Synchronet Message Base Specification
Version 1.10
Updated 03/28/94

PO Box 501
Yorba Linda, CA 92686

Voice: 714-529-6328 BBS: 714-529-9525 V.32/V.32bis
FAX: 714-529-9721 529-9547 V.FC
FIDO: 1:103/705 529-9721 ZyXEL

Table of Contents
==========

Introduction

Q. What is SMB?

A. SMB (Synchronet Message Base) is a technical specification for the st format of electronic mail messages. These e-mail messages may all be contained in one database, or, more commonly, separated into catagori databases. These message databases (or message bases) are also referr as 'sub-boards', 'forums', 'conferences,' and 'SIGs'. The messages ma directed to an individual person, sent to a group of individuals, or to everyone who can read messages in that message base. Messages may created and read soley at one physical location, or imported from and exported to a message network that may span continents. Message bases are connected to a message network are often called 'echoes'.

Q. Why SMB?

A. The Synchronet Message Base is designed to store high volumes of mess while maintaining optimum search, retrieval, and creation performance These messages are not defined as merely text. In addition to text, S defines the storage of digitized sound, MIDI, graphics, fonts, animat as well as other multimedia data and triggers for localized multimedi SMB thrives on a multi-user environment where messages are being crea read, modified, and deleted by multiple tasks simultaneously. With th large message networks of today being the rule, rather than the excep and high volumes of messages being imported on a daily, sometimes hou basis, creation and deletion speed is of the utmost importance. This where SMB really shines. Being extensible enough to handle message fo from networks of today and tomorrow, and fast enough to import more m that humanly readable, the SMB format will more than meet your messag storage needs.

Q. Why a specification?

A. Message bases are often accessed and modified by a number of differen programs. Often these programs are developed by individuals or compan other than the original designer of the message base format. This specification is an attempt to aid developers in creating programs th access or modify a message base stored in the SMB format.

Q. Who can use this specification?

A. Anyone that has interest in the Synchronet Message Base format at eit an educational or professional level. Specifically, software develope interested or currently involved in the development of message reader editors, echomail (toss/scan) programs, message transfer agents (MTAs network gateways, and bulletin board systems. Much of the information this specification is intended for those with preexisting programming knowledge, so those with little or no programming experience may find hard to comprehend.

Q. What does the SMB specification include?

A. The text you are reading is part of the SMB specification: a single t document that defines the storage format of each of the six files of SMB format message base and how they are related to each other.

Included with this specification is C source code to be used as an ex to programmers of how to access an SMB format message base and public library functions (SMBLIB) that can be compiled and linked into progr that access an SMB format message base developed by third parties. An utility program (SMBUTIL) is also included with C source code as an e of how to use the SMBLIB functions.

Q. Where did the SMB specification come from?

A. Digital Dynamics (southern California based software development comp released "Synchronet Multinode BBS Software Version 1a" in June of 19 one of the first BBS packages to be designed from the ground-up to op in a multinode environment with incredible speed and reliability, wit large suite of multinode specific features and design innovations.

The original message base format was designed with localized messagin low volume message networks in mind. By January of 1993, it was clear high volume message networks (FidoNet, RelayNet, Usenet, etc.) were t preference of most BBS users and a new message base format was requir allow for high volume message storage, improved storage, retrieval, a maintenance performance, as well as lower storage space requirements.

Rather than introduce another new message format, Digital Dynamics so to implement an existing public specification for a format that would current and future message storage needs. More than a few specificati were seriously considered at one time or another, but after careful examination, design flaws and lack of extensibility eliminated them f long term plans of Digital Dynamics and Synchronet BBS Software. Thus the design of the "Synchronet Message Base" (SMB) format.

At the request of many message related program developers, Digital Dy created and released the SMB specification before the release of "Syn Version 2.00" to allow lead-time on developing support programs for t format.

Digital Dynamics strongly encourages developers of message related pr (including software that directly competes with Synchronet or other D Dynamics products) to implement support for SMB. Though this is a pub specification and Digital Dynamics encourages developer suggestions, remain under the sole control of Digital Dynamics unless specifically otherwise in a future revision of this specification.

Digital Dynamics requests that any organizations that wish to adopt o ratify this specification, in part or whole, notify Digital Dynamics any of the contact methods listed at the beginning of this document.

Q. How does SMB store messages?

A. Each message base is stored in a set of six binary files. The base fi (maximum of eight characters) is the same for all six files of the sa message base and unique amoung the filenames of other message bases. files each have a different three character extension. The first char of the extension is always the letter 'S' (for SMB), while the second third characters define the contents of the file.

Two of the six files associated with each message base are not recrea and therefore are the most important when considering data integrity. two files are the data file (with a .SDT extension) and variable leng header file (.SHD extension). Both of these files use 256 byte blocks have associated block allocation tables (stored in .SDA and .SHA respectively) so that deleted message blocks may be used by new messa without creating odd sized unused 'holes' in the files. The block all table files (.SDA and .SHA) can be recreated with the information sto the header (.SHD) file.

For fast indexing, there is a small fixed length index file (with a . extension). This file allows for the immediate location of message he records based on sender's name or user number, recipient's name or us number, subject, message number, or message attributes. This file can recreated with the data stored in the header (.SHD) file.

The last file is an optional CRC history (.SCH) file. It contains 32- CRCs of a configurable number of messages imported or created locally is to help eliminate duplicate messages created by user or program er The CRC history file can be recreated with the combination of informa stored in the data (.SDT) and header (.SHD) files.

Q. How fast do messages import into an SMB message base?

A. This is a very important question for systems for that import large v of messages. Of course, the answer depends on the storage format whic are importing from, the average length of messages, the design of the program which is peforming the import process, as well as the hardwar system software being used. What's important is that SMB will allow t fastest import process possible with any given combination of the abo factors.

Since system storage capacity is rarely infinite, neither is the numb of messages which can be stored. System operators must define the max number of messages to be stored in a message base, the maximum age of messages in that message base, or a combination of both. Generally, t smaller the number of messages stored in a message base, the faster t import process. The SMB format is flexible enough to support multiple levels of import performance based on optimizations for storage space speed. Most system operators will almost invariably choose speed over space, but which choices are available is determined by the importing program.

Q. How much storage is required for an SMB message base?

A. The biggest factor in determining storage requirements for a message is the maximum number of messages to be stored in the base (defined b system operator) and the average size of each message. The minimum re storage for a message base is 32 bytes plus 535 bytes per message (53 per message if duplicate message checking is used).

The SMB format is designed to be "self-packing", meaning purged (dele message header and data blocks will be used automatically by new mess Relying solely on self-packing, an SMB format message base will never "shrink" in size. This is not to say that it will continually "grow" size, but that without specific packing procedures, deleted message b may remain unused for extended periods of time, meanwhile using some of storage space that could be freed using specific packing procedure

Limiting the maximum age of messages in an SMB message base is anothe to control the storage requirements. While maximum message age defini optional, the definition of the maximum number of messages is not.

Q. How many messages can be stored per SMB message base?

A. Without considering storage limitations or message data lengths great 256, the theoretical maximum number of messages that can be stored in single SMB message base is 16.7 million. Considering the variable len nature of message and header data, it is suggested that the system op allow no more than 1 million messages per base.

To determine an estimated maximum number of messages for a message ba using the average message data length as a factor, use the following formula:

4.2 billion divided by the average message length rounded up to be ev divisible by 256.

If the average message data length is 1500 bytes, the estimated maxim number of messages would be 2,734,375 (4.2 billion divided by 1536).

Implementation Levels

The SMB format can be implemented to varying degrees between programs wi creating compatibilty issues. Rather than have developers specifically s which features they have and have not implemented, we have defined five of implementation (represented by Roman numerals I through VII). For a p or software package to meet an implementation level, it must have all of features listed for that level and all of those for each level below it. minimum suggested imlementation is level I. The SMBUTIL program included this specification is an example of a level I implementation.

Level I

The minimum suggested level of implementation. Messages contain merely A text displayable on an ANSI terminal. Messages can be added to the messa base and if the maximum number of messages is exceeded, messages are rem or marked for deletion.

The SMBUTIL program included with this specification, is the perfect exa of level I implementation.

Level II

The addition of file attachments, multiple index/header entries per mess (multiple destinations), multiple text bodies for the separation of mess text and tag/origin lines (for example), forwarding, threading, and spec FidoNet kludge header field support makes this level of implementation m realistic for bulletin board system and EchoMail software implementation

Synchronet Multinode BBS Software v2.00 has a level II implementation of specification.

Level III

This implementation adds support for translation strings defined later i document for data compression, encryption, escaping, and encoding. This is still limited to basic ASCII text and ANSI escape sequence entry and retrieval.

Level IV

The storage and retrieval of embedded and attached images is added in th level of implementation. Supported images are limited to single binary o data blocks that can be displayed or transferred to the user (automatica or by request) if their display and translation protocols define specifi support for the image type.

Level V

This level of implementation adds support for embedded and attached soun This includes digitized sound and MIDI data. Supported sounds are limite single binary or text data blocks that can be played or transferred to t (automatically or by request) if their presentation and translation prot define specific support for the sound type.

Level VI

Localized sound and image data can be triggered by messages stored and retrieved in an implementation of this level.

Level VII

Complete multimedia support is reached in this implementation level with support for embedded and attached animation, sound, and video data.

Definitions

Control Characters

When specifying control characters (ASCII 1 through 31), the caret symbo or the abreviation "ctrl-" followed by a character will be used to indic value. ^A is equivalent to ASCII 1, ^B ASCII 2, etc. The case of the con character is not significant (i.e. ^z and ^Z are equivalent). The contro character ^@ (ASCII 0) will be specified as NULL or 0.

Hexidecimal

Base sixteen numbering system which includes the digits 0-9 and A-F. Hexidecimal numbers are represented in this document with a prefix of "0 "\x" or a suffix of "h". Hexidecimal letter digits are not case sensitiv (i.e. the number 0xff is the same as 0xFF).

File dump

When example file dumps are displayed, the format is similar to that of output from the DOS DEBUG program. With the exception of the ASCII chara all numbers are in hexidecimal.

```
Offset    Byte values                                          ASCII cha

000000    53 4D 42 1A 10 01 20 00   F4 01 00 00 F4 01 00 00    SMB... .⌠
000010    20 00 00 00 D0 07 00 00   D0 07 00 00 00 00 00 00     ...╨...╨
```

Bit values

Bit (or flag) values are represented in C notation as (1<<x) where x is number. (i.e. bit number 7 (1<<7) is the same as 0x80).

Word storage

All words (16-bit) and double words (32-bit) are stored in Intel 80x86 (endian) format with bytes stored from low to high (reverse of the Motoro 680x0 word storage format).

A 16-bit word with the value 1234h is stored as 34h 12h.

