System Independent Data Format

April 27, 1993
Rev 1.3

Novell® Inc.
122 East 1700 South
Provo, Utah 84606

Table Of Contents

List of Figures

Introduction

List of Figures

Term Definition

Important terms for using and understanding the System Independent Data Format (SIDF) are defined below:

- Transfer buffer -- A buffer or a logical collection of sectors transferred in a high level I/O request (e.g., 32K, 64K, 128K, 256K, etc.). The buffer size must be a multiple of the medium's physical sector size.

- Session -- A group of associated data sets.

- Logical sector -- The smallest unit of data that can be transferred between the media process and the process transferring the data. This size is 512 bytes.

- Physical sector -- The smallest number of bytes that can be transferred to/from the media (e.g., 512 bytes, 1K, 4K, etc.). Unless specifically stated otherwise, the term *sector* refers to a physical sector.

- Session Physical Sector Address -- The number of physical sectors from the session header (i.e., relative addresses). For example, Figure 1-1 shows physical sectors 22 through 2B. The figure shows two interleaved sessions and the addresses of their respective transfer buffers in parenthesis.

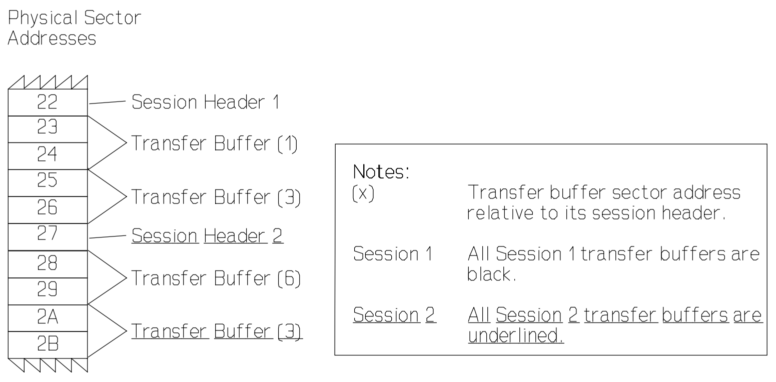

Physical Sector Addresses
22
23
24
25
26
27
28
29
2A
2B
Session Header 1
Transfer Buffer (1)
Transfer Buffer (3)
Session Header 2
Transfer Buffer (6)
Transfer Buffer (3)
Notes:
(x) Transfer buffer sector address relative to its session header.
Session 1 All Session 1 transfer buffers are black.
Session 2 All Session 2 transfer buffers are underlined.

Figure 1-1. Session Physical Sector Address Example

In this example, each transfer buffer occupies two sectors. Session 1 starts at sector 22 and has three transfer buffers at session physical sector addresses 1, 3, and 6. Session 2 starts at sector 27 and it has one transfer buffer whose address is 3. In essence, the transfer buffer's session physical sector address is the number of sectors from its session header.

- Absolute Physical Sector Address -- The physical sector address of the media. In Figure 1-1 these are addresses 27 through 2B.

Note: Physical sector zero is the first addressable sector within the SIDF space.

- Logical Sector Zero -- The first physical sector that can be addressed within the SIDF defined area.

- Data Set -- The data set data and data set information.

- Data Set Data -- A set of related data, attributes, and characteristics. For example, the data set data for a DOS file is the file data, file name, path, last modified date and time, and attributes (e.g., read only or system).

- Data Set Information -- A set of descriptive information used for display purposes only (e.g., data set name, or creation date). This is a separate set of information from the data set data and may contain duplicate information.

Task and Information Map

The following provides a map for readers to digest the material contained in this document efficiently. The following presents a list of tasks or information:

- Theory of SIDF - Read "System Independent Data Format" (page 1-5).

- Fast tour of SIDF - See Figure 1-2 (page 1-6), Figure 1-4 (page 1-9), Figure 1-5 (page 1-12), and Figure 1-6 (page 1-13).

- Developing your own FIDs - Read "Data Representation" (page 1-17).

- SMS Developers see Appendix E, "Notes for SMS Developers."

System Independent Data Format

System Independent Data Format (SIDF) is a specification that

- Provides a logical layout for media, session(s), data, indexes, and databases.
- Can provide a common data format across all platforms, thus allowing one engine to transfer data between multiple platforms. A platform is any object that has data that needs to be formatted (e.g., an OS or a service [e.g., printing]).
- Allows an engine to use many media types (e.g., tape, Magneto Optical, Write Once Read Many [WORM], random access media) without requiring media-specific knowledge. SIDF is not limited to currently defined media types.
- Ensures that the future and current engines and data are backwards and forwards compatible.
- Is extensible. New platform-specific and developer-specific data can be added while maintaining SIDF compliance.

Compatibility

SIDF helps to ensure compatibility among all future and current SIDF-compatible engine and stored-SIDF data because it provides a common logical data format for platforms (e.g., DOS and Macintosh) and media.

For instance, data sets from one platform can be restored to another platform. If the destination platform does not support parts of a data set, the parts may need to be discarded. For example, when restoring a session containing Macintosh data sets to a DOS target, the engine discards the resource forks but keeps the DOS equivalent (the data fork).

Another example is restoring an older data set to a newer file system that supports enhanced data sets (e.g., a new attribute was added). The restoring process creates the missing information for the old data set.

Security

SIDF does not enforce security, but provides the mechanism to allow engines to implement it. The mechanisms are authentication and data set encryption at various levels of SIDF. These are discussed later in this document (see "security" in the index).

Logical Format

SIDF starts from logical sector zero through the end of SIDF recorded data and defines a logical view of the medium, session, and data. It contains three levels, where the higher level encapsulates the lower level. The levels are, starting from the top, the media level, the transfer buffer level, and the data set level (see Figure 1-2). The media level contains one or more sessions. Within this level is the transfer buffer level, which divides the session data into logical segments called transfer buffers. The transfer buffers contain record(s) and subrecord(s), which in turn contain the session's data sets. The data set level contains the data set data (e.g., path and data).

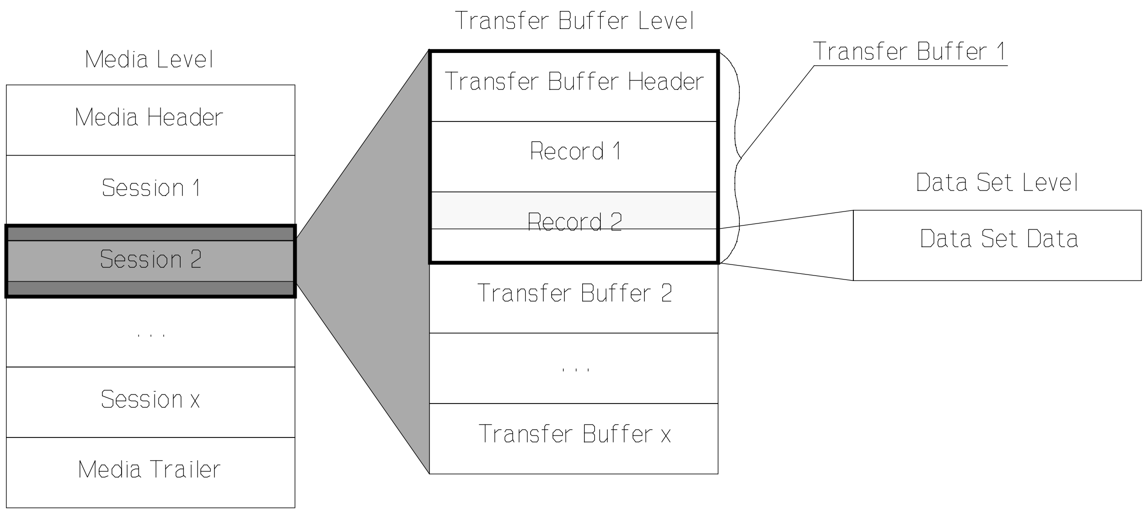


Figure 1-2. SIDF

Figure 1-3 shows a high level example of SIDF's components on the media.

