5

SIDF Functions and FID Macros

These functions construct and maintain sections and fields. The macros deal with FIDs. For more information about sections and fields, see *System Independent Data Format*. The field functions contain two routines, **SMDFPutFields** and **SMDFGetFields**, that you can use to parse FIDs and two routines, **SMDFPutNextField** and **SMDFGetNextField**, you can use to build your own parsing routines. FID macros contain macros that can help you build fields and parse the fields.

Header Functions

CCODE
NWSMSetMediaHeaderInfo

```
( NWSM_MEDIA_INFO *mediaInfo,
  BUFFERPTR buffer,
  UINT32 bufferSize,
  UINT32 *headerSize);
```

Parameters

mediaInfo	(INPUT) Passes the media information.
buffer	(OUTPUT) Returns the SIDF formatted media information. If you are using SDI, pass this buffer to **NWSMSDLabelMedia** (see *Storage Device API*).
bufferSize	(INPUT) Passes *buffer*'s size.
headerSize	(OUTPUT) Returns the size (in bytes) of the formatted media header.

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Prerequisites

None

Remarks

This function converts *mediaInfo* into an SIDF media header. The SME calls this function. *mediaInfo* uses the following structure:

```
typedef struct
{
   UINT32   mediaSetDateAndTime;
   UINT32   mediaDateAndTime;
   BUFFER   mediaSetLabel[NWSM_MAX_MEDIA_LABEL_LEN];
   UINT16   mediaNumber;
} NWSM_MEDIA_INFO;
```

mediaSetDateAndTime contains a DOS packed date and time value and is used for every medium in the set. To unpack or pack the date and time, see *Storage Management Services Utilities Library*.

mediaDateAndTime contains a DOS packed date and time value and indicates the when the media header was written. That is, if a media set contains two or more media and the back up started on July 3, 1991 11:30 pm, each media header's *mediaDateAndTime* value is:

Media Set	*mediaDateAndTime*
medium 1	July 3, 1992 11:300pm
medium 2	July 4, 1992 12:00am
medium 3	July 4, 1992, 12:30am

Note: SDI users only need to initialize this field to the same value as *mediaSetDateAndTime*. SDI will increment the date and time value for you.
. . .

mediaSetLabel contains the media set's label. NWSM_MEDIA_LABEL_LEN includes the NULL terminator.

mediaNumber contains the medium's sequence number (media numbers start from 1).

CCODE
NWSMGetMediaHeaderInfo

```
( BUFFERPTR headerBuffer,
  UINT32 headerBufferSize,
  NWSM_MEDIA_INFO *mediaInfo);
```

Parameters

headerBuffer	(INPUT) Passes a transfer buffer that was returned by **NWSMSDReturnMediaHeader** (see *Storage Device API*).
headerBufferSize	(INPUT) Passes the transfer buffer's size.
mediaInfo	(OUTPUT) Returns the media header information (see **NWSMSetMediaHeaderInfo** for information about this structure).

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFF9	NWSMUT_INVALID_FIELD_ID
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Prerequisites

None

Remarks

This function deformats the media header retrieved from the media.

Example

```
 /* example using SDI */
 NWSM_MEDIA_INFO mediaInfo;
 NWSMSD_CONTROL_BLOCK controlBlock;  /* SDI structure */
 NWSMSD_HEADER_BUFFER headerBuffer;

NWSMSDReturnMediaHeader(sdiConnection, selectedMediaHandle,
         &headerBuffer);

NWSMGetMediaHeaderInfo(headerBuffer.headerBuffer, headerBuffer.headerSize,
   &mediaInfo);
```

CCODE
NWSMSetSessionHeaderInfo

```
( NWSM_SESSION_INFO *sessionInfo,
  BUFFERPTR buffer,
  UINT32 bufferSize,
  UINT32 *headerSize);
```

Parameters

sessionInfo	(INPUT) Passes the session header information to be formatted.
buffer	(OUTPUT) Returns the formatted session information.
bufferSize	Contains the buffer's size.
headerSize	Contain's the header's size.

