

# Guide to the Open Structure Definition Language Utility

---

*This manual describes the Open Structure Definition Language (OpenSDL) and the OpenSDL translator.*

<b>Revision/Update Information:</b>	This revised manual supersedes HP and VAX SDL (Structure Definition Language) Version 3.2.
<b>Operating System and Version:</b>	Cygwin 2.11.2 or later
<b>Software Version:</b>	OpenSDL Version 3.4

**Jonathan D. Belanger**

---

This manual is for the OpenSDL Language and Processor Utility, 3.4

Copyright © 1985, 1987, 1989, 2007, 2019

Permission is granted to the source code and this manual, with no warrantee of any kind, implied or otherwise. This manual is for OpenSDL (version 3.4, February 2019).

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

**First Printing, June 1981**

**Revised, June 1985**

**Revised, November 1987**

**Revised, May 1989**

**Revised, October 2007**

**Revised, February 2019**

Published by Digital Equipment Corporation, Compaq Computer Corporation, Hewlett-Packard Corporation, VMS Software, Inc., and Jonathan D. Belanger

# Table of Contents

<b>Examples</b>	<b>1</b>
<b>Figures</b>	<b>3</b>
<b>Tables</b>	<b>5</b>
<b>Preface</b>	<b>7</b>
Intended Audience	7
Document Structure	7
Associated Documents	7
Conventions	8
Software and Documentation Reporting and Distribution	8
Summary of Technical Changes	9
New Features	10
<b>1 Overview</b>	<b>11</b>
1.1 The OpenSDL Translation Process	11
1.2 Description of a Sample OpenSDL Source File	12
<b>2 Creating, Editing, and Processing an     OpenSDL Source File</b>	<b>15</b>
2.1 Creating an OpenSDL Source File with Notepad++	15
2.1.1 Entering Source Code	16
2.1.2 Getting Started with OpenSDL	16
2.1.3 Compiling Source Code	36
2.2 Processing an OpenSDL Source File	36
<b>3 OpenSDL Language Elements</b>	<b>41</b>
3.1 Names	41
3.1.1 Local Symbol Names	42
3.1.2 Source Program Identifiers	42
3.2 Keywords	43
3.2.1 Declaration Keywords	43
3.2.2 Declaration Modifier Keywords	44
3.2.2.1 User-Specified TYPENAME keyword	49
3.2.2.2 PREFIX, MARKER, and TAG Keywords	49
3.2.2.3 RADIX Keyword	53
3.2.2.4 Alignment Keywords	53
3.2.2.5 Storage Class Keywords	54
3.2.2.6 OpenSDL Storage Classes and Typedef Syntax	55

3.2.2.7	Data Types .....	55
3.2.2.8	DIMENSION Keyword .....	56
3.2.3	Data Type Keywords .....	57
3.2.3.1	Pointer Data Types .....	57
3.2.3.2	ANY Data Type .....	57
3.2.3.3	BITFIELD Data Type .....	58
3.2.3.4	BOOLEAN Data Type .....	58
3.2.3.5	CHARACTER Data Type .....	59
3.2.3.6	COMPLEX Data Types .....	59
3.2.3.7	DECIMAL Data Type .....	60
3.2.3.8	Floating Point Data Types .....	60
3.2.3.9	Integer Data Types .....	60
3.3	Expressions .....	61
3.4	Local Comments .....	62
3.5	Output Comments .....	62
3.5.1	Line Comments .....	63
3.5.2	Block Comments .....	63
3.6	INCLUDE Statement .....	63
3.7	Conditional OpenSDL Compilation .....	64
3.7.1	Conditional OpenSDL Compilation using the IFLANGUAGE statement .....	64
3.7.2	Conditional OpenSDL Compilation using the IFSYMBOL statement .....	65
3.8	Text Pass-through .....	66
3.9	DECLARE Statement .....	67
<b>4</b>	<b>OpenSDL Declarations .....</b>	<b>71</b>
4.1	MODULE Description .....	71
4.2	MODULE Format .....	72
4.3	ITEM Description .....	73
4.4	ITEM Format .....	73
4.5	AGGREGATE Description .....	75
4.5.1	Subaggregate Declaration .....	75
4.5.2	STRUCTURE Declaration .....	76
4.5.3	UNION Declaration .....	76
4.5.4	Implicit Union Declarations .....	76
4.5.5	Implicit Union Declarations with the Optional DIMENSION ..	77
4.5.6	Forcing Negative Offsets .....	78
4.5.7	Forcing Data Alignment .....	78
4.5.8	Using Offset Symbols .....	80
4.6	AGGREGATE Format .....	81
4.7	CONSTANT Description .....	86
4.8	CONSTANT Format .....	88
4.9	ENTRY Description .....	91
4.10	ENTRY Format .....	91
<b>Appendix A OpenSDL Diagnostic Messages ...</b>		<b>95</b>

<b>Appendix B</b>	<b>OpenSDL Language</b>	
	<b>Translation Summaries .....</b>	<b>103</b>
	C/C++ Translation Summary .....	103
<b>Appendix C</b>	<b>ASCII Character Set .....</b>	<b>107</b>
<b>Index</b> .....		<b>111</b>



## Examples

Example 1.1: Sample OpenSDL Source File .....	12
Example 1.2: C Output File for the Sample OpenSDL Source File .....	14
Example 2.1: Creating a Sample OpenSDL Source File .....	17
Example 3.1: PREFIX, MARKER, and TAG Example .....	50
Example 3.2: PREFIX on AGGREGATE Example .....	52
Example 3.3: PREFIX on AGGREGATE C/C++ Output .....	52
Example 3.4: Conditional Compilation Example .....	65
Example 4.1: Subaggregate Example .....	75
Example 4.2: Implicit Union Example .....	76
Example 4.3: DIMENSION Example .....	77
Example 4.4: DIMENSION Output .....	78
Example 4.5: ORIGIN Example .....	78
Example 4.6: Alignment Example .....	79
Example 4.7: Byte Offset Symbol Example .....	81





## Figures

Figure 1.1: OpenSDL Translation Process.....	11
Figure 2.1: Openning File .....	15
Figure 2.2: Module Statement .....	18
Figure 2.3: Output Comment.....	19
Figure 2.4: Local Variable .....	20
Figure 2.5: Constant Start.....	21
Figure 2.6: Constant List .....	22
Figure 2.7: Constant First .....	23
Figure 2.8: Constant Last .....	24
Figure 2.9: Aggregate Start .....	25
Figure 2.10: More Aggregate Information .....	26
Figure 2.11: Aggregate Member .....	27
Figure 2.12: Aggregate Middle.....	28
Figure 2.13: Aggregate Almost Done.....	29
Figure 2.14: Aggregate Last .....	30
Figure 2.15: Aggregate End .....	31
Figure 2.16: Offset .....	32
Figure 2.17: Constant Offset.....	33
Figure 2.18: Item First .....	34
Figure 2.19: Item Last.....	35



## Tables

Table 2.1: OpenSDL Command Qualifiers and Their Defaults .....	37
Table 2.2: OpenSDL Output Language Options and File Types .....	38
Table 3.1: Keywords That Identify or End Declarations .....	43
Table 3.2: MODULE Declaration .....	44
Table 3.3: Item Declaration .....	44
Table 3.4: AGGREGATE Declaration .....	45
Table 3.5: CONSTANT Declaration .....	46
Table 3.6: ENTRY Declaration .....	47
Table 3.7: PARAMETER Declaration .....	47
Table 3.8: PARAMETER Modifiers .....	47
Table 3.9: RETURN Value Keywords .....	48
Table 3.10: ENTRY Description Keywords .....	48
Table 3.11: Default Tags Used by OpenSDL .....	51



## Preface

This manual describes the Open Structure Definition Language (OpenSDL) and the OpenSDL translator for use on any compatible operating systems. OpenSDL source code can be translated to output files in one or more target programming languages. OpenSDL is suitable for systems and application programming environments that use executable programs consisting of modules written in one or multiple programming languages.

## Intended Audience

This manual is intended for users who are familiar with one or more programming languages and who are currently involved in the design and development of multilanguage programming applications; however, users are not required to have previous experience with OpenSDL in order to use this manual.

## Document Structure

This manual contains the following chapters and appendixes.

Chapter 1	provides a brief overview of OpenSDL and the translation process.
Chapter 2	describes how to create, edit, and process an OpenSDL source file.
Chapter 3	describes the OpenSDL language elements that compose OpenSDL declarations.
Chapter 4	describes the function and format of OpenSDL declarations.
Appendix A	provides a list and descriptions of OpenSDL diagnostic messages.
Appendix B	shows translation summaries for all output languages supported by OpenSDL.
Appendix C	is a table showing the ASCII character set.

If OpenSDL is installed on your system, an online copy of this manual is contained in the **help/manpage** folders. You can print a copy of the manual file on a printer.

## Associated Documents

If OpenSDL is installed on your system, an online example of an OpenSDL source file is contained in **example.sdl** in the public directory/folder **/usr/lib/opensdl/examples**.

## Conventions

The following conventions are used in this document.

Convention	Meaning
<code>{ STRUCTURE }</code> <code>{ UNION }</code>	Stacked items within braces indicate that you must select one of the items.
<code>[ ]</code>	Simple square brackets indicate that the enclosed item(s) are optional.
<code>[ COMMON ]</code> <code>[ GLOBAL ]</code>	Stacked items within brackets indicate that only one item may be selected.
<code>MODULE name;</code>	Names shown in uppercase letters in examples and format descriptions are OpenSDL keywords that must be entered as shown. Names and syntactic elements shown in lowercase letters represent user-specified names and identifiers.
<code>arg,...</code>	A comma followed by an ellipsis means that the preceding item may be repeated one or more times, with commas separating two or more items.
<code>.</code> <code>.</code> <code>.</code>	A vertical ellipsis in an example or figure indicates that not all the statements or elements are shown.
<b>common storage</b>	Boldface words in text are used to introduce or define a new term, or to refer to a term used in a code example.
<code>\$ opensdl user.sdl</code>	In interactive examples, user input is <b>bold and mon-spaced</b> .

## Software and Documentation Reporting and Distribution

If OpenSDL is installed on your system, an online copy of this manual is contained in the `/usr/lib/opensdl/doc` folder.

## Summary of Technical Changes

The technical changes for this version of OpenSDL are described in the online release notes file `opensdl1034.release_notes` in `/usr/lib/opensdl/doc`.

The following is a summary of technical changes in OpenSDL Version 3.4. (For detailed descriptions and a summary of bug fixes, see the online release notes file.)

<b>Change</b>	<b>Description</b>
<b>DIMENSION *</b>	now yields <code>name[]</code> instead of <code>name[1]</code> in C language output.
<b>Output header</b>	is optionally included at the beginning of an output file now indicates that the file was created by OpenSDL instead of “VAX”, or “Open-VMS”, or “HP”, or “Compaq” SDL.
<b>DEFAULT</b>	This clause may now be specified for a parameter of an <b>AGGREGATE</b> type (i.e. <b>STRUCTURE</b> or <b>UNION</b> ).

## New Features

The following is a summary of new features in OpenSDL Version 3.4. (For detailed descriptions, see the online release notes file `opensdl034.release_notes` in `/usr/lib/opensdl/doc`.

Feature	Description
User-defined data types	OpenSDL provides the <code>TYPEDEF</code> keyword to allow you to define additional data types in some languages. <code>TYPEDEF</code> behaves like a Storage Class.
Entry point return types	This feature extends the syntax of the <code>ENTRY</code> statement to allow you to specify a user-defined data type as a return type.
<code>CONSTANT</code>	now includes support for string constants.
Conditional compilation	OpenSDL now allows you to compile a section of OpenSDL code conditionally, depending on whether output is being generated for a particular language or not.
Text pass-through	OpenSDL provides this feature to allow you to pass literal text through to the output language file without translation. This feature allows language-specific constructs that cannot be represented in OpenSDL to be emitted.
<code>DECLARE</code> statement	OpenSDL provides this statement to allow you to declare a data item of a type that you define, which may be unknown in the current OpenSDL compilation.
<code>--suppress</code> qualifier	OpenSDL provides this command line qualifier to allow you to suppress the addition of prefixes and/or tags to names.
<code>RADIX</code> modifier	OpenSDL provides this modifier on <code>CONSTANT</code> definitions to allow constant values to be written in Decimal ( <code>DEC</code> ), Octal ( <code>OCT</code> ), or Hexadecimal ( <code>HEX</code> ) format.
<code>ENUMERATE</code> modifier	OpenSDL provides this modifier on <code>CONSTANT</code> definitions to generate enumerations instead of constants. If the output language does not support enumerations, this modifier is ignored.
<code>/+</code> , <code>//</code> , and <code>/-</code>	These output comment characters can be used to define a comment block.



# 1 Overview

The Open Structure Definition Language (OpenSDL) is used to write source statements that describe data structures and that can be translated to source statements in other languages. You can include the resulting output files in a corresponding target language program for subsequent compilation.

Because OpenSDL is compiler- and language-independent, it is particularly useful for maintaining multilanguage implementations. For example, you can create and later modify a single OpenSDL source file that can be translated to multilanguage output files; any number of these output files can then be included in one or several multilanguage programming applications.

OpenSDL supports the following OpenVMS language:

- C/C++

## 1.1 The OpenSDL Translation Process

The translation of an OpenSDL source file occurs when you issue the OpenSDL command (Section 2.2 [Processing an OpenSDL Source File], page 36). The OpenSDL command activates the OpenSDL “front-end” translator (`opensdl.exe`), which is stored in the `/usr/bin` directory/folder. The front end parses the OpenSDL source code and, if you specify the `--lang` qualifier, passes the parsed data to one or more of the OpenSDL “back-end” translators.

Each back end translates only those OpenSDL declarations that can (or need) be expressed in that language. OpenSDL declarations, described in detail in Chapter 4 [Declarations], page 71, translate to the following types of data items:

- Scalar or dimensioned scalar data items (ITEM declarations)
- Non-scalar data items (AGGREGATE and subaggregate declarations)
- Named constants (CONSTANT declarations)
- External entries (ENTRY declarations)

Figure 1.1 shows each step in the OpenSDL translation process, and the key following the example describes each step. Section 2.2 [Processing an OpenSDL Source File], page 36, describes the OpenSDL command in detail.

1. Create a properly formatted OpenSDL file (`.sdl`) containing a `MODULE` and `END_MODULE`, and one or more `CONSTANT`, `ITEM`, or `AGGREGATE ...member... END` statements.
2. Process the file created above using the `opensdl` command, noting any errors that may be reported.
3. If errors are reported, then open the file in your favorite text editor and make corrections as appropriate. Then go back to step 2.
4. Verify that the resulting output file(s) contain the definitions expected. If not, then make corrections and reprocess the file.
5. Use the resulting output file(s) within the code they are defined to be used.

