and following section is a sequence of *Datum Specification Blocks*, each providing information required to identify and decode one frame entry. Thus, all of the Datum Specification Blocks, taken together, define a frame as it exists in the data records that follow. The order of these blocks is the order of the data in the frame.

The number of entries in a frame is represented by the following formula:

$$Number\ of\ Entries = \frac{Total\ Size}{Size\ per\ Entry}$$

For example, a waveform may consist of 512 sixteen-bit numbers measured versus time at a single depth. The Datum Specification Block contains the following information:

Data Specification Block	
Number of Samples	1
Size	1024 bytes
Representation Code	79 (a two-byte integer)

The total number of entries in the Frame is calculated as:

$$Number\ of\ Entries = \frac{Total\ Size}{Size\ per\ Entry} = \frac{1024}{2} = 512\ entries$$

A frame has one of several structures that are determined by examining the Size, Number of Samples, and Representation Code entries. Two structures are discussed separately below.

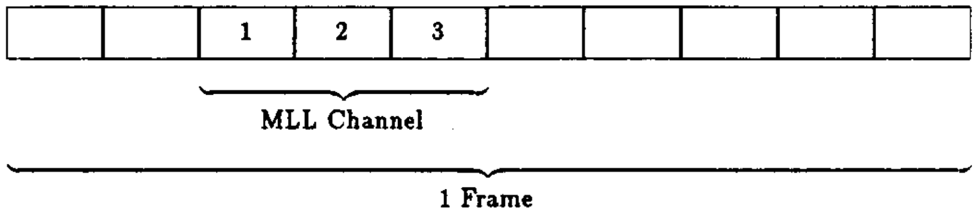
3.3.2.1. Standard Channel in a DFSR

A Standard Channel is one that is sampled once per frame. The Number of Samples is one, and the Size matches the size associated with the Representation Code.

3.3.2.2. Fast Channel in a DFSR

A Fast Channel is one that is sampled more than once per frame. A Fast Channel has a Number of Samples that is greater than one. The Size entry of the Datum Specification is the product of the Number of Samples multiplied by the number of bytes for each sample. The spacing between Fast Channel samples, which is equal, is the interval between frames divided by the Number of Samples. The depth (or other index) of the frame applies to the last of the multiple samples, with earlier samples' depths interpolated between the current and previous frame depth.

For example, a Micro LateroLog curve (MLL) is too active to be adequately sampled only once every six inches. It is common practice, for example, to sample it every two inches and store three successive samples in a single frame, as shown below. In this case, the log was run going up-hole, and frame spacing was six inches.



Sample	Frame Depth Calculation
1	frame depth + 4 in
2	frame depth + 2 in
3	frame depth

Figure 3.8: Fast Channel

3.4. Indirectly Formatted Logical Record

The syntax of the Logical Record Body of an Indirectly Formatted Logical Record is not specified by predefined LIS fixed format and can not be derived from an analysis of the record itself. Instead, it is defined by the contents of other Logical Records.

This is particularly useful for the representation and recording of acquired or computed log data whose format is not fixed but depends on the type and number of logging tools used in a particular log.

3.4.1. Data Record

Log data is recorded as logical records identified as Data Records. As already stated, a Data Record is composed of an integral number of frames whose format is defined in the Data Format Specification Record.

Programs that work with the LIS format cannot define data in terms of relative positions in an array. Instead, data must be identified by name. This method requires an extra level of software that can read the Data Format Specification Record and cross reference between names and numbers. However, the advantage of this approach is that application software does not have to interpret the various frame formats. In addition, data channels can be added or deleted with no effect on programs that do not use the new or missing channels.

3.5. Logical Configuration

An LIS tape may contain the following record types. If any other types occur, they may be safely ignored.

Logical Record Types	
Type Nb	*Type Name*
GROUP 0	DATA RECORDS
0	Normal Data
1	Alternate Data
GROUP 1	INFORMATION RECORDS
32	Job Identification
34	Wellsite Data
39	Tool String Info
42	Encrypted Table Dump
47	Table Dump
GROUP 2	DATA FORMAT SPECIFICATION RECORDS
64	Data Format Specification
65	Data Descriptor
GROUP 3	PROGRAM RECORDS (CSU ONLY)
95	TU10 Software Boot
96	Bootstrap Loader
97	CP-Kernel Loader Boot
100	Program File Header
101	Program Overlay Header
102	Program Overlay Load
GROUP 4	DELIMITERS
128	File Header
129	File Trailer
130	Tape Header
131	Tape Trailer
132	Reel Header
133	Reel Trailer
137	Logical EOF
138	Logical BOT
139	Logical EOT
141	Logical EOM
GROUP 7	MISCELLANEOUS RECORDS
224	Operator Command Inputs
225	Operator Response Inputs
227	System Outputs to Operator
232	FLIC Comment
234	Blank Record/CSU Comment
85	Picture
86	Image

Figure 3.9: Logical Record Types

Group 0 (Data Records) contains logging data stored in the logical tape and must be preceded by a Data Format Specification Record that will define the data structure within them.

Group 1 (Information Records) contains identification, computational, and environmental information about the logging process, and may appear anywhere within the logical file structure, but typically appears before the Data Format Specification Record.

Group 2 (Data Format Specification Records) defines the format of the data in the Data Records to follow and must, therefore, precede the first Data Record of a Logical Tape. Some LIS tapes will contain two identical copies of each Data Format Specification Record for redundancy.

Group 3 (Program Records) provides a mechanism for storing computer software used in acquiring and processing the log data. LIS has defined record types that contain the machine language used to boot and run the acquisition computer system. Programs or segments of programs are loaded as needed from an LIS tape.

Group 4 (Delimiter Records) provides cues for the beginning and end of logical structures. These are:

- The File Header Record is the first record of any file on the tape. It contains general information identifying the file. It also contains identification of the previous file of the logical tape.
- The File Trailer Record is the final record of any file. It may contain identification of the next file in the logical record.
- The Tape Header Record is the first record of a logical tape. It contains general information identifying the information content of this logical segment of the physical reel of tape. If many logical tapes are grouped into some larger structure, this record also contains identification of the previous logical tape in the structure.
- The Tape Trailer Record is the final record of the logical tape. It may contain identification of the next logical tape in the structure. If no additional logical tapes follow on the same reel, this record is optional.
- The Reel.Header Record is the first record of any physical reel and should appear nowhere else on the reel. It contains general information identifying the reel. If the reel is part of a multi-reel grouping, then the reel header also contains the identification of the previous reel of the group.
- The Reel Trailer Record is the final record of the physical reel. It contains identification of the next physical reel in the set. This is an optional record.
- The Logical End-of-File (LEOF) serves the same purpose as a Physical EOF. It may be used on a medium that does not have a Physical EOF. It will also be used to replace physical file marks at higher software levels.
- The Logical Beginning of Tape (LBOT) serves the same purpose as a Physical BOT. It may be used to indicate to a reading program that a physical BOT has been encountered.