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Remarks

This function formats the session header according to SIDF's specifications. Developers using SDI can use **NWSMSDOpenSessionForWriting** to write the header out to the medium (see *Storage Device API*).

```
typedef struct
{
  UINT32 sessionDateAndTime;
  BUFFER sessionDescription[NWSM_MAX_DESCRIPTION_LEN];
  BUFFER softwareName[NWSM_MAX_SOFTWARE_NAME_LEN];
  BUFFER softwareType[NWSM_MAX_SOFTWARE_TYPE_LEN];
  BUFFER softwareVersion[NWSM_MAX_SOFTWARE_VER_LEN];
  BUFFER sourceName[NWSM_MAX_TARGET_SRVC_NAME_LEN];
  BUFFER sourceType[NWSM_MAX_TARGET_SRVC_TYPE_LEN];
  BUFFER sourceVersion[NWSM_MAX_TARGET_SRVC_VER_LEN];
} NWSM_SESSION_INFO;
```

sessionDateAndTime contains a DOS packed date and time value. To unpack or pack the date and time, see *Storage Management Services Utilities Library*.

sessionDescription contains a user defined string. The size includes the NULL terminator.

softwareName contains the name of the software servicing the target.
softwareType and *sourceVersion* contains a software type and version string.

sourceName contains the target's name.

sourceType and *sourceVersion* contains the targets type and version string.

CCODE
NWSMGetSessionHeaderInfo

```
( BUFFERPTR headerBuffer,
  UINT32 headerBufferSize,
  NWSM_SESSION_INFO *sessionInfo);
```

Parameters

headerBuffer	(INPUT) Passes the transfer buffer returned by **NWSMSDOpenSessionForReading**.
headerBufferSize	(INPUT) Passes the header's size.
sessionInfo	(OUTPUT) Returns the unformatted session information. See **NWSMSetSessionHeaderInfo** for more information about this structure.

Completion Codes

0x0	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFF9	NWSMUT_INVALID_FIELD_ID
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW

Remarks

This function deformats the session header that was retrieved from the media.

The SME makes this call.

CCODE
NWSMSetRecordHeader

```
( BUFFERPTR *buffer,
  UINT32 *bufferSize,
  UINT32 *bufferData,
  NWBOOLEAN setCRC,
  NWSM_RECORD_HEADER_INFO *recordHeaderInfo);
```

Parameters

buffer	(INPUT/OUTPUT) Points within a transfer buffer to where the next record should be put. It returns a pointer to the end of the inserted record (where the next record should be).
bufferSize	(INPUT/OUTPUT) Passes the amount of free space in *buffer* and returns the leftover free space (free space - size of record).
bufferData	(INPUT) Passes the formatted data returned by **NWSMTSReadDataSe**t (the data set data).
setCRC	(INPUT) If set to TRUE, the function calculates the CRC and writes it into the record. if set to FALSE, the function does not calculate a CRC value.
recordHeaderInfo	(INPUT/OUTPUT) Passes the header information to be formatted according to SIDF's specifications and returns the results.

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Remarks

This function formats the data set information according to SIDF's specifications, places the result and the data set data into a data set or subdata set, and inserts it into a transfer buffer (*buffer*).

recordHeaderInfo uses the following structure:

```
typedef struct
{
   NWBOOLEAN                    isSubRecord;
   NWSM_DATA_SET_NAME_LIST      *dataSetName;
   NWSM_SCAN_INFORMATION        *scanInformation;
   UINT32                       headerSize;
   UINT32                       recordSize;
   UINT32                       *addressOfRecordSize;
   UINT32                       *addressForCRC;
   BUFFERPTR                    crcBegin;
   UINT32                       crcLength;
   UINT32                       archiveDateAndTime;
} NWSM_RECORD_HEADER_INFO;
```

isSubRecord is set to FALSE if a new record is being made or to TRUE if data overflows a record.

dataSetName contains information about the data set's name. This information was returned by one of the TS API scanning functions (**NWSMTSScanDataSetBegin** or **NWSMTSScanNextDataSet**). See *Target Service API* for more information.

scanInformation contains the data set's scan information that was returned from one of the scanning calls (see *dataSetName*).

headerSize - Contains the record's header size. The function sets and updates this field.

recordSize - Contains the record's size. The function sets and updates this field.

addressOfRecordSize - Points to the data that contains the record's size.

addressForCRC - Points to the data that contains the record's CRC value.

crcBegin - Points to the beginning of a section. The function sets this.

crcLength - Contains the length of the CRC value in bytes. The function set this.

archiveDateAndTime - Contains the archive date and time of the data set. The function sets this.

CCODE
NWSMUpdateRecordHeader
(NWSM_RECORD_HEADER_INFO *recordHeaderInfo);

Parameters

recordHeaderInfo	(INPUT/OUTPUT) Passes the recorder to update and returns the new record header size and CRC. See **NWSMUpdateRecordHeader** for more information about this structure.

Completion Codes

0x0	Successful

Remarks

This function updates the

Call this function each time you insert a record and just before you write the buffer to the medium.