Figure 1.1: OpenSDL Translation Process

Key to Figure 1.1:

1. The front end (`opensdl.exe`) parses the statements contained in the OpenSDL source file `test.sdl` as a result of issuing the OpenSDL command in the following way:
  - a. `$ opensdl --lang:cc test.sdl`  
 If you specify the `--lang` qualifier (for example, `--lang:cc`), the front end parses the source code and passes the parsed data to the back end for the specified language.
2. The specified back end (`cc`) produces an output file with the default file type of `.h`.

You can specify any or all of the language options on the `--lang` qualifier, and OpenSDL produces separate output files for each language specified.

## 1.2 Description of a Sample OpenSDL Source File

Example 1.1 is a typical OpenSDL source file, and the key following the example describes each of the numbered language elements. (Chapter 3 [Elements], page 41, describes all the language elements in detail.) You may want to familiarize yourself with this example because it shows the source file you will be creating in the tutorial in Section 2.1.2 [Getting Started with OpenSDL], page 16.

```
MODULE opr_descriptor IDENT "Version 2.0";1

/* define constants and node structure for operators2
#max_args = 10;3

CONSTANT (fixed_binary,floating,char,untyped) EQUALS 1 INCREMENT 1;4

AGGREGATE operator STRUCTURE5 PREFIX "opr_";6
    flink ADDRESS;7
    blink ADDRESS;
    opcount WORD;
    optype CHARACTER LENGTH 1;
    id WORD;
    operands LONGWORD DIMENSION 0:#max_args-1;8
END operator;

#opsize = .;9

CONSTANT opr_node_size EQUALS #opsize / 2;

ITEM current_node_ptr10 ADDRESS11 GLOBAL;12

END_MODULE opr_descriptor13;
```

Example 1.1: Sample OpenSDL Source File

1. All OpenSDL declarations are grouped within modules, and you must assign a name to each module. The `IDENT` keyword precedes any commented information that you may want to add to describe the `MODULE` declaration.

2. Output comments begin with a slash and an asterisk (`/*`) and are written to the language output file unless the `--nocomments` qualifier is specified.
3. Local symbols begin with a pound sign (`#`) and are not written to the output file. Local symbols may be used to express values in declarations. For example, the symbol `#max_args` is used in the declaration of the array operands.
4. `CONSTANT` declarations produce declarations of named constants. When the `INCREMENT` option and an increment value are specified, OpenSDL automatically increments the initial value for each of the declared output constants. In the example, `fixed_binary` will be assigned the value 1, `floating` will be assigned the value 2, and so on.
5. `AGGREGATE` declarations define data structures and their members.
6. When the `PREFIX` option and a prefix are specified, OpenSDL concatenates the prefix and a data type code to the declared aggregate member names in the language output files. Compare these aggregate member name declarations with the C output shown in Example 1.2.
7. Aggregate members are declared using reserved OpenSDL data type keywords.
8. An aggregate member or a scalar data item can be declared to be an array by specifying the `DIMENSION` option. In this example, the array operands has 10 elements, with subscripts 0 through 9.
9. The period (`.`) represents the current byte offset within an `AGGREGATE` declaration. The local symbol assignment `#opsize = .;` captures the size of the constant portion of the structure operator. The value of this local symbol is then used in the declaration of the constant `opr_node_size`.
10. `ITEM` declarations, such as the declaration of `current_node_ptr`, define scalar data items.
11. The `ADDRESS` keyword specifies a data type that is an address, or pointer.
12. The `GLOBAL` keyword specifies global storage to override the default language storage class for an `ITEM` or an `AGGREGATE` declaration.
13. The `END_MODULE` keyword ends a `MODULE` declaration; you may optionally specify the `MODULE` name after the `END_MODULE` keyword.

Example 1.2 shows the C/C++ language output file that results from translation of the OpenSDL source file shown in Example 1.1.

Chapter 3 [Elements], page 41, describes all the OpenSDL language elements, and Chapter 4 [Declarations], page 71, describes the function and format of each of the OpenSDL declarations.

```

/*****
/* Created 14-NOV-2018 14:18:38 by OpenSDL V3.4-20181114 */
/* Source: 14-NOV-2018 12:18:12 /usr/lib/opensdl/examples/example.sdl */
*****/

/** MODULE opr_descriptor IDENT = Version 2.0 **/
#include <ctype.h>
#include <stdint.h>
#include <stdbool.h>
#include <complex.h>

#ifndef _OPR_DESCRIPTOR_
#define _OPR_DESCRIPTOR_ 1
#ifdef __cplusplus
extern "C" {
#endif
/* define constants and node structure for operators */
#define fixed_binary 1
#define floating 2
#define char 3
#define untyped 4
#define opr_s_operator 28
struct operator
{
    void *opr_a_flink;
    void *opr_a_blink;
    int16_t opr_w_opcount;
    char opr_c_optype[1];
    int16_t opr_w_id;
    int32_t opr_l_operands;
};
#define opr_node_size 14
void *current_node_ptr __attribute__((aligned));

#ifdef __cplusplus
}
#endif
#endif /* _OPR_DESCRIPTOR_ */

```

Example 1.2: C Output File for the Sample OpenSDL Source File

## 2 Creating, Editing, and Processing an OpenSDL Source File

This chapter describes how to create an OpenSDL source file using a basic text editor (Notepad++) and how to process your source file using the OpenSDL command and its qualifiers.

You can use any text editor to create your source file.

Section 2.1.1 [Entering Source Code], page 16, describes how to use a text editor to enter OpenSDL source code. Section 2.1.2 [Getting Started with OpenSDL], page 16, provides a tutorial to get you started using a text editor to write OpenSDL source code. Section 2.1.3 [Compiling Source Code], page 36, describes how to compile your source code.

Section 2.2 [Processing an OpenSDL Source File], page 36, describes the OpenSDL command and the uses of the OpenSDL command qualifiers.

### 2.1 Creating an OpenSDL Source File with Notepad++

Open Notepad++ and create a file with a filename containing a `.sdl` file extension.

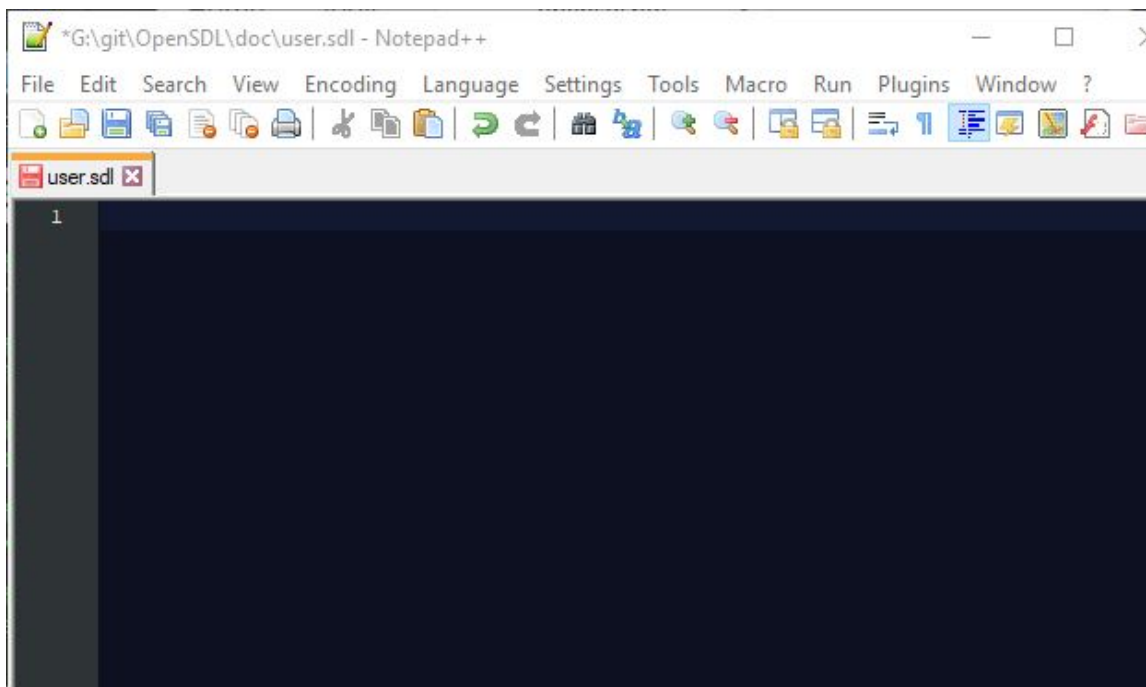


Figure 2.1: Opening File

**Note:** If you are unable to access OpenSDL or Notepad++, contact your system manager to ensure that they are installed on your system.

Section 2.1.1 [Entering Source Code], page 16, describes how to enter source code using Notepad++, and Section 2.1.2 [Getting Started with OpenSDL], page 16, provides a tutorial on using a text editor to create OpenSDL source code. Section 2.1.3 [Compiling Source Code], page 36, describes compiling the OpenSDL source code.

### 2.1.1 Entering Source Code

This section provides basic information about entering OpenSDL formatter code into a text editor.

The parser for OpenSDL ignores most whitespace characters, including SP, TAB, and ENTER. Therefore, these characters can be used to structure the file for easier understanding.

The OpenSDL keywords are not case sensitive. As such, they can be enter as all uppercase, all lowercase, or mixed (camel) case.

The identifiers associated with various OpenSDL keywords are case sensitive, depending upon the output language. Within the OpenSDL parser, a lowercase **AGGREGATE ID** will **not** match an uppercase (or camelcase) **END ID**, and will cause a warning message to be displayed.

Identifiers that are also an OpenSDL keyword can be used, but need to be enclosed in double-quotes ("). Any leading and trailing spaces will be ignored.

### 2.1.2 Getting Started with OpenSDL

This section provides a tutorial on using some common tokens and placeholders to write OpenSDL source code. The tutorial shows expansions of the following OpenSDL declarations and language elements:

- MODULE declaration
- Output line comment
- Local symbol declaration
- CONSTANT declaration
- AGGREGATE declaration
- ITEM declaration

Example 2.1 shows the sample OpenSDL source file described in Chapter 1 [Overview], page 11. You will be creating this sample source file in the following tutorial. The numbered callouts in the example correspond to the source code you will be entering in each step. Following each step, intermediate and resulting screen displays highlight the source code you just entered.

As you step through the tutorial, refer to Section 2.1.1 [Entering Source Code], page 16, for specifics about entering source code in a text editor. You can also access online help by typing `opensdl --help` at the command-line prompt.

```

MODULE opr_descriptor IDENT "Version 2.0";1

/* define constants and node structure for operators2
#max_args = 10;3
CONSTANT4 (fixed_binary,floating,char,untyped)5 EQUALS 16 INCREMENT 17;
AGGREGATE8 operator9 STRUCTURE10
    PREFIX11 "opr_";12
    flink13 ADDRESS;14
    blink ADDRESS;15
    opcount WORD;15
    optype CHARACTER LENGTH 1;15
    id WORD;15
    operands LONGWORD16 DIMENSION17 0:#max_args-1;18
END operator;19

#opsize = .;20

CONSTANT21 opr_node_size22 EQUALS #opsize23 / 2;24
ITEM25 current_node_ptr ADDRESS GLOBAL26;

END_MODULE opr_descriptor;27

```

#### Example 2.1: Creating a Sample OpenSDL Source File

The following numbered items describe and display each step. The numbered superscript corresponds to the equivalent numbered items. Not all steps will be followed by a screen capture, some steps are combined for display purposes.

1. Type `MODULE opr_descriptor IDENT "Version 2.0";` on the first line of the OpenSDL file we are creating, then hit the **Enter** key.

The `opr_descriptor`, will be used when generating files where the file should only be included a single time to prevent a file from being included multiple times. When used in this way, the string written to the *identifier* will be converted to all uppercase and an underscore `_` will be added to the beginning and end of the upcased string.

The value after the `IDENT` keyword must always be enclosed in double-quote characters (`"`). Your screen will look as follows:

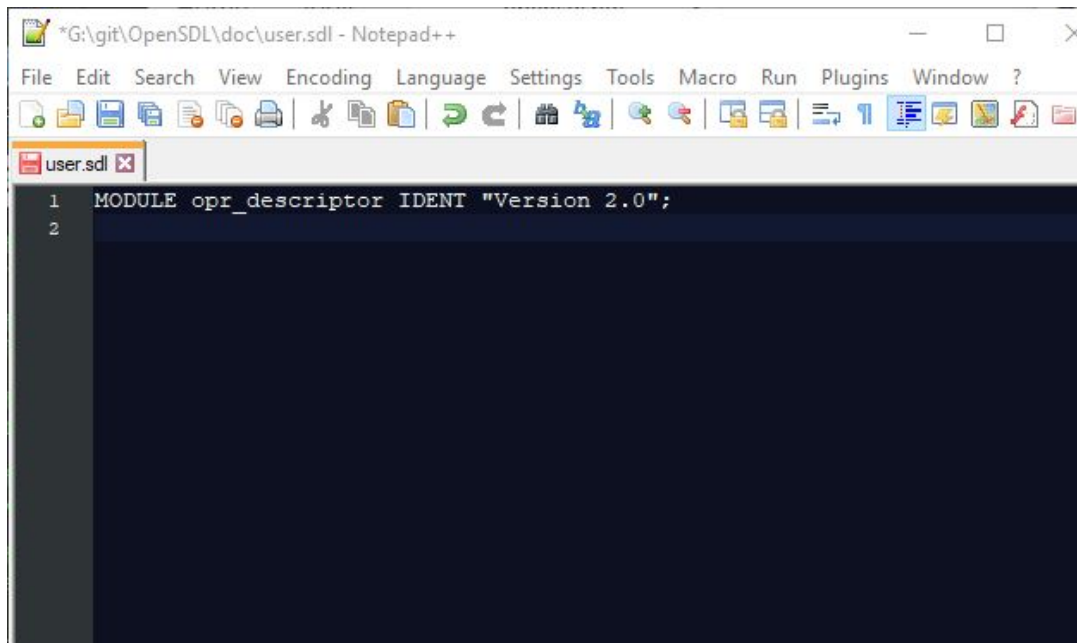


Figure 2.2: Module Statement



2. Insert another new-line by hitting the **Enter** key. Then type in `/* define constants and node structure for operators` and hit the **Enter** key.