Translation strings

Translation strings (xlat variables) are arrays of words (16-bit) in the
of the original storage translation. The last translation type is follow
16-bit zero (defined later as XLAT_NONE). If there are no translations,
the first and only element of the array is XLAT_NONE.

When translating data upon retrieval, the translation order must be reve
to obtain the proper data.

Acronyms:
========

ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
BBS	Bulletin Board System
C	The C programming language as defined by ANSI X3.159-198
CR	Carriage Return character (ASCII 13)
CRC	Cyclic Redundancy Check
CRC-16	Standard 16-bit CRC using 1021h polynomial
CRC-32	Standard 32-bit CRC using EDB88320h polynomial
CRLF	Carriage Return character followed by a Line Feed charac
FSC	FidoNet Standards Commitee (FTS proposal)
FTN	FidoNet Technology Network
FTS	FidoNet Technical Standard
LF	Line Feed character (ASCII 10)
QWK	Compressed message packet format for message reading/net
RFC	Request for Comments
SMB	Synchronet Message Base
UT	Universal Time (formerly called "Greenwhich Mean Time")

Data types

uchar Unsigned 8-bit value (0 through 255).
C example:

```
#define uchar unsigned char
```

short Signed 16-bit value (-32768 through 32767).
"short" is a C keyword indicating "short int".

ushort Unsigned 16-bit value (0 through 65535).
C example:

```
#define ushort unsigned short
```

ulong Unsigned 32-bit value (0 through 4294967295).
C example:

```
#define ulong unsigned long
```

time_t Unsigned 32-bit value.
Seconds since 00:00 Jan 01 1970 (Unix format).
Used for all time/date storage in SMB as part of the whe
data type. This time format will support dates through t
2105.
time_t is defined by ANSI C as a long (signed) which can
limit its date support to the year 2038 depending on the
library routines used.

ASCII String (aka character array) of 8-bit ASCII characters.
Characters with the bit 7 set (80h through FFh) represen
the IBM PC extended ASCII character set. When data or he
fields of this type are stored in the header, a NULL
terminator may or may not be present.
C example:

```
uchar str[80];
```

ASCIIZ ASCII string with (non-optional) NULL terminator.
C example:

```
uchar str[81];
```

nulstr ASCII string immediately terminated by NULL.
C example:

```
uchar *nulstr="";
```

undef Data buffer with undefined contents.
C example:

```
uchar buf[BUF_LEN];
```

when_t Date/Time stamp including time-zone adjustment informati
C example:

```
typedef struct {

    time_t  time;   // Time stamp (in local time)
    short   zone;   // Zone constant or Minutes (+/-) fr

    } when_t;
```

time:

A time value of 0 is invalid and indicates an uninitiali time stamp.

Time stamps are always stored in local time. i.e. Regard of what time zone, Jan 1st 1994 00:00 will always be sto as 2D250350h.

zone:

If the zone is the range -720 to +720, it represents the of minutes east or west of UT. Values in this range shou be used for time zones not otherwise represented here.

If the zone is greater than 720 or less than -720, then following bits have special meaning:

```
(1<<12)         // Non-US time zone     (east of UT)
(1<<13)         // Non-US time zone     (west of UT)
(1<<14)         // U.S. time zone
(1<<15)         // Daylight savings
```

The lower 12 bits (0 through 11) contain the number of m east or west of UT (not accounting for daylight savings)

If the time zone is one specified in the U.S. Uniform Ti
the following values represent the zone:

```
AST 0x40F0      // Atlantic             (-04:00)
EST 0x412C      // Eastern              (-05:00)
CST 0x4168      // Central              (-06:00)
MST 0x41A4      // Mountain             (-07:00)
PST 0x41E0      // Pacific              (-08:00)
YST 0x421C      // Yukon                (-09:00)
HST 0x4258      // Hawaii/Alaska        (-10:00)
BST 0x4294      // Bering               (-11:00)
```

With bit 15 set, the following values represent the same
with the presence of daylight savings:

```
ADT 0xC0F0      // Atlantic             (-03:00)
EDT 0xC12C      // Eastern              (-04:00)
CDT 0xC168      // Central              (-05:00)
MDT 0xC1A4      // Mountain             (-06:00)
PDT 0xC1E0      // Pacific              (-07:00)
YDT 0xC21C      // Yukon                (-08:00)
HDT 0xC258      // Hawaii/Alaska        (-09:00)
BDT 0xC294      // Bering               (-10:00)
```

The following non-standard time zone specifications may
used:

```
MID 0x2294      // Midway               (-11:00)
VAN 0x21E0      // Vancouver            (-08:00)
EDM 0x21A4      // Edmonton             (-07:00)
WIN 0x2168      // Winnipeg             (-06:00)
BOG 0x212C      // Bogota               (-05:00)
CAR 0x20F0      // Caracas              (-04:00)
RIO 0x20B4      // Rio de Janeiro       (-03:00)
FER 0x2078      // Fernando de Noronha  (-02:00)
AZO 0x203C      // Azores               (-01:00)
LON 0x1000      // London               (+00:00)
BER 0x103C      // Berlin               (+01:00)
ATH 0x1078      // Athens               (+02:00)
MOS 0x10B4      // Moscow               (+03:00)
DUB 0x10F0      // Dubai                (+04:00)
KAB 0x110E      // Kabul                (+04:30)
KAR 0x112C      // Karachi              (+05:00)
BOM 0x114A      // Bombay               (+05:30)
KAT 0x1159      // Kathmandu            (+05:45)
DHA 0x1168      // Dhaka                (+06:00)
BAN 0x11A4      // Bangkok              (+07:00)
HON 0x11E0      // Hong Kong            (+08:00)
TOK 0x121C      // Tokyo                (+09:00)
SYD 0x1258      // Sydney               (+10:00)
NOU 0x1294      // Noumea               (+11:00)
WEL 0x12D0      // Wellington           (+12:00)
```

fidoaddr_t FidoNet address stored as four ushorts that represent th network, node, and point (in that order).
C example:

```
typedef struct {

    ushort zone,
           net,
           node,
           point;

    } fidoaddr_t;
```

typestr_t ASCIIZ string with ushort type prefix.
C example:

```
typedef struct {

    ushort  type;   // Specifier for type of 'str'
    uchar   str[];  // ASCIIZ filename or other string d

    } typestr_t;
```

mattach_t File attachment information with type prefix, translatio string, and filename.
C example:

```
typedef struct {

    ushort  type;   // Attachment type
    ushort  xlat[]; // Translations of data in attachmen
    uchar   str[];  // ASCIIZ filename

    } mattach_t;
```

vattach_t Video file attachment information with type, compression translation string, and filename.
C example:

```
typedef struct {

    ushort  type;   // Attachment type
    ushort  comp;   // Compression method
    ushort  xlat[]; // Translations of data in attachmen
    uchar   str[];  // ASCIIZ filename

    } vattach_t;
```

mtext_t — Message text with translation string prefix.
C example:

```
typedef struct {

    ushort  xlat[]; // Translations of text
    uchar   text[]; // Actual text data

    } mtext_t;
```

ftext_t — Formatted message text with translation string prefix an format type.
C example:

```
typedef struct {

    ushort  type;   // See Image Types for valid types
    ushort  xlat[]; // Translations of data
    uchar   data[]; // Actual formatted text data

    } ftext_t;
```

membed_t — Embedded data with type prefix, translation string, and description.
C example:

```
typedef struct {

    ushort  type;   // Specifier for type of 'dat'
    ushort  xlat[]; // Translations of embedded data
    uchar   name[]; // ASCIIZ char description of embedd
    uchar   dat[];  // Binary data

    } membed_t;
```

vembed_t — Embedded video data with type, compression method, trans string, and ASCIIZ description.
C example:

```
typedef struct {

    ushort  type;   // Specifier for type of 'dat'
    ushort  comp;   // Compression method
    ushort  xlat[]; // Translations of embedded data
    uchar   name[]; // ASCIIZ char description of embedd
    uchar   dat[];  // Binary data

    } vembed_t;
```

File formats

Index File (*.SID)

The index file for each message base contains one record per message in base. Each record is fixed length using the following format:

Index Record:

C example:

```
typedef struct {

    ushort  to;             // 16-bit CRC of recipient name (lower case)
    ushort  from;           // 16-bit CRC of sender name (lower case) or
    ushort  subj;           // 16-bit CRC of title/subject (lower case)
    ushort  attr;           // attributes (read, permanent, etc. flags)
    ulong   offset;         // offset into header file
    ulong   number;         // message number
    time_t  time;           // import date/time stamp (Unix format)

    } idxrec_t;
```

Example file dump (16 messages starting with message number 15):

```
000000    36 4F 13 07 2A 77 00 00   20 00 00 00 0F 00 00 00    6O..*w..
000010    BE 62 76 2C 36 4F 46 0A   7F B2 00 00 20 01 00 00    ╛bv,6OF.⌂
000020    10 00 00 00 C7 29 78 2C   36 4F 70 6F 46 FF 00 00    ....╟)x,6
000030    20 02 00 00 11 00 00 00   AD D3 7A 2C 70 6F 13 07     .......¡
000040    46 FF 00 00 20 03 00 00   12 00 00 00 D6 F8 7F 2C    F .. ....
000050    36 4F E1 EA E7 E9 00 00   20 04 00 00 13 00 00 00    6Oß τΘ..
000060    1E 7B 85 2C 37 0D 2E DF   4D 79 00 00 20 05 00 00    .{à,7..▀M
000070    14 00 00 00 5C E1 A1 2C   90 54 2D 5A 86 62 00 00    ....\ßí,É
000080    20 06 00 00 15 00 00 00   39 2E A2 2C 70 6F 1A 8B     .......9
000090    46 FF 00 00 20 07 00 00   16 00 00 00 D0 7B A8 2C    F .. ....
0000A0    2E DF 1A 8B 4D 79 00 00   20 08 00 00 17 00 00 00    .▀.ïMy..
0000B0    FF 7B A8 2C B4 D9 35 7C   23 B1 00 00 20 09 00 00     {¿,┤┘5│#
0000C0    18 00 00 00 CE D4 BA 2C   36 4F BC D8 B2 E7 00 00    ....╬╘║,6
0000D0    20 0A 00 00 19 00 00 00   14 5F C3 2C BA A8 4E B0     ........
0000E0    67 76 00 00 20 0B 00 00   1A 00 00 00 6F 89 C3 2C    gv.. ....
0000F0    36 4F 0C 01 19 9C 00 00   20 0C 00 00 1B 00 00 00    6O...£..
000100    F8 30 C6 2C 36 4F FA 48   0E 55 00 00 20 0D 00 00    °0╞,6O·H.
000110    1C 00 00 00 6A 94 D3 2C   36 4F F1 CE CF A2 00 00    ....jö╙,6
000120    20 0E 00 00 1D 00 00 00   53 DB D5 2C 8D A6 21 CE     .......S
000130    F7 AB 00 00 20 0F 00 00   1E 00 00 00 31 29 DC 2C    ≈½.. ....
```

Field descriptions:

To:
The 'To' field is the CRC-16 of the name of the intended recipient agent this message or the intended recipient's user number. If the CRC is stor text must be converted to lower case (A-Z changed to a-z) before the CRC calculated. If the message is forwarded to another agent, the original o index record must be changed to contain the CRC-16 of the new recipient user number.