- The Logical End-of-Tape (LEOT) serves the same purpose as a Physical EOT. It may be used to indicate to reading programs that a physical EOT has been encountered.
- The Logical End-of-Medium (LEOM) serves the same purpose as a Physical EOM. It defines the end of a medium. Upon encountering a physical EOM on a medium, the reading program may be given a Logical EOM to indicate this condition.

Group 7 (Miscellaneous Records) consists of operator inputs to the system, system outputs to the operator, general comments, and bulk records like picture descriptions and images and may be stored in logical records anywhere within the logical file structure.

A typical grouping of the various Program Records follows:

Reel 1	Reel Head	TU10 Boot	EOF	File 1	EOF	...	EOF	File N	EOF	EOF

File 1	CP-Kernel	Loader Boot	Loader Boot	Bootstrap	Overlay 1	...	Overlay M

Overlay 1	Program Overlay Header	Program Overlay Load

Figure 3.10: Program Records in a Logical Structure

A typical grouping of the record types follows:

Reel Header	Tape Header	EOF	File Header	Info	Data Format	Data	Data	...

...	Data	Data	Data	File Trailer	EOF

File Header	Many Files	File Trailer	EOF	Tape Trailer	Reel Trailer	EOF	EOF

Figure 3.11: Record Types in a Logical Structure

Chapter 4.

LOGICAL RECORD SEMANTICS

4.1. Detailed Logical Record Type Descriptions

This section contains specifications for all defined record types, arranged in numeric order for easy reference. All numbers are decimal (base ten) unless otherwise noted. The descriptions may have information listed under the following headings:

- *Entry* - A short description of the entry defined by this line. If the entry is a constant alphanumeric text, this text is shown in quotes (i.e., "." means a single period character.)
- *Size* - A decimal number specifying the length of the entry in bytes. "V" is used to indicate variable size (i.e., defined on the tape but not in this manual).
- *Representation Code* - A numeric code used to indicate the representation of a corresponding entry. Representation Codes are listed in Appendix B.
- *Comments* - Most entries have comments, numerically ordered and appearing after the figure of the logical record to which they refer.

4.1.1. Type 0: Normal Data Record

Normal Data Record (Type 0)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Depth	V		2
First Frame	V		3
Last Frame	V		3

Figure 4.1: Normal Data Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Depth* may appear here (only once in each data record), or it may be contained in each data frame. Entry Block 13 in the Type 64 Data Format Specification Record specifies which mode of recording depth is used.
   (a) If Depth is recorded just once per record, then Entry Block Type 13 will be present and will have a value of 1. In this case the size, representation code, and units for the Depth entry are also given in the Data Format Specification Record in Entry Block Types 14 and 15. The Depth given is one for the first frame in the record. Other information required for computing the depth of other frames is given in Entry Block Types 4, 8, and 9.
   (b) If Entry Block Type 13 has a value of 0 (or if it is absent), then Depth (or other index information) is contained as a normal frame entry in each frame and does not appear in this special position at the beginning of each data record.
3. *Frame* organization, size, and representation codes are not pre-defined but are described in detail within the Data Format Specification Record (Type 64) by means of Datum Specification Blocks.

4.1.2. Type 1: Alternate Data Record

This record type has never been implemented. It was intended as a way of specifying another frame type so that multiple frame types could be mixed in a logical record. The mechanism making this possible exists, in fact, since Entry 1 of the Data Format Specification Record specifies the Data Record Type. Because only Data Record Type 0 exists, Entry 1 of the Data Format Specification Record is always 0.

4.1.3. Types 32, 34, 39: Information Record

Information Record (Type 32,34,39)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Datum Block 1	2		2
Datum Block 2	M1		
.			
.			
.			
Datum Block m	Mm		

Figure 4.2: Information Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Datum Block* contains a logical unit of Component Blocks–that is, identifying, computational, or environmental information about the logging process.

Datum Block			
Entry	*Size*	*Repr Code*	*Comments*
Component Block 1	L1		2a
Component Block 2	L2		
.			
.			
.			
Component Block n	Ln		

(a) *Component Block* contains the information needed to define and represent each individual piece of information.

Component Block			
Entry	*Size*	*Repr Code*	*Comments*
Component Type Nb	1	66	2b
Component Repr Code (r)	1	66	2c
Component Size (Ji)	1	66	2d
Component Category	1	66	2e
Component Mnemonic	4	65	2f
Component Units	4	65	2g
Component	Ji	r	2h

(b) *Component Type Nb* is a number indicating the type of datum represented.

(c) *Component Repr Code* is a number indicating the manner in which the datum is represented. The various representation codes are listed in Appendix B.

(d) *Component Size* is a number indicating the size in bytes reserved for the datum.

(e) *Component Category* is an undefined field.

(f) *Component Mnemonic* is the name of the datum.

(g) *Component Units* indicate the units of measurement for the datum.

(h) *Component* is the actual datum, represented in the Representation Code r that is stored in the Component Representation Code entry.

The length of a Component Block may be calculated with the following equation:

$$L_i = J_i + \mathit{12\ bytes}$$

The length of a Datum Block may be calculated with the following equation:

$$M_i = \sum_{i=1}^{n} (J_i + \mathit{12})\ \mathit{bytes}$$

The length of an Information Record may be calculated with the following equation:

$$\mathit{Length} = 2 + \sum_{i=1}^{m} M_m\ \mathit{bytes}$$

4.1.4. Type 42: Encrypted Table Dump Record

Some tables contained in standard Information Records are classified as proprietary. In order to secure these tables on a real-time client tape, the tables are encrypted and stored on the tape as a new logical record type. These records are re-typed as Type 42. Only the body of the logical record following the logical record header is encrypted. The memo reference above contains examples of records that are encrypted and those that are not.

4.1.5. Type 47: Table Dump Record

Table Dump Record (Type 47)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Table Type	4	65	2
Format Code	1	66	3
Table Size	2	79	4
S-Size of Mask	1	66	5
Mask	S	77	6
1st Table			6
.			
.			
.			
Last Table			

Figure 4.3: Table Dump Record

Comments:

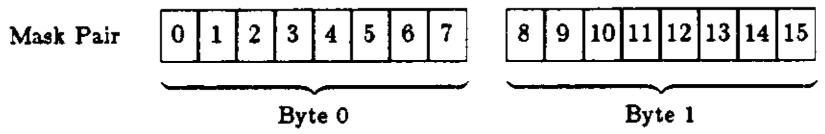
1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Table Type* is an alphanumeric definition of the type of table in the record. These include:

Table Types	*Table Definition*
TSCA	TTR scale table
TOOL	tool tables
INPU	input tables
EQUI	equipment tables
OUTP	output tables
PRES	presentation tables
AREA	area tables
FILM	film tables
CONS	constant tables
SONI	sonic constant tables
SHOT	shot tables

3. *Format Code* defines the format for tables being dumped. Each time the table format is changed, this entry will change.
4. *Table Size* is the number of bytes in one table.
5. *S-Size of Mask* is the number of bytes of mask.
6. Each bit of the *Mask* refers to one pair of bytes in the table.