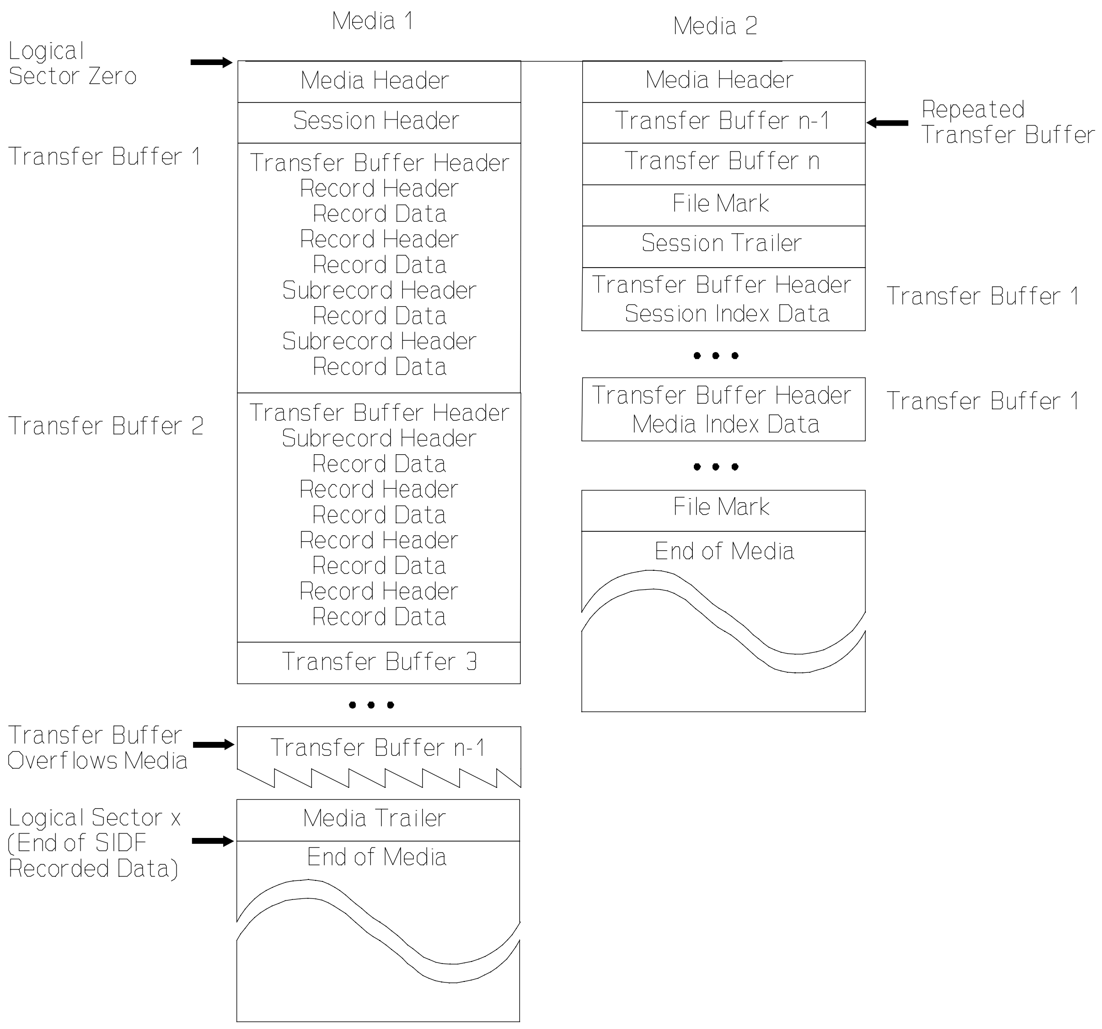


Figure 1-3. SIDF Layout

Figure 1-3 shows a session having n transfer buffers and that media 2 entirely repeats transfer buffer n-1 because it cannot fit on media 1. The other blocks are discussed later in this document.

Media level

The media level defines the start and end of SIDF data, the media set, and the session(s). The start of SIDF data begins at logical sector 0, which contains a media header. Logical sector 0 is the first sector that begins the SIDF data. If sector 27 contained the media header, that sector is logical sector 0.

The media set is a set of related media. Each medium within a media set can be identified through one of two methods:

1. Each medium in the media set has the same label and a unique sequence number.

2. Every medium may have an *alternate media label*.

Each medium within the media set can be different. That is, the device used, physical sector size, etc. does not have to be the same.

A session is a set of related data sets. The user defines the relationship (e.g., the user backs up one directory).

Figure 1-4 shows the media level. Notice that all media must begin with a media header. Following this are one or more sessions. Each session contains, in order, a session header, session data, a file mark, session trailer, and session index. The media index follows the session trailer, and is optional, however, there must be at least one in a media set. Set marks begin every session, except the first one in a media set.

Note: Set marks and file marks can be emulated on devices that do not support them. Also, the file mark and media trailer are recommended if a session overflows the medium.

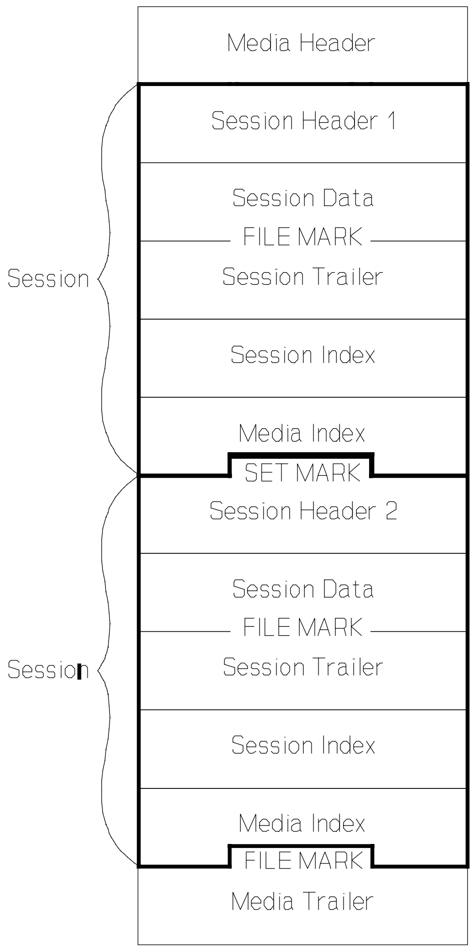


Figure 1-4. Media level

The session index is a log of serviced data sets and their location on the media, and provides redundancy. The media index is a log that records the location of every session that precedes the media index.

