OMF166 Description

Rev 1.1, 11/94

Table of contents

OMF166 Description

This document describes the relevant records for absolute object files using the OMF166 format. The document is based on the OMF166 specification originated by Siemens and contains the Keil extensions to OMF166 which deal with type descriptors.

For the OMF166 base document, consult Siemens.

List of Records

The following table lists the record types contained in OMF166. Those records commented with extension are not defined in the Siemens base document and serve the purpose of project management and provide the information for high level debug.

Recordname	Recordnumber	Comment
RTXDEF	0x30	extension to Siemens OMF
DEPLST	0x70	extension to Siemens OMF
REGMSK	0x72	extension to Siemens OMF
TYPNEW	0xF0	extension to Siemens OMF
BLKEND	0x7C	
THEADR	0x80	
LHEADR	0x82	
COMMENT	0x88	
MODEND	0x8A	
LINNUM	0x94	
LNAMES	0x096	
LIBLOC	0xA8	
LIBNAMES	0xA6	
LIBDICT	0xAA	
LIBHDR	0xBA	
PHEADR	0xE0	
PECDEF	0xE4	
SSKDEF	0xE5	
MODINF	0xE7	
TSKDEF	0xE1	
REGDEF	0xE3	
SECDEF	0xB0	
TYPDEF	0xB2	
GRPDEF	0xB1	
PUBDEF	0xB3	
GLBDEF	0xE6	
EXTDEF	0x8C	
LOCSYM	0xB5	
BLKDEF	0xB7	
DEBSYM	0xB6	
LEDATA	0xB8	
PEDATA	0xB9	
VECTAB	0xE9	
FIXUPP	0xB4	
TSKEND	0xE2	
XSECDEF	0xC5	

Conventions

Record format

The OMF-Records have the basic format as shown in the example below:

```
****************************************
* RecType | RecLen | Content | CheckSum *
****************************************
```

The Record-Type field 'RecType' is the first byte in each record and identifies the record by an the 8 bit record number.

The Record-Length field 'RecLen' contains the number of bytes in the record exclusive the RecTyp and RecLen field. RecLen is a 16 Bit value.

The format of the Content field depends upon record type. The number of bytes and there layout depends upon the record type.

The Checksum field is always the last field in each record and contains the check sum, which is the 2's complement of the sum (modulus 256) of all other bytes in the record. Therefore, the sum of all bytes in a record modulus 256 equals zero.

Index values

Many of the OMF166 records use some index to refer to other records. The high order bit of the first (and possibly the only one) byte determines whether the index occupies one or two bytes. If the bit is 0, then the index is a number in range 0 to 0x7F, occupying one byte. If the bit is 1, then the index is a number in range 0x80 and 0x7FFF, occupying two bytes; the value is constructed as follows: the low order 8 bits are in the second byte, and the high order 7 bites are in the first byte.

Throughout this document, names with the suffix 'Index' specify an index of the form just described, for example GroupIndex, SectionIndex, TypeIndex.

Representation of an Address Base (BASE-FIELD)

The address base is used in various records to specify relocatable and absolute addresses (example: block-base in BLKDEF). This document refers to absolute addresses only.

The basic layout of an address field is as follows:

If both the GroupIndex (GI) and the SectonIndex (SI) are zero, then a 16 Bit frame number follows the two index fields. The frame number is interpreted as follows:

```
if (FrameNumber & 0x8000) != 0) then  PAGE Number
else                                  SEGMENT Number
```

For absolute object files, the base-field always has GI=0 and SI=0. Base fields with GI or SI not zero represent relocatable items and should be treated as an error within a loader for absolute object files.

Representation of Names (Name format)

A name is represented by the leading length of the name, which is a byte value followed by the name itself, for example:

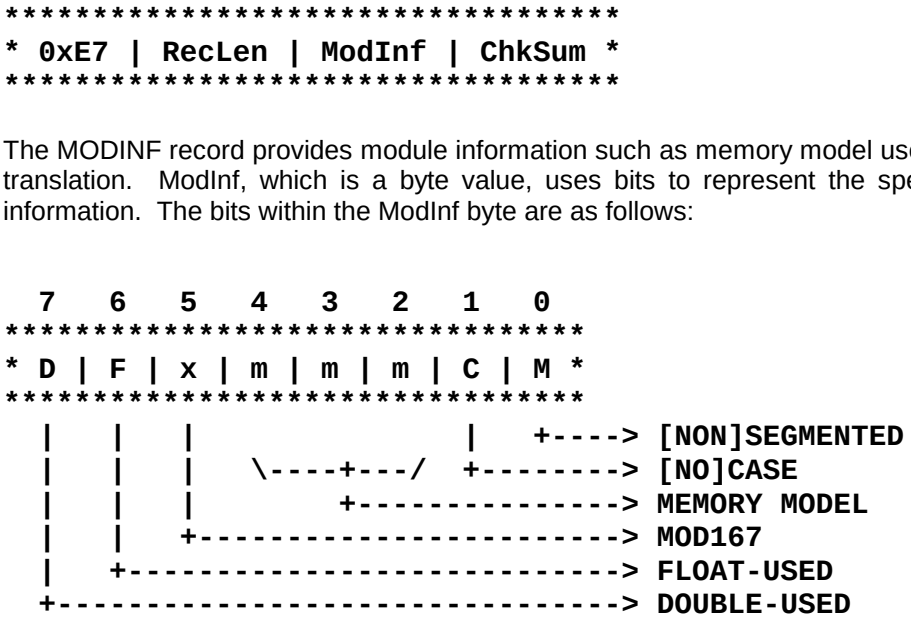
A name may represent a null name, which is denoted by a value 0 with no other bytes following the zero length name:

Note that names represented in this manner never have a null terminator as is the case with C language style strings.

Names are used in almost all symbolic debug records to specify symbolic names. Null names may be used by BLKDEF records to specify unnamed do-blocks.

Record Descriptions

MODINF-Record



The MODINF record provides module information such as memory model used in translation. ModInf, which is a byte value, uses bits to represent the specific information. The bits within the ModInf byte are as follows:

[Non]Segmented:

If bit is set, then the segmented cpu mode was choosen for the module.

[No]Case:

If bit is set, then names are to be considered case sensitive. This info is intended for the linker when combining object modules.

Memory Model:

The three bit model specifier gives the memory model choosen on translation:

1: Tiny
2: Small
3: Compact
4: Medium
5: Large

Mod167:

If bit is set, then the module is intended to be executed on an 80C167 CPU, otherwise the module is for a 80C166 CPU.

Float used:

The module contains single precision float operations. This bit is intended for the linker for automatic selection of libraries.

Double used:

The module contains double precision float operations. This bit is intended for the linker for automatic selection of libraries.

NEWTYP-Record

Each compound type will force creation of a type record, which describes the type of a variable or function. The layout of the type record is as shown:

```
*****************************///***************
* 0xF0 |   RecLen  | Type-Descriptor | Chks *
*****************************///***************
```

The NEWTYPE records are implictely numbered by sequenece, i.e. the first record has number 0, the second number one and so on. The debug records (LOCSYM, PUBDEF, ...) refer to a type record using an index, called TI (TypeIndex) for short. The index uses the general format used within OMF166 and has a special interpretation.

Interpretation of Index values in symbol records

A TI value in range 0 to 127 specifies the final type without referring to a type record. As of now, only values in range 0x40 to 0x54 are used to represent final scalar types:

Value	represented final type
0x40	**untyped**
0x41	**bit**
0x42	**char**
0x43	**unsigned char**
0x44	**int**
0x45	**unsigned int**
0x46	**long**
0x47	**unsigned long**
0x48	**float (32-Bit IEEE)**
0x49	**double (64-Bit IEEE)**
0x4A	**void**
0x4B	**label**
0x4C	**< a166 BITWORD >**
0x4D	**< a166 NEAR >**
0x4E	**< a166 FAR >**
0x4F	**< a166 DATA3 >**
0x50	**< a166 DATA4 >**
0x51	**< a166 DATA8 >**
0x52	**< a166 DATA16 >**
0x53	**< a166 INTNO >**
0x54	**< a166 REGBANK >**

The types prefixed by **a166** are generated by the assembler A166. These types are special to the assembler and are not created by the C-compiler.

Application of the Type-Index 'TI'

When analyzing a type index, the following rules apply:

A type index within the debug symbols records is interpeted as follows:

> If the index value is lower then 128, it represents one of the final types (basic scalar type).

An index value greater than 127 refers to a type descriptor respresented by a type record. Subtract 128 from the TI value to get the sequence number of the type record which TI refers to. The selected type record must be analyzed to get the type description (recursive analysis).