Bit in Mask	*Meaning*
0	character pair is alphanumeric
1	character pair is binary

This is useful since all data is represented with the most significant byte first. In some 16-bit computers–i.e., DEC PDP-11–the internal representation of binary numbers is low-order byte first. In such computers it is convenient to have a simple rule that determines which 16-bit words require byte swapping.



7. *Table* entries and information depend on the Table type.

4.1.6. Type 64: Data Format Specification Record

Data Format Specification Record (Type 64)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
1st Entry Block	V		2
.			
.			
.			
Last Entry Block	V		
1st Datum Spec Block	40		3
.			
.			
.			
Last Datum Spec Block	40		

Figure 4.4: Data Format Specification Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Entry Block* is a sequence of miscellaneous entries, each of which tagged with Type, Size, and Representation Code. These are used to record various properties of a data record–i.e., a group of Frames–as a whole.

Entry Block			
Entry	*Size*	*Repr Code*	*Comments*
Entry Type	1	66	2(a)
n - Size	1	66	
r - Repr Code Nb	1	66	
Entry	n	r	

(a) *Entry Type* has been defined as follows:

Entry Types		
Entry	*Short Definition*	*Comments*
0	Terminator (no more entries)	2(a)i
1	Data Record Type	2(a)ii
2	Datum Spec Block Type	2(a)iii
3	Data Frame Size	2(a)iv
4	UP/DOWN Flag	2(a)v
5	Optical Log Depth Scale Units	2(a)vi
6	Data Reference Point	2(a)vii
7	Units of Above	2(a)viii
8	Frame Spacing	2(a)ix
9	Units of Above	2(a)x
10	Currently Undefined	2(a)xi
11	Maximum Frames/Record	2(a)xii
12	Absent Value	2(a)xiii
13	Depth Recording Mode	2(a)xiv
14	Units of Depth in Data Records	2(a)xv
15	Repr. Code for Output Depth	2(a)xvi
16	Datum Spec Block Sub-type	2(a)xvii

Some example Entry Block entries are shown in the following table.

Entry Block Example		
Entry Nb	*Value*	*Meaning*
1	0	Type 0 Data Records being used
2	0	Type 0 Datum Spec Blocks being used
3	44	Frame length defined by Datum Spec Blocks
4	1	File logged going up-hole (decrease depth)
5	1	Original field log was scaled in feet
8	60	Frame space is 60 tenths of an inch
9	.1IN	in units of .1 inches
12	-999.25	This value is written in a frame when no valid data is present
13	1	Depth will be written once/data record
14	.1IN	in units of .1 inches
15	70	and encoded as a 32 bit integer
16	1	Datum Spec Block Sub-Type 1 used
0	1	Pad out cumulative entry block lengths to even nb of bytes

i. *Terminator* is required as the last entry in the string of Entry Blocks (all other blocks are optional). A zero entry terminates the string. The terminator entry size must be chosen as 0 or 1 so that the number of bytes in the record through the terminator block is even.

ii. *Data Record Type* indicates the logical record type number which will be used for data records with this format specification. Thus, several different types of Data Records can be simultaneously defined. The default Data Record Type is 0.

iii. *Datum Spec Block Type* defines which Block type is being used. The default is type 0.

iv. *Data Frame Size* is the size, in bytes, of a data frame. This is not generally needed, but if required and absent, the size can be calculated from the size entries in the Datum Specification Blocks.

v. *UP/DOWN Flag* corresponds to the direction in which the data was taken (1= up, 255 = down, 0 = neither). The default is up.

vi. *Optical Log Depth Scale Units* flag specifies the depth units used on the optical log on the original recording (1 = feet, 255 = meters, 0 = time). The default is feet.

vii. *Data Reference Point* is a point fixed relative to the tool string. There is another point on the tool string called the tool reference point. Its distance from the surface corresponds to measured depth. At any instant in time during the real-time acquisition of data, the data reference point stands opposite the part of the hole to which the then current output corresponds. This value gives the distance of the data reference point above the tool reference point. It may have either sign. It is useful in determining the significance of data, such as tension or frame duration, which are not actually a function of depth. If absent, the value is undefined on the tape.

viii. *Units for Data Reference Point* value are expressed as 4 alphanumeric characters–i.e., .1IN. The default is inches in tenths.

ix. *Frame Spacing* is the depth difference between frames. For FLIC tapes, the value may be calculated from successive data frames.

x. *Units for Frame Spacing* value are expressed as 4 alphanumeric characters–i.e., .1IN. The default is inches in tenths.

xi. *Currently Undefined* field.

xii. *Maximum Frames/Record* is the maximum number of frames which will fit in a data record. This will normally be the number of frames for all but the last record. This is not generally needed, but if required and absent, this value may be computed from the record size minus the headers and trailers, the frame size, and the record depth, if present.

xiii. *Absent Value* defines a value which, if found as a frame entry, implies that the entry has no valid data and should be ignored. The default is -999.25, but absence of this entry block may indicate that no absent value was used.

xiv. *Depth Recording Mode* flag, if present with a value of 1, means that the depth occurs only once per data record preceding the first frame. When depth is recorded in this mode, an Entry Block 14 will be present to specify the units of depth, and an Entry Block Type 15 will be present to define the representation code of the depth. The frame sampling interval in this case is constant, and the depth for each successive frame in Entry Blocks 4, 8, 9 supply the necessary information to compute the depths of these successive frames. This mechanism makes it possible to maximize the amount of data on the tape. If this entry block is not present or the value indicates that depth does not occur once per data record (that is, its value is 0), depth normally appears in each frame as described by a Datum Specification Block. By including depth in every frame, the value is explicit, and the frame sample interval is known implicitly (1 = present, 0 = absent). If this entry is absent, the default value of 0 is assumed.

xv. *Units of Depth in Data Records* value is expressed as 4 alphanumeric characters–i.e., .1IN. The default is inches in tenths.

xvi. *Repr. Code for Output Depth* is the representation code that applies to depth if it is stored at the beginning of a data record.

xvii. *Datum Spec Block Sub-type* gives the sub-type number of the Datum Specification Block. This is used to indicate minor variations of Datum Specification Block form. If absent, a sub-type 0 is the default. Sub-type 1 Datum Specification Blocks have a new entry called "Process Indicators", and the API codes are represented as a 32-bit integer.