This line contains one of the three comment formats. Comments starting with a brace (`{`) will **not** be written to the output file. Comments starting with a forward-slash followed by an asterics `/*` will be written to the output file as a line comment formatted for the output language. The third comment format is the block comment, see for more information about comments in the OpenSDL language in sections Section 3.4 [Local Comments], page 62, and Section 3.5 [Output Comments], page 62. Your screen will look as follows:

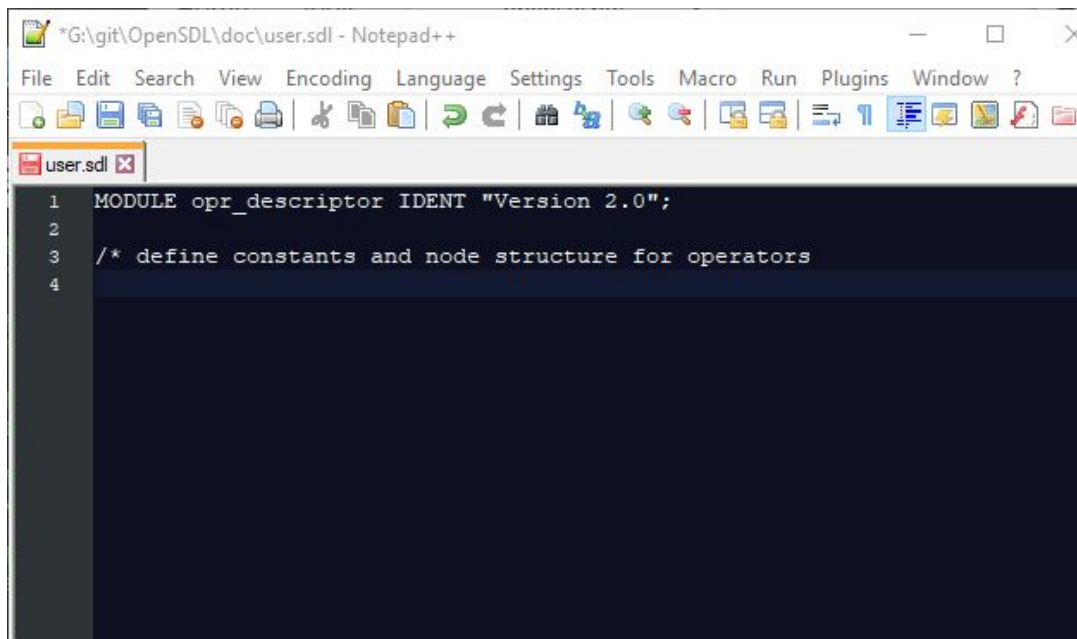
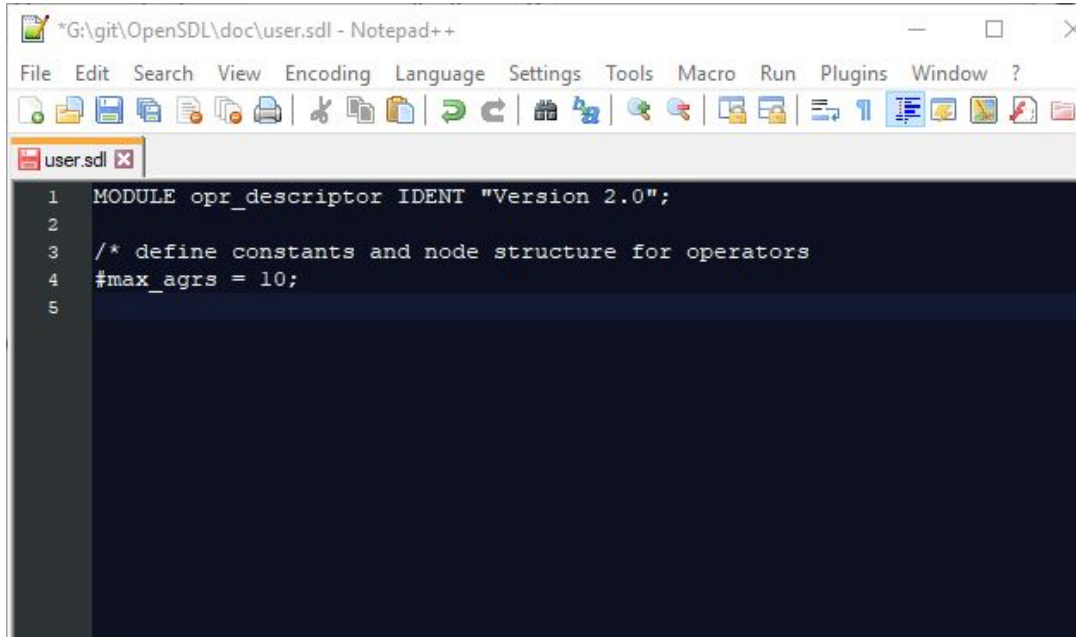


Figure 2.3: Output Comment

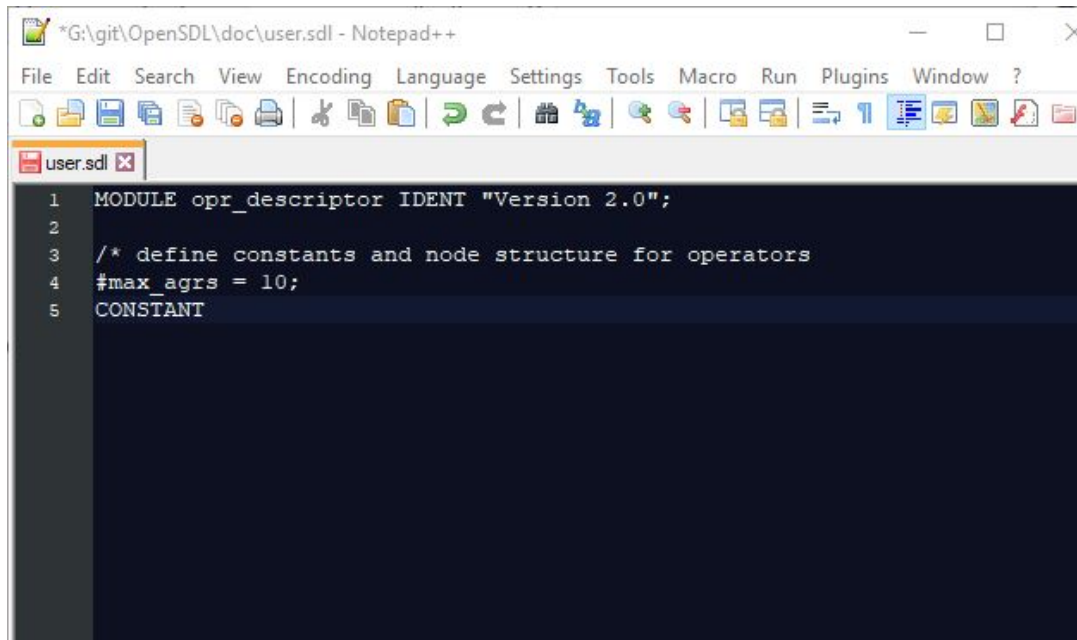
3. Insert another new-line, and now we are going to defined a local variable to be used later on in this example. Local variable name always starts with a hash-symbol (#), followed by alpha-numeric characters, and is case-sensitive. Now type in `#max_args = 10;` (don't forget the semicolon at the end). Your screen will look as follows:



```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
user.sdl
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5
```

Figure 2.4: Local Variable

4. Now we are going to define a set of constants that start from a specific value to be assigned to the first constant and then incremented by another value to be assigned to each of the following constant definitions. Type `CONSTANT`, with a trailing space character. Your screen will look as follows:



```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
user.sdl
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_agrs = 10;
5 CONSTANT
```

Figure 2.5: Constant Start

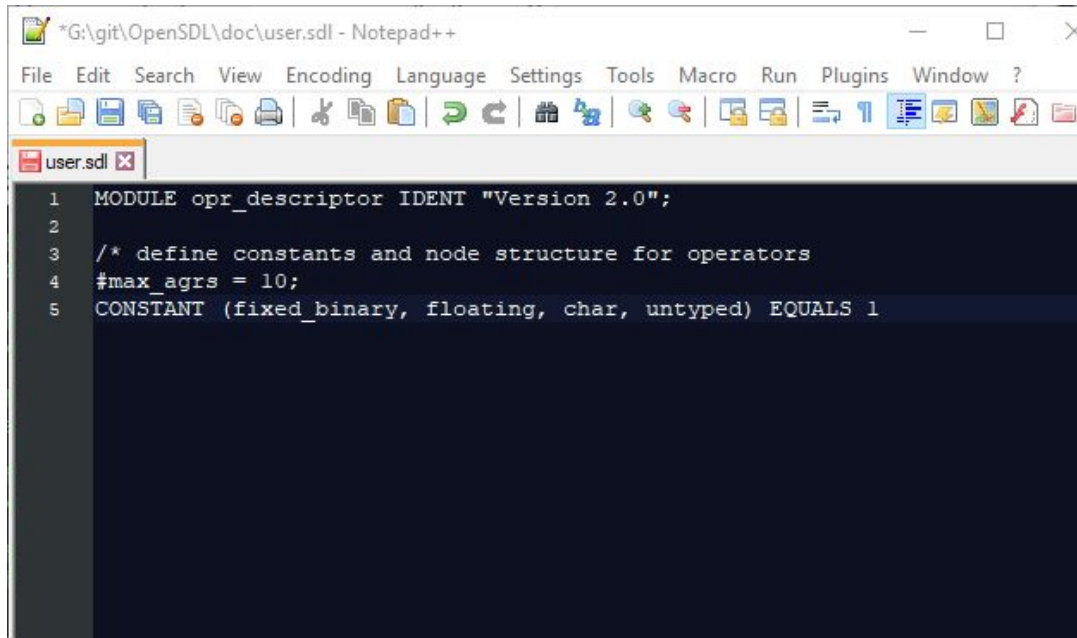
5. Now we need to defined a set of constants to be defined. This is a comma separator list enclosed in an open parenthesis ( and close parenthesis ). Type in (**fixed\_binary**, **floating**, **char**, **untyped**), with a trailing space character. Your screen will look as follows:

A screenshot of a Notepad++ window titled '\*G:\git\OpenSDL\doc\user.sdl - Notepad++'. The window shows a file named 'user.sdl' with the following content:

```
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_agrs = 10;
5 CONSTANT (fixed_binary, floating, char, untyped)
```

Figure 2.6: Constant List

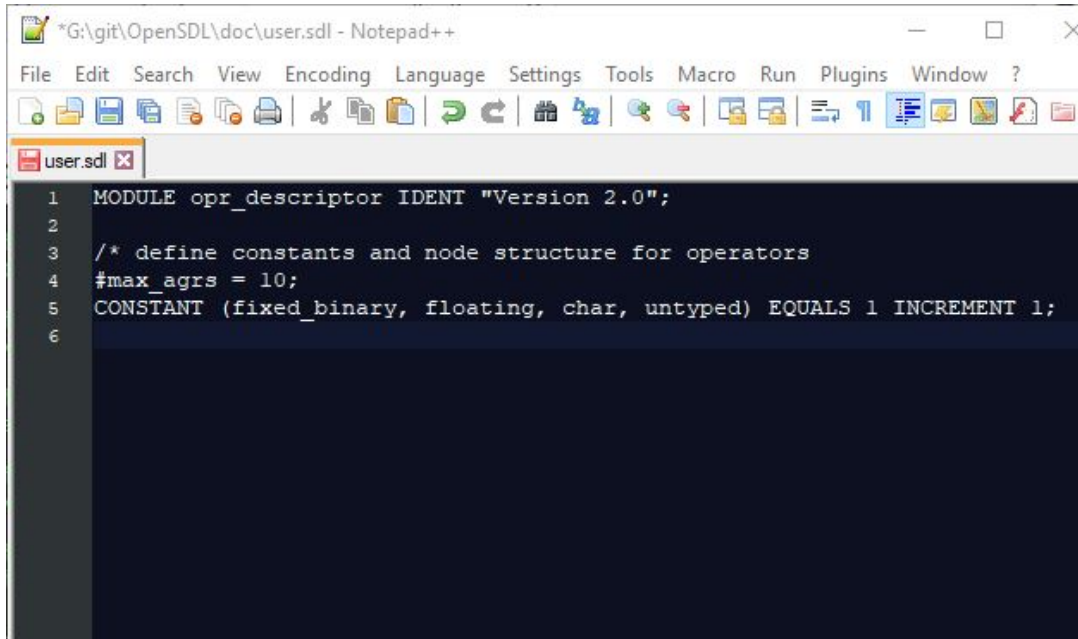
6. Now we add the value to be associated with the first constant to be defined. Type in `EQUALS 1`, followed by a space character. Your screen will look as follows:



```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
user.sdl
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1
```

Figure 2.7: Constant First

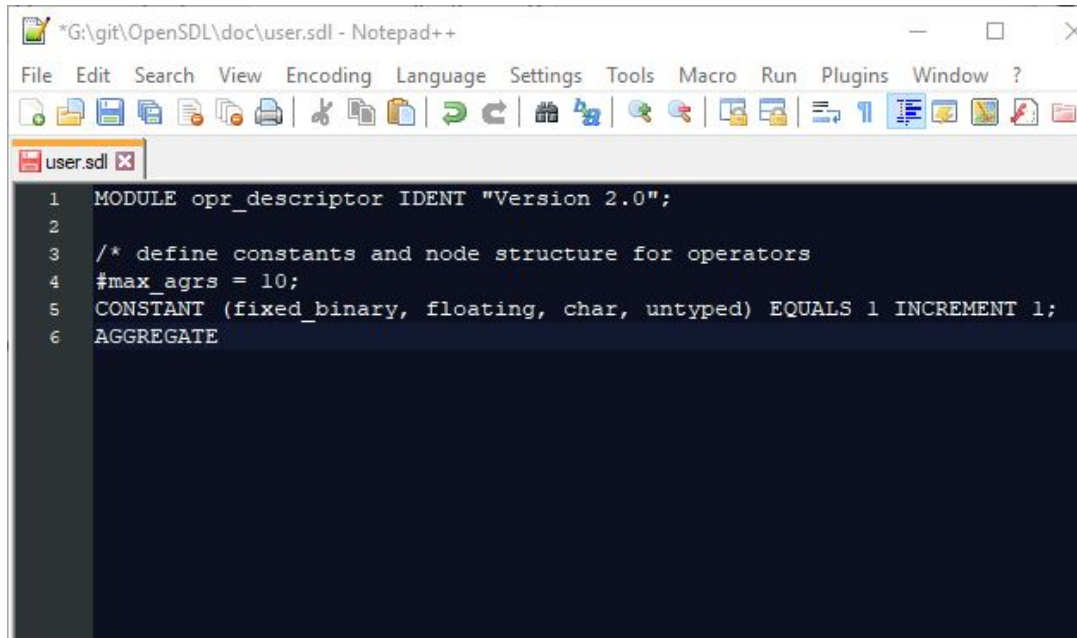
7. Finally we add the value to be used to increment the remaining constants to be defined. Type in `INCREMENT 1;`, making sure the semi-colon has been included, then hit the **Enter** key.



```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
user.sdl
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_agrs = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6
```