> ### Note
>
> *A type record may again contain type indices (TI's) which refer to previous (or following) type records. Indices contained within type records are always 16-Bit, even if such an index represents a final scalar type !*

Compound type descriptors

The following sections describe the various type descriptors. For convention, the numeric suffix 8 specifies a byte, 16 specifies a word and 32 specifies a double word.

COMPONENT-LIST Descriptor

Specifies the number of components (NrOfComp16). Used in function and structure types.

```
*****************************///********
* 0x20 | NrOfComp16 | Components [*] *
*****************************///********
```

Each component is described as follows:

```
+------+--------+------+------+----------+
| TI16 | OFFS32 | REP8 | POS8 | n,'name' |
+------+--------+------+------+----------+
```

TI16: members type index

OFFS32: members offset

REP8:

relevant on function parameter lists, otherwise REP8 and POS8 will be zero. The possible values are as follows:

1: RegBit, POS8=BitPos (0-15), OFFS32=RWn (0-15)
2: StackVar (auto/parameter) OFFS32=StackOffs ([R0+#n])
3: RegVar (auto/parameter) OFFS32=RWn (0-15)

The register number 'RWn' is contained in OFFS32, which is actually interpreted as a 16 bit word. The value 0 represents R0, 1 R1 and so on.

NAME: member name in OMF166 name format

POS8: contains a bit position if REP8 contains method 1 (RegBit)

POINTER Descriptor

The Pointer descriptor is used to describe the type which a pointer refers to and specific attributes of the pointer:

```
*********************************************
* 0x21 | SIZE8 | ATTRIB8 | RESERVED16 | TI16 *
*********************************************
```

SIZE8: the size of the pointer (either 16 or 32 bits)

ATTRIB8: 1 = Data pointer (PAGE:OFFSET)
2 = Function pointer (SEG:OFFSET)
4 = Huge pointer (linear 32-Bit)
8 = Xhuge pointer (linear 32-Bit)

RESERVED16: reserved, set to zero.

TI16: reference to referred type.

The TI16 refers to the final type or another type record. For example, if TI16 contains 0x4A, which is the type 'void', then the meaning is '**void ***'. The SIZE8 specifier will define further details of the type, for example '**void near ***' or '**void far ***'.

The ATTRIB8 byte defines the interpretation of a pointer. Data pointers use the PAG:POF convention, where PAG is the page number of a physical 16k page and POF is the offset within the page. Function pointers use the SEG:SOF convention which specifies the 64k segment (SEG) and a segment-offset (SOF). Huge pointers use linear addressing undergoing the paged addressing scheme of the 80C166 CPU.

ARRAY Descriptor

An Array descriptor is used to describe array types:

```
*********************************************
*0x22 | DIMS8 | ATTRIB8 | TI16 | DIMSZ32 [*] *
*********************************************
```

DIMS8: number of array dimensions

ATTRIB8: 1 = Huge-Array (0 ... 64K)
2 = Xhuge-Array (0 ... 16MByte)

TI16: refers to the type which the array consist of

DIMSZ32: the dimension size of each dimension

The DIMSZ32 field contains 'DIMSZ8' repeated sizes of the dimensions. A special case is a DIMSZ32 field containing -1L, which specifies an array dimension of unknown size. This may be the case on external arrays when the size is not known to the translator.

Example: the array declaration 'int array [5][3][2];' creates the type

```
| 0x22 | 3 | 0 | 0x44 | 5 | 3 | 2 |
```

Hint for the Linker:

The declaration	extern char array[];
creates the type descriptor	0x22,1,0,0x42,-1L

The linker should replace the incomplete type by the type of the corresponding PUBDEF/GLBDEF symbol which represents the exact type of the array. Since one of the input modules to the linker must have the complete type, the final output module from the linker should not contain any incomplete types.

FUNCTION Descriptor

A Function type descriptor is used to describe a function return type and the types of the parameters of the function:

```
**********************************************
* 0x23 | ATTRIB8 | RTYPE-TI16 | PARMLIST-TI16 *
**********************************************
```

ATTRIB8: 1 = Near-Function
2 = Far-Function

RTYPE-TI: TypeIndex of the function return type

PARMLIST-TI: TypeIndex of the parameter list (a component list)

Functions without parameters and with return type int/uint/void will not create a function type descriptor at all. Such functions will be represented by a TI value of 0x4B which means 'label'. The intention of this short form is to avoid unnecessary descriptors with almost no information. This has no impact on local variables of such a function.