Data Set Databases. Besides the session and media indexes, databases also can track the serviced data sets. The database may keep track of all backup instances of a data set. This allows a user to instantaneously see variations of a data set through time without having to search through many media or log files.

SIDF allows the database to be kept on the medium in one of several places. Searching for the database can consume a relatively large portion of time, therefore the placement of databases is an important issue. The following paragraphs describe SIDF's options for placing and finding a database.

Database Location Information -- Finding a database on the media with many databases can take a considerable amount of time. Consequently, SIDF provides 5 methods to place them. The device determines the method selected. Information about the method used (and sometimes the parameters used) should be recorded in the media header. The following paragraphs describe the supported database placement methods.

Method 1: Databases can occur only after a session trailer or media index. The database must occur after the session trailer and before the media index (if present).

Method 2: Two partitions - Partition one (fixed in size) contains the addresses of the databases. Partition zero contains intermixed session data and databases.

Method 3: Two logical partitions converging from opposite ends of the same physical partition - The partition growing from the end of the media contains the database addresses. The partition growing from the beginning contains intermixed session data and databases.

Method 4: Two logical partitions converging from opposite ends of the same physical partition - The partition growing from the end contains the databases. This method is not preferred, because locating the databases requires more time, and it must be written in reverse.

Method 5: One partition with databases allowed only at specified offsets - Align databases on an offset or multiples of an offset. To find a database, look at each offset to learn if it is there.

When selecting an offset value, it is necessary to consider:

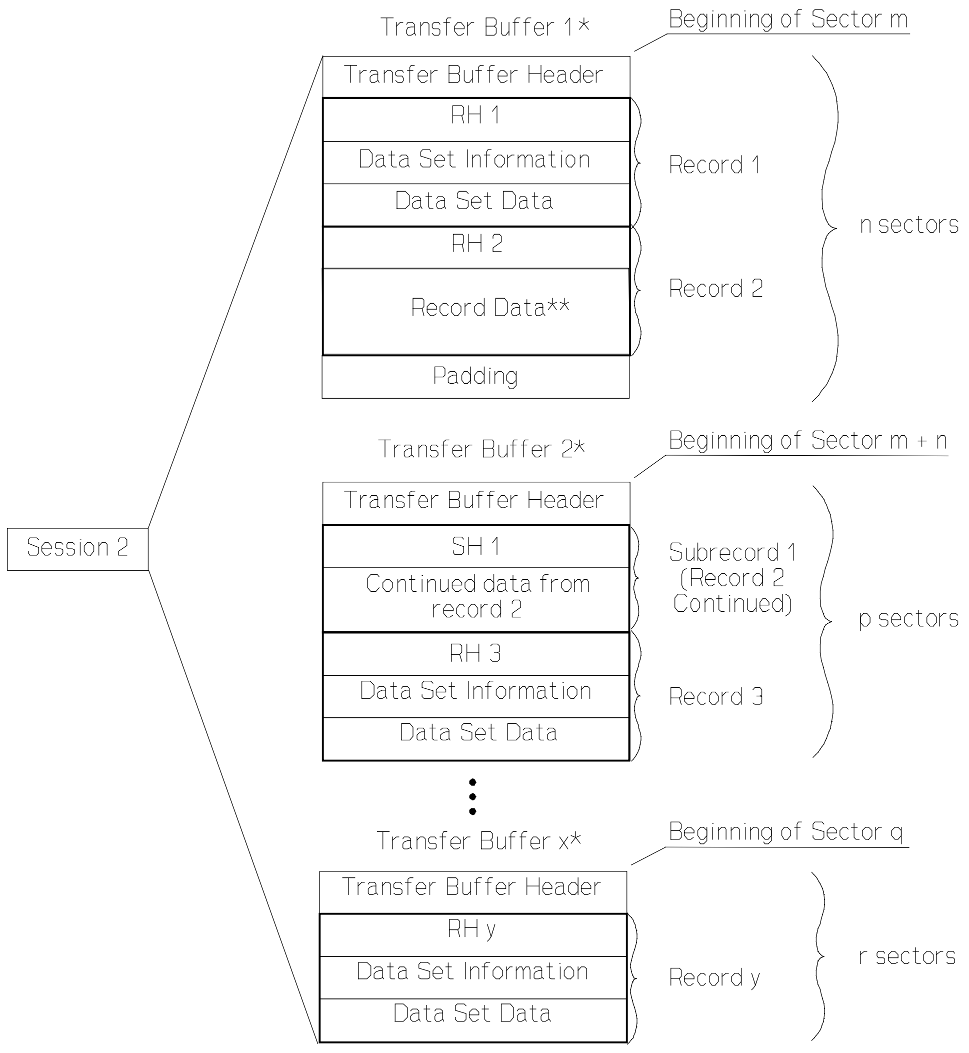
- The number of databases that will be written on the medium.
- The amount of time required to find them.
- The amount of space wasted caused by skipping from the end of the session data to the next offset location because the session data are not aligned to the offsets. That is, to write the database information, the media engine must jump to the next offset location.

Transfer Buffer Level

The transfer buffer level divides the session data into one or more groups called transfer buffers. These transfer buffers contain a transfer buffer header and one or more record and\or subrecords.

The records and subrecords contain the session's data sets. Each record contains a record header and one data set, which consists of the data set information and the data set data. The subrecord contains a subrecord header and the rest of the data that could not fit into the record (repeat the subrecord as often as necessary).

The data set information contains descriptive data that is used to display the serviced data set (this information is not needed to restore the data set). The data set data contains the actual data set. "Data Set Level" in this chapter further defines the data set data. Figure 1-5 shows the data set level.



*Transfer buffers are aligned on physical sector bounderies and must fully occupy the sectors they use (e.g., a transfer buffer cannot occupy part of a sector).

** May contain partial Data Set Information or Data Set Data. The remaining data (overflow) is in subrecord 1.

Figure 1-5. Transfer Buffers Level

Media Overflow. If a transfer buffer overflows the media, the engine should do the following:

1. Write a file mark if possible.

2. Write a media trailer if possible.

3. Write a media header onto the next medium.

4. Rewrite the complete transfer buffer.

Data Set Level

The data set level defines the format of the data set data. SIDF divides the data set data into logical sections such as the data streams, extended attributes, trustees, etc. Figure 1-6 and Figure 1-7 show the data set level (other data sets can be added here).