CCODE
NWSMGetRecordHeader

```
( BUFFERPTR *buffer,
  UINT32 *bufferSize,
  NWSM_RECORD_HEADER_INFO *recordHeaderInfo);
```

Parameters

buffer	(INPUT/OUTPUT) Passes a pointer to a transfer buffer. Before calling this function, *buffer* must be moved to the beginning of the record. When the function returns, buffer points to the data set data.
bufferSize	(INPUT/OUTPUT) Passes the size of the transfer buffer minus the transfer buffer header size, and returns how much data is left in the transfer buffer.
recordHeaderInfo	(OUTPUT) Returns the record header, and if they exist the scan information and the data set name list, and other data. See **NWSMSetRecordHeader** for more information about this structure.

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER
0xFFFBFFFB	NWSMUT_OUT_OF_MEMORY

Remarks

This function returns the unformatted record header information.

Before calling this function, you must set *recordHeaderInfo->scanInfo* to NULL or to an allocated NWSM_SCAN_INFORMATION structure. This buffer may be reallocated if it is to small. *recordHeaderInfo->scanInfo.bufferSize* must be set to the buffers's size. *recordHeaderInfo->scanInfo->otherInformation* is allocated by the function.

If you plan on using the scan information later on, you must copy *recordHeaderInfo->scanInformation* before calling the function again. Calling this function again, overwrites the previous information.

You must free *recordHeaderInfo->scanInformation,* when done with this function.

recordHeaderInfo->dataSetNames points to the names contained in the transfer buffer.

Note: *dataSetName* in *recordHeaderInfo* may have an invalid *bufferSize* field and must not be freed. Set the *scanInformation* pointer to NULL or to a valid NWSM_SCAN_INFORMATION structure with a valid *bufferSize*.

Field Functions

These functions parses or construct a section from a table of fields or one field at a time. These routines are:

SMDFPutFields
SMDFGetFields
SMDFPutNextField
SMDFGetNextField

SMDFPutFields and **SMDFGetFields** write a section to and read a section from a buffer by using the **SMDFPutNextField** and **SMDFGetNextField** field functions. The last two routines can build a parser. These routines do not handle buffer overflow or buffer underflow, but return a completion code indicating the condition. Buffer overflow exists when a complete field or section cannot fit into the buffer, but must span buffers.

Buffer underflow exits when the receiving buffer cannot contain a complete field or section. Three structures contain the field and other information as shown below:

```
typedef struct
{
   UINT32   fid;
   UINT64   dataSize;
   void     *data;
   UINT32   bytesTransfered;
   UINT64   dataOverflow;
} SMDF_FIELD_DATA;
```

fid contains a FID value. For more information about FIDs, see *System Independent Data Format*.

dataSize specifies the data's total size for data sizes over and including 128 bytes (i.e., if the data is 64 kb, but only 32 kb is being transferred now, this field contains the value 64 kb). The validity of this value is determined by *dataSizeMap*, and is defined later on. If *dataSizeMap* indicates that *dataSize* contains the size, then *dataSize* is valid; otherwise it contains invalid information. For data sizes under 128 bytes see *dataSizeMap*. *dataSize* is used for FIDs using size format 2. *dataSizeMap* is for data sizes in data size format 1's range.

data points to the field data.

bytesTransferred specifies the number of bytes moved into or out of a buffer.

dataOverFlow, if greater than zero, indicates the amount of data that could not be transferred into the specified buffer.

UINT64 has the following structure:

```
typedef struct
{
      UINT16 v[4];
} UINT64;
```

Parser.c contains various support routines that manage the UINT64 data type.

The second structure used by the put and get routines is shown below:

```
typedef struct
{
   SMDF_FIELD_DATA field;
   UINT32  sizeOfData;
   void   *addressOfData;
   UINT8   dataSizeMap;
   UINT8   reserved[3];
} NWSM_FIELD_TABLE_DATA;
```

Each field requires a NWSM_FIELD_TABLE_DATA structure and is used only by **SMDFPutFields**.

field is defined in structure SMDF_FIELD_DATA.

sizeOfData contains the *field->data*'s size. This field is used by the calling routine to tell **SMDFPutNextField** if all the data is being passed or if only a subset of the whole is being passed. *sizeOfData* and *field.dataSize* must contain the same value.

addressOfData returns a pointer to where the data (i.e.*field->data*) was put into the buffer. This field is valid only when the calling routine requests that it needs a pointer into the buffer to where the data was stored or where space was allocated for a field's data. See **SMDFPutFields**' remark section for more information about *addressOfData*.

dataSizeMap defines if *field->data* is less than 128 bytes or greater. If it contains 0x7F (127) or less, then *dataSizeMap* contains the size of the total data. However, if it is set to 0x80 (128), then *field->dataSize* contains the data's total size.