The following two examples show the case when the short form is used:

```
int test () { ... }          // no params
int test (void) { ... }      // no params
```

STRUCT/UNION Descriptor

The Struct/Union descriptor is used to describes the details of structure and unions:

```
**************************************************
** 0x24 | ATTRIB8 | SIZE32 | MEMBER-TI16 | tagname *
**************************************************
```

ATTRIB8: 1 = struct, 2 = union

SIZE32: sizeof struct or union

MEMBER-TI16: reference to component list or <void>

tagname: struct/union-tag name in OMF166 name format

The member TI may be void on structures or unions which do not have defined the members. Such a descriptor should be replaced by the linker with the complete type which is probably defined in another module.

BITFIELD Descriptor

The Bitfield descriptor is used to describe ANSI-C style bit fields:

```
**********************************
* 0x25 | TI16 | OFFSET8 | WIDTH8 *
**********************************
```

TI16: base scalar type of the field [uchar, uint, long]

OFFSET8: field offset in base scalar in bits

WIDTH8: field width in bits

DEPLST-Record

The DEPLST record is used to describe the dependency list of the module. This information is used by the automatic project maintenance utility **AutoMAKE** for recreation of projects.

```
********************///**********
* 0x70 | RecLen |  Info  | ChkS *
********************///**********
```

The DEPLST records describes the components, which the current module consist of. The current module may be one single object file or a completely bound application consisting of many object files.

The Info field delivers all information necessary to recreate one or more components of the module and has the follwing format:

```
  iTyp   Mark8   Time32   Name(s)
+------+-------+--------+-------///--------+
| 0x00 | mark8 | time32 | Path_OutputFile  |
+------+-------+--------+-------///--------+
+------+-------+--------+-------///--------+
| 0x01 | mark8 | time32 | Path_InputFile   |
+------+-------+--------+-------///--------+
+------+-------+--------+-------///--------+
| 0x02 | mark8 | time32 | Path_IncludeFile |
+------+-------+--------+-------///--------+
+------+-------+--------+-------///--------+
| 0x03 | mark8 | time32 | Path_CommandFile |
+------+-------+--------+-------///--------+
+------+-------+--------+-------///--------+
| 0x04 | mark8 | time32 | ObjInputFile     |
+------+-------+--------+-------///--------+
+------+------///--------+
| 0xFF | Invocation_Line |
+------+------///--------+
```

iTyp: specifies the type of the dependency descriptor

iTyp-0: Outputfile descriptor. Specifies path and name of the output file created by a translator or linker

iTyp-1: Inputfile descriptor. Specifies path and name of the input file to the translator.

iTyp-2: Includefile descriptor. If the Input file contains more than one include file, then each include file is listed with an iTyp-2 descriptor.

iTyp-3: Commandfile descriptor; used when @file was given in the invocation line.

iTyp-4: Object-Inputfile descriptor. Used to specify an object file as input for L166.

iTyp-5: Commandline descriptor. Contains the invocation line to the translator including all invocation controls.

Mark8: Byte, required to be zero.

Time32: File creation date in Microsoft's 'fstat()' format.

Name(s): specifies the Pathname of one file. In case of iTyp 4, more than one pathname may be specified.

REGMSK-Record

This record is used to describe the register usage of one or more functions. Note that this record is created only by the C166 compiler and updated by the linker L166. The record is used to perform apllication wide register optimization by recoloring registers use in functions by means of retranslations.

```
*****************************///**************
* 0x72 |   RecLen  | RegMask [...]   | Chks *
*****************************///**************
```

One RegMsk-Record may contain zero, one or more RegMask descriptors. The layout of the RegMask field is as follows:

```
+------+-------+-----+
|  E8  |  R16  |  N  |
+------+-------+-----+
```

E8: 0 = Public Function (Regmask is definitely known)
1 = External Function (Regmask is assumed to a pessimistic value)

R16: 16-Bit Value RegisterMask description.

N: Name of the function which 'R' belongs in OMF166 name format.

Consult the compiler documentation for details on the format of the R field.