From:
This field, similar to the 'To' field, contains the CRC-16 of the name o the sending agent of this message or the sender's user number. If the CR is stored, the text be converted to lower case (A-Z changed to a-z) befo CRC is calculated. If the message is forwarded to another agent, the ori or new index record must be changed to contain the CRC-16 of the new sen name or user number.

Subj:
The 'Subj' field contains the CRC-16 of the message's subject. The subje must be converted to lower case (A-Z changed to a-z) and all preceeding "re: "'s and "re:"'s removed before calculating the CRC-16.

Attr:
This field is a ushort bit-map of the specific attributes for this messa It is a clone of the 'attr' element of the smbhdr_t structure.

Offset:
This ulong is the offset (in bytes) in the header file for this message' header record.

Number:
This ulong is the serial number of this message. Valid values are 1 thro 0xffffffff. No two index records in the same message base may have the s message number.

Time:
This field is the date/time stamp the message was imported to or posted the message base. It is a clone of the 'when_imported.time' element of t smbhdr_t structure.

Header File (*.SHD)

Each SMB header file is made up of two distinct sections: base header re and message header records.

Base Header Records:

Base header records are blocks of data that apply to the entire message and are of variable length. This specification defines only one base hea record, the "Status info" (smbstatus_t) record. This status info record the first base header record in the file and must be modified if additio base header records are added.

Additional header records allow other developers to store configuration status information particular to their application needs. It also allows future header record definitions as part of this specification without c backward compatibility issues.

Each base header record contains a fixed length portion (smbhdr_t) and a optional variable length portion.

Whenever a base header record is read or updated (written), it must firs be successfully locked and subsequently unlocked.

Message Header Records:

Following the last base header record is the first message header record header record is stored in one or more 256 byte blocks. There must be ex one active message header record for every index record in the index fil (Note: This does not include deleted message headers that have not been overwritten by a new message header).

Each message header record contains a fixed length portion (msghdr_t), a of zero or more fixed length data fields (dfield_t), and a list of three more variable length header fields (hfield_t).

The value of the data stored in the zero or more unused bytes of the las header record block have an undefined value, though whenever possible developers should initialize to binary zero for human readability.

Whenever a message header record is read or updated (written), it must f be successfully locked and subsequently unlocked.

Base Header Record (Fixed Portion):

C example:

```
typedef struct {

    uchar   id[4];          // text or binary unique hdr ID
    ushort  version;        // version number (initially 100h for 1.00)
    ushort  length;         // length including this struct

    } smbhdr_t;
```

Base Header #1 (Status info) Record (Variable Portion):

C example:

```
typedef struct {

    ulong   last_msg;       // last message number
    ulong   total_msgs;     // total messages
    ulong   header_offset;  // byte offset to first header record
    ulong   max_crcs;       // Maximum number of CRCs to keep in history
    ulong   max_msgs;       // Maximum number of messages to keep in bas
    ushort  max_age;        // Maximum age of messages (days) to keep in
    ushort  reserved;       // Reserved for future use

    } smbstatus_t;
```

Base Header #1 (Status info) Record Contents:

```
smbhdr.id="SMB\x1a";        // SMB^Z
smbhdr.version=0x110;       // v1.10
smbhdr.length=sizeof(smbhdr_t)+sizeof(smbstatus_t);
smbstatus_t status;
```

Additional Base Headers:

Additional headers from developers must have initial 8 bytes in smbhdr_t format, length must include size of smbhdr_t, and header_offset of smbst must be changed to include the size of the additional header(s).

Example file dump (base header portion only):

```
000000    53 4D 42 1A 10 01 20 00   F4 01 00 00 F4 01 00 00    SMB... .⌠
000010    20 00 00 00 D0 07 00 00   D0 07 00 00 00 00 00 00     ...╨...╨
```

Message Header Record (Fixed portion):

C example:

```
typedef struct {

    uchar   id[4];          // SHD^Z (same for all types and versions)
    ushort  type;           // Message type (this is the definition of t
    ushort  version;        // Version of type (initially 100h for 1.00)
    ushort  length;         // Total length of fixed portion + all field
    ushort  attr;           // Attributes (bit field) (duplicated in SID
    ulong   auxattr;        // Auxillary attributes (bit field)
    ulong   netattr;        // Network attributes (bit field)
    when_t  when_written;   // Date/Time message was originally created
    when_t  when_imported;  // Date/Time message was imported (locally)
    ulong   number;         // Message number (unique, not necessarily s
    ulong   thread_orig;    // Original message number in thread
    ulong   thread_next;    // Next message in thread
    ulong   thread_first;   // Number of first reply to this message
    uchar   reserved[16];   // 16 reserved bytes for future use
    ulong   offset;         // Offset for buffer into data file (0 or mo
    ushort  total_dfields;  // Total number of data fields

    } msghdr_t;

typedef struct {

    ushort  type;           // See "Data Field Types" values
    ulong   offset;         // Offset into buffer
    ulong   length;         // Length of data field in buffer

    } dfield_t;

typedef struct {

    ushort  type;           // See "Header Field Types" for values
    ushort  length;         // Length of buffer
    uchar   dat[length];

    } hfield_t;
```

Example file dump (one header record, both fixed and variable length por

```
---------------------------------------------------------------------------
000020    53 48 44 1A 00 00 10 01   F5 00 00 00 00 00 00 00    SHD.....⌡
000030    00 00 00 00 46 DB F7 2C   00 00 7D D7 29 2D 00 00    ....F▀≈,..
000040    01 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ........
000050    00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00    ........
000060    00 00 00 00 02 00 00 00   00 00 00 00 4A 01 00 00    ........
000070    02 00 4A 01 00 00 53 00   00 00 00 00 13 00 4D 61    ..J...S..
000080    72 69 61 6E 6E 65 20 4D   6F 6E 74 67 6F 6D 65 72    rianne Mo
000090    79 30 00 0C 00 43 61 72   6F 6C 20 47 61 69 73 65    y0...Caro
0000A0    72 60 00 07 00 46 61 72   6E 68 61 6D A4 00 14 00    r`...Farn
0000B0    31 3A 31 33 38 2F 31 30   32 2E 30 20 32 63 66 38    1:138/102
0000C0    30 35 37 36 A5 00 14 00   31 3A 33 34 33 2F 31 30    0576Ñ...1
0000D0    30 2E 30 20 32 63 66 33   62 39 30 61 A3 00 23 00    0.0 2cf3b
0000E0    31 33 38 2F 31 30 32 20   31 20 32 37 30 2F 31 30    138/102 1
0000F0    31 20 32 30 39 2F 32 30   39 20 31 30 33 2F 30 20    1 209/209
000100    33 35 35 02 00 02 00 02   00 03 00 08 00 01 00 8A    355......
000110    00 66 00 00 00 00 00 00   00 00 00 00 00 00 00 00    .f.......
```

Contents of example header:

```
--------------------------
id                      SHD^Z
type                    0000h
version                 0110h
length                  245
attr                    0000h
auxattr                 00000000h
netattr                 00000000h
when_written            Sat Nov 27 17:57:10 1993
when_imported           Tue Jan 04 15:54:21 1994
number                  1
thread_orig             0
thread_next             0
thread_first            0
reserved[16]
offset                  0
total_dfields           2

dfield[0].type          00h
dfield[0].offset        0
dfield[0].length        330
dfield[1].type          02h
dfield[1].offset        330
dfield[1].length        83

hfield[0].type          00h
hfield[0].length        19
hfield[0]_dat           Marianne Montgomery
hfield[1].type          30h
hfield[1].length        12
hfield[1]_dat           Carol Gaiser
hfield[2].type          60h
hfield[2].length        7
hfield[2]_dat           Farnham
hfield[3].type          A4h
hfield[3].length        20
hfield[3]_dat           1:138/102.0 2cf80576
hfield[4].type          A5h
hfield[4].length        20
hfield[4]_dat           1:343/100.0 2cf3b90a
hfield[5].type          A3h
hfield[5].length        35
hfield[5]_dat           138/102 1 270/101 209/209 103/0 355
hfield[6].type          02h
hfield[6].length        2
hfield[6]_dat           02 00
hfield[7].type          03h
hfield[7].length        8
hfield[7]_dat           01 00 8A 00 66 00 00 00
```

Fixed Portion Field descriptions:

Id:
This field (regardless of the header type or version) must always contai
the string "SHD^Z". This is to aid in the restoration of a corrupted hea
file and give a visual indication of the beginning of a new header recor
viewing hex/ASCII dumps of the header file.

Type:
This is the message header type. Only one type is currently defined by t
specification (type 0). Any and all future header types will have the fi
4 fields (10 bytes) in the same format of type 0. This allows other type
(with different lengths) to be skipped because the 4th field (length) wi
always be in the same position.

Version:
This is the version of this header type. This specification defines vers
1.10 of message header type 0 (stored as 110h).

Length:
This is the total length of this message header record (including both f
and variable length portions, but NOT including unused block space).

Attr:
This is a bit field (16-bit) containing basic message attributes (flags)
this message. An exact duplicate of this field is stored in the index fi
well. They must always match.

Auxattr:
This is a bit field (32-bit) containing the auxillary attributes (flags)
this message. The attributes stored in this variable are more specific i
nature and less critical than those in the Attr field.

Netattr:
This is a bit field (32-bit) containing the network attributes (flags) f
message. The attributes stored in this variable are related solely to me
networking.

When_written:
This is the date and time when the message was originally created.

When_imported:
This is the date and time when the message was posted on or imported int
local message system.

Number:
This is the message's unique serial number (from 1 to FFFFFFFFh). This f
is duplicated in the index file. They must always match.

Thread_orig:
If this message is a reply, then this field contains the number of the o message that was replied to. If this message was not a reply, this field contain the value 0.

Thread_next:
If this message is a reply, and there are later replies to that message (the message number contained in the Thread_orig field), then this field contain the number of the next reply in the chain. If this message is th reply to the orignal message, this field will contain the value 0.

Thread_first:
If there are any replies to this message (after it has been posted), thi will contain the number of the first reply to this message. If there are replies to this message, this field will contain the value 0.

Reserved:
Unused bytes, reserved for future definition in the message header type specification.