The third data structure is shown below:

```
typedef struct
{
   UINT32 fid;
   void *data;
   UINT32 dataSize;
   NWBOOLEAN found;
} NWSM_GET_FIELDS_TABLE;
```

NWSM_GET_FIELDS_TABLE is used as a table to contain the fields parsed from a buffer that **SMDFGetFields** received. Only **SMDFGetFields** uses this structure. Before calling **SMDFGetFields**, all fid fields must be filled with FID values that the application is expecting. For **SMDFGetFields**, the order of the table entries are not important. The settings of the structure's fields are discussed below.

fid is set to the FID value you are looking for (e.g., RECORD_SIZE).

data points to a buffer that is allocated by the calling routine. If the buffer is too small the routine returns BUFFER_OVERFLOW.

dataSize indicates the size of *data* and is set by the calling routine.

found must be set to FALSE by the calling routine. When **SMDFGetFields** returns, *found* will indicate if the field was found.

CCODE

SMDFPutFields

```
( NWSM_FIELD_TABLE_DATA table[],
  BUFFERPTR *buffer,
  UINT32 *bufferSize,
  UINT32 crcFlag);
```

Parameters

table[]	(INPUT) *table* passes a table defining a section (i.e., a header).
buffer	(OUTPUT) *buffer* passes a pointer to where *table* will be stored.
bufferSize	(INPUT/OUTPUT) *bufferSize* is initially set to *buffer*'s size. Upon return, *bufferSize* contains how much room is left in *buffer*.
crcFlag	(INPUT) Tells **SMDFPutFields** when to calculate a CRC for the field group. CRC_YES: now CRC_NO: never CRC_LATER

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Remarks

SMDFPutFields writes a section into a buffer using **SMDFPutNextField** during a back up session. It will not handle buffer overflow conditions. Buffer overflow exists when a complete field or section cannot fit into a buffer, but must span buffers.

SMDFPutFields puts an NWSM_FIELD_TABLE_DATA table that defines the section (i.e., a header or trailer) into *buffer*. The last field must be NWSM_END. The table for a block header section is shown below:

```
NWSM_FIELD_TABLE_DATA blockHeaderTable[] =
{
   { { NWSM_BLOCK_HEADER, UINT64_ZERO, &blockHeader, 0, UINT64_ZERO },
       0, NULL, 0 },
#if defined(DEBUG_CODE)
   { { DEBUG_HEADER_STRING_FIELD }, DEBUG_HEADER_FIELD },
#endif
   { { NWSM_OFFSET_TO_END, UINT64_ZERO, &offset, 0, UINT64_ZERO },
       2, GET_ADDRESS, 2 },
   { { NWSM_BLOCK_SIZE, UINT64_ZERO, &blockSize, 0, UINT64_ZERO },
       0, NULL, 0 },
   { { NWSM_UNUSED, UINT64_ZERO, &unused, 0, UINT64_ZERO },
       4, GET_ADDRESS, 4 },
   { { NWSM_SESSION_DATE_TIME, UINT64_ZERO, &session, 0, UINT64_ZERO },
       4, NULL, 0 },
   { { NWSM_SOURCE_NAME, UINT64_ZERO, sourceName, 0, UINT64_ZERO },
       strlen(sourceName), NULL, 0 },
   { { NWSM_SOURCE_TYPE, UINT64_ZERO, sourceType, 0, UINT64_ZERO },
       strlen(sourceType), NULL, 0 },
   { { NWSM_SOURCE_VERSION, UINT64_ZERO, sourceVersion, 0, UINT64_ZERO },
       strlen(sourceVersion), NULL, 0 },
   { { NWSM_END } }
};
```

Notice that there is a debug FID. This FID allows the placement of a string into the buffer to show what kind of field group is there. This is helpful when looking at a dump of the buffer or media.

GET_ADDRESS is a (void *)1 and tells **SMDFPutFields** that the calling routine needs a pointer to the data area of the field in *buffer*. Your application can use this pointer to modify the field's data area later on. *.table[x]->addressOfData* contains that pointer.