3. *Datum Specification Block* provides information required to identify and decode one frame entry. Thus, all of the Datum Specification Blocks, taken together, define a frame as it exists in the data records that follow. The total size of each block is 40 bytes.

A type 0, sub-type 0 Datum Specification Block has the following format.

Datum Spec Block (sub-type 0)			
Entry	*Size*	*Repr Code*	*Comments*
Mnemonic	4	65	3(a)
Service ID	6	65	3(b)
Service Order Nb	8	65	3(c)
Units	4	65	3(d)
API Log Type	1	66	3(e)
API Curve Type	1	66	3(e)
API Curve Class	1	66	3(e)
API Modifier	1	66	3(e)
File Nb	2	79	3(f)
Size	2	79	3(g)
"0"	2	79	
Process Level	1	66	3(h)
Nb Samples	1	66	3(i)
Representation Code	1	66	3(j)
"0"	1	66	
"0"	4	73	

A type 0, sub-type 1 Datum Specification Block has the following format.

Datum Spec Block (sub-type 1)			
Entry	*Size*	*Repr Code*	*Comments*
Mnemonic	4	65	3(a)
Service ID	6	65	3(b)
Service Order Nb	8	65	3(c)
Units	4	65	3(d)
API Codes	4	73	3(k)
File Nb	2	79	3(f)
Size	2	79	3(g)
"0"	3	79	
Nb Samples	1	66	3(i)
Representation Code	1	66	3(j)
Process Indicators	5	77	3(l)

(a) *Mnemonic* is the name of the channel.

(b) *Service ID* identifies the tool, the tool string used to measure the datum, or the name of the computed product. This identification is not a controlled dictionary mnemonic. It may be free form.

(c) *Service Order Nb* is a unique number which identifies the logging trip to the wellsite.

(d) *Units* indicate the units of measurement for the datum–i.e., inch.

(e) *API Codes (Sub-type 0)* form a largely outdated log/curve code system utilizing a 2 digit curve code (Ref: API Bulletin D-9 Feb '79. The user is also referred to an updating by the Canadian Well Society in 1975).

(f) *File Nb* indicates the file number at the time the data was first acquired (for well-site data acquisition tapes only) and written. This number, along with Service Order Nb and Service ID, will uniquely identify any data string for the purpose of merging or other processing.

(g) *Size* is the number of bytes reserved for the datum in the frame. If size is negative, output has been suppressed, though space is still reserved in the frame. The amount of space reserved is the absolute value of this entry.

(h) *Process Level* is a measure of the amount of processing done to obtain the datum. The size of the number increases in proportion to the amount of processing, but the system has never been objectively defined.

(i) *Nb Samples* indicates the number of samples of the datum per frame. This number times the size associated with the representation code equals the size of the block reserved for this datum.

(j) *Representation Code* is the numerical representation of the datum.

(k) *API Codes (Sub-type 1)* form a log/curve system featuring a 3 digit curve code (Ref: API Bulletin D-9 Jul '78). The API Codes are represented as a 32-bit integer. The single number is obtained by concatenating the decimal representations of the API Codes, and the resulting decimal number is converted to 32-bit binary. For example, assume that the main run Gamma Ray Curve from a Density Tool has the following API Codes:

Log Type = 45
Curve Type = 310
Curve Class = 01
Modifier = 1

The eight digit decimal integer is, then: 45310011.

(l) *Process Indicators* are used to designate the processing or corrections performed on the datum. Each bit in the Process Indicators represents a different computation. A value of 1 means processing has been done; a value of 0 means processing has not been done. Since a datum may undergo several types of correction, multiple flags may be set. The 40 bits are assigned as follows:

Process Indicators	
Bit Nb	*Definition*
0	Original logging direction
1	
2	True vertical depth correction
3	Data channel not on depth
4	Data channel is filtered
5	Data channel is calibrated
6	Computed (processed thru a function former)
7	Derived (computed from more than one tool)
8	Tool defined correction Nb 2
9	Tool defined correction Nb 1
10	Mudcake correction
11	Lithology correction
12	Inclinometry correction
13	Pressure correction
14	Hole size correction
15	Temperature corrrection
16	Unassigned
17	Unassigned
18	Unassigned
19	Unassigned
20	Unassigned
21	Unassigned
22	Auxiliary data flag
23	Schlumberger proprietary
24-39	Unassigned

Bits 0 and 1 form a single entry that defines the original logging direction for this particular channel. A value of 01 indicates the original logging direction was down-hole. A value of 10 indicates the original logging direction was up-hole. A value of 00 indicates an ambiguous original logging direction (i.e., stationary). A value of 11 is currently undefined.

4.1.7. Type 85: Picture Record

This type of record is used to record any picture descriptions. It may appear anywhere between the Reel Header Record and the final double EOF.

Picture Record (Type 85)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Picture Description	V		2

Figure 4.5: Picture Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Picture Description* may contain any valid picture description.

4.1.8. Type 86: Image Record

This type of record is used to record any image. It may appear anywhere between the Reel Header Record and the final double EOF.

Image Record (Type 86)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Image	V		2

Figure 4.6: Image Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Image* may contain a number of naked objects (in LIS 84 terminology), which characterize the Image Record, followed by bytes of the actual image.

4.1.9. Type 95: TU10 Software Boot Record

This record (the second one on all program tapes) initiates loading of a program tape for those programs which automatically load and execute the second record on a program tape.

TU10 Software Boot Record (Type 95)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Start Address	2	66	
TU10 Boot	592	66	2

Figure 4.7: TU10 Software Boot Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *TU10 Boot* is written in the machine language of the PDP-11. It supports TU-10 9-track, 800 bpi. The normal address for the controller status register is 172520.

4.1.10. Type 96: Bootstrap Loader Record

This record contains a program which will cause programs to be loaded into the computer.

Bootstrap Loader Record (Type 96)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Loader	V	66	2

Figure 4.8: Bootstrap Loader Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Loader* is written in the machine language of the particular computer for which it will be used. To execute an absolute loader, a user may have to load it using an appropriate loader booter.

4.1.11. Type 97: CP-Kernel Loader Boot Record

This record contains a program which causes the Bootstrap Loader to be loaded.

CP-Kernel Loader Boot Record (Type 97)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Reserved	4		2
Controller Codes	V	66	3
Loader Booter	V	66	4

Figure 4.9: CP-Kernel Loader Boot Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Reserved* bytes contain machine dependent code.
3. *Controller Codes* define codes for the controllers supported by this Loader Booter. Each code is one byte long, and the set is terminated with a zero.