VECTAB-Record

```
*****************************************///*********
*
* 0xE9 | RecLen | ABS-Address | DatTyp |  Data | Chks
*
*****************************************///*********
*
```

This record provides contiguous data, which represents the interrupt vector table.

The ABS-Address field has the following format:

```
+----------------+------------+-------------+
| SegmentNumber8 | OffsetLow8 | OffsetHigh8 |
+----------------+------------+-------------+
```

The segment number specifies the segment, which is in range 0 to 3 for the 80C166 and 0 to 256 for the 80C167.

The 'DatTyp' field is a byte and may have the following values:

0: BIT
1: DATA
2: CODE
3: CONST

Note that DatTyp values 0 and 1 do not apply to the INTVEC record.

The 'Data' field provides consecutive bytes of vector table image. The number of bytes are the rest of the record not counting the checksum field.

PEDATA-Record

```
*****************************************///*********
*
* 0xB9 | RecLen | ABS-Address | DatTyp |  Data | Chks
*
*****************************************///*********
*
```

This record provides contiguous data, from which a portion of a memory image is to be constructed.

The ABS-Address field has the following format:

```
+----------------+------------+-------------+
| SegmentNumber8 | OffsetLow8 | OffsetHigh8 |
+----------------+------------+-------------+
```

The segment number specifies the segment, which is in range 0 to 3 for the 80C166 and 0 to 256 for the 80C167.

The 'DatTyp' field is a byte and may have the following values:

0: BIT
1: DATA
2: CODE
3: CONST

The 'Data' field provides consecutive bytes of the memory image. The number of bytes are the rest of the record not counting the checksum field.

BLKDEF-Record

```
******************** ///
************************************
* 0xB7 | RecLen | BlockBase | BlockInfo | PInfo | TI |
ChkSum *
******************** ///
************************************
```

This record provides information about blocks that were defined in the source program input to the tanslator which produced the module. A BLKDEF record will be generated for each procedure and each block that contains variables. The purpose of this information is to specify the live range (scope) of the debug symbol record(s) enclosed by a BLKDEF and a BLKEND record.

BLKDEF records may be nested. Each BLKDEF record is matched by a corresponding BLKEND record in the the same nesting level. The maximum nesting is 32 (introduced by the C166 compiler).

The sequence of BLKDEF records defines the implicite number of each BLKDEF record. This sequence number may be referred to by a BlockIndex, as may be the case in DEBSYM records.

The BlockBase field has the format shown below and is used to specify the starting address of some block:

```
+------------+-------------+ - - - - - - +
| GroupIndex | SectionIndex | FrameNumber |
|            |              |  (optional) |
+------------+-------------+ - - - - - - +
```

The BlockInfo field has the following format:

```
+--------+---------------+---------------+
|  NAME  | BlockOffset16 | BlockLength16 |
+--------+---------------+---------------+
```

NAME: is the block name. If the record describes an unnamed block, then a null name is used.

BlockOffset16: is a 16 Bit value which is the offset of the first byte of the block with respect to the referent value specified by 'BlockBase'.

BlockLength16: this field gives the length of the block in bytes.

The PInfo field has the following format:

```
+-------------+---------+
|P|Z|Z|Z|Z|Z|Z|  Null16 |
+-------------+---------+
    (Byte)       (Word)
```

P: is the high order Bit of the first byte. If the bit is set, then the BLKDEF record was generated from a procedure.

Z: indicates an unused bit. These bits are required to be zero.

Null16: a 16 bit value always representing zero.

TI: The TypeIndex field represents a final type or refers to a previous NEWTYP record by sequence, depending upon the type index value.

BLKEND-Record

```
**************************
* 0x7C | RecLen | ChkSum *
**************************
```

This record, together with the BLKDEF record provides information about the scope of variables of a source program. Each BLKDEF is matched by a BLKEND record. The order of BLKDEF, debug symbol record(s) and BLKEND reflects the order of declarations in the source module.

LINNUM-Record

```
************************************************************
**
* 0x94 | RecLen | AddressBase | LinNum16 | Offset16 | ChkSum
*
************************************************************
**
                              |                     |
                              +-----> repeated <----+
```

This record provides the correspondence between line number of a source program and the associated object code created by a translator.

Since several modules may be linked together to form an output module, the line numbers have to be associated to some source or list file. This file is identified using a comment record with comment type 'K'. The comment record is preceeded by a THEADR record which signals the start of a module within the object file.