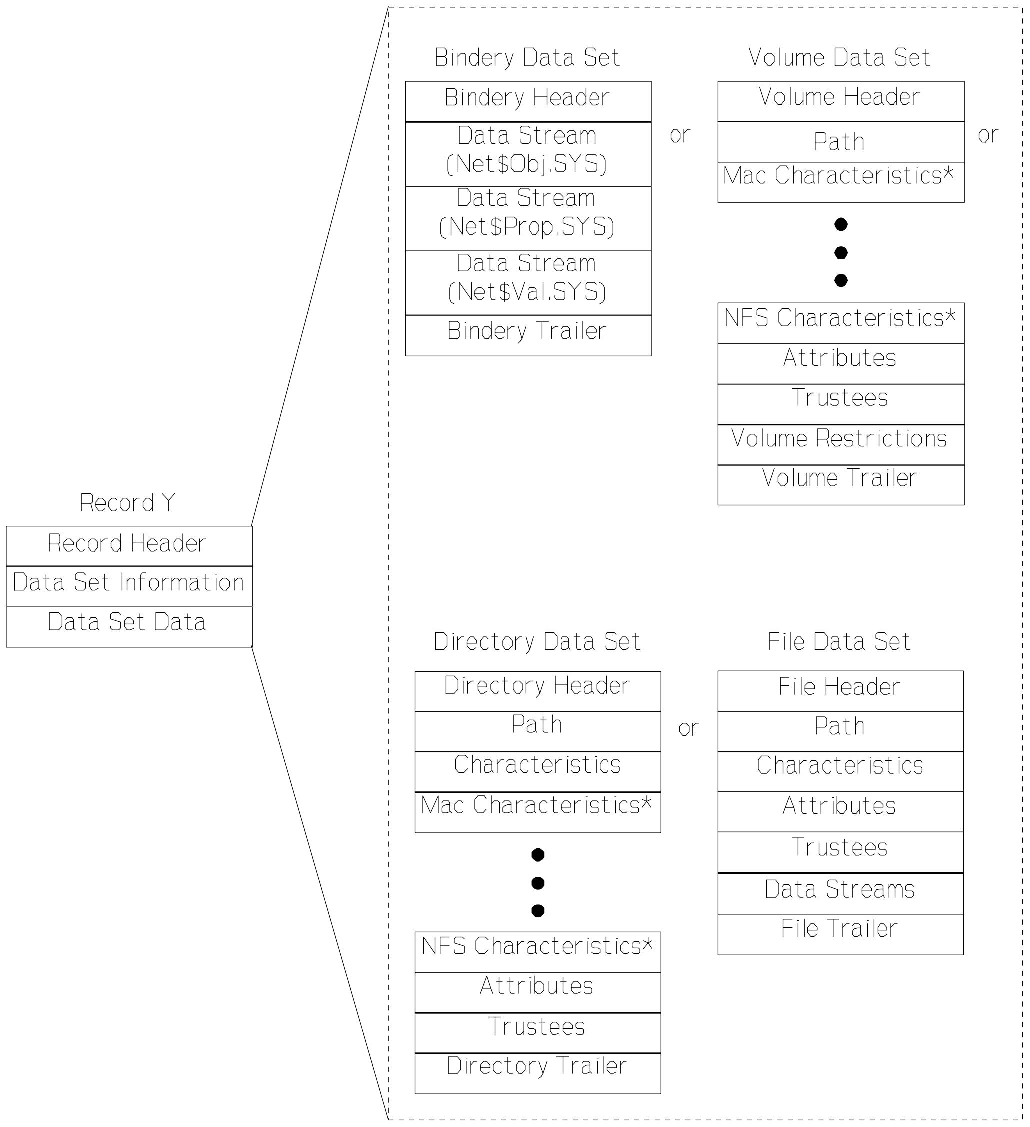


Figure 1-6. Data Set Level

Figure 1-7 shows the format of a transaction set (i.e., files for one database). Notice that the whole database is treated as one data set.

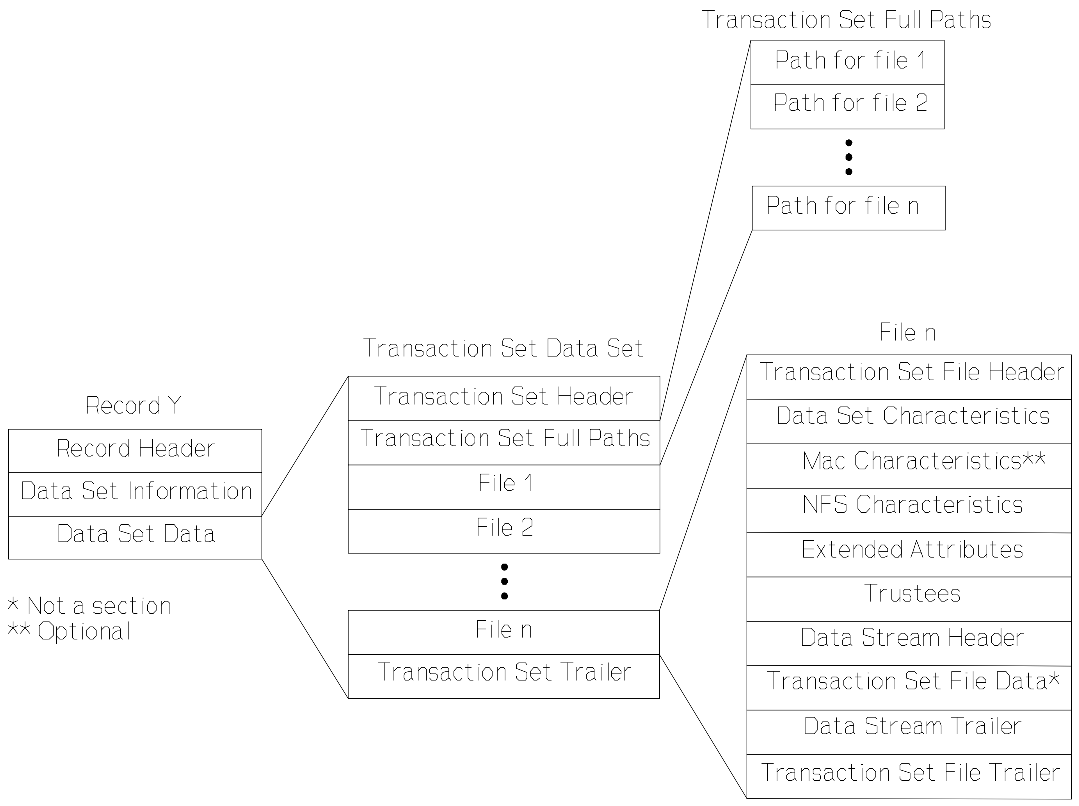


Figure 1-7. Transaction Sets

Figure 1-6 and Figure 1-7 show that the path and characteristic sections are usually the second and third sections (except in special cases like the bindery). The other sections can occur in any order and in any number (e.g., data streams, name spaces, data streams, data streams, extended attributes, etc.).

Note: The target's name space type, not SIDF, defines the case sensitivity for objects such as a path, data set name, etc. Therefore, the engine must ensure that the objects received from the user, objects passed to the target, etc., have the proper case.