Offset:
The byte offset into the data file, specifying the start of the buffer f all data associated with this message. This value must be either 0 or mo 256. When retrieving the actual data portion of data fields, the physica offset into the file will be the offset of the message data buffer (this plus the offset of the individual data field (msghdr_t.offset+dfield_t.o

Total_dfields:
This field contains the total number of data fields associated with this message. The value of this field must match the actual number of data fi stored in the header (dfield_t data types following the fixed portion of message header).

Variable Portion Field descriptions:

See the Header Field Type and Data Field Type sections for the descripti of the values contained in these fields.

Message Header Block Allocation (*.SHA)

This file contains no header or signature data. Each byte (uchar) in the specifies the allocation state of the corresponding 256 byte block in th header (*.SHD) file. A value of 0 indicates a free header block, and a v 1 indicates an allocated block. Other non-zero values are undefined.

This file must always be opened DENY ALL (non-shareable).

Message Data (*.SDT)

This file contains no header or signature data. It contains the text and embedded data for the messages in a single message base. The data for ea message always begins on a 256 byte block boundary. The data in the unus portion of a data block is undefined, but should be initialized to NULL whenever possible.

This file must always be opened DENY NONE (shareable).

Data fields of type TEXT_BODY and TEXT_TAIL must have all trailing white and control characters removed (i.e. the last character of the data reco must be in the range 21h to FFh). The only exception to this rule, is if TEXT_BODY is terminated with multiple contiguous CRLFs, only the last CR should be removed. A CRLF should always be appended to the text data whe displayed.

Message Data Block Allocation (*.SDA)

This file contains no header or signature data. Each word (ushort) in th
specifies the allocation state of the corresponding 256 byte block in th
(*.SDT) file. A value of 0 indicates a free block, and a non-zero value
indicates the number of message header records associated with this mess
data (most often 1). Each block can be used by up to 65,535 header recor

This file must always be opened DENY ALL (non-shareable).

CRC history for duplicate message checking (*.SCH)

This file is optional and contains no header or signature data. Each lon (ulong) in the file contains a CRC-32 of previously posted/imported mess These CRCs can be used to check a candidate message for posting/import t sure the message isn't a duplicate created by human or program error. Th maximum number of CRCs to store is defined in the first message base hea record (smbstatus_t.max_crcs).

This file must always be opened DENY ALL (non-shareable).

Header Field Types:
==========

These are the defined valid values for hfield_t.type:

Name   : SENDER
Value   : 00h
Data   : ASCII
Multiple : Yes, order significant
Required : Yes
Summary : Name of agent that sent this message

If blank (0 length or nulstr), assumed "Anonymous". If multiple SENDER f
exist, then the message has been forwarded and the order of the fields i
record must match the forwarding order (chronologically). When forwardin
message, the original SENDER field should be left intact and new SENDER,
FORWARDED, and RECIPIENT fields added to the end of the record.

Name   : SENDERAGENT
Value   : 01h
Data   : ushort
Multiple : Yes, order significant
Required : No
Default : AGENT_PERSON or previous SENDERAGENT if exists
Summary : Type of agent that sent this message

If multiple SENDER fields exist, then the message has been forwarded. If
forwarding agents is of a type other than AGENT_PERSON, then this field
follow that SENDER field to specify the agent type.

Name   : SENDERNETTYPE
Value   : 02h
Data   : ushort
Multiple : Yes, order significant
Required : No
Default : NET_NONE or previous SENDERNETTYPE if exists
Summary : Type of network message was sent from

If multiple SENDERNETADDR fields are included, a SENDERNETTYPE field sho
included before each to determine what data type the address is stored i

Name   : SENDERNETADDR
Value   : 03h
Data   : undef
Multiple : Yes, order significant
Required : No
Default : Previous SENDERNETADDR if exists
Summary : Network address for agent that sent this message

The SENDERNETTYPE field indicates the data type of this field. If the
SENDERNETTYPE is of type NET_INTERNET, the local-part of the Internet
address is optional. If the local-part separator character ('@') is omit
the SENDER field is assumed to be the local-part of the address.

Name   : SENDEREXT
Value   : 04h
Data   : ASCII
Multiple : Yes, order significant
Required : No
Default : Previous SENDEREXT if exists
Summary : Extension of sending agent

This field is useful for storing the sending agent's extension, when the agent's extension binds more tightly than the agent's name.

For example, Synchronet Multinode BBS Software stores local e-mail with sending and receiving agent's user numbers stored as their respective extensions. This is done so that if a user name changes for some reason, messages will not "disappear" from the users's mail box. In Synchronet l e-mail, user numbers bind more tightly than user names.

If the SENDEREXT field is specified, then the "From" field in the index contain the CRC-16 of this field rather than the SENDER (name) field.

Name   : SENDERPOS
Value   : 05h
Data   : ASCII
Multiple : Yes, order significant
Required : No
Default : Previous SENDERPOS if exists
Summary : Position of sending agent

Primarily for documentary purposes, this field contains the position of sending agent (i.e. President, Sysop, C.E.O., MIS Director, etc).

It can also be useful for getting a message or reply to the intended recipient when the agent name is not located or is unknown, but the posi of the agent is known and specified.

Name   : SENDERORG
Value   : 06h
Data   : ASCII
Multiple : Yes, order significant
Required : No
Default : Previous SENDERORG if exists
Summary : Organization name of sending agent

Primarily for documentary purposes, this field contains the organization which the sending agent belongs (i.e. Microsoft, Joe's BBS, SoCal User's etc).

Name   : AUTHOR
Value   : 10h
Data   : ASCII
Multiple : Yes
Required : No
Default : First SENDER
Summary : Name of agent that created this message

This field can only be added by the process that originally creates the message. It should not be included if same as first SENDER field. If mul AUTHOR fields exist, then the message was created by multiple agents and considered valid. The order of multiple AUTHOR fields in the record is n significant.

Name   : AUTHORAGENT
Value   : 11h
Data   : ushort
Multiple : Yes, order significant
Required : No
Default : SENDERAGENT or previous AUTHORAGENT if exists
Summary : Type of agent that created this message

This field can only be added by the process that originally creates the message. It should not be included if same as first SENDERAGENT field. I multiple AUTHOR fields exist, then the message was created by multiple a and if the agent type for any of the authors is other than AGENT_PERSON, AUTHORAGENT field must follow to specify the agent type.

Name   : AUTHORNETTYPE
Value   : 12h
Data   : ushort
Multiple : Yes, order significant
Required : No
Default : SENDERNETTYPE or previous AUTHORNETTYPE if exists
Summary : Type of network this author is member of

Name   : AUTHORNETADDR
Value   : 13h
Data   : undef
Multiple : Yes, order significant
Required : No
Default : SENDERNETADDR or previous AUTHORNETADDR if exists
Summary : Network address of this author

```
Name     : AUTHOREXT
Value    : 14h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : SENDEREXT or previous AUTHOREXT if exists
Summary  : Extension of this author

Name     : AUTHORPOS
Value    : 15h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : SENDERPOS or previous AUTHORPOS if exists
Summary  : Position of this author

Name     : AUTHORORG
Value    : 16h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : SENDERORG or previous AUTHORORG if exists
Summary  : Organization this author belongs to
```

```
Name     : REPLYTO
Value    : 20h
Data     : ASCII
Multiple : Yes, but only last is valid
Required : No
Default  : SENDER
Summary  : Name of agent that replies should go to

Name     : REPLYTOAGENT
Value    : 21h
Data     : ushort
Multiple : Yes, but only last is valid
Required : No
Default  : SENDERAGENT
Summary  : Type of agent that replies should go to

Name     : REPLYTONETTYPE
Value    : 22h
Data     : ushort
Multiple : Yes, but only last is valid
Required : No
Default  : SENDERNETTYPE
Summary  : Type of network that replies should go to

Name     : REPLYTONETADDR
Value    : 23h
Data     : undef
Multiple : Yes, but only last is valid
Required : No
Default  : SENDERNETADDR
Summary  : Network address that replies should go to
```

```
Name     : REPLYTOEXT
Value    : 24h
Data     : ASCII
Multiple : Yes, but only last is valid
Required : No
Default  : SENDEREXT
Summary  : Extension of agent that replies should go to

Name     : REPLYTOPOS
Value    : 25h
Data     : ASCII
Multiple : Yes, but only last is valid
Required : No
Default  : SENDERPOS
Summary  : Position of agent that replies should go to

Name     : REPLYTOORG
Value    : 26h
Data     : ASCII
Multiple : Yes, but only last is valid
Required : No
Default  : SENDERORG
Summary  : Organization of agent that replies should go to
```

```
Name     : RECIPIENT
Value    : 30h
Data     : ASCII
Multiple : Yes, order significant
Required : Yes
Default  : "All"
Summary  : Name of agent to receive this message
```

If multiple RECIPIENT fields exist, the message has been forwarded and f additional RECIPIENT field (after the initial RECIPIENT), there should b FORWARDED field. The order of the RECIPIENT fields in the record must ma order in which the message was sent and forwarded (chronologically).

```
Name     : RECIPIENTAGENT
Value    : 31h
Data     : ushort
Multiple : Yes, order significant
Required : No
Default  : AGENT_PERSON or previous RECIPIENTAGENT if exists
Summary  : Type of agent to receive this message
```

If multiple RECIPIENT fields exist, the message has been forwarded. If a the recipient agents are of a type other than AGENT_PERSON, this field m follow the RECIPIENT field to specify the agent type.

```
Name     : RECIPIENTNETTYPE
Value    : 32h
Data     : ushort
Multiple : Yes, order significant
Required : No
Default  : NET_NONE or previous RECIPIENTNETTYPE if exists
Summary  : Type of network to receive this message
```

```
Name     : RECIPIENTNETADDR
Value    : 33h
Data     : undef
Multiple : Yes, order significant
Required : No
Default  : Previous RECIPIENTNETADDR if exists
Summary  : Address of network to receive this message
```

```
Name     : RECIPIENTEXT
Value    : 34h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : Previous RECIPIENTEXT if exists
Summary  : Extension of agent to receive this message
```