AddressBase: Specifies the address base of the following line numbers using the base address format.

LinNum16: gives the line number in range 0 to 32767. The most significant bit is reserved for future use and is always zero.

Offset16: gives a 16 bit value which represents the line number offset of the line number with respect to the address base given by the AddressBase field.

LOCSYM-Record

```
*********************************************************
* 0xB5 | RecLen | Base | Name | Ofs16 | Rep8 | TI | ChkSum *
*********************************************************
                       |                          |
                       +-------> repeated <-------+
```

This record provides information about symbols that were used in the source program. The purpose of this information is to specify the name, offset and type of one or more variables.

Base: specifies the local address base for the following symbolic formation using the base address format.

TI: the TypeIndex field represents a final type or refers to a previous NEWTYP record by sequence, depending upon the type index value.

Name: represents the symbol name.

Ofs16: is a 16 Bit offset of the symbol with respect to the referent value pecified by 'LocBase'.

REP8: a byte specifying the representation value as follows:

```
Bit:    7   6   5   4   3   2   1   0
      +---+---+---+---+---+---+---+---+
      | V |    REP    |     bpos      |
      +---+---+---+---+---+---+---+---+
```

V: represents the sign of the value stored in the local symbol offset 'Offset16'. V=0 means a positive, V=1 a negative value.

REP: these three bits encode the representation type of the Offset16 field as follows:

0: BIT - the symbol is a bit symbol. The 'bpos' field contains the position of the bit in the bitaddressable word. If V=1, then the Offset16 field specifies a register (0=R0, 1=R1, 15=R15).

1: VAR - the symbol is a variable, whose type is specified with the type index.

2: LAB - the symbol represents a label or procedure.

3: REGBANK - the symbol represents the name of a register bank. 'Offset16' is an address relative to segment zero.

4: INTNO - the symbol represents a symbolic interrupt number.'Offset16' is the absolute interrupt number

5: CONST - the symbol represents the numeric constant given by Offset16.

6: REGVAR - the symbol represents a register variable. The register number is defined by the Offset16 field. The type of the variable given by TypeIndex decides the interpretation of the register number (WORD or BYTE register).

7: AUTO - the symbol represents a an automatic variable, which are located on the stack. Automatics are relative to R0 with an offset given by Offset16 [R0+Offset16]).

GLBDEF-Record

```
*************************************************************
* 0xE6 | RecLen | Base | Name | Ofs16 | Rep8 | TI | ChkSum *
*************************************************************
                      |                           |
                      +-------> repeated <-------+
```

This record provides information about global symbols. The representation of the fields are exactly the same as shown in the LOCSYM record. The purpose of this record is to specify the name, offset and type of one or more global variables.

DEBSYM-Record

```
*************************************************************
***
* 0xB6 | RecLen | FrameInfo | Name | Ofs | REP8 | TI |
ChkSum *
*************************************************************
***
                            |                         |
                            +------> repeated <------+
```

This record provides information about all local symbols including stack based symbols. The sy,bols in the record were originally defined in the source module of name given by the most recently preceeding T-MODULE HEADER record.

The scope of the symbols in the record is defined to be the most recently preceding BLKDEF whose extent has not yet been closed by a BLKEND record. If no such BLKDEF exists, then the symbols are defined at file level of the module identified by the most recent THEADER record.

FrameInfo:

This field gives information about the frame of the symbols defined in the record. It's format is as follows:

```
*********************
* FRAMEINFO | DATUM *
*********************
```

The FRAMEINFO field is a byte containing the following fields:

```
  7   6   5   4   3   2   1   0
+-------------------------------+
| B | L | 0 | 0 | 0 | f | r | m |
+-------------------------------+
```

B: based bit, not used.
L: long value bit, not used.
frm: 3 bit field representing the frame method used for the symbol.

Frame method 0: DATUM specifies an address base field (GI,SI,Framenumber)
Frame method 2: DATUM specifies a BLKDEF index.
All other frame methods are illegal in absolute object files.

Note: A DEBSYM record whose FRAMEINFO field is 0 is functionally equivalent to a LOCSYM record.

Name: this field provides the symbol name in name format.

Ofs: this 16 bit field provides the offset with respect to the base given by DATUM. In case of frame method 2, Ofs is the byte offset in the activation record of the block specified by the block index.