File Service Example

The following shows a high-level view of formatting a file service to SIDF's specifications. File service Ed has the following structure:

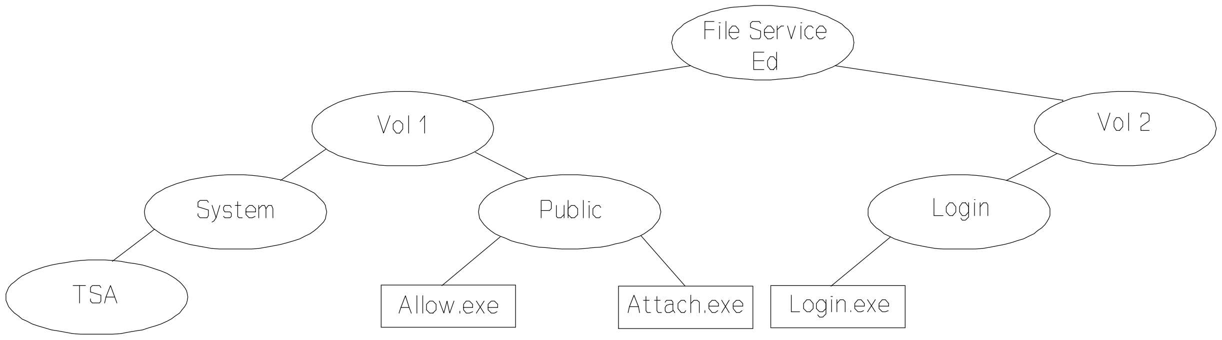


Figure 1-8. File Service Ed

Representing Ed according to SIDF's specifications could result in the following as it would appear on the media:

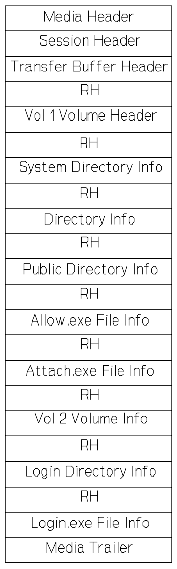


Figure 1-9. SIDF Representation of Ed

Each information block in Figure 1-9 represents a group of sections as shown in Figure 1-6. Figure 1-9 assumes that each information block fit into one record and all records fit into one transfer buffer. The name, Ed, is not explicitly represented because it is contained in the session header. The full paths section preserves the structure of Ed (see "Full Paths" in Chapter 2).

Data Representation

SIDF's specifications are explained below. For those defining new data sets, the following information is critical to designing your sections.

Sections

A section is a logical grouping of data. SIDF identifies items such as the session header, media index, volume header, path, and characteristics as sections (see Figure 1-2 and Figure 1-6). The session data item (see Figure 1-2) is not a section, but consists of many sections as shown in the data set level (see Figure 1-6).

The section is a table of related fields. The first field in the table identifies the section and contains parsing resynchronization data (see "Resynchronization" in this chapter for more information). The last field marks the end of the section and usually contains, a CRC[1] value for all fields in the section except the last field. The position of the first, second (offset to end), and last fields are fixed; other fields can be in any order (except where noted). If the second field is not needed, its space is used by any field.

For example, the media trailer section contains three fields:

Media Trailer
Close Date and Time
Media Trailer

The first and last field, media trailer shows the start and end of the section. Close date and time contains the date and time the media was closed. The following paragraphs discuss the format of a field.

Fields

As described above, a group of related fields makes a section. A field:

- Gives the data a name (a Field IDentifier [FID])
- Shows the data's size
- Contains the data

[1] The CRC polynomial is $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x^{1} + x^{0}$. This can be found in the ANSI X3.66 (X.25).

Figure 1-10 shows a logical view of a field. Some fields physically have three subfields, while others imply the subfields. This will become apparent as the discussion progresses.

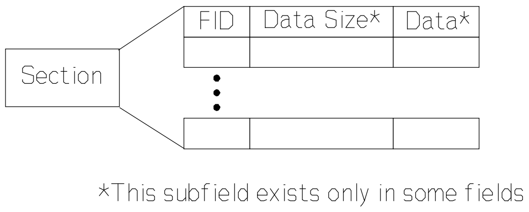


Figure 1-10. Fields

FIDs are contained in a byte stream, where the previous byte determines how the next byte is interpreted. Each FID is a bit pattern that uniquely identifies the field and its data (e.g., media trailer, software version, or owner ID).

The data size is a complex descriptor that shows the data's size in bytes or bits. In some FIDs the data's size is part of the FID. Consequently, there is no separate "data size" subfield as shown in Figure 1-10.

The data subfield exists as a separate entity for most FIDs; in some cases however, the FID contains the data. The data's byte order is little endian (low-high byte order). The following paragraphs further describe the subfields.

The Field IDentifier (FID)

The basic FID has three or four parts, depending upon how it is used. These parts are the

- FID size bit (it shows if a FID is small, short, or long)
- Variable or fixed data size indicator
- FID number
- Size of data

Note: FIDs are contained in a byte stream, where the first byte determines how the next byte is interpreted.

The FID size bit is size of the FID, not the data's size. The small FID is the basic FID and consists of only one byte. The short FID adds one byte to the small FID and the long FID adds another byte. These other bytes serve to increase the number of available FIDs. More bytes can be added can be added to further increase the number of FIDs.

The "variable or fixed data size indicator" shows if the data's size varies over time or is fixed over time. Depending upon the FID, this information is contained in one or more bits. When set to a proper value, it suggests a variable data size and one or more data size bytes follow the FID (these bytes contain the data's size). "Data Size," in this chapter, further discusses the data size bytes.

If the "variable or fixed data size indicator" is set to another value, four bits in the FID contains the data's size in powers of two. Fixed data sizes from 1 byte to 32 kb is representable.

The FID number is a unique number given to a type of data. For example, one number could represent the general idea of a path as used in DOS, Macintosh, and UNIX.

The size of data subfield, exists as a separate component for variable FIDs and is part of the FID for fixed FIDs. This idea is important to note, because it helps to decide what is the field identifier. For variable FIDs, only the FID and not the data size bytes are used to identify the data. For fixed FIDs however, the whole FID, including the four data size bits,[2] identifies the data.

As mentioned in the above paragraphs, additional bytes can be added to the basic FID to increase the number of FIDs. The following bytes have been added:

- The standard byte
- The developer byte
- The extended developer byte

The following paragraphs discuss these FIDs.

Note: special case FID.

Standard FIDs. The standard FID defines data common to most targets and contains the

- Small Standard FIDs (the basic FID)
- Short Standard FID
- Long Standard FID

[2] The last four bits contains the data's size in powers of two.

Small FIDs are 1 byte, and are identified by the seventh bit, which is set to zero. The FID is variable if bits 4 through 6 are less than or equal to 3. If the bits are greater than or equal to 4, the FID is fixed. If the FID is variable, the lower six bits contain the FID number (the value of these bits must be greater than or equal to 1). If the FID is fixed then bits four and five contain the FID number, while bits 0 through 3 contains the data's size in powers of two. Figure 1-11 shows the bit pattern for the small standard FIDs.

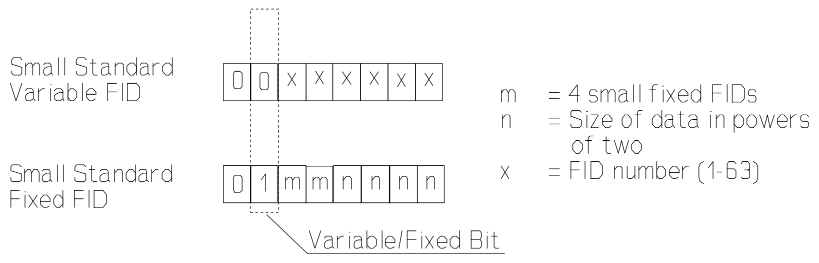


Figure 1-11. Small Standard FIDs

The table below shows the range of small standard FIDs.

Note: m and n are explained in Figure 1-11.

Small Standard FID Range	
FID Type	**Valid FID Values**
Variable Size	0x01 - 0x3F[3]

Fixed Size[4]	0xmn - 0xmn[5]
	m = 4 to 7
	n = 0 to F

[3] 64 standard, variable size, small FIDs.

[4] Fixed size means that the data's size is encoded within the FID.