Figure 2.8: Constant Last

8. Next we are going to enter an **AGGREGATE** definition. Start by typing in **AGGREGATE** with a trailing space character. Your screen will look as follows:

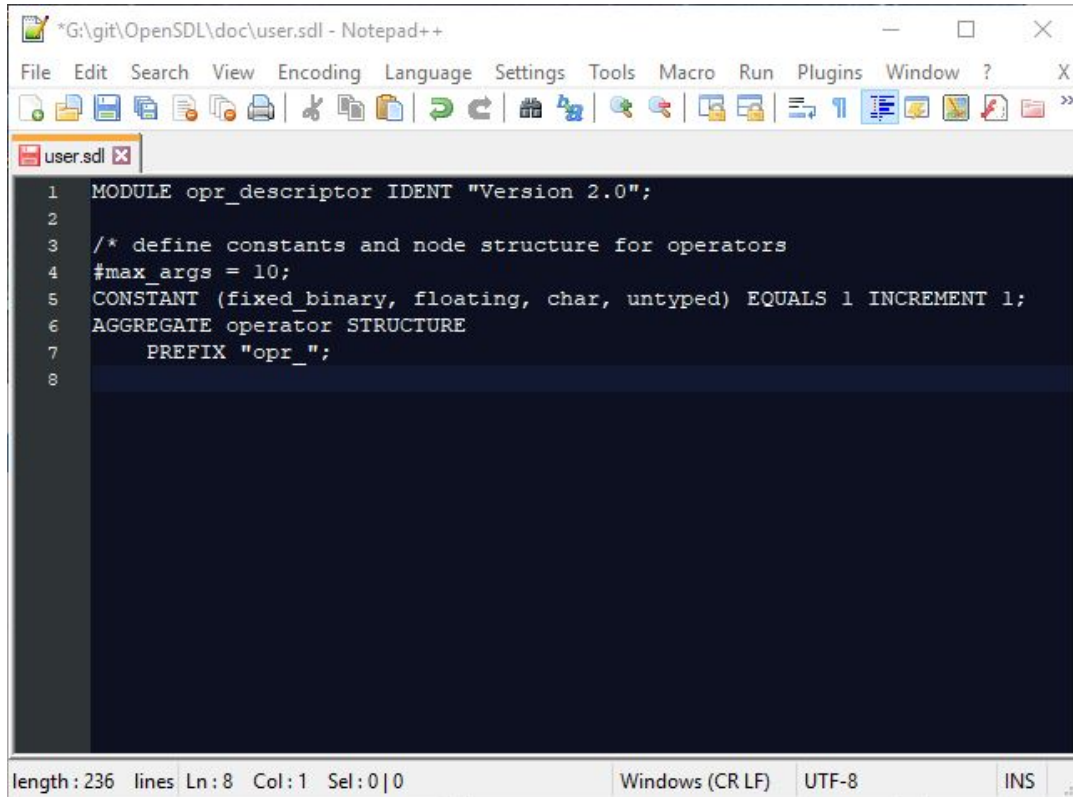


The screenshot shows a Notepad++ window titled "G:\git\OpenSDL\doc\user.sdl - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations and editing. The text area shows the following code:

```
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_agrs = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE
```

Figure 2.9: Aggregate Start

9. Now type in `operator`, with a trailing space character.
10. Next type in `STRUCTURE`, entering a `Enter` key.
11. Add any space or tab characters to keep the code looking clean, then type in `PREFIX`, followed by a trailing space.
12. Type `"opr_"` followed by a semi-colon (`;`) and `Enter` key. Your screen will look as follows:



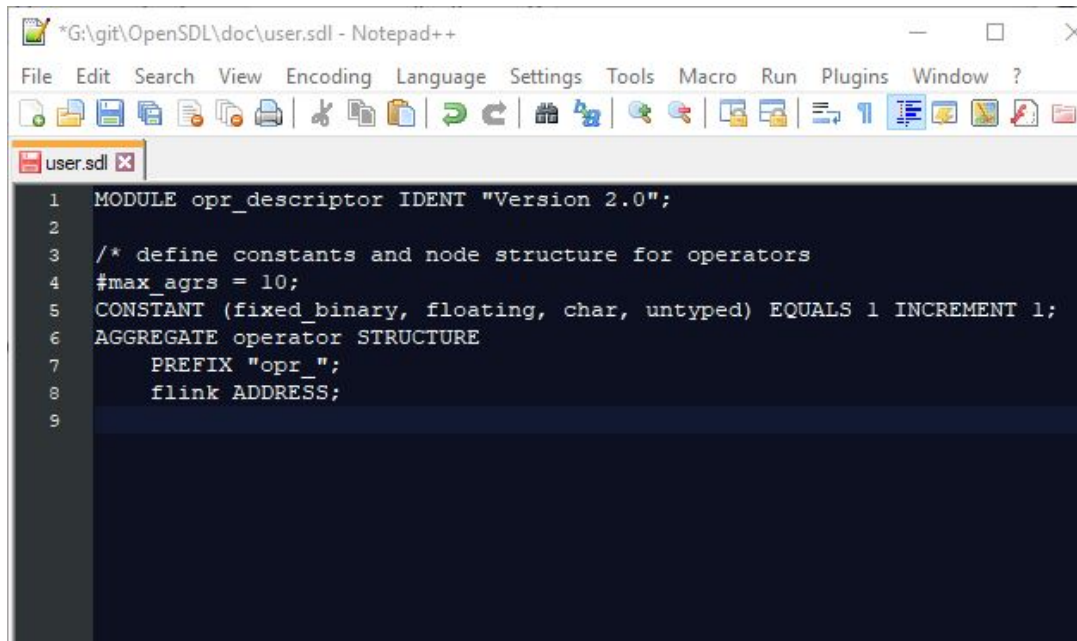
```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
user.sdl x
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr_";
8
```

length: 236 lines Ln: 8 Col: 1 Sel: 0|0 Windows (CR LF) UTF-8 INS

Figure 2.10: More Aggregate Information



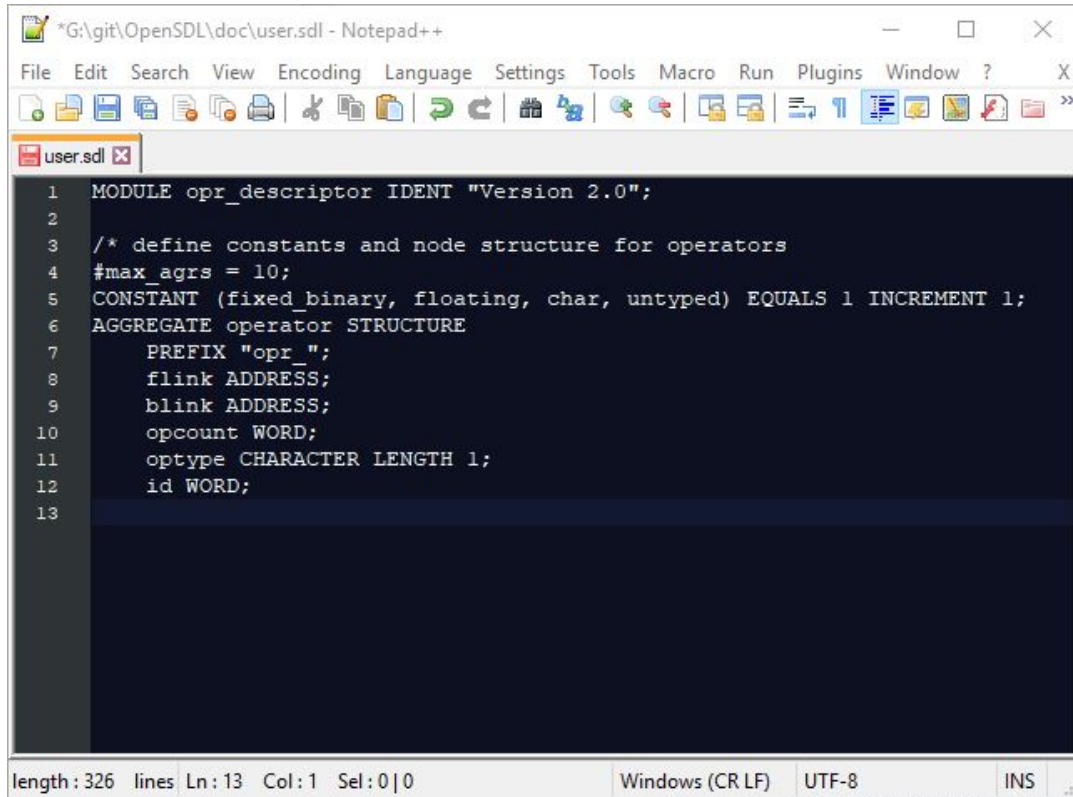
13. Type the first member name **flink**, followed by a space character.
14. Now type in **ADDRESS;**, not forgetting to include the semi-colon **;** at the end. Type the **Enter** key. Your screen will look as follows:



```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
user.sdl x
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr_";
8     flink ADDRESS;
9
```

Figure 2.11: Aggregate Member

15. Follow the steps described in the previous step for the **blink**, **opcount**, **optype**, and **id** members shown on each line in the example callout. Note the differences in syntax required for some of the member names. Your screen will look as follows:

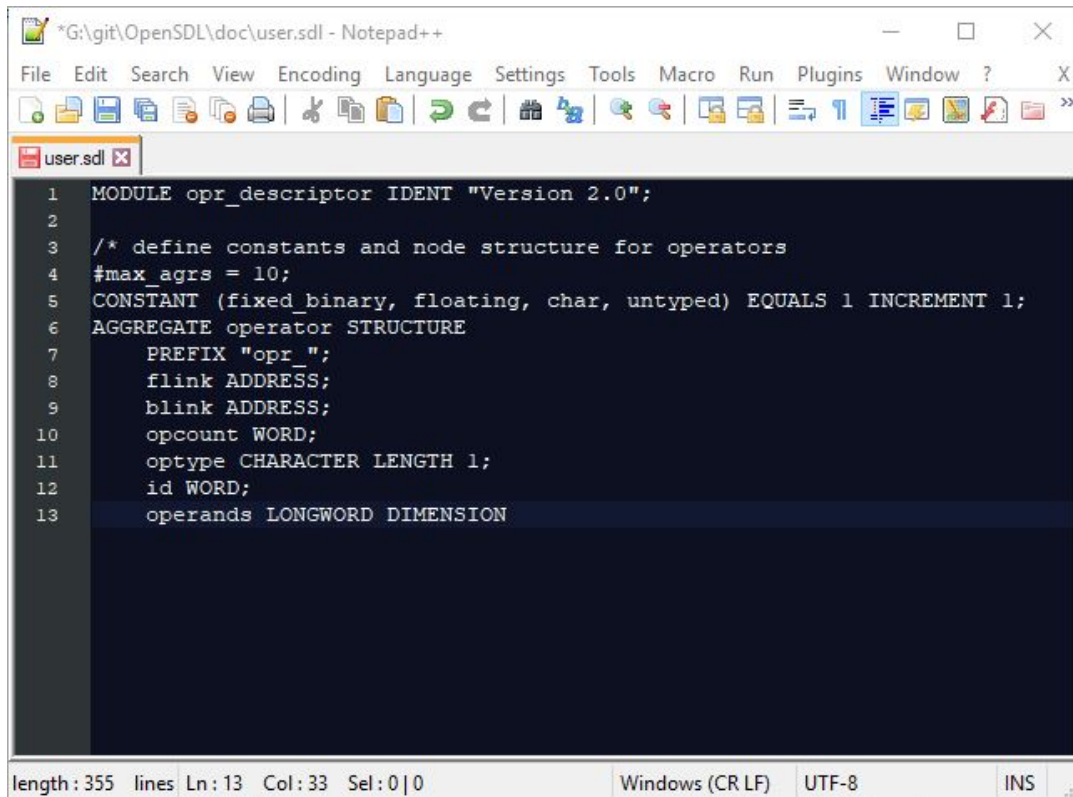


```
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr_";
8     flink ADDRESS;
9     blink ADDRESS;
10    opcount WORD;
11    optype CHARACTER LENGTH 1;
12    id WORD;
13
```

length: 326 lines Ln: 13 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS

Figure 2.12: Aggregate Middle

16. Now we are going to add the last member in this particular definition. Type in `operands LONGWORD` followed by a space character.
17. Next type in `DIMENSION` followed by a space character. Your screen will look as follows:

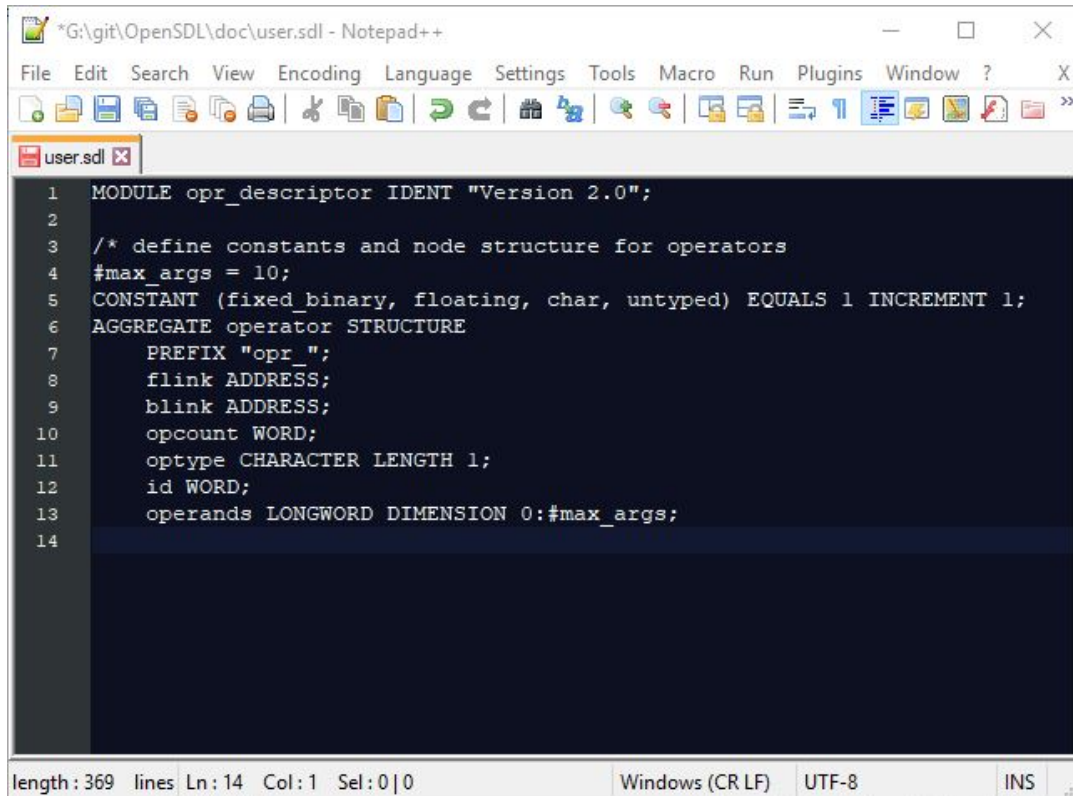


```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
user.sdl
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_agrs = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr ";
8     flink ADDRESS;
9     blink ADDRESS;
10    opcount WORD;
11    optype CHARACTER LENGTH 1;
12    id WORD;
13    operands LONGWORD DIMENSION
```

length: 355 lines Ln: 13 Col: 33 Sel: 0|0 Windows (CR LF) UTF-8 INS

Figure 2.13: Aggregate Almost Done

18. Now we will defined the lower and upper bounds for the `DIMENSION` definition. For languages that do not have a lower bound, the range is adjusted to be consistent with the particular language, and still have the exact same number of entries. Type in `0:#max_args-1;`, not forgetting the semi-colon at the end. Hit the **Enter** key (deleting any characters inserted by the text editor, if added). Your screen will look as follows:

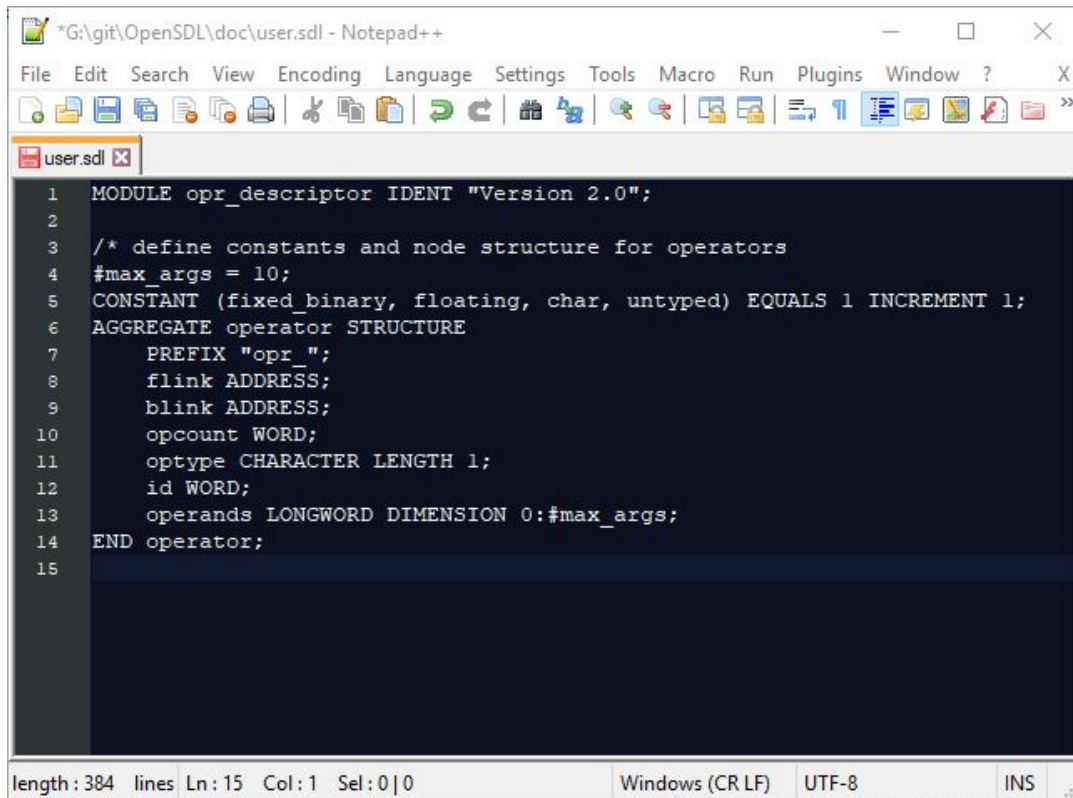


```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
user.sdl
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr_";
8     flink ADDRESS;
9     blink ADDRESS;
10    opcount WORD;
11    optype CHARACTER LENGTH 1;
12    id WORD;
13    operands LONGWORD DIMENSION 0:#max_args;
14
```

length: 369 lines Ln: 14 Col: 1 Sel: 0|0 Windows (CR LF) UTF-8 INS

Figure 2.14: Aggregate Last

19. Finally, we are going to close off the **AGGREGATE** definition. Type in **END operator;** and type the **Enter** key. Make sure the *id* specified on the **AGGREGATE** statement matches the one on the **END**, keeping the case the same. **Note:** Putting the *id* on the **END** statement is optional, but if specified, must match. Your screen will look as follows:



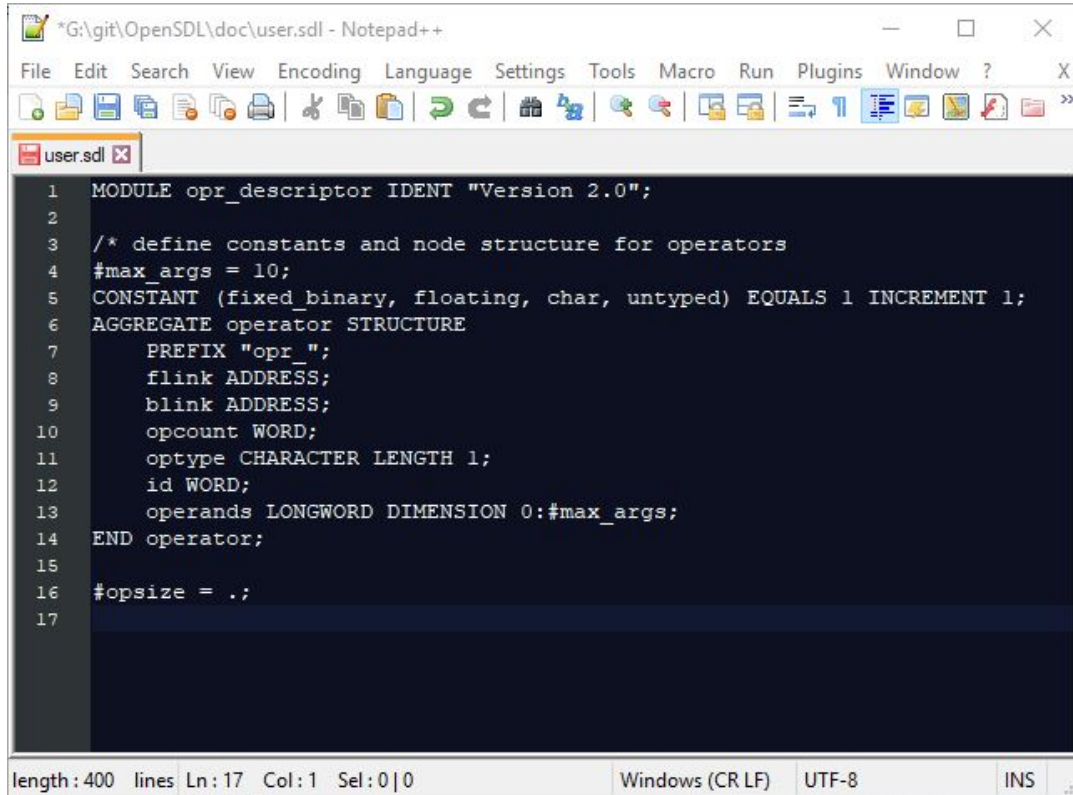
The screenshot shows a Notepad++ window titled '\*G:\git\OpenSDL\doc\user.sdl - Notepad++'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and Help. The toolbar contains various icons for file operations and editing. The text area shows the following code:

```
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr_";
8     flink ADDRESS;
9     blink ADDRESS;
10    opcount WORD;
11    optype CHARACTER LENGTH 1;
12    id WORD;
13    operands LONGWORD DIMENSION 0:#max_args;
14 END operator;
15
```

The status bar at the bottom indicates 'length: 384 lines', 'Ln: 15 Col: 1 Sel: 0|0', 'Windows (CR LF)', 'UTF-8', and 'INS'.

Figure 2.15: Aggregate End

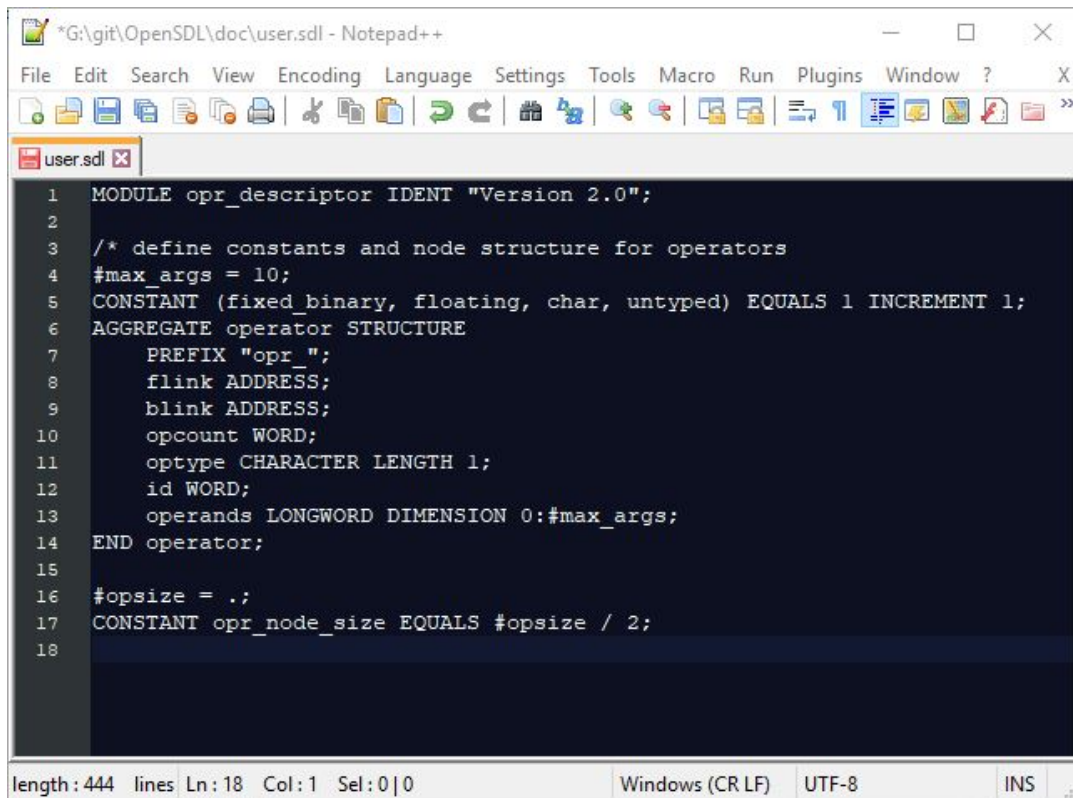
20. Again type the **Enter** key. Next we are going to define another local variable to contain a value representing the current offset within a **AGGREGATE** definition. Type **#opsize = .;**, again make sure to include the semi-colon character. Type the **Enter** key again. Your screen will look as follows:



```
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr_";
8     flink ADDRESS;
9     blink ADDRESS;
10    opcount WORD;
11    optype CHARACTER LENGTH 1;
12    id WORD;
13    operands LONGWORD DIMENSION 0:#max_args;
14 END operator;
15
16 #opsize = .;
17
```

Figure 2.16: Offset

21. Because local variables are not written to the output file, if we want the offset values, we need to define a `CONSTANT` value. To do this we start by typing in `CONSTANT` followed by a space character.
22. Type `opr_node_size` followed by a space character.
23. Type `EQUALS #opsize`, which does not have to be followed by a space character. We do so here for clarity.
24. Type `/ 2;`, remembering to include the semi-colon, plus typing the `Enter` key. Your screen will look as follows:



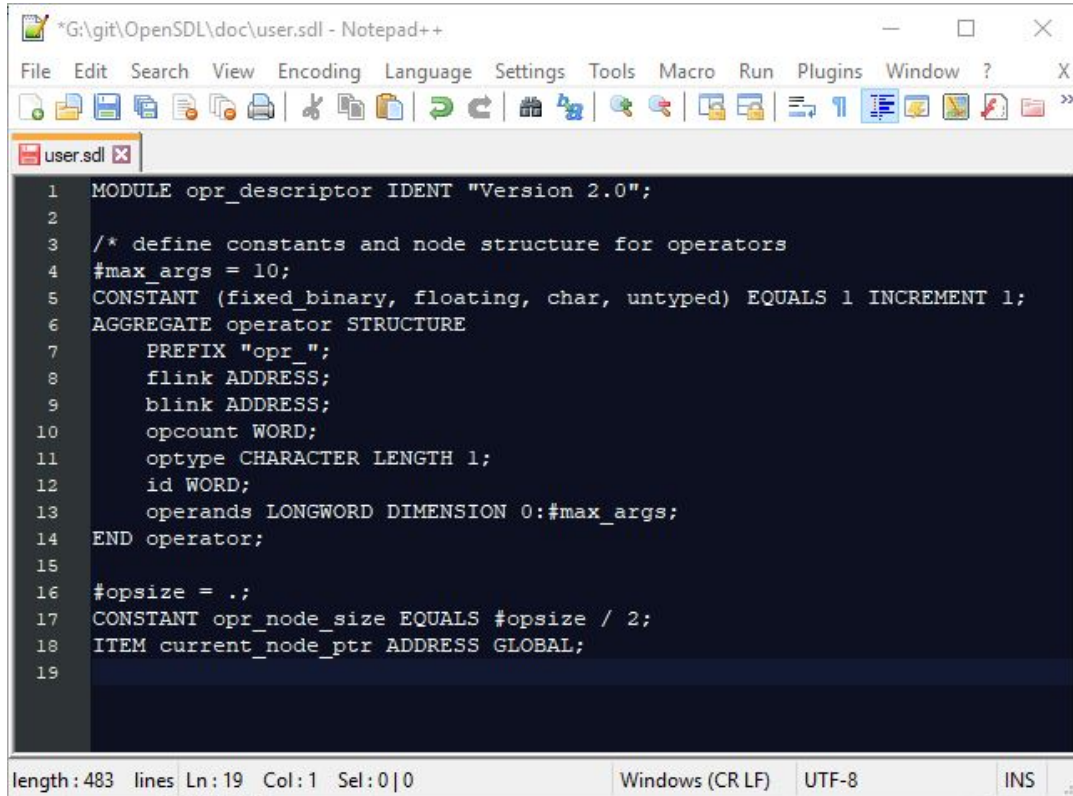
```
1  MODULE opr_descriptor IDENT "Version 2.0";
2
3  /* define constants and node structure for operators
4  #max_args = 10;
5  CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6  AGGREGATE operator STRUCTURE
7      PREFIX "opr ";
8      flink ADDRESS;
9      blink ADDRESS;
10     opcount WORD;
11     optype CHARACTER LENGTH 1;
12     id WORD;
13     operands LONGWORD DIMENSION 0:#max_args;
14 END operator;
15
16 #opsize = .;
17 CONSTANT opr_node_size EQUALS #opsize / 2;
18
```

length: 444 lines Ln: 18 Col: 1 Sel: 0|0 Windows (CR LF) UTF-8 INS

Figure 2.17: Constant Offset



25. Now we are going to define a stand alone variable. Type ITEM followed by a space character.
26. Type `current_node_ptr ADDRESS GLOBAL;` followed by the Enter key. Your screen will look as follows:



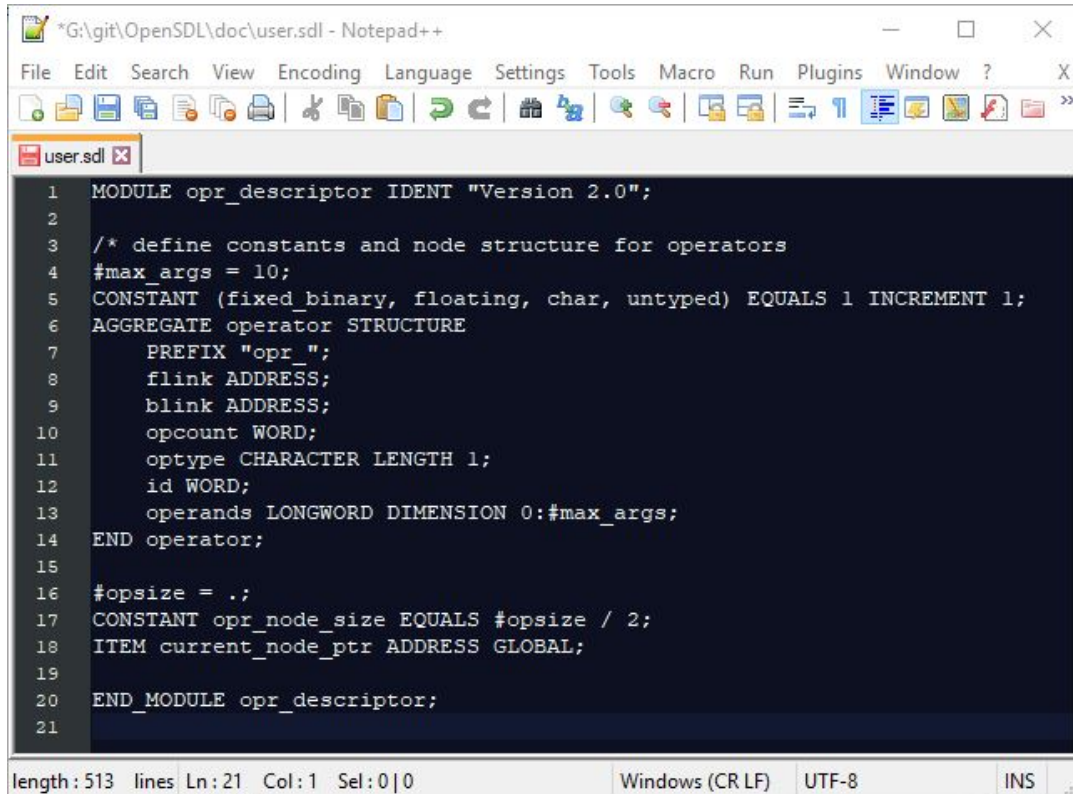
```
*G:\git\OpenSDL\doc\user.sdl - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
user.sdl
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed_binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr ";
8     flink ADDRESS;
9     blink ADDRESS;
10    opcount WORD;
11    optype CHARACTER LENGTH 1;
12    id WORD;
13    operands LONGWORD DIMENSION 0:#max_args;
14 END operator;
15
16 #opsize = .;
17 CONSTANT opr_node_size EQUALS #opsize / 2;
18 ITEM current_node_ptr ADDRESS GLOBAL;
19

length: 483 lines Ln: 19 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 INS
```

Figure 2.18: Item First



27. Lastly, type the **Enter** key again and then type `END_MODULE opr_descriptor;`, making sure the semi-colon is present. An **Enter** key can be typed again, but is not required. Your screen will look as follows, with the complete OpenSDL file:

A screenshot of a Notepad++ window titled '\*G:\git\OpenSDL\doc\user.sdl - Notepad++'. The window displays the complete OpenSDL source file 'user.sdl' with the following code:

```
1 MODULE opr_descriptor IDENT "Version 2.0";
2
3 /* define constants and node structure for operators
4 #max_args = 10;
5 CONSTANT (fixed binary, floating, char, untyped) EQUALS 1 INCREMENT 1;
6 AGGREGATE operator STRUCTURE
7     PREFIX "opr_";
8     flink ADDRESS;
9     blink ADDRESS;
10    opcount WORD;
11    optype CHARACTER LENGTH 1;
12    id WORD;
13    operands LONGWORD DIMENSION 0:#max_args;
14 END operator;
15
16 #opsize = .;
17 CONSTANT opr_node_size EQUALS #opsize / 2;
18 ITEM current_node_ptr ADDRESS GLOBAL;
19
20 END_MODULE opr_descriptor;
21
```

The status bar at the bottom shows 'length: 513 lines Ln: 21 Col: 1 Sel: 0|0', 'Windows (CR LF)', 'UTF-8', and 'INS'.

Figure 2.19: Item Last

You may insert a specific copyright-statement, but for the purposes of this tutorial, it is not being shown.

Your resulting screen display should be the OpenSDL source file shown in Example 2.1.

### 2.1.3 Compiling Source Code

Compiling source code is very straightforward. You use the command qualifiers to read in the input file and generate one or more output files. At a minimum the command line consists of the following:

1. The `opensdl` command.
2. One output language (e.g. `--lang:cc`).
3. The input file `test.sdl`.

When compiling an input file, as errors are detected, they are displayed to `stderr`. Additionally, the `--list` qualifier can be specified to generate a listing file, where the same errors will be displayed at the line where the error was detected.

**Note:** The line numbers displayed within the error message text may not be exactly the line of code in a listing file where the message will be written. The error message text will refer to the first line of a multiline statement, where this same text will be written immediately after the line of code that actually caused the condition to be detected.

A single line of code can generate multiple errors, the errors will be displayed in the order in which they were detected.

Section 2.2 [Processing an OpenSDL Source File], page 36, describes how to process an OpenSDL source file using the OpenSDL command and all its qualifiers.

## 2.2 Processing an OpenSDL Source File

The `opensdl` command invokes the OpenSDL translator from the command line to produce output files for one or more target languages. The `opensdl` command has the following format:

```
$ opensdl [qualifier[...]] file-spec [qualifier[...]]
```

### Command Parameter

#### `file-spec`

Specifies one or more OpenSDL source files to be translated. A file specification must specify a file name; if it does not include a file type, OpenSDL uses the default file type `.sdl`. You can specify one a single input file.

## Command Line Qualifiers

Command qualifiers may be specified following the `opensdl` command. Table 2.1 lists all the optional `opensdl` command qualifiers and their defaults.

Qualifier	Default
<code>-a, --align</code>	No alignment
<code>-32 or -64</code>	<code>-64</code>
<code>-k, --[no]check</code>	<code>--nocheck</code>
<code>-c, --[no]comments</code>	<code>--comments</code>
<code>-C, --[no]copy</code>	<code>--nocopy</code>
<code>-H, --[no]header</code>	<code>--header</code>
<code>-h, --help</code>	Do not display help information.
<code>-l, --lang</code>	No languages
<code>-L, --[no]list</code>	<code>--nolist</code>
<code>-m, --[no]member</code>	<code>--nomember</code>
<code>-S, --[no]suppress</code>	<code>--nosuppress</code>
<code>-s, --symbol</code>	No symbols
<code>-t, --trace</code>	No trace memory allocations/deallocations
<code>-v, --verbose</code>	No verbose output
<code>-V, --version</code>	Don't display OpenSDL version information.

Table 2.1: OpenSDL Command Qualifiers and Their Defaults

`-a:value`

`--align:value`

The assumed alignment. Integer value greater than zero. If specified, diagnostic messages are emitted for data items that do not fall on the assumed alignment.

`-32 or -64`

The number of bits the represent a LONGWORD.

`-k`

`--check`

`--nocheck`

If specified, diagnostic messages are emitted for data items that do not fall on their natural alignment.

`-c`

`--comments`

`--nocomments`

Controls whether output comments are included in the output file. For more compact target language representation, use the `--nocomments` qualifier to save file space. The default is `--comments`.

`-C`

`--copy`

`--nocopy`

Controls whether a standard copyright header is produced in the output file. The `--copy` qualifier causes the OpenSDL translator to precede the

output with a comment containing the copyright claim as specified in the `/usr/lib/opensdl/copyright.sdl` file. The default is `--nocopy`.

`-H`

`--header`

`--noheader`

Controls whether a header containing the date and the source file name is included at the beginning of the output file. The default is `--header`.

`-h`

`--help`

Controls whether the help/usage information is specified to standard output.

`-l:{language[=file-spec]}`

`--lang:{language[=file-spec]}`

`{language[=file-spec]}` specifies a single language option listed in Table 2.2 for which the OpenSDL translator is to produce a source output file. By default, OpenSDL writes output files into separate files in the current default directory/folder. The default file name for each output file is taken from the file name of the corresponding source file and the default target file type for each language name. Multiple `--lang` qualifiers can be specified on the command line, but each language should only be specified once.

#### Language

#### Option

C/C++

`--lang:cc`

`-l:cc`

Table 2.2: OpenSDL Output Language Options and File Types

The `--lang` qualifier also allows you to override the default output file specification for a language output files. You can specify a language option followed by the destination file specification for that language. You must separate the language from the destination file specification with an equal sign (=).

`-L[:file-spec]`

`--list[:file-spec]`

`--nolist`

Controls whether a listing file is produced.

The `--list` qualifier causes the OpenSDL translator to produce a listing file with numbered lines of source code and descriptions of any compilation errors. The listing file has the same name as the related source file and a file type of `.lis`.

`-m`

`--member`

`--nomember`

Specifies that every item in aggregates should be aligned. This is the same as specifying `ALIGN` on all aggregates.

`-S:{suppress-option}`

`--suppress:{suppress-option}`  
`{(suppress-option,...)}`

The `--supress` qualifier has the following format: `opensdl --suppress=prefix,tag`

Note the following:

- The suppress-options can be either **prefix** or **tag**.
- The qualifier may appear anywhere in the command line where a qualifier is valid.
- Output in all languages in that compilation is affected.
- Either **prefix** or **tag**, or both, may be included in the list (with no spaces between when specifying both).
- If both prefixes and tags are suppressed, the connecting underscore is also suppressed.
- The effect is as though null prefixes and/or tags had been specified throughout the source.

`-s:{symbol=value}`

`--symbol:{symbol=value}`

It is possible to specify symbols and values which can be used in the `IFSYMBOL` statement (kind of conditional compilation). Multiple `--symbol` qualifiers can be specified on the command line. If the same symbol is specified more than ones, then the last one will be in effect. See `IFSYMBOL`.

`-t`

`--trace`

Specifies that the OpenSDL Utility should display memory allocations and deallocations used by the tool as it processes the input file and generates the output file(s).

`-v`

`--verbose`

Specifies that the OpenSDL Utility should generate verbose messages to both standard output and standard error as it processes the input file and generates the output file(s). This can be used to debug issues with the utility.

`-V`

`--version`

Specifies that the OpenSDL Utility should display the version information for the utility to standard output.

## Examples

The following are examples and descriptions of the `opensdl` command.

1. `$ opensdl blocknode.sdl --lang:cc=./block_node.h)`  
 OpenSDL translates the declarations in `blocknode.sdl` to C/C++ and writes the C/C++ output to `block_node.h` in the parent directory/folder.
2. `$ opensdl --list --lang:cc iodef.src`  
 OpenSDL translates the declarations in `iodef.src` and writes the output to `iodef.h`, while also generating the list file `iodef.lis`



## 3 OpenSDL Language Elements

This chapter describes the function and syntax of the following OpenSDL language elements that compose the OpenSDL declarations described in Chapter 4 [Declarations], page 71:

- User-specified names, which can be either local symbol names or source program identifiers
- Reserved OpenSDL keywords
- Expressions

The following can also be used within a `MODULE` declaration; although they are not OpenSDL language elements (Section 3.4 [Local Comments], page 62, through Section 3.6 [INCLUDE Statement], page 63):

- Local and output, line and block, comments
- `INCLUDE` statement
- Conditional compilation
- Text pass-through
- `DECLARE` statement

The space, tab, or carriage return character delimits the language elements, and a semicolon (;) terminates each declaration. In `MODULE` and `AGGREGATE` declarations, the semicolon also terminates separate parts of the declaration.

Appendix B [Summaries], page 103, shows language translation summaries of all the OpenSDL language elements.

### 3.1 Names

An OpenSDL name can be either a user-specified local symbol name that is not translated to the output file or a user-specified source program identifier that is translated to the output file. Names are composed of upper- and lowercase letters (A-Z, a-z), numbers (0-9), the dollar sign (\$), and the underscore (\_). Specifying a name is subject to the following rules:

**Note:** The use of the dollar sign (\$) in symbol names is not supported by all languages or compilers. As a result, its use may cause compiler or portability issues. It's use is discouraged. It is supported for backward compatibility with OpenVMS versions of the software.