Controller Codes	
Octal Code	*Controller*
102	MTUB
115	MTUC
124	TU10-9 track, 800 bpi
123	TU10-7 track, dump mode
103	Cartridge tape, 1600 bpi

4. *Loader Booter* is written in the machine language of the computer for which it is used.

4.1.12. Type 101: Program Overlay Header Record

This record defines an overlay load (Record Type 102) which follows.

Program Overlay Header Record (Type 101)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Program name	10	65	2
2 Blanks	2	65	2
Phase name	6	65	2
1 Blank	1	65	2
Version	9	65	2
Date of Tape Generation	8	65	2
2 Blanks	2	65	2
Date of Link	8	65	2
Load address	2	79	2
Load size (in words)	2	79	2
Transfer address	2	79	2

Figure 4.10: Program Overlay Header Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. The exact format of this record depends upon the machine into which the program is to be loaded. This shows a typical implementation of this record type.

4.1.13. Type 102: Program Overlay Load Record

This record is an overlay load, which is in a form to be loaded into computer core by the Absolute Loader (Record Type 96). This record must be preceded by a Program Overlay Header (Record Type 101).

Program Overlay Load Record (Type 102)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Load	V		2

Figure 4.11: Program Overlay Load Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Load* is written in the machine language of the computer for which it is used.

4.1.14. Type 128: File Header Record

The File Header Record is 58 bytes in length. It contains general information identifying the file.

File Header Record (Type 128)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
File Name	10	65	2
(Service Name)	(6)	(65)	(2a)
(".")	(1)	(65)	(2b)
(File Number)	(3)	(65)	(2c)
2 blanks	2	65	3
Service Sub Level Name	6	65	4
Version Number	8	65	5
Date of Generation	8	65	6
(Year)	(2)	(65)	(6)
("/")	(1)	(65)	(6)
(Month)	(2)	(65)	(6)
("/")	(1)	(65)	(6)
(Day)	(2)	(65)	(6)
1 blank	1	65	3
Maximum Physical Record Length	5	65	7
2 blanks	2	65	3
File Type	2	65	8
2 blanks	2	65	3
Optional Previous File Name	10	65	9

Figure 4.12: File Header Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *File Name* is a unique name for a file within a logical tape and consists of following parts:
   (a) *Service Name* is the name of the service or program that created the tape.
   (b) "." is simply a separator.
   (c) *File Number* is a 3-character counter (001, 002, etc, 999) that counts the files in a logical tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Service Sub Level Name* is a subdivision of the Service ID that is used to further classify the source of data.
5. *Version Number* is the version number for the software that wrote the original data.

6. *Date of Generation* is the date of generation for the software that wrote the original data. The format is:

   Year/Month/Day

   For example, 84/12/25.
7. *Maximum Physical Record Length* is the representation in alphanumeric digits of the maximum physical record length.
8. *File Type* is a 2-character indicator of the kind of information in the file. For example, LL for Label, LO for Log Data, CA for Calibration.
9. *Optional Previous File Name* is intended for disk-based implementations of LIS in which there may be no obvious predecessor or successor file. When used, it has the same format as the File Name (comment 2). When unused, it consists of 10 alphanumeric space characters. File Headers, File Trailers, Tape Trailers, and Reel Trailers are identical except for the record type and the definition of these 10 bytes.

4.1.15. Type 129: File Trailer Record

The File Trailer Record is 58 bytes in length. If present, it must be the last logical record in a file. It may be followed by a Tape Trailer but are always followed by End-of-File marks (EOF's).

The File Trailer Record is used primarily to check the data without having to backspace to the beginning of the last file.

File Trailer Record (Type 129)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
File Name	10	65	2
(Service Name)	(6)	(65)	(2a)
(".")	(1)	(65)	(2b)
(File Number)	(3)	(65)	(2c)
2 blanks	2	65	3
Service Sub Level Name	6	65	4
Version Number	8	65	5
Date of Generation	8	65	6
(Year)	(2)	(65)	(6)
("/")	(1)	(65)	(6)
(Month)	(2)	(65)	(6)
("/")	(1)	(65)	(6)
(Day)	(2)	(65)	(6)
1 blank	1	65	3
Maximum Physical Record Length	5	65	7
2 blanks	2	65	3
File Type	2	65	8
2 blanks	2	65	3
Optional Next File Name	10	65	9

Figure 4.13: File Trailer Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *File Name* is a unique name for a file within a logical tape and consists of following parts:
   (a) *Service Name* is the name of the service or program that created the tape.
   (b) *"."* is simply a separator.
   (c) *File Number* is a 3-character counter (001, 002, etc, 999) that counts the files in a logical tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Service Sub Level Name* is a subdivision of the Service ID that is used to further classify the source of data.

5. *Version Number* is the version number for the software that wrote the original data.
6. *Date of Generation* is the date of generation for the software that wrote the original data. The format is:

   Year/Month/Day

   For example, 84/12/25.
7. *Maximum Physical Record Length* is the representation in alphanumeric digits of the maximum physical record length.
8. *File Type* is a 2-character indicator of the kind of information in the file. For example, LL for Label, LO for Log Data, CA for Calibration.
9. *Optional Next File Name* is intended for disk-based implementations of LIS in which there may be no obvious predecessor or successor file. When used, it has the same format as the File Name (comment 2). When unused, it consists of 10 alphanumeric space characters. File Headers, File Trailers, Tape Trailers, and Reel Trailers are identical except for the record type and the definition of these 10 bytes.

4.1.16. Type 130: Tape Header Record

The Tape Header Logical Record is 128 bytes in length. It is used

- to identify the beginning of a set of Logical Files that constitute an LIS Logical Tape
- to provide the consumer with some information about a specific Logical Tape.

Tape Header Record (Type 130)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Service Name	6	65	2
6 blanks	6	65	3
Date	8	65	4
(Year)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Month)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Day)	(2)	(65)	(4)
2 blanks	2	65	3
Origin of Data	4	65	5
2 blanks	2	65	3
Tape Name	8	65	6
2 blanks	2	65	3
Tape Continuation Number	2	65	7
2 blanks	2	65	3
Previous Tape Name	8	65	8
2 blanks	2	65	3
Comments	74	65	9

Figure 4.14: Tape Header Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Service Name* is the name of the service or program that created the tape. The first six characters of this name are used in all File Header and File Trailer Records. In this fashion, all of the file names within a Logical Tape will be unique. The construction is to use the six-character service name and a ".", followed by a three-character counter (001, 002, etc, 999), which counts the files in a Logical Tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Date* is the date when the data was originally acquired. The format is:

   Year/Month/Day

   For example, 84/12/25.
5. *Origin of Data* is the system that originally acquired or created the data.
6. *Tape Name* is an ID that can be used to identify the Logical Tape, where applicable.