[5] 4 standard, fixed-size, small FIDs per developer number. Each FID can have one of the following sizes: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, and 32K Bytes.

Figure 1-12 shows that the short standard FID adds a byte, 0x80, in front of the small FID. The FID is variable if bits 4 through 6, of the low byte, are less than or equal to 3. If the bits are greater than or equal to 4, the FID is fixed.

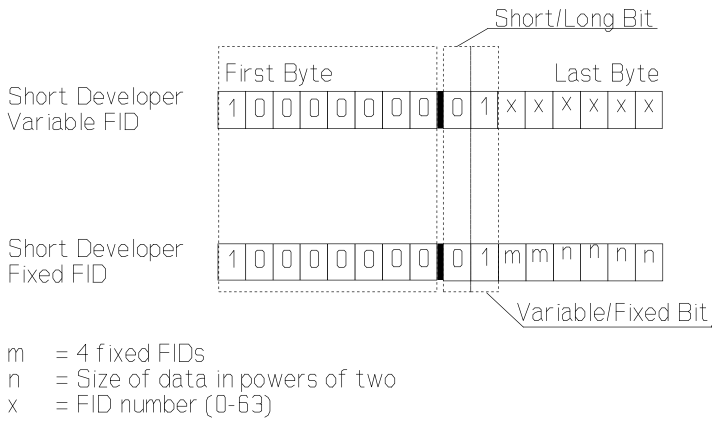

Short/Long Bit
First Byte
Last Byte
Short Developer Variable FID
1 0 0 0 0 0 0 0 0 1 x x x x x x
Short Developer Fixed FID
1 0 0 0 0 0 0 0 0 1 m m n n n n
Variable/Fixed Bit
m = 4 fixed FIDs
n = Size of data in powers of two
x = FID number (0-63)

Figure 1-12. Short Standard FIDs

The following table shows the range of the short standard FIDs.

Note: See Figure 1-12 for a description of m and n.

Short Standard FID Ranges	
Type	**Range**
Variable Size	0x8000 - 0x803F[6]

Fixed Size	0x80mn - 0x80mn[7]
	m = 0x4 to 0x7
	n = 0x0 to 0xF

[6] 64 standard, variable-size, short FIDs.

[7] 4 standard, fixed-size, short FIDs. Each FID can have one of the following sizes: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, and 32K bytes.

Like the short standard FID, the long standard FID has the 0x80 byte, but it also has an additional byte at the end. If the high nibble of the middle byte is greater than or equal to 0x8 and less than or equal to 0xE, then it is a long standard variable FID. If it is 0xF, then it is a long standard fixed FID (see Figure 1-13).

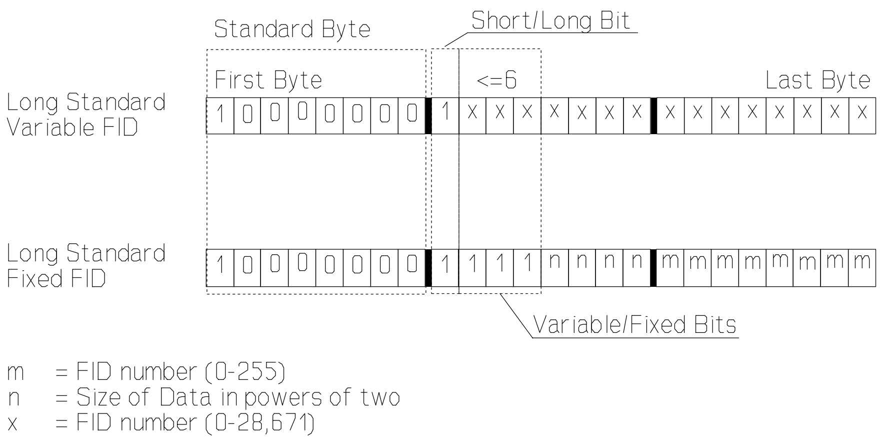

Standard Byte
Short/Long Bit
First Byte
<=6
Last Byte
Long Standard Variable FID
1 0 0 0 0 0 0 0 1 x x x x x x x x x x x x x x x
Long Standard Fixed FID
1 0 0 0 0 0 0 0 1 1 1 1 n n n n m m m m m m m m
Variable/Fixed Bits
m = FID number (0-255)
n = Size of Data in powers of two
x = FID number (0-28,671)

Figure 1-13. Long Standard FID

The following table shows the range of the long standard FIDs.

Note: See Figure 1-13 for a description of m and n.

Long Standard FIDs

Type	Range
Variable Size	0x808000 - 0x80EFFF[8]

Fixed Size	0x80Fnmm - 0x80Fnmm[9]
	mm = 0x00 to 0xFF
	n = 0x0 to 0xF

8 28K standard, variable-size, long FIDs.

9 256 standard, fixed-size, long FIDs. Each FID can one of the 16 following sizes: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, and 32K bytes.

Developer FIDs. The developer FIDs allow for developer specific information and will be assigned to the first 62 developers. The developer FID adds another byte to the front of the small standard FID.

The developer FID is identified by the seventh and sixth bits of the first byte, which is always set to 10. The lower six bits contain the developer number.

Short FIDs set the seventh bit of the low byte to 0. The FID is variable if bits 4 through 6, of the lower byte, are less than or equal to 3. If the bits are greater than or equal to 4, the FID is fixed. For fixed FIDs, bits four and five show the ID number, and bits 0 - 3 tell the data size in powers of two. Figure 1-14 shows the bit pattern.

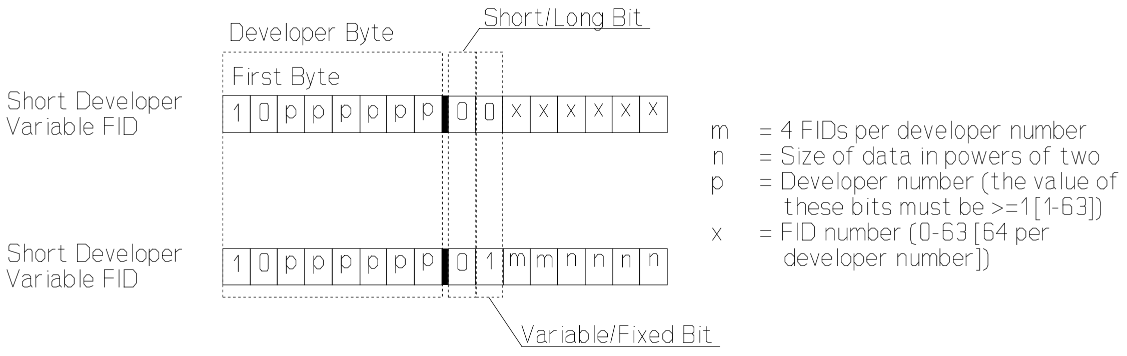

Developer Byte
Short/Long Bit
First Byte
Short Developer Variable FID
1 0 p p p p p p 0 0 x x x x x x
Short Developer Variable FID
1 0 p p p p p p 0 1 m m n n n n
Variable/Fixed Bit
m = 4 FIDs per developer number
n = Size of data in powers of two
p = Developer number (the value of these bits must be >=1 [1-63])
x = FID number (0-63 [64 per developer number])

Figure 1-14. Short Developer FIDs

The following table shows the FID's range.

Note: See Figure 1-14 for a description of m, n, and p.