CCODE
SMDFGetFields
```
( UINT32 headFID,
  NWSM_GET_FIELDS_TABLE table[],
  BUFFERPTR *buffer,
  UINT32 *bufferSize);
```

Parameters

headFID	(INPUT) Specifies the next FID head (i.e., the first field of a section) the calling routine expects to find in *buffer* (i.e., block header, media trailer).
table	(OUTPUT) Defines the section (i.e., header) the calling routine expects to find in buffer.
buffer	(INPUT) Passes the buffer to be parsed.
bufferSize	(INPUT/OUTPUT) Passes the size of buffer and returns the buffer's size minus the size of the returned section.

Completion Codes

0x0	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFF9	NWSMUT_INVALID_FIELD_ID
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW

Remarks

SMDFGetFields reads a section from a buffer using **SMDFGetNextField** during a restore session. It does not handle buffer underflow or overflow conditions. Buffer underflow exists when a complete field or a complete section cannot be taken from a buffer.

If **SMDFGetFields** encounters a field(s) not defined in *table*, the field(s) is ignored. The table must end with field NWSM_END. An example is shown below:

```
NWSM_GET_FIELDS_TABLE blockHeaderTable[] =
{
   { NWSM_BLOCK_SIZE, &blockSize, 4, FALSE },
   { NWSM_UNUSED, &unused, 4, FALSE },
   { NWSM_SESSION_DATE_TIME, &sessionDateAndTime, 4, FALSE },
   { NWSM_SOURCE_NAME, sourceName,
       NWSM_MAX_TARGET_SRVC_NAME_LEN - 1, FALSE },
   { NWSM_SOURCE_TYPE, sourceType,
     NWSM_MAX_TARGET_SRVC_TYPE_LEN - 1, FALSE },
   { NWSM_SOURCE_VERSION, sourceVersion,
     NWSM_MAX_TARGET_SRVC_VER_LEN - 1, FALSE },
   { NWSM_END }
};
```

CCODE
SMDFPutNextField

```
( BUFFERPTR buffer,
  UINT32 bufferSize,
  SMFD_FIELD_DATA *field,
  UINT8 dataSizeMap,
  UINT32 sizeOfData);
```

Parameters

buffer	(INPUT) Passes the pointer to a buffer.
bufferSize	(INPUT) Passes the size of *buffer*.
field	(INPUT) Passes a pointer to an SMDF_FIELD_DATA structure.
dataSizeMap	(INPUT) Indicates where to find the data size.
sizeOfData	(INPUT) Specifies how many bytes of data are in *field->data*.

Completion Codes

0x0	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW
0xFFFBFFFD	NWSMUT_INVALID_PARAMETER

Remarks

SMDFPutNextField puts a field into a buffer. If *field->data* contains a subportion of the total field data, the caller must put the rest of the data into the buffer. To figure out where to put the rest of the data, the caller must track the offset into the buffer through *field->bytesTransferred*.

The value for *dataSizeMap* depends upon the amount of data in field->data. If the total amount of data to be put into a field is less than 128 bytes, then set *dataSizeMap* to the data's size. If the total amount of data is 128 bytes or more, then set *dataSizeMap* to NWSM_VARIABLE and *field->dataSize* to the total amount of data that the field will contain.

If **SMDFPutNextField** successfully completes, the caller should always check for a buffer overflow condition. If the buffer overflows, *field->dataOverflow* contains the amount of data not transferred. The caller must deal with this overflow condition

Example

```
See the example parser in System Independent Data Format's appendix.
```

See Also

SMDFGetNextField

CCODE
SMDFGetNextField

```
( BUFFERPTR buffer,
  UINT32 bufferSize,
  SMDF_FIELD_DATA *field);
```

Parameters

buffer	(INPUT) Pointer to the buffer containing the fields to be parsed.
bufferSize	(INPUT) Passes the size of *buffer*.
field	(OUTPUT) Passes a pointer to a SMDF_FIELD_DATA structure.

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW

Remarks

SMDFGetNextField parses the next field into various field components. For more information about the field types, see *System Independent Data Format*. If this function completes successfully, the caller should always check for a buffer overflow condition. If *field->data* overflows, *field->dataOverflow* contains the amount of data not transferred. The caller must deal with this overflow condition.