7. *Tape Continuation Number* is a number sequentially ordering multiple Logical Tapes stored on the same reel.
8. *Previous Tape Name* is an ID that can be used to identify the previous Logical Tape, where applicable. If this is the first Logical Tape, then this entry should be all blanks.
9. *Comments* are any relevant remarks concerning the Logical Tape or information contained within the Logical Tape.

4.1.17. Type 131: Tape Trailer Record

The Tape Trailer Logical Record is 128 bytes in length. If several Logical Tapes are stored on one Physical Reel, then this type indicates the end of a Logical Tape. This record is optional; in its absence a Logical Tape is assumed to be terminated when a new Tape Header Logical Record is encountered.

Tape Trailer Record (Type 131)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Service Name	6	65	2
6 blanks	6	65	3
Date	8	65	4
(Year)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Month)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Day)	(2)	(65)	(4)
2 blanks	2	65	3
Origin of Data	4	65	5
2 blanks	2	65	3
Tape Name	8	65	6
2 blanks	2	65	3
Tape Continuation Number	2	65	7
2 blanks	2	65	3
Next Tape Name	8	65	8
2 blanks	2	65	3
Comments	74	65	9

Figure 4.15: Tape Trailer Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Service Name* is the name of the service or program that created the tape. The first six characters of this name are used in all File Header and File Trailer Records. In this fashion, all of the file names within a Logical Tape will be unique. The construction is to use the six-character service name and a ".", followed by a three-character counter (001, 002, etc, 999), which counts the files in a Logical Tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Date* is the date when the data was originally acquired. The format is:

   Year/Month/Day

   For example, 84/12/25.
5. *Origin of Data* is the system that originally acquired or created the data.
6. *Tape Name* is an ID that can be used to identify the Logical Tape, where applicable.

7. *Tape Continuation Number* is a number sequentially ordering multiple Logical Tapes stored on the same reel.
8. *Next Tape Name* is an ID that can be used to identify the next Logical Tape, where applicable.
9. *Comments* are any relevant remarks concerning the Logical Tape or information contained within the Logical Tape.

4.1.18. Type 132: Reel Header Record

The Reel Header Logical Record is 128 bytes in length and is the first record on any physical reel. It is intended to identify the reel of tape.

Reel Header Record (Type 132)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Service Name	6	65	2
6 blanks	6	65	3
Date	8	65	4
(Year)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Month)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Day)	(2)	(65)	(4)
2 blanks	2	65	3
Origin of Data	4	65	5
2 blanks	2	65	3
Reel Name	8	65	6
2 blanks	2	65	3
Reel Continuation Number	2	65	7
2 blanks	2	65	3
Previous Reel Name	8	65	8
2 blanks	2	65	3
Comments	74	65	9

Figure 4.16: Reel Header Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Service Name* is the name of the service or program that created the tape. The first six characters of this name are used in all File Header and File Trailer Records. In this fashion, all of the file names within a Logical Tape will be unique. The construction is to use the six-character service name and a ".", followed by a three-character counter (001, 002, etc, 999), which counts the files in a Logical Tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Date* is the date when the physical reel was created. The format is:

   Year/Month/Day

   For example, 84/12/25.
5. *Origin of Data* is the system that originally acquired or created the data.
6. *Reel Name* is an eight-character name used to physically identify a specific reel of tape. This name matches the visual identification written on the tape canister.

7. *Reel Continuation Number* is a number sequentially ordering multiple Physical Reels and is an alphanumeric from 1 to 99.
8. *Previous Reel Name* is an ID that can be used to identify the previous Physical Reel, where applicable.
9. *Comments* are any relevant remarks describing the Reel of tape.

4.1.19. Type 133: Reel Trailer Record

The Reel Trailer Logical Record is 128 bytes in length and may optionally be used as the last record on a Physical Reel of tape.

Reel Trailer Record (Type 133)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Service Name	6	65	2
6 blanks	6	65	3
Date	8	65	4
(Year)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Month)	(2)	(65)	(4)
("/")	(1)	(65)	(4)
(Day)	(2)	(65)	(4)
2 blanks	2	65	3
Origin of Data	4	65	5
2 blanks	2	65	3
Reel Name	8	65	6
2 blanks	2	65	3
Reel Continuation Number	2	65	7
2 blanks	2	65	3
Next Reel Name	8	65	8
2 blanks	2	65	3
Comments	74	65	9

Figure 4.17: Reel Trailer Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Service Name* is the name of the service or program that created the tape. The first six characters of this name are used in all File Header and File Trailer Records. In this fashion, all of the file names within a Logical Tape will be unique. The construction is to use the six-character service name and a ".", followed by a three-character counter (001, 002, etc, 999), which counts the files in a Logical Tape.
3. *Blanks* are ASCII blanks used as filler characters.
4. *Date* is the date when the physical reel was created. The format is:

   Year/Month/Day

   For example, 84/12/25.
5. *Origin of Data* is the system that originally acquired or created the data.
6. *Reel Name* is an eight-character name used to physically identify a specific reel of tape. This name matches the visual identification written on the tape canister.

7. *Reel Continuation Number* is a number sequentially ordering multiple Physical Reels and is an alphanumeric from 1 to 99.
8. *Next Reel Name* is an ID that can be used to identify the next Physical Reel, where applicable.
9. *Comments* are any relevant remarks describing the Reel of tape.

4.1.20. Type 137: Logical EOF Record

A logical End-of-File (LEOF) serves the same purpose as a physical EOF. It may be used on a medium which does not have a physical EOF, or it may be used to replace physical file marks at higher software levels. A logical EOF consists of only a logical record header.

Logical EOF Record (Type 137)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1

Figure 4.18: Logical EOF Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.

4.1.21. Type 138: Logical BOT Record

A Logical Beginning of Tape (LBOT) serves the same purpose as a physical BOT. It may be used to indicate to a reading program that a physical BOT has been encountered. A Logical BOT consists of only a logical record header.

Logical BOT Record (Type 138)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1

Figure 4.19: Logical BOT Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.

4.1.22. Type 139: Logical EOT Record

A Logical End-of-Tape (LEOT) serves the same purpose as a physical EOT. It may be used to indicate to reading programs that a physical EOT has been encountered. A Logical EOT consists of only a logical record header.

Logical EOT Record (Type 139)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1

Figure 4.20: Logical EOT Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.

4.1.23. Type 141: Logical EOM Record

A Logical End-of-Medium (LEOM) serves the same purpose as a physical EOM. It defines the end of a medium. Upon encountering a physical EOM on a medium, the reading program may be given a Logical EOM to indicate this condition. A Logical EOM consists of only a logical record header.