Short Developer FID Ranges

Type	**Range**
Variable Size	0xpp00 - 0xpp3F[10]

Fixed Size	0xppmn - 0xppmn[11]
	m = 0x4 to 0x7
	n = 0x0 to 0xF
	pp = 0x81 to 0xBF

[10] 64 variable-size short FIDs per developer number (62 developer numbers).

[11] 4 fixed-size FIDs per developer number (62 developer numbers). Each FID can represent one of the following 16 sizes: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, and 32K bytes.

The long developer FID adds a byte to the end of the short developer FID, which increases the number of FIDs per developer number from 64 to 28k. Like the short developer FID, the long developer FID also sets the seventh and sixth bit of the first byte to 10.

Since this is a long FID, the seventh bit of the second byte is set to 1. Bits 4 - 6 suggest a variable or fixed FID. If the value is less than or equal to 6, we have a variable FID. If the value is 7, we have a fixed FID (see Figure 1-15).

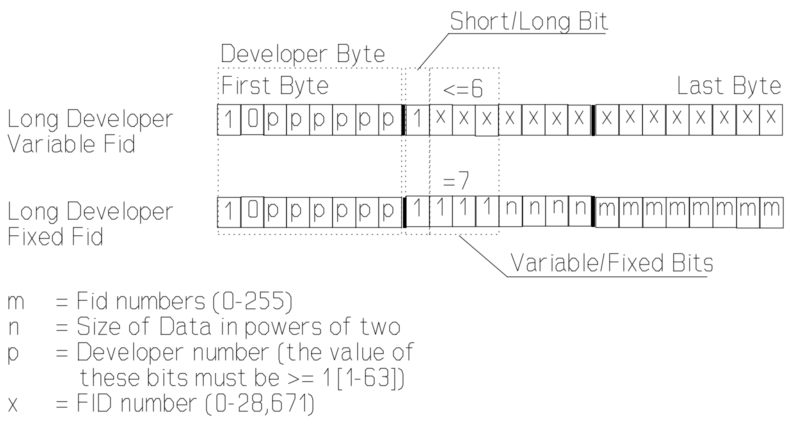

Short/Long Bit
Developer Byte
First Byte
<=6
Last Byte
Long Developer Variable Fid
1 0 p p p p p p 1 x x x x x x x x x x x x x x x
=7
Long Developer Fixed Fid
1 0 p p p p p p 1 1 1 1 n n n n m m m m m m m m
Variable/Fixed Bits
m = Fid numbers (0-255)
n = Size of Data in powers of two
p = Developer number (the value of these bits must be >= 1 [1-63])
x = FID number (0-28,671)

Figure 1-15. Long Developer FIDs

The following table shows the range of the long FIDs.

Note: See Figure 1-15 for a description of m, n, and p.

Long Developer FID Ranges

Type	Range
Variable Size	0xpp8000 - 0xppEFFF[12]
Fixed Size	0xppFnmm - 0xppFnmm[13]
	mm = 0x00 to 0xFF
	n = 0x0 to 0xF
	pp = 0x81 to 0xBF

[12] 28K variable-long FIDs per developer number (62 developer numbers).

[13] 256 fixed-size long FIDs per developer number (62 developer number). Each FID can have one of the following sizes: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, and 32K bytes.

Extended Developer FIDs. After the 62 developer FIDs are assigned, the following FIDs will be assigned to other developers. This group of FIDs, like the developer FIDs, only contain the short and long FIDs. The extended developer FID adds two bytes to the front of the small standard FID. The two highest bits of the high byte are set to 11 (see Figure 1-16).

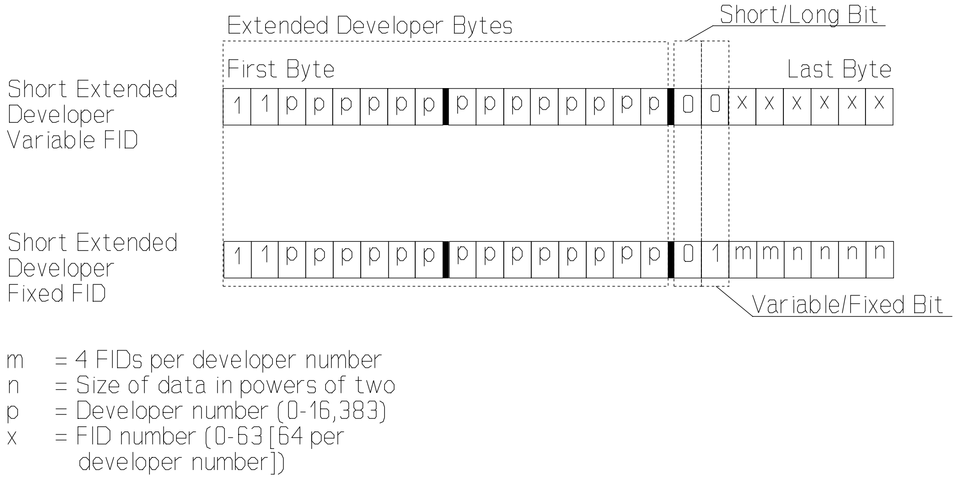

Extended Developer Bytes
Short/Long Bit
First Byte
Last Byte
Short Extended Developer Variable FID
1 1 p p p p p p p p p p p p p p 0 0 x x x x x x
Short Extended Developer Fixed FID
1 1 p p p p p p p p p p p p p p 0 1 m m n n n n
Variable/Fixed Bit
m = 4 FIDs per developer number
n = Size of data in powers of two
p = Developer number (0-16,383)
x = FID number (0-63 [64 per developer number])

Figure 1-16. Short Extended Developer FIDs

The following table shows the range for the short FIDs.

Note: See Figure 1-16 for a description of m, n, and p.

Short extended Developer FID Ranges	
Type	**Range**
Variable Size	0xpppp00 - 0xpppp3F[14]

Fixed Size	0xppppmn - 0xppppmn[15]
	m = 0x4 to 0x7
	n = 0x0 to 0xF
	pppp = 0xC000 to 0xFFFF

[14] 62 extended variable-size short FIDs per developer number (16K of developer numbers).

[15] 4 extended fixed-size short FIDs per developer number (16K of developer numbers). Each FID can have one of the 16 following sizes: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, and 32K bytes.

The long developer FID contains 4 bytes (see Figure 1-17).