Example

```
See the example parser in System Independent Data Format's appendix.
```

See Also

SMDFPutNextField

FID Macros

This chapter explains the macros used by SIDF for dealing with FIDs and UINT64 values. For more information about FIDs, see the *System Independent Data Format*. This chapter refers to the following data structures:

```
typedef struct
{
   UINT16 v[4];
} UINT64;


typedef struct
{
   UINT32   fid;
   UINT64   dataSize;
   void    *data;
   UINT32   bytesTransfered;
   UINT64   dataOverflow;
} SMDF_FIELD_DATA;
```

fid is the field identifier.

dataSize indicates the size of the data and its value is determined by *"dataSizeMap"* . Please see **SMDFSizeOfFieldData** for a complete description of both of these variables.

data points to the data.

bytesTransferred refers to how many bytes were taken out of or put into a buffer.

dataOverflow specifies the amount of data (in bytes) that could not fit into a buffer or read from a buffer.

int
SMDFFixedFid
(SMDF_FIELD_DATA fid);

Parameters

fid	(INPUT) Passes the fid to be tested.

Completion Codes

!0	Non-zero if *fid* is a fixed fid (true)
0x0	Zero if *fid* isn't a fixed fid (false)

Remarks

SMDFFixedFid returns a true value (non zero) if the FID is fixed and false (zero) if it isn't. A variable "longFid" needs to be set before calling this macro. Set *longFid* to true if fid is a long fid or false (zero) if fid is a short fid. For more information about fixed FIDs, see *System Independent Data Format*. The macro is defined as follows:

```
#define SMDFFixedFid(fid) ((longFid) ?
    (((fid) AND 0xF000) == 0xF000) : ((fid) & 0x40))
```

Example

```
SMDF_FIELD_DATA fid;
UINT8 longFid;
BUFFERPTR ptr;

...

ptr = (BUFFERPTR)&fid;
if (*ptr & 0x80)  /* The FID is 2 bytes (a long fid) */
  longFid = TRUE;

else /* The FID is 1 byte */
  longFid = FALSE;

if (SMDFFixedFid(fid))
{ ... }

else
{ ... }
```

UINT32
SMDFFixedSize
(SMDF_FIELD_DATA fid);

Parameters

fid	(INPUT) Passes a field.

Completion Codes

None	

Remarks

SMDFFixedSize returns the data size for a fixed FID. Variable "longFid" must be set before calling this macro. Set *longFid* to true (one) if fid is a long fid or false (zero) if fid is a short fid. For more information about fixed FIDs see, *System Independent Data Format*. The macro is defined as follows:

```
#define SMDFFixedSize(fid)
    (1L << ((longFid) ? (*((UINT8 *)&(fid) + 1)\
    & 0x0F) : ((fid) & 0x0F)))
```

Example

```
SMDF_FIELD_DATA fid;
UINT8 longFid;
BUFFERPTR ptr;
UINT32 size;

...

ptr = (BUFFERPTR)&fid;
if (*ptr & 0x80)  /* The FID is 2 bytes (a long fid) */
  longFid = TRUE;

else /* The FID is 1 byte */
  longFid = FALSE;

size = SMDFFixedSize(fid);
```

void
SMDFPutUINT64
(UINT64 *dest,
UINT32 src);

Parameters

dest	(OUTPUT) Returns a UINT64 equivalent of *src*.
src	(INPUT) Passes the value to be converted to a UINT64.

Completion Codes

None	

Remarks

SMDFPutUINT64 converts a UINT32 value to a UINT64 value. The macro is defined as follows:

```
#define SMDFPutUINT64(dest, src)
    (*((UINT32 *)(dest) + 1) = 0, *(UINT32 *)(dest) = (src))
```

Example

```
UINT32 src;
UINT64 dest;

SMDFPutUINT64(&dest, src);
```

int
SMDFBitNIsSet
(char c);

Parameters

c	(INPUT) Passes a character.

Completion Codes

!0	True (non zero) if bit N is set.
0x0	False (zero) if bit N is not set.

Remarks

SMDFBit**N**IsSet is actually a set of macros where "N" stands for the bit to be tested. "N" ranges from 1 to 6 as shown below:

SMDFBit1Set()
SMDFBit2Set()
....

The function tests if the appropriate bit is set and returns a true value if it is set or a false value if it is not set. The macro is defined as follows:

```
#define SMDFBit1IsSet(v)
    (((v) &SMDF_BIT_XXX) == SMDF_BIT_XXX)
```

where SMDF_BIT_XXX is one of the following:

SMDF_BIT_ONE
SMDF_BIT_TWO
SMDF_BIT_THREE
SMDF_BIT_FOUR
SMDF_BIT_FIVE
SMDF_BIT_SIX

Example

```
char c;

if (SMDFBit1IsSet(c))
{ ... }

else
{ ... }
```

void
SMDFSetBitN
(char c);

Parameters

c	(OUTPUT) Passes the character to be set.