If the RECIPIENTEXT field is specified, then the "To" field in the index contain the CRC-16 of this field rather than the RECIPIENT (name) field.

```
Name     : RECIPIENTPOS
Value    : 35h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : Previous RECIPIENTPOS if exists
Summary  : Position of agent to receive this message

Name     : RECIPIENTORG
Value    : 36h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : Previous RECIPIENTORG if exists
Summary  : Type of agent to receive this message
```

```
Name     : FORWARDTO
Value    : 40h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Summary  : Name of agent this message is to be forwarded to

Name     : FORWARDTOAGENT
Value    : 41h
Data     : ushort
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTAGENT or previous FORWARDTOAGENT if exists
Summary  : Type of agent this message is to be forwarded to

Name     : FORWARDTONETTYPE
Value    : 42h
Data     : ushort
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTNETTYPE or previous FORWARDTONETTYPE if exists
Summary  : Type of network this message is to be forwarded to

Name     : FORWARDTONETADDR
Value    : 43h
Data     : undef
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTNETADDR or previous FORWARDTONETADDR if exists
Summary  : Network address this message is to be forwarded to
```

```
Name     : FORWARDTOEXT
Value    : 44h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTEXT or previous FORWARDTOEXT if exists
Summary  : Extension of agent this message is to be forwarded to

Name     : FORWARDTOPOS
Value    : 45h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTPOS or previous FORWARDTOPOS if exists
Summary  : Position of agent this message is to be forwarded to

Name     : FORWARDTOORG
Value    : 46h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTORG or previous FORWARDTOORG if exists
Summary  : Organization of agent this message is to be forwarded to

Name     : FORWARDED
Value    : 48h
Data     : when_t
Multiple : Yes, order significant
Required : Yes, if forwarded
Summary  : Date/Time this message was forwarded to another agent
```

```
Name     : RECEIVEDBY
Value    : 50h
Data     : ASCII
Multiple : Yes, order significant
Required : Yes, if receiving agent is other than RECIPIENT
Summary  : Name of agent that received this message

Name     : RECEIVEDBYAGENT
Value    : 51h
Data     : ushort
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTAGENT or previous RECEIVEDBYAGENT if exists
Summary  : Type of agent that received this message

Name     : RECEIVEDBYNETTYPE
Value    : 52h
Data     : ushort
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTNETTYPE or previous RECEIVEDBYNETTYPE if exists
Summary  : Type of network that received this message

Name     : RECEIVEDBYNETADDR
Value    : 53h
Data     : undef
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTNETADDR or previous RECEIVEDBYNETADDR if exists
Summary  : Network address that received this message
```

```
Name     : RECEIVEDBYEXT
Value    : 54h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTEXT or previous RECEIVEDBYEXT if exists
Summary  : Extension of agent that received this message

Name     : RECEIVEDBYPOS
Value    : 55h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTPOS or previous RECEIVEDBYPOS if exists
Summary  : Position of agent that received this message

Name     : RECEIVEDBYORG
Value    : 56h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Default  : RECIPIENTORG or previous RECEIVEDBYORG if exists
Summary  : Organization of agent that received this message

Name     : RECEIVED
Value    : 58h
Data     : when_t
Multiple : Yes, order significant
Required : Yes, if received
Default  : NULL
Summary  : Date/Time this message was received
```

```
Name     : SUBJECT
Value    : 60h
Data     : ASCII
Multiple : No
Required : Yes, but may be blank (0 length or nulstr)
Summary  : Subject/title of message

Name     : SUMMARY
Value    : 61h
Data     : ASCII
Multiple : No
Required : No
Summary  : Summary of message contents, created by AUTHOR

Name     : COMMENT
Value    : 62h
Data     : ASCII
Multiple : Yes
Required : No
Summary  : Comment about this message, created by SENDER

This field is useful for adding notes to a message when forwarding to a
recipient.

Name     : CARBONCOPY
Value    : 63h
Data     : ASCII
Multiple : Yes
Required : No
Summary  : List of agents this message was also sent to

This field is optional and only for the use of notifying the recipient o
else received the message.

Name     : GROUP
Value    : 64h
Data     : ASCII
Multiple : Yes
Required : No
Summary  : Name of group of users to receive message on recipient system

This field is used when sending to a group name across a network, where
group can be expanded into multiple header records for each agent on the
destination system.

Name     : EXPIRATION
Value    : 65h
Data     : when_t
Multiple : No
Required : No
Summary  : Date/Time that this message will expire
```

```
Name     : PRIORITY
Value    : 66h
Data     : ulong
Multiple : No
Required : No
Default  : 0
Summary  : Message priority (0 is lowest, FFFFFFFFh is highest)
```

```
Name     : FILEATTACH
Value    : 70h
Data     : ASCII
Multiple : Yes
Required : No
Summary  : Name/file specification of attached file(s)
```

Name of attached file(s). Wildcards allowed. MSG_FILEATTACH attribute mu
set. If the MSG_FILEATTACH attribute is set but this field is not includ
the SUBJECT field is assumed to be the filename(s).

```
Name     : DESTFILE
Value    : 71h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Summary  : Destination name for attached file(s)
```

Wildcards allowed. FILEATTACH field must also be included.

```
Name     : FILEATTACHLIST
Value    : 72h
Data     : ASCII
Multiple : Yes
Required : No
Summary  : Name of ASCII list of attached filenames
```

Wildcards not allowed in ASCII list filename. Wildcards allowed in ASCII
MSG_FILEATTACH attribute must be set.

```
Name     : DESTFILELIST
Value    : 73h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Summary  : Name of ASCII list of destination filenames
```

Wildcards not allowed in ASCII list filename. Wildcards allowed in ASCII

```
Name     : FILEREQUEST
Value    : 74h
Data     : ASCII
Multiple : Yes
Required : No
Summary  : Name of requested file
```

Wildcards allowed. MSG_FILEREQUEST attribute must be set

```
Name     : FILEPASSWORD
Value    : 75h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Summary  : Password for FILEREQUEST
```

```
Name     : FILEREQUESTLIST
Value    : 76h
Data     : ASCII
Multiple : Yes
Required : No
Summary  : Name of ASCII list of filenames to request

Wildcards allowed.

Name     : FILEPASSWORDLIST
Value    : 77h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Summary  : Name of ASCII list of passwords for FILEREQUESTLIST
```

```
Name     : IMAGEATTACH
Value    : 80h
Data     : mattach_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of attached image file for display

MSG_FILEATTACH attribute must be set. See Image Types for valid
mattach_t.type values.

Name     : ANIMATTACH
Value    : 81h
Data     : mattach_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of attached graphical animation file for di

MSG_FILEATTACH attribute must be set. See Animation Types for valid
mattach_t.type values.

Name     : FONTATTACH
Value    : 82h
Data     : mattach_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of attached font definition file

MSG_FILEATTACH attribute must be set. See Font Types for valid mattach_t
values.

Name     : SOUNDATTACH
Value    : 83h
Data     : mattach_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of attached sound file for playback

MSG_FILEATTACH attribute must be set. See Sound Types for valid mattach_
values.

Name     : PRESENTATTACH
Value    : 84h
Data     : mattach_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of attached presentation definition file

MSG_FILEATTACH attribute must be set. See Present Types for valid
mattach_t.type values.
```

Name   : VIDEOATTACH
Value   : 85h
Data   : vattach_t
Multiple : Yes, order significant
Required : No
Summary : Type and filename of attached interleaved video/sound file

MSG_FILEATTACH attribute must be set. See Video Types for valid vattach_t.type values and Video Compression Types for valid vattach_t.co values.

Name   : APPDATAATTACH
Value   : 86h
Data   : mattach_t
Multiple : Yes, order significant
Required : No
Summary : Name of attached application data file for process/display

MSG_FILEATTACH attribute must be set. See Application Data Types for val mattach_t.type values.

```
Name     : IMAGETRIGGER
Value    : 90h
Data     : typestr_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of image file to trigger for display

See Image Types for valid typestr_t.type values.

Name     : ANIMTRIGGER
Value    : 91h
Data     : typestr_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of animation file to trigger for display

See Animation Types for valid typestr_t.type values.

Name     : FONTTRIGGER
Value    : 92h
Data     : typestr_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of font definition file to trigger

See Font Types for valid typestr_t.type values.

Name     : SOUNDTRIGGER
Value    : 93h
Data     : typestr_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of sound file to trigger for playback

See Sound Types for valid typestr_t.type values.

Name     : PRESENTTRIGGER
Value    : 94h
Data     : typestr_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of presentation definition file to trigger

See Present Types for valid typestr_t.type values.

Name     : VIDEOTRIGGER
Value    : 95h
Data     : typestr_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of interleaved video/sound file to trigger

See Video Types for valid typestr_t.type values.
```

```
Name     : APPDATATRIGGER
Value    : 96h
Data     : typestr_t
Multiple : Yes, order significant
Required : No
Summary  : Type and filename of application data file to trigger
```

See Application Data Types for valid typestr_t.type values.

```
Name     : FIDOCTRL
Value    : A0h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Format   : keyword ":" [" "] appdata
Summary  : FTS/FSC-compliant control information line
```

Any FidoNet FTS/FSC-compliant control information ("kludge") line that does not have an equivalent representation here. All data not unique to actual control line, including leading and trailing white space, Ctrl-A character and terminating CR must be ommited. Defined in FTS-0001.

```
Name     : FIDOAREA
Value    : A1h
Data     : ASCII
Multiple : No
Required : No
Summary  : FTN EchoMail conference name.
```

Defined in FTS-0004.

```
Name     : FIDOSEENBY
Value    : A2h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Format   : net"/"node [" "[net"/"]node] [...]
Summary  : Used to store two-dimensional (net/node) SEEN-BY information
```

Often used in FTN EchoMail environments. Only the actual SEEN-BY data is and SEEN-BY: is stripped along with any leading and trailing white space characters. Defined in FTS-0004.

```
Name     : FIDOPATH
Value    : A3h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Format   : net"/"node [" "[net"/"]node] [...]
Summary  : Used to store two-dimensional (net/node)
```

Defined in FTS-0004. ^aPATH: is stripped along with any leading and trai white space characters.

```
Name     : FIDOMSGID
Value    : A4h
Data     : ASCII
Multiple : No
Required : No
Format   : origaddr " " serialno
Summary  : MSGID field as specified in FTS-0009.

Name     : FIDOREPLYID
Value    : A5h
Data     : ASCII
Multiple : No
Required : No
Format   : origaddr " " serialno
Summary  : REPLY field as specified in FTS-0009.

Name     : FIDOPID
Value    : A6h
Data     : ASCII
Multiple : No
Required : No
Format   : pID " " version [" "serialno]
Summary  : Indentification string of program that created this message

Defined FSC-0046. "^aPID:" and any white space is not included.

Name     : FIDOFLAGS
Value    : A7h
Data     : ASCII
Multiple : Yes
Required : No
Summary  : Used to store the FTN FLAGS kludge information

Note that all FLAG options that have binary representation in the messag
header must be removed from the FLAGS string prior to storing it. Only t
actual flags option string is stored and ^aFLAGS is stripped along with
leading and trailing white space characters. Defined in FSC-0053.
```

```
Name     : RFC822HEADER
Value    : B0h
Data     : ASCII
Multiple : Yes, order significant
Required : No
Format   : field-name ":" [field-body] [CRLF]
Summary  : Undefined RFC-822 header field
```

Internet Message storage format, that does not have an equivalent representation here. Folded header fields are allowed. Terminating CRLF ommited.

```
Name     : RFC822MSGID
Value    : B1h
Data     : ASCII
Multiple : No
Required : No
Format   : "<" addr-spec ">"
Summary  : Message-ID field as specified in RFC-822.

Name     : RFC822REPLYID
Value    : B2h
Data     : ASCII
Multiple : No
Required : No
Format   : "<" addr-spec ">"
Summary  : In-Reply-To field as specified in RFC-822.
```

Name   : UNKNOWN
Value   : F0h
Data   : undef
Multiple : Yes
Required : No
Summary : Undefined header field of undefined type

This field is useful for retaining binary header fields (that do not hav equivalent representation here) between message storage formats.