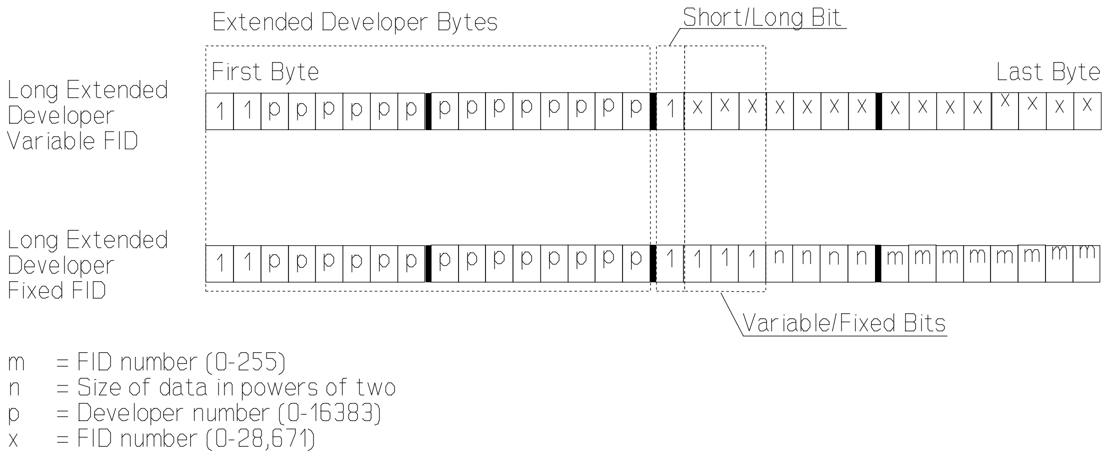

Extended Developer Bytes
Short/Long Bit
First Byte
Last Byte
Long Extended Developer Variable FID
1 1 p p p p p p p p p p p p p p 1 x x x x x x x x x x x x x x x
Long Extended Developer Fixed FID
1 1 p p p p p p p p p p p p p p 1 1 1 1 n n n n m m m m m m m m
Variable/Fixed Bits
m = FID number (0-255)
n = Size of data in powers of two
p = Developer number (0-16383)
x = FID number (0-28,671)

Figure 1-17. Long Extended Developer FIDs

The following table shows the range of the long extended developer FIDs.

Note: See Figure 1-17 for a description of m, n, and p.

Long Extended Developer FID Ranges

Type	Range
Variable Size	0xpppp8000 - 0xppppEFFF[16]
Fixed Size	0xppppFnmm - 0xppppFnmm[17] mm = 0x00 to 0xFF n = 0x0 to 0xF pppp = 0xC000 to 0xFFFF

NULL FID. All space within a session must be occupied by data or zeroed bytes. The blank space section allows you to pad the sectors with zeros. In cases where this section does not fit, you must fill the rest of the sector with zeros. In

[16] 28K extended variable-size long FIDs per developer number (16K of developer numbers).

[17] 256 extended fixed-size FIDs per developer number (16K of developer numbers). Each FID can have one of the following sizes: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, and 32K bytes.

either case, padding produces a field that begins with a zeroed byte or a NULL FID. The FID parsing functions must dump these bytes until it encounters a non-zeroed byte, in which case, the function begins parsing the next field.

Data Size

All variable data size FIDs in the standard, developer, and extended developer groups have an accompanying data size descriptor. The data size descriptor represents the data's size by using two formats that represent the data's size in bytes. A third format shows the data's size in bits and contains the data itself. The following paragraphs describe these size formats.

Size Format 1. The first format represents data sizes from 1 through 127 bytes. This format uses one byte with the seventh bit set to zero. The lower seven bits represent data sizes from 1 through 127 bytes. The format is shown below:

Format 1: 0nnnnnnn

Size Format 2. The second format shows how many size bytes follow the FID. This format sets the two highest bits to 1 and 0. The lower two bits (nn) represent the number of size bytes that follow (shown below):

Format 2: 10xxxxnn

Value of nn	Number of Following Size Bytes
00	1
01	2
10	4
11	8

Therefore, a variable data size field containing a large amount of data could be represented as:

<FID> 0x82 0xC0000000 <3,145 MB of data>

where FID identifies the FID as variable data size, 0x82 is a format two data size descriptor that has 4 data size bytes following it, and 0xC0000000 is the data's size.

Size Format 3. The third format represents bit data; it does not show the data's size. The seventh and sixth bits are set to 11, while the lower six bits contain bit data values from 0 to 0x3F. This format is shown below:

Format 3: 11bbbbbb

Size Summary. The following table shows a summary of the three data size formats:

Size of *Data*	*Data Size*
Format 1 0 - 127 bytes	0x00 - 0x7F
Format 2 128 - 255 bytes 256 - 65,535 bytes 65,536 - 4,294,967,295 bytes 4,294,967,296 - 18,4x10^{18} bytes	1 additional byte 0x8080 - 0x80FF 2 additional bytes 0x810100 - 0x81FFFF[18] 4 additional bytes 0x8200010000 - 0x82FFFFFFFF[13] 8 additional bytes 0x830000000100000000 - 0x83FFFFFFFFFFFFFFFF[13]
Bit data (Format 3) 1 - 6 bits	11bbbbbb

Byte Order

The FID and data size are a byte stream; however, the data is in little endian order (low-high byte order). To decipher the FID and data size, parse the first byte, then the second, and so on until the data's size is retrieved.

Field Example

This is a very simple example of how to design and parse a field. Appendix B, "SIDF Example" gives an in-depth parsing example.

[18] Intel low-high byte order

Designing the Field. To design a field containing a software's version do the following:

1. Consider all instances of the field. Since the version can contain numerical and character values, we design the data portion of the field to contain a string of characters. Also, we want to allow for future possibilities, so we select a variable size FID to contain a null-terminated string.

2. If the FID already existed, we would create the following entry into a section:

   <00001111> - 00000110 - VER 1

   where <00001111> is the FID's unique bit pattern (a small variable FID) that shows this is a software version field, 00000110 is the data's size (6 bytes), and "VER 1" is the data.

Parsing the Field. To parse this field:

3. We take the first byte and analyze its bit pattern (start at the seventh bit). The high bit tells us that we have a small standard FID, so we know that the sixth bit tells us if this is a variable or fixed size FID. Since it is 0, we know that it is a variable FID.

4. Next, because this is a small standard FID, we know that the lower six bits tell us what kind of data this field contains. We see that it is a software version field.

5. Next, because this is a variable field, we grab the next byte to see what kind of size type this field has. Again, we start with the seventh bit. Because it is 0, we know the size descriptor uses size format one (1) to suggest the data's size.

6. Because of this, we grab the lower seven bits to get the data's size. This size information tells us that there are six bytes of data.

Error Recovery

Hard and Soft Aborts

Hard aborts are caused by hardware failure, power failure, etc. and can be identified by the missing or incomplete session trailer, session or media index, expected data is missing, etc.

Resynchronization

SIDF provides a resynchronization field for times when an engine becomes lost while parsing formatted data. The first field in every section not only identifies the section, but also contains resynchronization data "0xA55A." If the engine becomes lost, it should look for this bit pattern and do the following steps after finding it:

1. Check the FID value that the resynchronization pattern belongs to, to see if the engine is where it should be (if possible).

2. Begin parsing the next few fields after the resynchronizing field.

If you are successful in parsing the next three or four fields, it is highly probable that your parsing mechanism has resynchronized with the data.

Direct Seeking

Direct seeking to a data set is possible under SIDF by recording the data set's name and media location to a log file during a backup session.

Information for the log can be obtained as follows:

1. SIDF specifies that before writing a transfer buffer to the media, its session physical sector address is recorded into the transfer buffer header.

2. The engine writing the transfer buffer could return the transfer buffer header to the engine requesting the write.

3. Since the engine requesting the write operation knows the offset to each record in the transfer buffer, it could store the transfer buffer's address, offset of each record, and the data set's name (contained in the record) into the log file.

During a restore session, an engine can use this log file to seek to the transfer buffer that contains the desired data set. After seeking to and reading the desired transfer buffer, the offset information is used to get the specified record.

Note: A database also can be used to provide a sophisticated solution than a log file.