1. Local symbol names must begin with a pound sign (#).
2. Source program identifiers must begin with an alphabetic character (A-Z, a-z), a dollar sign (\$), or an underscore (\_).
3. Source program identifiers that are reserved OpenSDL keywords or that contain invalid OpenSDL characters must be enclosed in quotation marks (" "). (For more information on reserved OpenSDL keywords, Section 3.2 [Keywords Used in Declarations], page 43).
4. OpenSDL passes all source program identifiers to the output file in the same case in which they are defined, except where noted.

### 3.1.1 Local Symbol Names

A local symbol name is known only within an OpenSDL source file and cannot be translated directly to the output file. A local symbol name can be assigned a value anywhere within a source file, but must begin with the pound sign (#). A local symbol is declared when it is first assigned a value. This value can be any valid expression (Section 3.3 [Expressions], page 61). If you reference a local symbol before assigning it a value, OpenSDL displays an error message and does not produce an output file. A local symbol assignment has the following syntax:

```
#local-name = expression;
```

#local-name is any valid OpenSDL name.

expression is any valid OpenSDL expression resulting in a quadword integer value. Signed integer quadword data types are described in Section 3.2.3.9 [Integer Data Types], page 60. The following are examples of local symbol assignments:

```
#max_args = 255;
#counter = #counter + 1;
```

The values of these local symbols may be referenced by subsequent declarations, as shown in the following example:

```
CONSTANT block_node_size EQUALS #max_args + #counter;
```

### 3.1.2 Source Program Identifiers

Source program identifiers (identifiers) specify declaration names, **AGGREGATE** member names, and **ENTRY** parameter names that are passed to the output file. Optional user-specified prefixes and tags can be appended to these identifiers. If a prefix is specified without a tag, OpenSDL concatenates a default tag (corresponding to the data type) to the identifier in the output file. (Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49, describes the **PREFIX**, **MARKER** and **TAG** keyword options.) The identifier `block_node_size` in the following example names the **CONSTANT** declaration:

```
CONSTANT block_node_size EQUALS #max_args + #counter;
```

To avoid compilation errors, each reference to a particular OpenSDL identifier must be a case sensitive match because an identifier is passed to the output file in the same case in which it appears in the source file. You can use reserved OpenSDL keywords and characters that are not valid in identifiers if you enclose them in quotation marks (" "). For example:

```
ITEM "length" LONGWORD;
```

This declaration produces the identifier `length`, which is a reserved OpenSDL keyword typically used to specify the length of a bit-string or character-string data type (see Section 3.2.3.3 [BITFIELD Data Type], page 58, and Section 3.2.3.5 [CHARACTER Data Type], page 59).



## 3.2 Keywords

Reserved OpenSDL keywords are used to specify the following:

- Declarations (Section 3.2.1 [Declaration Keywords], page 43)
- Declaration modifiers (Section 3.2.2 [Declaration Modifier Keywords], page 44)
- Prefixes, markers, and tags (Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49)
- Alignment (Section 3.2.2.4 [Alignment Keywords], page 53)
- Storage classes (Section 3.2.2.5 [Storage Class Keywords], page 54)
- Arrays (Section 3.2.2.8 [DIMENSION Keyword], page 56)
- Datatypes (Section 3.2.3 [Data Type Keywords], page 57)

Reserved OpenSDL keywords can be entered in either upper- or lowercase letters, but they cannot be truncated.

The following sections describe the format and function of each of the reserved OpenSDL keywords.

### 3.2.1 Declaration Keywords

Table 3.1 alphabetically lists and defines the keywords for the declarations described in detail in Chapter 4 [Declarations], page 71.

Keyword	Definition
MODULE	Declaration of a module
END_MODULE	Delimiter for the end of a module
CONSTANT	Declaration of a named constant
ITEM	Declaration of an item
AGGREGATE	Declaration that produces a structure or union body
STRUCTURE	Declaration that is a type of aggregate or subaggregate
UNION	Declaration that is a type of aggregate or subaggregate
END	Delimiter for the end of an aggregate body
ENTRY	Declaration of an entry

Table 3.1: Keywords That Identify or End Declarations

### 3.2.2 Declaration Modifier Keywords

Table 3.1 lists and defines other keywords that are used in declarations. The prefix and tag, storage class, and array keywords have special functions that are described in detail in the sections following Section 3.2 [Keywords Used in Declarations], page 43. All the other keywords defined in Section 3.2 [Keywords Used in Declarations], page 43, are described in greater detail in Chapter 4 [Declarations], page 71.

#### MODULE Declaration

Keyword	Description
IDENT	Optional keyword used to pass information describing the MODULE declaration to the output file

Table 3.2: MODULE Declaration

#### ITEM Declaration

Keyword	Description
ALIGN, NOALIGN, BASEALIGN	Optional keywords used to specify alignment; see Section 3.2.2.4 [Alignment Keywords], page 53,
COMMON and GLOBAL	Optional keywords used to specify common and global storage; see Section 3.2.2.5 [Storage Class Keywords], page 54,
DIMENSION	Optional keyword used to specify that the ITEM declaration is an array; see Section 3.2.2.8 [DIMENSION Keyword], page 56,
PREFIX	Optional keyword used to concatenate a user-defined prefix to ITEM names, AGGREGATE member names, and named constants; see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49,
TAG	Optional keyword used to override the default OpenSDL code assigned to a name and to assign a user-defined tag instead; see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49,

Table 3.3: Item Declaration

## AGGREGATE Declaration

Keyword	Description
COMMON, GLOBAL, BASED, TYPEDEF	Optional keywords used to specify common, global, or based storage, or a TYPEDEF; see Section 3.2.2.5 [Storage Class Keywords], page 54,
DIMENSION	Optional keyword used to specify that the <b>AGGREGATE</b> declaration is an array; see Section 3.2.2.8 [DIMENSION Keyword], page 56,
FILL	Optional keyword used to indicate whether the associated aggregate or member occurs only as a fill to force byte alignment on the following member or aggregate, respectively.
MARKER	Optional keyword used to assign a user-defined prefix to the aggregate name; see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49,
ORIGIN	Optional keyword used to define the beginning of an aggregate with respect to an aggregate member
PREFIX	Optional keyword used to concatenate a user-defined prefix to <b>AGGREGATE</b> member names, <b>ITEM</b> names, and named constants; see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49,
TAG	Optional keyword used to override the default OpenSDL code assigned to a name and to assign a user-defined tag instead; see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49,

Table 3.4: AGGREGATE Declaration

## CONSTANT Declaration

Keyword	Description
COUNTER	Optional keyword that saves the last assigned value in a local symbol declaration for subsequent use.
EQUALS	Required keyword used in assigning the value to the first named constant <code>STRING</code> Optional keyword specified immediately after <code>EQUALS</code> to indicate the definition of a string constant.
INCREMENT	Optional keyword used to specify constants with incremental values
PREFIX	Optional keyword used to concatenate a user-defined prefix to aggregate member names and named constants; see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49,
RADIX	Optional keyword used to indicate how a constant should be written in the output file; see Section 3.2.2.3 [RADIX Keyword], page 53,
TAG	Optional keyword used to override the default OpenSDL code assigned to a name and to assign a user-defined tag instead; see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49,
TYPENAME	Optional keyword not currently used, but may be introduced when Ada or PL/I back-ends are developed; see Section 3.2.2.1 [User Specified TYPENAME Keyword], page 49,

Table 3.5: CONSTANT Declaration

## ENTRY Declaration

Parameter Passing Mechanism	Description
REFERENCE	Optional parameter-passing mechanism keyword used to specify that a parameter must be passed by reference (address); REFERENCE is the default
VALUE	Optional parameter-passing mechanism keyword used to specify that the parameter must be passed BY immediate VALUE

Table 3.6: ENTRY Declaration

Parameter Mode Keywords	Description
IN	Parameter description keyword used to indicate that a parameter is an input parameter; can be used with OUT to indicate that the parameter is both an input and an output parameter; IN is the default
OUT	Parameter description keyword used to indicate that a parameter is an output parameter; can be used with IN to indicate that the parameter is both an input and an output parameter

Table 3.7: PARAMETER Declaration

Other Parameter Modifier	Description
DEFAULT	Optional parameter description keyword used to specify a default parameter value
DIMENSION	Optional keyword used to specify that the parameter is an array; see Section 3.2.2.8 [DIMENSION Keyword], page 56,
LIST	Optional parameter description keyword used to indicate that the routine may be called with one or more parameters of the type being described
NAMED	Optional parameter description keyword used only by the Ada back end to name the parameter
OPTIONAL	Optional parameter description keyword used to specify that the parameter may or may not appear in the sequence of parameters using the entry point name
TYPENAME	Optional parameter description keyword used only by the Ada and PL/I back ends to specify a user-defined data type name; see Section 3.2.2.1 [User Specified TYPENAME Keyword], page 49,

Table 3.8: PARAMETER Modifiers

<b>Entry Return Value Keywords</b>	<b>Description</b>
NAMED	Optional keyword used to specify the name of the parameter (in an Ada IMPORT_VALUED_PROCEDURE) into which the return value is returned
RETURNS	Optional keyword used to specify the data type returned by the external entry
TYPENAME	Optional keyword used only by the Ada and PL/I back ends to specify a user-defined name that is the data type returned by the external entry; see Section 3.2.2.1 [User Specified TYPENAME Keyword], page 49,

Table 3.9: RETURN Value Keywords

<b>Entry Description Keywords</b>	<b>Description</b>
ALIAS	Optional keyword used to indicate an alternate internal name that can be used to designate the entry point
LINKAGE	Optional keyword used only by the MACRO back end to indicate that a special call macro will be used in the expansion of the entry macro
PARAMETER	Optional keyword used to describe the parameters of the external entry
VARIABLE	Optional keyword used to indicate that the entry point can be invoked with a variable number of parameters; see also the LIST parameter modifier keyword

Table 3.10: ENTRY Description Keywords

### 3.2.2.1 User-Specified TYPENAME keyword

**Note:** The TYPENAME keyword is part of the original SDL language format. This particular implementation does not currently support either the Ada or PL/I languages. As a result, this keyword is parsed but ignored in the current implementation.

The TYPENAME keyword is used to specify a data type name that is not an OpenSDL data type. Depending on which back end is specified, this name may or may not override the OpenSDL data type. The Ada and PL/I back ends use these data type names as parameter data types, as return value data types, and as CONSTANT declaration data types.

The TYPENAME keyword has the following syntax:

```
TYPENAME name
```

The Ada language translation (as a result of processing the OpenSDL source file `/usr/lib/opensdl/example/example.sdl`) shows an example of the TYPENAME keyword. The following is an example of the TYPENAME keyword on each of the parameters in an ENTRY declaration:

```
ENTRY SYS$FAO
  ALIAS $FAO
  PARAMETER (CHARACTER REFERENCE NAMED CTRSTR IN TYPENAME CHARDESC,
             WORD UNSIGNED NAMED OUTLEN OUT DEFAULT 0 TYPENAME NUMBER,
             CHARACTER REFERENCE NAMED OUTBUF OUT TYPENAME CHARDESC,
             LONGWORD VALUE NAMED P1 OPTIONAL LIST TYPENAME VARIES)
  RETURNS LONGWORD TYPENAME CONDVALLU;
```

### 3.2.2.2 PREFIX, MARKER, and TAG Keywords

User-defined prefixes, markers, and tags are optional character strings that help to uniquely identify the names associated with a particular facility or system. When the `--suppress` qualifier is specified on the command line, the inclusion of prefixes and/or tags on output symbol names is suppressed. See Section 2.2 [Processing an OpenSDL Source File], page 36, for a description of the `--suppress` qualifier.

#### PREFIX Keyword

The PREFIX option may be specified on an AGGREGATE, subaggregate, CONSTANT, or ITEM declaration to cause OpenSDL to concatenate a user-specified prefix and the name specified in the declaration.

When you specify a prefix for an aggregate, OpenSDL concatenates the prefix and the name of each member or named constant declared within the aggregate. The name of the aggregate itself is not altered by the use of the PREFIX option. The PREFIX option has the following syntax:

```
PREFIX prefix-string
```

#### prefix-string

Is a 0 to 32-character string that can be any valid OpenSDL identifier. If you specify the PREFIX option, OpenSDL constructs the identifier of each member by concatenating the prefix, a tag, an underscore, and the member name.

You can override a prefix that is currently in effect by specifying a new prefix for a particular aggregate member. If this member happens to be a subaggregate, the new prefix is applied to all the members of that subaggregate. Otherwise, if no prefix is specified for the subaggregate, all subaggregate members are assigned the same prefix as that specified on the parent aggregate.

## MARKER Keyword

You can use the **MARKER** keyword to assign a user-defined prefix to the aggregate name. The **MARKER** option has the following syntax:

**MARKER** *marker-string*

### **marker-string**

Is a 0 to 32-character string that can be any valid OpenSDL name that may or may not be enclosed in quotation marks (" ") and may be null.

## TAG Keyword

The **TAG** option overrides the default tags that OpenSDL uses in forming identifiers. You can specify tags for **CONSTANT**, **ITEM**, and **AGGREGATE** declarations and aggregate members. The tag that you specify, however, affects only the outer-level identifier. For example, a tag you supply in an **AGGREGATE** declaration affects only the aggregate name; if you wish to change all the tags in an aggregate, you must do it on a member-by-member basis. The **TAG** option has the following syntax:

**TAG** *tag-string*

### **tag-string**

Is a 0 to 32-character string specifying the tag to use in forming the name. If the **TAG** option is not specified, OpenSDL uses a default code based on the data type of the name.

If you specify the **TAG** option, OpenSDL appends the tag-string, which may be null (" "), and an underscore character (\_) to the current prefix-string. A tag consisting of a single underscore character produces a single underscore character in any resulting identifier.

The following is an example of the **PREFIX**, **TAG**, and **MARKER** keywords:

```
AGGREGATE operator STRUCTURE MARKER doowop_ PREFIX beebop_ TAG shoo;
    flink ADDRESS;
    blink WORD;
END;
```

The resulting output using the C/C++ backend is as follows:

```
struct doowop_shoo_operator
{
    void *beebop_a_flink;
    void *beebop_a_blink;
};
```

### Example 3.1: PREFIX, MARKER, and TAG Example

Table 3.11 shows the default tags that OpenSDL uses when the **TAG** option is not specified on an aggregate member.