Logical EOM Record (Type 141)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1

Figure 4.21: Logical EOM Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.

4.1.24. Type 224: Operator Command Inputs

On files generated by a real-time system, operator inputs may be recorded on the tape in records of this type.

Operator Command Inputs Record (Type 224)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Input Message	V	65	2

Figure 4.22: Operator Command Inputs Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Input Message* may contain any valid alphanumeric characters. It is generally terminated by a carriage return.

4.1.25. Type 225: Operator Response Inputs

This type of record is used to store input issued to the operator in response to a system request for information.

Operator Response Inputs Record (Type 225)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Input Message	V	65	2

Figure 4.23: Operator Response Inputs Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Input Message* may contain any valid alphanumeric characters. It is generally terminated by a carriage return.

4.1.26. Type 227: System Outputs to Operator

This type of record is used to store system output messages issued by the operator.

System Outputs to Operator Record (Type 227)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Output Message	V	65	2

Figure 4.24: System Outputs to Operator Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Output Message* may contain any valid alphanumeric characters. It is generally terminated by a carriage return.

4.1.27. Type 232: Comment Record

This type of record is used to record any desired comments. It may appear anywhere between the Reel Header Record and the final double EOF.

Comment Record (Type 232)			
Entry	*Size*	*Repr Code*	*Comments*
Logical Record Header	2		1
Comment	V	65	2

Figure 4.25: Comment Record

Comments:

1. *Logical Record Header* is described in Section 2.2.1.1.
2. *Comment* may contain any valid alphanumeric characters. It is generally terminated by a carriage return (but it can contain general free-form text of great length).

4.1.28. Type 234: Blank Record

Content is undefined. One use of this type of record is to permit writing a blank file which can, at a later time, be overwritten. Extreme caution should be exercised in using this technique. This is also used as a CSU-D comment record.

Chapter 5.

LIS IMPLEMENTATION

5.1. General Software Structure

Software to implement the LIS format may be classified into several levels. The table below lists some levels and the type of information from the LIS format that they do or do not know about.

LEVEL	*KNOWS ABOUT*	*DOES NOT KNOW ABOUT*
Device Handlers	The medium	The content of physical records
Physical Record Handler	Physical Record Headers and Trailers	The medium or the content of physical records beyond the header and trailer
Logical Record Handlers	Logical Record Types and Blocking Factors	Physical records
Application service routines	Constructing Frames retrieving general info	Logical records
Applications software	Frame and other general info	LIS format

5.2. Device Handlers

Device Handlers know all about the media aspects of information storage but virtually nothing about the content of the data they handle. For magnetic tape, these routines know how to read, write, and, in general, move tape; but the most knowledge they have about the content will be physical size. These routines must be able to sense and relay status information about the physical devices.

5.3. Physical Record Handlers

At a more sophisticated level are the physical record handlers. These routines know about the construction or decoding of physical record headers and trailers. Proceeding toward the physical medium, information is altered only by hardware beyond these routines. Conversely, moving toward the applications software, information must be in a coherent form when it leaves this level. Consequently, the following operations will occur at this level:

- Compute and check the checksum (if one is present). Set an error indicator if it is wrong.
- Retrieve and store file number information if it is present.
- Retrieve and store record number information if it is present.
- Pass on error indications from the Device Handler level.

Routines at this level receive raw physical records from device handlers and pass checksum, file number, and record number information to higher level routines. In addition, these routines should pass all the information between the physical record header and trailer upward in a transparent fashion. Conversely, they should be able to receive such information and append physical record headers and trailers. In particular, physical record boundaries should be invisible at higher levels.

5.4. Logical Record Handlers

The record content handlers must be able to distinguish logical record types. By knowing the logical record types, these routines will be able to decode or construct the various logical records. It is at this level that the information in the logical record is first examined. Routines at this level will block and unblock frames from logical data records.

5.5. Application Service Routines

The routines at this level provide direct support to the general applications software. General data handling protocol, such as opening and closing files, or logical tapes are implemented at this level. At the data level, frames are processed from external representations to internal formats.

Application Service Routines will have a multitude of entry points. The information required to understand an LIS-formatted data structure will be entered through these calls directly into table structures from which the appropriate logical records will be constructed by lower level routines. Entry points at this level will also be able to interrogate the status of data structures and logical devices.

5.6. Applications Software

The software at the applications level knows little about the LIS format other than that a minimum set of preliminary data is required. This level of program is concerned almost exclusively with interpretation of information.

Chapter 6.

ENCRYPTION

6.1. Encrypted Logical Records

Some tables contained in standard Information Records are classified as proprietary. In order to secure these tables on a real-time client tape, the tables are encrypted and stored on the tape as a new logical record type. These records are re-typed as Type 42. Only the body of the logical record following the logical record header is encrypted.

Appendix A.

ASCII CODES

Character	*ASCII* 7-Bit*	*Character*	*ASCII* 7-Bit*	*Character*	*ASCII* 7-Bit*
Space	040	@	100		140
!	041	A	101	a	141
"	042	B	102	b	142
#	043	C	103	c	143
$	044	D	104	d	144
%	045	E	105	e	145
&	046	F	106	f	146
'	047	G	107	g	147
(	050	H	110	h	150
)	051	I	111	i	151
*	052	J	112	j	152
+	053	K	113	k	153
,	054	L	114	l	154
-	055	M	115	m	155
/	056	N	116	n	156
.	057	O	117	o	157
0	060	P	120	p	160
1	061	Q	121	q	161
2	062	R	122	r	162
3	063	S	123	s	163
4	064	T	124	t	164
5	065	U	125	u	165
6	066	V	126	v	166
7	067	W	127	w	167
8	070	X	130	x	170
9	071	Y	131	y	171
:	072	Z	132	z	172
;	073	[	133	{	173
≺	074	\	134	|	174
=	075	]	135	}	175
≻	076		136	~	176
?	077	_	137	Delete	177
Horizontal Tab	011	Vertical Tab	013		
Line Feed	012	Form Feed	014	Carriage Return	015

*7-bit ASCII stored as 8-bit byte in octal. High order bit is 0.

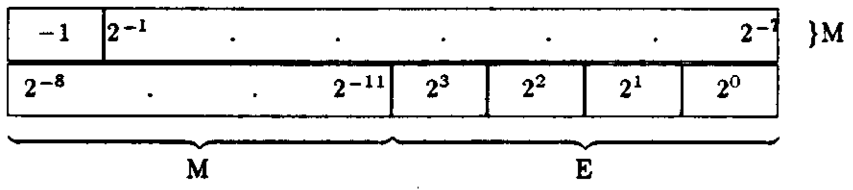
Appendix B.