REP8: a byte specifying the representation value as follows:

```
Bit:    7   6   5   4   3   2   1   0
      +---+---+---+---+---+---+---+---+
      | V |    REP    |     bpos      |
      +---+---+---+---+---+---+---+---+
```

V: represents the sign of the value stored in the local symbol offset 'Offset16'. V=0 means a positive, V=1 a negative value.

REP: these three bits encode the representation type of the Offset16 field as follows:

0: BIT - the symbol is a bit symbol. The '***bpos***' field contains the position of the bit in the bitaddressable word. If V=1, then the Offset16 field specifies a register (0=R0, 1=R1, 15=R15).

1: VAR - the symbol is a variable, whose type is specified with the type index.

2: LAB - the symbol represents a label or procedure.

3: REGBANK - the symbol represents the name of a register bank. 'Offset16' is an address relative to segment zero.

4: INTNO - the symbol represents a symbolic interrupt number.'Offset16' is the absolute interrupt number

5: CONST - the symbol represents the numeric constant given by Offset16.

6: REGVAR - the symbol represents a register variable. The register number is defined by the Ofs field. The type of the variable given by TypeIndex decides the interpretation of the register number (WORD or BYTE register).

7: AUTO - the symbol represents a an automatic variable, which are located on the stack. Automatics are relative to R0 with an offset given by Ofs [R0+Ofs]).

TI: NEWTYP the TypeIndex field represents a final type or refers to a previous record by sequence, depending upon the type index value.

COMENT-Record

```
*****************************///*************
* 0x88 | RecLen | ComTyp | Comment | ChkSum *
*****************************///*************
```

ComTyp: This field indicates the type of comment carried out in this record. The field consists of two bytes as follows:

Byte 1:

```
  7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+
| N | bits - 0 to 6 not used    |
+---+---+---+---+---+---+---+---+
```

N: NOPURGE bit; 1 = comment may not be purged from the file

Byte 2:

```
+---+---+---+---+---+---+---+---+
|    Comment - Class            |
+---+---+---+---+---+---+---+---+
```

The comment class is a byte encoding the meaning of the comment. As of now, the following values are defined:

0: Language translator comment
'K': comment specifies the name of the input file

Comment: this field provides the commentary text. Note that this text string does not have a leading length specifier.

THEADR-Record

```
**************************************
* 0x80 | RecLen | T-Module Name | ChkS *
**************************************
```

Every module output from a translator must have a THEADR-Record. It's purpose is to provide the identity of the original defining module for all line numbers and symbols. The T-Module Name represents the module name assigned by the translator.

LHEADR-Record

```
**************************************
* 0x82 | RecLen | L-Module Name | ChkS *
**************************************
```

Every module output from the linker must have a LHEADR-Record. It's purpose is to identify a module that has been processed by the linker but is still subject to be input to the linker (incremental linking). The L-Module Name represents the module name assigned by the linker.

PHEADR-Record

```
**************************************
* 0xE0 | RecLen | P-Module Name | ChkS *
**************************************
```

Every absolute module output from the linker must have a PHEADR-Record. It's purpose is to identify a module that has been processed by the linker.

MODEND-Record

```
******************************
* 0x8A | RecLen | 00H | Chks *
******************************
```

This record denotes the end of a module.

XSECDEF-Record

The XSECDEF record is almost identical to the OMF166 SECDEF record with minor changes to represent sections which are bigger than 64K.

Changes have been made in the 'SecTyp' field and the 'SecLen' field, the remaining fields and meanings are left unchanged.

```
****************************************** ///
************
* 0xC5 | RecLen | SecTyp | SecAtr | Seclen |         |
ChkSum * ****************************************** ///
************
```

SecTyp Field:

```
Bit-7       .....   Bit-0
************************
* Type | X | H | bitpos *
************************
```

The 'Type' field is two bits and specifies the type of the section as follows:
0:=BIT, 1:=DATA, 2:=CODE, 3:=CONST

The 'X' bit is set if the section is of type 'xhuge' (length 0 ... 16M).
The 'H' bit is set if the section is of type 'huge' (length 0 ... 64K).
The 'bitpos' field has the same meaning as defined in the Siemens OMF166 spec.

SecLen-Field:

The SecLen field is now a 32 bit value which represents the length of the section.
The SecLen field of the OMF166-Secdef record is only 16 Bits.