Name   : UNKNOWNASCII
Value   : F1h
Data   : ASCII
Multiple : Yes
Required : No
Summary : Undefined header field of type ASCII

This field is useful for retaining ASCII header fields (that do not have equivalent representation here) between message storage formats.

Name   : UNUSED
Value   : FFh
Data   : undef
Multiple : Yes
Required : No
Summary : Unused (deleted) header field

The data contained in this header field is of an unknown type and should processed.

Note:

Specifically, not defined are the values F000h through FFFFh. These valu are to be used for user or system defined header fields. Digital Dynamic requests that any developers or organizations that wish to have addition header fields added to this specification notify Digital Dynamics throug of the contact methods listed at the beginning of this document.

Data Field Types:
==========

These are the defined valid values for dfield_t.type:

Val	Name	Data	Description
00h	TEXT_BODY	mtext_t	Displayable text (body of message). Included in duplicate message checki All terminating white space and cont characters are to be truncated from (except when multiple contiguous CRL terminate the text, only the last CR is removed).
01h	TEXT_SOUL	mtext_t	Non-displayed text. Not normally displayed. Not necessar displayable. Included in duplicate message checki
02h	TEXT_TAIL	mtext_t	Displayable text (tag/tear/origin li etc). Not included in duplicate message ch All terminating white space and cont characters are to be truncated from
03h	TEXT_WING	mtext_t	Non-displayed text. Not normally displayed. Not necessar displayable. Not included in duplicate message ch
10h	FTEXT_BODY	ftext_t	Formatted equivalent of TEXT_BODY to displayed in place of TEXT_BODY if f is supported. See Image Types for va values of ftext_t.type.
12h	FTEXT_TAIL	ftext_t	Formatted equivalent of TEXT_TAIL to displayed in place of TEXT_TAIL if f is supported. See Image Types for va values of ftext_t.type.

20h IMAGEEMBED	membed_t	Type and data of embedded raster ima for display. See Image Types for valid membed.typ values.
21h ANIMEMBED	membed_t	Type and data of embedded graphical animation file for display. See Animation Types for valid membed values.
22h FONTEMBED	membed_t	Type and data of embedded font defin file. See Font Types for valid membed_t.type values.
23h SOUNDEMBED	membed_t	Type and data of embedded sound file playback. See Sound Types for valid membed_t.t values.
24h PRESENTEMBED	membed_t	Type and data of embedded presentati definition file. See Present Types for valid membed_t values.
25h VIDEOEMBED	vembed_t	Type and data of embedded video/soun for playback. See Video Types for valid vembed_t.t values. See Video Compression Types for vali vembed_t.comp values.
26h APPDATAEMBED	membed_t	Type and data of embedded applicatio file for process/display. See Application Data Types for valid membed_t.type values.
FFh UNUSED	undef	Space allocated for future update/ex

Specifically, not defined are the values F000h through FFFFh. These valu are to be used for user or system defined data fields. Digital Dynamics requests that any developers or organizations that wish to have addition data fields added to this specification notify Digital Dynamics through of the contact methods listed at the beginning of this document.

Message Attributes:

These are the bit values for idxrec_t.attr and msghdr_t.attr:

```
MSG_PRIVATE         (1<<0)  // Private
MSG_READ            (1<<1)  // Read by addressee
MSG_PERMANENT       (1<<2)  // Permanent
MSG_LOCKED          (1<<3)  // Msg is locked, no editing possible
MSG_DELETE          (1<<4)  // Msg is marked for deletion
MSG_ANONYMOUS       (1<<5)  // Anonymous author
MSG_KILLREAD        (1<<6)  // Delete message after it has been read
MSG_MODERATED       (1<<7)  // This message must be validated
MSG_VALIDATED       (1<<8)  // This message has been validated by a mode
```

Auxillary Attributes:

These are the bit values for msghdr_t.auxattr:

```
MSG_FILEREQUEST     (1<<0)  // File request
MSG_FILEATTACH      (1<<1)  // File(s) attached to Msg
MSG_TRUNCFILE       (1<<2)  // Truncate file(s) when sent
MSG_KILLFILE        (1<<3)  // Delete file(s) when sent
MSG_RECEIPTREQ      (1<<4)  // Return receipt requested
MSG_CONFIRMREQ      (1<<5)  // Confirmation receipt requested
MSG_NODISP          (1<<6)  // Msg may not be displayed to user
```

Network Attributes:

These are the bit values for msghdr_t.netattr:

```
MSG_LOCAL           (1<<0)  // Msg created locally
MSG_INTRANSIT       (1<<1)  // Msg is in-transit
MSG_SENT            (1<<2)  // Sent to remote
MSG_KILLSENT        (1<<3)  // Kill when sent
MSG_ARCHIVESENT     (1<<4)  // Archive when sent
MSG_HOLD            (1<<5)  // Hold for pick-up
MSG_CRASH           (1<<6)  // Crash
MSG_IMMEDIATE       (1<<7)  // Send Msg now, ignore restrictions
MSG_DIRECT          (1<<8)  // Send directly to destination
MSG_GATE            (1<<9)  // Send via gateway
MSG_ORPHAN          (1<<10) // Unknown destination
MSG_FPU             (1<<11) // Force pickup
MSG_TYPELOCAL       (1<<12) // Msg is for local use only
MSG_TYPEECHO        (1<<13) // Msg is for conference distribution
MSG_TYPENET         (1<<14) // Msg is direct network mail
```

Translation Types:

Definition for values of *.xlat[x]:

```
XLAT_NONE           0       // No translation/End of translation list
XLAT_LF2CRLF        1       // Expand sole LF to CRLF
XLAT_ESCAPED        2       // 7-bit ASCII escaping for ctrl and 8-bit d
XLAT_HUFFMAN        3       // Static and adaptive Huffman coding compre
XLAT_LZW            4       // Limpel/Ziv/Welch compression
XLAT_MLZ78          5       // Modified LZ78 compression
XLAT_RLE            6       // Run length encoding compression
XLAT_IMPLODE        7       // Implode compression (PKZIP)
XLAT_SHRINK         8       // Shrink compression (PKZIP)
```

Agent Types:

```
AGENT_PERSON        0       // To or from person
AGENT_PROCESS       1       // Unknown process, identified by agent name
```

Agent types E000h through EFFFh are reserved for Synchronet process type (defined specifically by Digital Dynamics).

Note:

Specifically not defined are agent types F000h through FFFFh. These valu are to be used for user or system defined agent types. Digital Dynamics requests that any developers or organizations that wish to have addition agent types added to this specification notify Digital Dynamics through of the contact methods listed at the beginning of this document.

Network Types:

```
NET_NONE            0       // Locally created
NET_UNKNOWN         1       // Unknown network type
NET_FIDO            2       // FTN network
NET_POSTLINK        3       // PostLink network
NET_QWK             4       // QWK based network
NET_INTERNET        5       // The Internet
```

Media Types:
==========

Image Types:

```
IMAGE_UNKNOWN       0x00    // Use image signature header to determine f
IMAGE_ASC           0x01    // ASCII text/IBM extended ASCII graphics
IMAGE_ANS           0x02    // ANSI X3.64 terminal escape sequences
IMAGE_AVT           0x03    // AVATAR terminal escape sequences
IMAGE_LVI           0x04    // LVI terminal escape sequences
IMAGE_GIF           0x05    // Compuserve Graphics Interchange Format (G
IMAGE_TIF           0x06    // Tagged Image Format (AKA TIFF)
IMAGE_JPG           0x07    // Joint Photographers Electronics Group (JP
IMAGE_T16           0x08    // TrueVision 16-bit bitmap (TGA)
IMAGE_T24           0x09    // TrueVision 24-bit bitmap (TGA)
IMAGE_T32           0x0a    // TrueVision 32-bit bitmpa (TGA)
IMAGE_PCX           0x0b    // ZSoft PaintBrush graphics
IMAGE_BMP           0x0c    // Windows bitmap
IMAGE_RLE           0x0d    // Windows bitmap (compressed)
IMAGE_DIB           0x0e    // Display independant bitmap
IMAGE_PCD           0x0f    // Kodak PhotoCD
IMAGE_G3F           0x10    // Group 3 FAX
IMAGE_EPS           0x11    // Ecapsulated PostScript
IMAGE_RTF           0x12    // Rich text format
IMAGE_RIP           0x13    // Remote Imaging Protocol Script (RIPscrip)
IMAGE_NAP           0x14    // NAPLPS
IMAGE_CDR           0x15    // Corel Draw!
IMAGE_CGM           0x16    // Computer graphics metafile
IMAGE_WMF           0x17    // Windows metafile
IMAGE_DFX           0x18    // Autodesk AutoCAD
```

Animation Types:

```
ANIM_UNKNOWN        0       // Use file signature header to determine fo
ANIM_FLI            1       // Autodesk animator
ANIM_FLC            2       // Autodesk
ANIM_GL             3       // Grasprt
```

Video Types:

```
VIDEO_UNKNOWN       0       // Use file signature header to determine fo
VIDEO_QTIME         1       // Apple Quick-time
VIDEO_FQTIME        2       // Apple Flattened Quick-time
VIDEO_AVI           3       // Windows Auto/Video Interleave
VIDEO_ULT           4       // OS/2 Ultimotion
```

```
Video Compression Types:
------------------------

VCOMP_UNKNOWN       0       // Use file signature header to determine co
VCOMP_RLE           1       // Apple animation
VCOMP_SMC           2       // Apple graphics
VCOMP_RPZA          3       // Apple video
VCOMP_KLIC          4       // Captain crunch
VCOMP_CVID          5       // CinePak
VCOMP_RT21          6       // Intel indeo R2
VCOMP_IV31          7       // Intel indeo R3
VCOMP_YVU9          8       // Intel YVU9
VCOMP_JPEG          9       // JPEG
VCOMP_MRLE          10      // Microsoft RLE
VCOMP_MSVC          11      // Microsoft video 1


Font Types:
-----------

FONT_UNKNOWN        0       // Use file signature header to determine fo
FONT_TTF            1       // Windows TrueType
FONT_PFB            2       // PostScript Type 1 Font Binary
FONT_PFM            3       // PostScript Type 1 Font Metric


Sound Types:
------------

SOUND_UNKNOWN       0       // Use file signature header to determine fo
SOUND_MOD           1       // MOD format
SOUND_VOC           2       // Sound Blaster VOC format
SOUND_WAV           3       // Windows 3.1 WAV RIFF format
SOUND_MID           4       // MIDI format
SOUND_GMID          5       // General MIDI format (standardized patches
SOUND_SMP           6       // Turtle Beach SampleVision format
SOUND_SF            7       // IRCAM format
SOUND_AU            8       // Sun Microsystems AU format


Application Data Types:
-----------------------

APPDATA_UNKNOWN     0       // Use file signature header to determine fo
APPDATA_WORDPERFECT 1       // WordPerfect Document
APPDATA_WKS         2       // Lotus 123 Worksheet (?)
APPDATA_WK1         3       // Lotus 123 Worksheet rev 1
APPDATA_WK2         4       // Lotus 123 Worksheet rev 2
APPDATA_WK3         5       // Lotus 123 Worksheet rev 3
APPDATA_DBF         6       // dBase III data file
APPDATA_PDX         7       // Paradox data file
APPDATA_EXCEL       8       // Excel data file
APPDATA_QUATRO      9       // Borland Quatro Pro file
APPDATA_WORD        10      // Microsoft Word
```