Completion Codes

None	

Remarks

SMDFSetBit**N** is actually a set of macros where "N" stands for the number of the bit to be set. "N" ranges from 1 to 6 as shown below:

SMDFSetBit1()
SMDFSetBit2()
....

The function sets the appropriate bit. The macro is defined as follows:

```
#define SMDFSetBitN(v)
    ((v) |= SMDF_BIT_XXX)
```

where "N" is a bit number from "1" to "6" and SMDF_BIT_XXX is one of the following:

SMDF_BIT_ONE
SMDF_BIT_TWO
SMDF_BIT_THREE
SMDF_BIT_FOUR
SMDF_BIT_FIVE
SMDF_BIT_SIX

Example

```
char c;

SMDFSetBit1(c);
```

int
SMDFSizeOfFID
(UINT32 fid);

Parameters

fid	(INPUT) Passes a field identifier.

Completion Codes

None	

Remarks

SMDFSizeOfFID receives a FID and returns its size in bytes. The return value does not describe the size of the data associated with the FID, but only the FID's size.

```
#define SMDFSizeOfFID(d)
    ((*((UINT8 *)&d + 3)) ? 4 :\
    ((*((UINT8 *)&d + 2)) ? 3 :\
    ((*((UINT8 *)&d + 1)) ? 2 : 1)))
```

Example

```
UINT32 fid;

SMDFSizeOfFID(fid);
```

int
SMDFSizeOfUINT32Data
(UINT32 dataSize);

Parameters

dataSize	(INPUT) Passes the size of the data.

Completion Codes

None	

Remarks

SMDFSizeOfUINT32 returns the number bytes the FID's data size descriptor occupies.

```
#define SMDFSizeOfUINT32Data(d)
    ((*((UINT16 *)&(d) + 1)) ? 4 :\
    (((*((UINT8 *)&(d) + 1))) ? 2 : 1))
```

Example

```
UINT32 fid;
int size;

size = SMDFSizeOfUINT32Data(fid);
```

int
SMDFSizeOfUINT32Data0
(UINT32 dataSize);

Parameters

dataSize	(INPUT) Passes the data's size.

Completion Codes

None	

Remarks

SMDFSizeOfUINT32Data0 calculates how many bytes the data size descriptor uses. The macro is defined as follows:

```
#define SMDFSizeOfUINT32Data0(d)
    ((*((UINT8 *)&(d) + 3)) ? 4 :\
    ((*((UINT8 *)&(d) + 2)) ? 3 :\
    ((*((UINT8 *)&(d) + 1)) ? 2 : 1)))
```

Example

```
UINT32 dataSize;
int numberOfBytes;

numberOfBytes = SMDFSizeOfUINT32Data0(fid);
```

int
SMDFSizeOfUINT64Data
(UINT64 dataSize);

Parameters

dataSize	(INPUT) Passes the data's size.

Completion Codes

None	

Remarks

SMDFSizeOfUINT64Data calculates how many bytes the size descriptor actually uses. The macro is defined as follows:

```
#define SMDFSizeOfUINT64Data(d)
    ((*((UINT32 *)&(d) + 1)) ? 8 :\
    ((*((UINT16 *)&(d) + 1)) ? 4 :\
    ((*((UINT8  *)&(d) + 1)) ? 2 :\
    ((*(UINT8   *)&(d)) ? 1 : 0))))
```

Example

```
UINT64 dataSize;

SMDFSizeOfUINT64Data(dataSize);
```

int
SMDFSizeOfFieldData
(UINT64 dataSize,
UINT8 dataSizeMap);

Parameters

dataSize	(INPUT) Passes the size of the data set.
dataSizeMap	(OUTPUT) Returns a size format 2 descriptor that indicates how many size bytes should follow it. If the function returns a zero, this parameter is not valid. The following values are returned: 0x80 2^0 size bytes follow 0x81 2^1 size bytes follow 0x82 2^2 size bytes follow 0x83 2^3 size bytes follow

Completion Codes

0x0	The data's size is between 0 and 127 bytes (size format 1 should be used). *dataSizeMap* is invalid.
0x1	The data size value occupies one byte
0x2	The data size value occupies 2 bytes
0x3	The data size value occupies 4 bytes
0x8	The data size value occupies 8 bytes

Remarks

This function sets the bits for a size format 2 descriptor (see *System Independent Data Format* for more information). *dataSizeMap* returns the size descriptor. The macro is defined as follows:

```
#define SMDFSizeOfFieldData(d, m)
    (m = 0, ((*((UINT32 *)&(d) + 1)) ? (m = 0x83, 8) :\
    ((*((UINT16 *)&(d) + 1)) ? (m = 0x82, 4) :\
    ((*((UINT8 *)&(d) + 1)) ? (m = 0x81, 2) :\
    ((*(UINT8 *)&(d) & 0x80) ? (m = 0x80, 1) : 0)))))
```