REPRESENTATION CODES

Code Number	*Code as ASCII*	*Size (bytes)*	*Format*	*Comment*
49	1	2	16-bit Floating Point	B.1
50	2	4	32-bit Low Resolution Floating Point	B.2
56	8	1	8-bit 2's complement Integer	B.3
65	A		Alphanumeric	
66	B	1	Byte Format	B.4
68	D	4	32-bit Floating Point	B.5
70	F	4	32-bit Fixed Point	B.6
73	I	4	32-bit 2's complement Integer	B.7
77	M		Mask	B.8
79	O	2	16-bit 2's complement Integer	B.9
≻127				B.10

These codes indicate the manner in which the data is represented. Note that the corresponding ASCII character has been used to identify the codes in this document, but the representation of the representation codes themselves is 66 (byte format).

B.1. Code 49: 16-bit Floating Point



$$Value = M * 2^E$$

2's complement 12-bit fractional mantissa with 4-bit unsigned integer exponent.

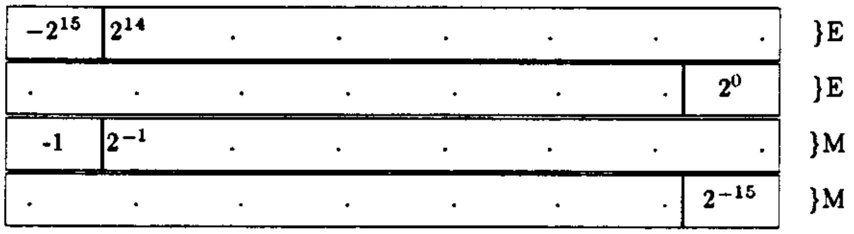
$$153_{10} = 231_8 = .462_8 \; x \; 2^8 =$$

Bit Number	
0-11	12-15
010011001000	1000

$$-153_{10} = -231_8 = -.462_8 \; x \; 2^8 =$$

Bit Number	
0-11	12-15
101100111000	1000

B.2. Code 50: 32-bit Low Resolution Floating Point



$$Value = M * 2^E$$

2's complement 16-bit fractional mantissa with 2's complement 16-bit integer exponent.

$$153_{10} = 231_8 = .462_8 \; x \; 2^8$$

Bit Number	
0-15	16-31
0000000000001000	0100110010000000

$$-153_{10} = -231_8 = -.462_8 \; x \; 2^8$$

Bit Number	
0-15	16-31
0000000000001000	1011001110000000

B.3. Code 56: 8-bit Integer

-2^7	2^6 . . . . . 2^0

2's complement 8-bit integer.

$$89_{10} = 131_8$$

Bit Number
0-7
01011001

$$-89_{10} = -131_8$$

Bit Number
0-7
10100111

B.4. Code 66: Byte

2^7 . . . . . . 2^0

Unsigned 8-bit integer.

B.5. Code 68: 32-bit Floating Point

Bit Number		
0	1-8	9-31
Sign	2^7-2^0	2^{-1}-2^{-23}
Sign (S)	Exponent (E)	Fraction (F)

Sign bit appended to the fraction yields a 24-bit Mantissa (M). If S=0, then the number is positive, and:

1. The mantissa is a binary fraction.
2. The exponent is expressed as excess 128.

$$Value = M * 2^{E-128}$$

If S=1, then the number is negative, and:

1. The mantissa is the two's complement of a binary fraction.
2. The exponent contains the one's complement of the normal exponent code.

$$Value = M * 2^{127-E}$$

If M=0, then the entire word is set to 0.

$$Value = O$$

$$153_{10} = 231_8 = +.462_8 \; x \; 2^8 =$$

Bit Number		
0	1-8	9-31
0	10001000	10011001000000000000000

$$-153_{10} = -231_8 = -.462_8 \; x \; 2^8 =$$

Bit Number		
0	1-8	9-31
1	01110111	01100111000000000000000

B.6. Code 70: 32-bit Fixed Point

-2^{15}	2^{14}	2^{13} . . . . .
. . . . 2^{0}		
2^{-1}	2^{-2}	
. . . . 2^{-1}		

2's complement with binary point in the middle.

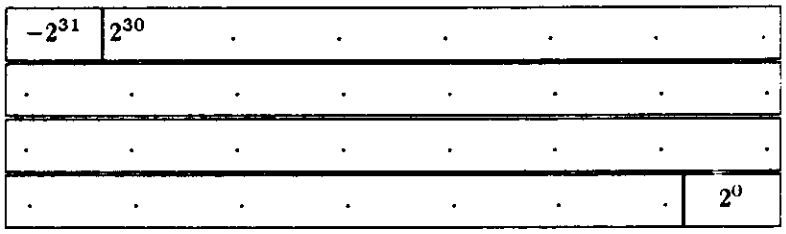
$$153.25_{10} = 231.2_8$$

Bit Number	
0-15	16-31
0000000010011001	0100000000000000

$$-153.25_{10} = -231.2_8$$

Bit Number	
0-15	16-31
1111111101100110	1100000000000000

B.7. Code 73: 32-bit Integer



2's complement 32-bit integer.

$153_{10} = 231_8$

Bit Number
0-31
00000000000000000000000010011001

$-153_{10} = -231_8$

Bit Number
0-31
11111111111111111111111101100111

B.8. Code 77: Mask

1	1	1	1	1	1	1	1

Each bit position has its own meaning.

B.9. Code 79: 16-bit Integer

-2^{15}	2^{14} .	.	.	.	.	.	.
.	.	.	.	.	.	.	2^0

2's complement 16-bit integer.

$$153_{10} = 231_8$$

Bit Number
0-15
0000000010011001

$$-153_{10} = -231_8$$

Bit Number
0-15
1111111101100111

B.10. Code Greater Than 127

If the representation code is equal to or greater than 128, it indicates the datum is a raw data block. That means that the data is a string of bytes with an unknown internal structure (as far as the LIS Format is concerned). This internal structure must be indicated externally to the tape.

Appendix C.

CHECKSUM ALGORITHM

The checksum may appear in the physical record trailer. It includes the physical record header and trailer, except for the checksum value itself. The checksum is a 16-bit integer quantity computed using a cyclic-redundancy type checksum algorithm. This algorithm is described below. Note that this algorithm assumes that there are an even number of bytes in the data block.

```
1) checksum=0                 initialize checksum to zero
2) loop i=1,n,2               loop over the data two bytes at a time
3)   t=byte(n+1)*256+byte(n) compute a 16-bit addend by concatenating
                              the next two bytes of data
4)   c=c+t                    add the addend to the checksum
5)   if carry c=c+1           add carry to checksum
6)   c=c*2                    left shift checksum
7)   if carry c=c+1           add carry to checksum
8) endloop
```