Message Storage Protocol

1. Open SDT, SHD, and SID files read/write deny-none (shareable).
2. Determine length of all message data and number of 256 byte blocks required to store the data.
3. Open SDA file read/write deny-all.
4. If fast allocation mode, seek to end of SDA file and go to step 6.
5. Search SDA file for enough consecutive unused blocks to store all of message data. If found, seek back to beginning of unused blocks. Oth stay at end of file.
6. Write to the SDA file the number of index entries that are going to to this data (normally 1) for the number of blocks that will be used data block(s) have now been allocated.
7. Close the SDA file.
8. Determine length of header record and number of 256 byte blocks required to store the record.
9. Open SHA file read/write deny-all.
10. If fast allocation mode, seek to end of SHA file and go to step 12.
11. Search SHA file for enough consequetive unused blocks to store all o header record. If found, seek back to beginning of unused blocks. Ot stay at end of file.
12. Write to the SHA file a 1 (single byte) for each block that will be The header block(s) have now been allocated.
13. Close the SHA file.
14. Lock message base header in SHD file.
15. Read SHD base header #1 (config info).
16. Increment the total number of messages and last message number in he
17. Write SHD base header #1 (config info).
18. Write header record to SHD file.
19. Write index record to SID file.
20. Unlock SHD base header.
21. Write message data to SDT file.

Message Retrieval Protocol

1. Open SDT, SHD, and SID files read/write deny-none (shareable).
2. Read index record from SID file.
3. Seek to the byte offset in the SHD file specified in the index recor
4. Lock the message header record.
5. Read the message header record.
6. Unlock the message header record.
7. Compare the message number to the one specified in the index record. they don't match, re-read the index record and goto step 3. If they continue to mismatch, the index has been corrupted and must be recre
8. For each data field specified in the header, seek to the byte offset in the SDT file plus the offset specified in the data field, read fr the SDT file the length (in bytes) specified in the data field.

SMBUTIL
=======

SMBUTIL is a utility that can perform various functions on an SMB format message base. The primary purpose of SMBUTIL is as an example to C progr of how to use the SMBLIB functions to access and modify an SMB message b The complete C source code for SMBUTIL is included and functions from it be used or modified by developers at their own discretion. The following make up SMBUTIL:

```
SMBUTIL.EXE     Compiled and linked (ready to run)
SMBUTIL.C       C functions
SMBUTIL.H       C definitions and variable prototypes
SMBUTIL.MAK     Makefile (for Borland C++)
CRC32.H         C header file for CRC-32 calculations
```

The usage syntax is as follows:

```
SMBUTIL [/opts] cmd smb_filespec.shd
```

where cmd is one or more of the following:

```
       l[n] = list msgs starting at number n
       r[n] = read msgs starting at number n
       v[n] = view msg headers starting at number n
       k[n] = kill (delete) n msgs
       i<f> = import from text file f
       s    = display msg base status
       c    = change msg base status
       m    = maintain msg base - delete old msgs and msgs over max
       p    = pack msg base
```

where opts is one or more of the following:

```
       a    = always (force) packing
       f    = fast msg creation mode
       d    = disable duplicate message checking
       z<n> = set time zone (n=min +/- from UT or 'EST','EDT','CST',etc)
```

and smb_filespec is the base filename or file specification (wildcards) message base. If wildcards are used, the ".SHD" extension must be specif

An example command line:

```
SMBUTIL R FORSALE
```

would read all the messages in the forsale message base. If the forsale base files are not stored in the current directory, the complete path mu specified. (i.e. smbutil r c:\msgs\forsale)

```
SMBUTIL MP C:\SBBS\DATA\SUBS\*.SHD
```

would maintain and pack all the message bases found in the C:\SBBS\DATA\ directory.

CHKSMB

CHKSMB is a utility that performs a comprehensive analysis of a message to find any possible errors. It does not "fix" a message base if any err are found, it only reports the specific errors (and exits with a non-zer error level).

C source code for CHKSMB is also included as an example to programmers o to use SMBLIB functions.

The usage syntax is as follows:

```
CHKSMB [/opts] smb_filespec.shd
```

where opts is one or more of the following:

```
        q   = quiet mode (no beeps)
        s   = stop after an errored message base (for use with wildcards
        p   = pause after an errored message base (wait for key press)
```

An example command line:

```
CHKSMB /QP C:\SBBS\DATA\SUBS\*.SHD
```

would check all the message bases in the C:\SBBS\DATA\SUBS directory, wi beeping on errors, and pausing after an errored message base.

SMBLIB
======

SMBLIB is a library of C functions for accessing and storing messages in SMB format message base. It can eliminate much of the development time f developers that wish to use the library in whole or in part, or use the functions as examples for their own message base function library. The l consists of the following files:

SMBDEFS.H	Constant definitions, macros, and data types
SMBLIB.H	Function prototypes
SMBLIB.C	Function definitions
SMBVARS.C	Global variable definitions (doubles as declaration file

For developers to use this library with their program, they must include "SMBLIB.H" header file at the top of each C file that uses any of the li functions, global variables, data types, macros, constants. This can be by simply add the following line to each .C file:

```
#include "smblib.h"
```

If SMBLIB.H is included, there is no need to include SMBDEFS.H or SMBVAR

To link the library functions and variables with a main program, the fil SMBVARS.OBJ and SMBLIB.OBJ must be linked with the main program .OBJ fil If the operating system is DOS, be sure that all .OBJ files are compiled the same memory model.

An example MAKEFILE for compiling and linking SMBUTIL with Borland C++ i included.

SMBDEFS.H
=========

The SMBDEFS.H file contains important constant definitions and data type defined in this document). If ever this document and SMBDEFS.H are incon with each other, then SMBDEFS.H is to be considered correct and this doc in error. If such a discrepency is found, please notifiy Digital Dynamic can be corrected in a future revision of the specification.

Most notable of the data types is a structure called smbmsg_t (not defin in this document). It contains the fixed and variable portions of a mess header record as well as convenience pointers to the sender's name (smbmsg_t.to), recipient's name (smbmsg_t.from), network addresses, and If multiple SENDER header fields are included (for example), then smbmsg will point to the last SENDER header field in the header record. Conveni pointers for other data items work in the same fasion if multiple header of the same type exist in the header record.

Variables of the smbmsg_t data type (and pointers to variables of smbmsg type) are used as arguments to many of the SMBLIB functions.

SMBVARS.C
=========

The SMBVARS.C file contains definitions of the global variables used by SMBLIB functions. It is a fairly small file since their are a small numb global variables (by design). This file is used for both definitions and declarations, so no "extern" declarations need to be made in developers code as long as SMBVARS.C or (preferably) SMBLIB.H is included in the so code.

SMBLIB.H

The SMBLIB.H file contains prototypes of all the functions in the SMBLIB file. It is necessary to include this file in C source code if any of th SMBLIB functions are used. The following C source line will include this

```
#include "smblib.h"
```

and should be placed near the top of all C source files that use SMBLIB functions, variables, constants, or data types.

Function prototypes are necessary for compilers to know the correct call syntax of a function and detect incorrect usage. Prototypes are also use as a quick reference for programmers as to the correct calling syntax of specific function.

SMBLIB.C
=======

The SMBLIB.C file contains the actual SMBLIB library functions. This sou
file is not a stand alone program, but instead must be compiled and link
with a main source file to create the executable program.

The functions in this file are organized in a logical order, but their o
is actually irrelevant to the compiling, linking, and execution of the
resulting program.

A comment block preceeds each function, explaining what the function doe
how the passed parameters are used, and what the return code (if any)
indicates. A more detailed explanation of each function is included here

int smb_open(int retry_time)

The smb_open() function must be called before the message base is access
(read from or written to). The parameter, retry_time, is the maximum num
of seconds to wait while retrying to lock the message base header. The g
variable smb_file must be initialized with the path and base filename of
message base. This function returns 0 on success, 1 if the .SDT file cou
be opened, 2 if the .SHD file could not be opened, and 3 if the .SID fil
not be opened. If the message base header could not be locked, this func
returns -1. If the message base ID is incorrect, it returns -2. And if t
message base is of an incompatible version, it returns -3.