Example

```
UINT64 dataSize;
UINT8 dataSizeMap;

SMDFPutUINT64( &dataSize, 45000);
SMDFSizeOfFieldData( dataSize, dataSizeMap);
```

void
SMDFZeroUINT64
(UINT64 *a);

Parameters

a	(OUTPUT) Returns a zero value

Completion Codes

None	

Remarks

SMDFZeroUINT64 sets the variable *a* to zero. The macro is defined as follows:

```
#define SMDFZeroUINT64(a)
    (*(UINT32 *)(a) = *((UINT32 *)(a) + 1) = 0)
```

Example

```
UINT64 dataSize;

SMDFZeroUINT64(&dataSize);
```

CCODE
SMDFSetUINT32Data

```
( UINT64 *sizeOfDataSize,
  BUFFERPTR buffer,
  UINT32 *data);
```

Parameters

sizeOfDataSize	(INPUT) Passes the number of bytes the data size occupies.
buffer	(INPUT) Passes the data size.
data	(OUTPUT) Returns the data size.

Completion Codes

0x0	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: sizeOfDataSize is greater than four bytes.

Remarks

This function retrieve the data's size. *buffer* contains SIDF data and points to the data's size. *sizeOfDataSize* shows the number of bytes the data's size occupies.

CCODE
SMDFSetUINT64

```
( UINT64 *uint64,
  void *buffer,
  UINT16 sizeOfDataSize);
```

Parameters

uint64	(OUTPUT) Receives the transferred value.
buffer	(INPUT) Passes the data's size.
sizeOfDataSize	(INPUT) Passes the number of bytes data size occupies.

Completion Codes

0x0	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW: *sizeOfDataSize* is greater than 8.

Remarks

This function is used during the departing process to retrieve the data's size. *buffer* contains SIDF data and points to the data's size. This function transfers *sizeOfDataSize* bytes from *buffer* to *uint64*.

Note: If *buffer* is NULL, *uint64* is set to zero, and zero (0) is returned.

See Also

SMDFGetUINT64

CCODE
SMDFGetUINT64
```
    ( UINT64 *uint64,
      UINT32 *uint32);
```

Parameters

uint64	(INPUT) Passes the a UINT64 value.
uint32	(OUTPUT) Receives the lower 4 bytes of the UINT64 value.

Completion Codes

0x0	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW

Remarks

This function returns the lower 4 bytes of a UINT64 value. If the upper 4 bytes contain a value, SMDF_BUFFER_OVERFLOW is returned.

See Also

SMDFSetUINT64

CCODE
SMDFIncrementUINT64

```
( UINT64 *a,
  UINT32 b);
```

Parameters

a	(OUTPUT) Passes one operand and receives the additive result.
b	(INPUT) Passes the second operand.

Completion Codes

0x0	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW

Remarks

This function adds *a* to *b* and puts the result into *b*.

CCODE
SMDFAddUINT64

```
( UINT64 *a,
  UINT64 *b,
  UINT64 *sum);
```

Parameters

a	(INPUT) Passes a UINT64 value.
b	(INPUT) Passes a UINT64 value.
sum	(OUTPUT) Receives the sum of a and b.

Completion Codes

0x0	Successful
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW

Remarks

This function adds *a* and *b* and puts the result into *sum*.

CCODE
SMDFSubUINT64

```
( UINT64 *a,
  UINT64 *b,
  UINT64 *dif);
```

Parameters

a	(INPUT) Passes the left hand operand.
b	(INPUT) Passes the right hand operand.
diff	(OUTPUT) Receives result of a - b.

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW

Remarks

This function returns the results of *a* - *b*. If *b* is larger than *a*, SMDF_UNDERFLOW is returned.

CCODE

SMDFDecrementUINT64

```
( UINT64 *a,
  UINT32 b);
```

Parameters

a	(OUTPUT) Passes the value to be decremented and receives the decremented results.
b	(INPUT) Passes the amount *a* is to be decremented by.

Completion Codes

0x0	Successful
0xFFFBFFF0	NWSMUT_BUFFER_UNDERFLOW
0xFFFBFFFA	NWSMUT_BUFFER_OVERFLOW

Remarks

This function subtracts *b* from *a* and puts the results into *a*.