The errno global variable (standard of most C libraries) will most likel
contain the error code for open failure.

int smb_open_da(int retry_time)

The smb_open_da() function is used to open the data block allocation fil
writing messages to a message base. The parameter, retry_time, is the ma
number of seconds to wait while retrying to open the file. This function
returns 0 on success. -1 is returned if an open error other than "Access
Denied" is returned from the operating system, and the global variable e
will contain the error code. -2 is returned if the retry_time has been
reached, and -3 is returned if the file descriptor could not be converte
a stream by the fdopen() function.

fclose(smb_sda) should be called immediately after all necessary file ac
has been completed.

int smb_open_ha(int retry_time)

The smb_open_ha() function is used to open the header block allocation f writing messages to a message base. The parameter, retry_time, is the ma number of seconds to wait while retrying to open the file. This function returns 0 on success. -1 is returned if an open error other than "Access Denied" is returned from the operating system, and the global variable e will contain the error code. -2 is returned if the retry_time has been reached, and -3 is returned if the file descriptor could not be converte a stream by the fdopen() function.

fclose(smb_sha) should be called immediately after all necessary file ac has been completed.

int smb_create(ulong max_crcs, ulong max_msgs, ushort max_age, int retry
---
The smb_create() function is used to create a new message base or reset existing message base. The parameters max_crcs, max_msgs, and max_age ar to set the initial status of the message base status header. The paramet retry_time is the maximum number of seconds to wait while retrying to lo message base header. This functions returns 0 on success or 1 if the mes base header could not be locked.

int smb_trunchdr(int retry_time)

The smb_trunchdr() function is used to truncate the header file when pac the message base and writing the new header information back to the head file. The parameter, retry_time is the maximum number of seconds to wait retrying to truncate the header file. Returns 0 on success, -1 if error other than "Access Denied", or -2 if retry_time reached.

int smb_locksmbhdr(int retry_time)

The smb_locksmbhdr() function is used to lock the first message base (st header. The parameter, retry_time is the number of seconds to wait while retrying to lock the header. The smb_unlocksmbhdr() function should alwa used to unlock the header after accessing the message base header (usual with smb_getstatus() and/or smb_putstatus()). Returns 0 if successful, - unsuccessful.

int smb_unlocksmbhdr()

The smb_unlocksmbhdr() function is used to unlock a previously locked me base header (using smb_lockmsghdr()). Returns 0 on success, non-zero on failure.

int smb_getstatus(smbstatus_t *hdr)

The smb_getstatus() function is used to read the status message base hea into the hdr structure. Returns 0 on success, 1 on failure.

```
int smb_putstatus(smbstatus_t hdr)
----------------------------------
```
The smb_putstatus() function is used to write the status information to first message base header. The parameter hdr, contains the status inform to be written. Returns 0 on success, 1 on failure.

```
int smb_getmsgidx(smbmsg_t *msg)
--------------------------------
```
The smb_getmsgidx() function is used to get the byte offset for a specif message header in the message header file based on the message base inde

If msg->hdr.number is non-zero when this function is called, then the in will be searched for this message number. If the message number is found the index, the msg->idx.offset is set to the byte offset of the message record in the header file and msg->offset is set to the record offset of index record in the index file, and the function returns 0. If the messa number is not found in the index, the function returns 1.

If msg->hdr.number is zero, msg->idx.offset and msg->idx.number are obta from the index record at record offset msg->offset. If msg->offset is an invalid record offset when this function is called, the function returns Otherwise, the function returns 0.

```
int smb_getmsghdrlen(smbmsg_t msg)
----------------------------------
```
The smb_getmsghdrlen() function is used to calculate the total length of message header msg including both fixed and variable length portions. Th function returns the length of the header record in bytes.

```
long smb_getmsgdatlen(smbmsg_t msg)
-----------------------------------
```
The smb_getmsgdatlen() function is used to calculate the total length of data for message msg. This function returns the length of all data field combined.

```
int smb_lockmsghdr(smbmsg_t msg, int retry_time)
------------------------------------------------
```
The smb_lockmsghdr() function is used to lock the header record for mess msg. The parameter retry_time is the maximum number of seconds to wait w retrying to lock the header. Returns 0 on success, -1 on failure. The fu smb_unlockmsghdr() should immediately be called after accessing the mess header (usually with smb_getmsghdr() or smb_putmsghdr()).

```
int smb_getmsghdr(smbmsg_t *msg)
--------------------------------
```
The function smb_getmsghdr() is used to read the header record for messa msg. msg->idx.offset must be initialized to the byte offset of the heade record in the header file before this function is called. The function smb_freemsgmem() must be called to free the memory allocated by this fun for the header and data felds. This function returns 0 on success, -1 if the fixed portion of the message header record could not be read, -2 if message header ID was incorrect, -3 if memory could not be allocated, -4 if a data field could not be read, -5 if the fixed length portion of a h field could not be read, -6 if the variable length portion of a header f could not be read, -7 if one or more of the mandatory header fields (SEN RECIPIENT, or SUBJECT) are missing, -8 if total_dfields extends beyond t end of the header reocord, or -9 if incompatible header version.

Several convenience pointers in the msg structure are initialized by thi function to point to the last occurance of the SENDER (msg->from), RECIP (msg->to), SUBJECT (msg->subj), etc.

```
int smb_unlockmsghdr(smbmsg_t msg)
----------------------------------
```
The smb_unlockmsghdr() function is used to unlock a previously locked me header (with smb_lockmsghdr()). This function returns 0 on success, non- on failure.

```
int smb_addcrc(ulong max_crcs, ulong crc, int retry_time)
---------------------------------------------------------
```
The smb_addcrc() function is used to add a CRC-32 to the CRC history fil for a message base, automatically checking for duplicates. The parameter max_crcs should be the max_crcs defined in the status header of the mess base. The parameter crc, is the CRC-32 of the TEXT_BODY and TEXT_SOUL da fields for the message. The parameter retry_time is the maximum number o seconds to wait when retrying to open the CRC history file.

This function returns -1 if there was an open error, -2 if the retry_tim was reached, -3 if there was a memory allocation error, 1 if the CRC alr exists in the CRC history file (indicating a duplicate message), or 0 on success (and no duplicate).

```
int smb_hfield(smbmsg_t *msg, ushort type, ushort length, void *data)
----------------------------------------------------------------------
```
The smb_hfield() function is used to add a header field to the structure The parameters type, length, and data, must be specified according to th header field values listed in this specification. This function returns on success, non-zero on memory allocation error. The function smb_freems must be called to free the memory allocated by this function.

```
int smb_dfield(smbmsg_t *msg, ushort type, ulong length)
--------------------------------------------------------
```
The smb_dfield() function is used to add a data field to the structure m The parameters type and length must be specified according to the data f values listed in this specification. This function returns 0 on success, non-zero on memory allocation error. The function smb_freemsgmem() must called to free the memory allocated by this function.

int smb_addmsghdr(smbmsg_t *msg, smbstatus_t *status, int fast, int retr
---
The smb_addmsghdr() function is used to add a new message header to the header file. The msg and status structures are updated to reflect the ne total messages, last message number, etc. The fast parameter is used to indicate if the fast allocation mode should be used. If the fast paramet 0 (off), the header block allocation file will be searched for unused bl to store this header. If the fast parameter is 1 (on), the header is sto the end of the header file. Returns 0 on success, non-zero on failure. T parameter retry_time is the maximum number of seconds to wait while retr to lock and open files.

int smb_putmsg(smbmsg_t msg)

Ths smb_putmsg() function calls both the smb_putmsghdr() and smb_putmsgi functions to write the header and index elements of a message to the appropriate files. Returns 0 on success, non-zero on failure.

int smb_putmsgidx(smbmsg_t msg)

The smb_putmsgidx() function is used to store a message index in the mes index file. The message index can be for a new message or an existing message. Returns 0 on success, non-zero on failure.

int smb_putmsghdr(smbmsg_t msg)

The smb_putmsghdr() function is used to store a message header in the me header file. The message header can be for a new message or an existing message. Returns 0 on success, non-zero on failure.

void smb_freemsgmem(smbmsg_t msg)

Frees allocated memory for the header and data fields in the msg structu This function must be called to free the memory allocated by the functio smb_hfield(), smb_dfield(), and smb_getmsghdr().

long smb_hdrblocks(ulong length)

The smb_hdrblocks() function is used to calculate the number of blocks required to store a message header of length size (in bytes). This funct returns the number of blocks required.

long smb_datblocks(ulong length)

The smb_datblocks() function is used to calculate the number of blocks required to store message data of length size (in byte). This function r the number of blocks required.

long smb_allochdr(ulong length)

The smb_allochdr() function is used to search for free blocks to store a message header of length bytes and mark the free blocks as allocated in header allocation file. This function returns the byte offset to the hea record or a negative number on error. The function smb_open_ha() should called prior to calling this function and fclose(sha_fp) should be calle after.

long smb_fallochdr(ulong length)

The smb_fallochdr() function works exactly the same as the smb_allochdr(function except it is much faster because the header allocation file is searched for free blocks.

long smb_allocdat(ulong length, ushort headers)

The smb_allocdat() function is used to search for free blocks to store l amount of data for a message. The parameter headers, indicates the numbe message headers that are associated with this data. Normally, the header parameter will be 1, unless this message is part of a mass mailing. The to the allocated data blocks is returned, or a negative value on error. function smb_open_da() should be called prior to calling this function a fclose(sda_fp) should be called after.

long smb_fallocdat(ulong length, ushort headers)

The smb_fallocdat() function works exactly the same as the smb_allocdat(function except it is much faster because the data allocation file is no searched for free blocks.

int smb_incdat(ulong offset, ulong length, ushort headers)

The smb_incdat() function is used to increment the header counter in the allocation file for the data starting at the byte offset and length size bytes. The parameter headers, indicates the number of headers to add to current allocation value in the data allocation file. Returns 0 on succe non-zero on failure.

int smb_freemsg(smbmsg_t msg, smbstatus_t status)

The smb_freemsg() function is used to free the memory allocated for the and data fields in the msg structure. Returns 0 on success, non-zero on failure. The parameter, status, must be the current status from the mess base header for this message base.

int smb_freemsgdat(ulong offset, ulong length, ushort headers)

The smb_freemsgdat() function is used to decrement the data block alloca records in the data allocation file associated with the data in the data by the value of the headers parameter (normally 1). The parameter offset indicates the byte offset to the beginning of the message data in the da file and the parameter length is the total length of the message data. Returns 0 on success, non-zero on failure.

int smb_freemsghdr(ulong offset, ulong length)

The smb_freemsghdr() function is used to set the header block allocation records in the header allocation file to 0 (indicated non-allocated bloc The parameter offset indicates the byte offset to the beginning of the h record being freed and the parameter length indicates the total length o header record. Returns 0 on success, non-zero on failure.

Bibliography

Title   : The C Programming Language
Publisher : Prentice Hall
Author   : Brian W. Kernighan and Dennis M. Ritchie

Document : ARPANET Request for Comments (RFC) #822
Title   : Standard for the Format of ARPA Internet text messages
Publisher : SRI International
Author   : David H. Crocker, University of Delaware

Document : FTS-0001
Publisher : FSC
Author   : Randy Bush, Pacific Systems Group

Document : FTS-0004
Title   : EchoMail Specification
Publisher : FSC
Author   : Bob Hartman

Document : FTS-0009
Title   : A standard for unique message identifiers and reply chain li
Publisher : FSC
Author   : Jim Nutt

Document : FSC-00046
Title   : A Product Idenfifier for FidoNet Message Handlers
Publisher : FSC
Author   : Joaquim H. Homrighausen

Document : FSC-00053
Title   : Specifications for the ^aFLAGS field
Publisher : FSC
Author   : Joaquim H. Homrighausen

Implementations
===============

Product   : Synchronet Multinode BBS Software
Developer : Digital Dynamics
Level   : II
Version   : 2.00

Product   : Synchornet/FidoNet Import/Export Utility (SBBSFIDO)
Developer : Digital Dynamics
Level   : II
Version   : 2.00

Product   : Synchronet UTI (Universal Text Interface) Driver
Developer : Digital Dynamics
Level   : II
Version   : 2.00