<b>Data Type</b>	<b>Default Tag</b>
CONSTANT	K
BYTE [UNSIGNED]	B
WORD [UNSIGNED]	W
LONGWORD [UNSIGNED]	L
QUADWORD [UNSIGNED]	Q
OCTAWORD [UNSIGNED]	O
D_FLOATING	D
F_FLOATING	F
G_FLOATING	G
H_FLOATING	H
S_FLOATING	S
T_FLOATING	T
X_FLOATING	X
D_FLOATING COMPLEX	DC
F_FLOATING COMPLEX	FC
G_FLOATING COMPLEX	GC
H_FLOATING COMPLEX	HC
S_FLOATING COMPLEX	SC
T_FLOATING COMPLEX	TC
X_FLOATING COMPLEX	XC
DECIMAL	P
BITFIELD	V for BITFIELD offset S for BITFIELD size <sup>1</sup> M for BITFIELD mask
CHARACTER	T
ADDRESS	A
BOOLEAN	B
VOID	Z
INTEGER	IS
INTEGER_BYTE	IB
INTEGER_WORD	IW
INTEGER_LONG	IL
INTEGER_QUAD	IQ
INTEGER_HW	IH
POINTER_HW	PH
POINTER_LONG	PL
POINTER	PS
POINTER_QUAD	PQ
HARDWARE_ADDRESS	HA
HARDWARE_INTEGER	HI
STRUCTURE	R
UNION	R

Table 3.11: Default Tags Used by OpenSDL

<sup>1</sup> Identifiers with size and mask (if `MASK` is specified) tags are generated regardless of whether a `PREFIX` or `TAG` option is specified.

The following example shows the use of a user-specified prefix on an **AGGREGATE** declaration:

```

AGGREGATE operator STRUCTURE
    PREFIX opr_;
    id WORD;
    "typename" CHARACTER;
    CONSTANT (fixed_bin_,float_) EQUALS 0 INCREMENT 1;
    bits STRUCTURE;
        variable_size BITFIELD;
        size_units BITFIELD LENGTH 3;
    END bits;
END operator;

```

### Example 3.2: PREFIX on AGGREGATE Example

In the previous example, the member name "typename" is enclosed in quotation marks because it is an OpenSDL keyword.

The previous declaration produces the following names, with the prefix `opr_` and default tags, in the C/C++ output file:

```

#define opr_k_fixed_bin_ 0
#define opr_k_float_ 1
#define opr_s_operator 4 /* not sure if I generate this */
struct opr_r_operator
{
    int16_t opr_w_id;
    char opr_t_typename;
    struct opr_r_bits
    {
        uint8_t opr_v_variable_size : 1;
        uint8_t opr_v_size_units : 3;
        uint8_t opr_v_fill_0 :4;
    };
};

```

### Example 3.3: PREFIX on AGGREGATE C/C++ Output

The name `opr_v_fill_0` in the previous list is the result of a **BITFIELD** declaration that OpenSDL supplied because the subaggregate did not end on a byte boundary. The name (`opr_v_fill_0`) ensures that the next aggregate begins on a byte boundary. Section 4.5.7 [Forcing Data Alignment], page 78, describes data alignment in detail. OpenSDL uses default codes followed by an underscore (`_`) for the tag portion of a prefix when the **TAG** option is not specified. You can override the default OpenSDL codes by specifying a tag, which may be null (`""`). For example:

```

CONSTANT (abc,def,ghi) EQUALS 0 INCREMENT 1
    PREFIX new
    TAG "";

```

This declaration results in the names `new_abc`, `new_def`, and `new_ghi`.

### 3.2.2.3 RADIX Keyword

The RADIX keyword can be used to indicate how a constant value should be written out to the output file. It can have one of three possible values.

- DEC — for Decimal
- OCT — for Octal
- DEC — for Hexadecimal

For example:

```
CONSTANT MyConst EQUALS 42 RADIX HEX;
```

will generate the following C code header file:

```
#define MyConst 0x2a
```

**Note:** When a BITMASK has the optional MASK keyword, constant values are generated consistent with the RADIX HEX keyword.

### 3.2.2.4 Alignment Keywords

Both the ALIGN and BASEALIGN keywords can ensure that items are properly aligned. The BASEALIGN keyword takes an argument, which specifies the alignment, whereas the ALIGN keyword uses the natural alignment of the item.

- If the ALIGN keyword is included in the definition of an aggregate, every member will be aligned. Both i1 and i2 will be aligned.

```
AGGREGATE MyStruct STRUCTURE ALIGN;
    c1 CHARACTER;
    i1 LONGWORD;
    c2 CHARACTER;
    i2 LONGWORD;
END;
```

- If the ALIGN keyword is included in the definition of a member of an aggregate, this member will be aligned, even if the AGGREGATE itself does not have the ALIGN attribute. i2 will be aligned whereas i1 will not be aligned.

```
AGGREGATE MyStruct STRUCTURE NOALIGN;
    c CHARACTER;
    i1 LONGWORD;
    i2 LONGWORD ALIGN;
END;
```

- If the NOALIGN keyword is included in the definition of a member of an aggregate, no action will be taken to ensure that this member will be aligned, even if the AGGREGATE has the ALIGN attribute. i2 will be aligned, i1 will not be aligned.

```
AGGREGATE MyStruct STRUCTURE ALIGN;
    c CHARACTER;
    i1 LONGWORD NOALIGN;
    i2 LONGWORD;
END;
```

- If the `NOALIGN` keyword is included in the definition of an aggregate, no action will be taken to ensure that any member of this aggregate will be aligned. Neither `i2` nor `i1` will be aligned.

```
AGGREGATE MyStruct STRUCTURE NOALIGN;
    c CHARACTER;
    i1 LONGWORD;
    i2 LONGWORD;
END;
```

- If the `BASEALIGN` keyword is included in the definition of a member of an aggregate, this member will be aligned according to the given alignment. In this case, `i2` will have an offset of 256 ( $2^8$ ),

```
AGGREGATE MyStruct STRUCTURE NOALIGN;
    c CHARACTER;
    i1 LONGWORD;
    i2 LONGWORD BASEALIGN(8);
END;
```

- If the `BASEALIGN` keyword is included in the definition of an aggregate or an item, the aggregate or item itself will be padded, so that in an array of elements of this aggregate or item type, all elements will be aligned according to the given alignment. The syntax is as follows:

```
AGGREGATE MyStruct STRUCTURE BASEALIGN (2);
    c CHARACTER;
    i1 LONGWORD;
    i2 LONGWORD;
END;
```

The size of the aggregate will be a multiple of 4 ( $2 \sum 2$ ), in this case 12, and neither `i1` nor `i2` will be aligned. In the following example, the item will have a size of 8 ( $2^3$ ).

```
ITEM MyItem LONGWORD UNSIGNED BASEALIGN (3);
```

### 3.2.2.5 Storage Class Keywords

Storage class refers to the way in which the target language compiler allocates storage for scalar items and aggregates. In general, declarations produce a template describing data for which the compiler allocates storage dynamically at run time, rather than at compile time. This type of storage is the default and is specified using the `BASED` option in some languages, although the default storage class option is language-dependent. The back end can generate the `BASED` storage class option, which has the following syntax:

```
BASED (pointer-name)
```

In languages that support the construct, you can use the `BASED` pointer-name option on an `AGGREGATE` declaration to bind a named pointer to that aggregate. In all target languages, the aggregate resulting from such a declaration has the default storage class (`BASED`).

A subaggregate always acquires the storage class of the aggregate to which it belongs. The default storage class associated with any declaration is language-dependent. You can override the default storage class by specifying either of the following storage classes:

- **Common storage** is allocated in an external program section (Psect) with the OVR option and is shared by all routines that reference it. You declare data in common storage by using the **COMMON** option on an **AGGREGATE** or **ITEM** declaration.
- **Global storage** represents data in a global storage location whose value is defined elsewhere. You declare global data by using the **GLOBAL** option on an **AGGREGATE** or **ITEM** declaration.
- **TYPDEF** behaves like a storage class. In C/C++, examples of storage classes are **static**, **globaldef/ref**, **extern**, etc. Syntactically, you can replace a **static** in any declaration with **typedef** and have it compiled. Storage classes (including **TYPDEF**) are mutually exclusive in an OpenSDL declaration.

Example of C/C++ type definition:

```
typedef struct {int32_t jg_l_i1; int32_t j_l_i2;} MyStruct;
static MyStruct foo;
```

is equivalent to:

```
static struct {int32_t jg_l_i1; int32_t jg_l_i2;} foo;
```

or:

```
static struct MyStruct {int32_t jg_l_i1; int32_t jg_l_i2;} foo;
```

or:

```
struct MyStruct {int32_t jg_l_i1; int32_t jg_l_i2;};
static struct MyStruct foo;
```

If **COMMON** and **GLOBAL** appear together in a declaration, a **SDL\_DUPCONATT** (duplicate or conflicting attributes) error is given. The **COMMON**, **GLOBAL** and **TYPDEF** storage classes are mutually exclusive.

### 3.2.2.6 OpenSDL Storage Classes and Typedef Syntax

Explicit OpenSDL Storage classes are **COMMON** and **GLOBAL**. To maintain orthogonal syntax, **TYPDEF** is permitted wherever **COMMON** and **GLOBAL** are permitted.

The Storage class definition in OpenSDL is included as an option on an **ITEM** or **AGGREGATE** declaration. This means that an **AGGREGATE TYPDEF** for the preceding example would be:

```
AGGREGATE MyStruct STRUCTURE TYPDEF PREFIX jg_;
    i1 LONGWORD;
    i2 LONGWORD;
END;
```

### 3.2.2.7 Data Types

In most cases it is possible to refer to a user-defined Data Type where a standard built-in OpenSDL Data Type (**BYTE**, **LONGWORD** etc.) can be referenced. Reference to the type **MyStruct** defined in the preceding example is made as follows:

```
ITEM foo MyStruct;
```

### 3.2.2.8 DIMENSION Keyword

You can apply a dimension to **AGGREGATE** (as well as members of aggregates and subaggregates) and **ITEM** declarations, which means that you can define an array of structures, a structure that contains one or more arrays, or an array of structures each of which contains one or more arrays. The **DIMENSION** option is valid when specified with any of the data types described in Section 3.2.3 [Data Type Keywords], page 57, and has the following syntax:

```
DIMENSION [lbound:]hbound
```

**lbound**

Is any valid OpenSDL expression giving the value of the lowest-numbered element of the array. If **lbound** is not specified, OpenSDL supplies a default **lbound** of 1.

**hbound**

Is any valid OpenSDL expression giving the number of elements in the array, or, if **lbound** is specified, the highest-numbered element.

The following is an example of the **DIMENSION** option specified on an **AGGREGATE** declaration:

```
AGGREGATE array_info STRUCTURE;
  bound STRUCTURE DIMENSION 8;
    lower LONGWORD;
    upper LONGWORD;
    multiplier LONGWORD;
    constant_lower BITFIELD LENGTH 1;
    constant_upper BITFIELD LENGTH 1;
    constant_multiplier BITFIELD LENGTH 1;
    reserved BITFIELD LENGTH 13;
  END bound;
END array_info;
```

The subaggregate bound has eight elements. Each element consists of the members upper, lower, multiplier, and so on. Because bound is an array, each of its members (upper, lower, multiplier, and so on) can also be considered an array of eight elements. When you specify a single value after the **DIMENSION** keyword, as in the previous example, the specified back end assumes that the value represents the high bound value and supplies a default low bound value of one. You can override this default by specifying both a low bound and a high bound value as follows:

```
ITEM node_pointers DIMENSION 0:255 ADDRESS;
```

This declaration results in a declaration of the array `node_pointers`, whose low bound is 0 and whose high bound is 255. Only one dimension can be specified for an **AGGREGATE** or **ITEM** declaration. This restriction ensures that there is no interlanguage conflict in an array declaration.

You can see how OpenSDL translates the **DIMENSION** option for each output language by processing the OpenSDL source file `/usr/lib/opensdl/examples/example.sdl`.

### 3.2.3 Data Type Keywords

Data type keywords specify the data types of scalar objects, which can be declared as members of aggregates or as individual items. Data type keywords are also used to describe the data types of parameters, as well as the return value of an **ENTRY**. They can also be used in an **AGGREGATE** declaration to generate an implicit union. The data type declaration also specifies, either implicitly or explicitly, the size of a member.

The following sections describe the data types and the keywords you use to specify them.

#### 3.2.3.1 Pointer Data Types

The keywords **ADDRESS**, **POINTER**, **POINTER\_LONG**, **POINTER\_QUAD**, **POINTER\_HW**, and **HARDWARE\_ADDRESS** specify a data type that is an address, or pointer. The **ADDRESS** data type has the following syntax:

```
pointer-type [ (object-type [ basealign-attribute ] ) ]
```

**pointer-type**

is one of the keywords **ADDRESS**, **POINTER**, **POINTER\_LONG**, **POINTER\_QUAD**, **POINTER\_HW**, and **HARDWARE\_ADDRESS**.

**ADDRESS**, **POINTER**, **POINTER\_LONG** are 4-byte-addresses. **POINTER\_HW** and **HARDWARE\_ADDRESS** are 4-byte-addresses if **-32** is specified and 8-byte-addresses if **-64** is specified. **POINTER\_QUAD** is an 8-byte-address.

**object-type**

is the optional data type of the object to which the address refers. This construct is ignored for languages in which pointers are distinct data types. Object-type is either a built-in object type, like **LONGWORD** or **ANY**, or a user-defined object type, optionally followed by a **DIMENSION** specification, or an **ENTRY** declaration.

**basealign-attribute**

Here, a **BASEALIGN** attribute can be specified, as described in Section 3.2.2.4 [Alignment Keywords], page 53,

The following is an example of an aggregate with a member of pointer type:

```
AGGREGATE any_node STRUCTURE;
    flink ADDRESS (any_node);
    blink ADDRESS (any_node);
END;
```

#### 3.2.3.2 ANY Data Type

The **ANY** keyword specifies that the parameter being described in an **ENTRY** declaration can be of any data type. The **ANY** data type can be used only within the context of a parameter description and has the following syntax:

**ANY**

**ANY**

Specifies that the parameter can be of any data type. The following is an example of the use of the **ANY** data type:

```
ENTRY sys_abc PARAMETER (ANY);
```

### 3.2.3.3 BITFIELD Data Type

The **BITFIELD** keyword specifies a bit field variable that must be a member of an **AGGREGATE** declaration. The **BITFIELD** data type has the following syntax:

```
BITFIELD [LENGTH n] [MASK] [SIGNED] [RADIX x];
[LENGTH n]
```

Is any valid OpenSDL expression giving the number of bits in the bitfield. If no length is specified, OpenSDL uses a default length of 1 bit.

```
[MASK]
```

Is a keyword specifying that OpenSDL generate both a bitfield variable and a constant bit mask representing the bits defined in this field.

```
[SIGNED]
```

Is a keyword specifying that OpenSDL treat the output as a signed field.

```
[RADIX x]
```

Is a keyword where 'x' is one of **DEC**, **OCT**, or **HEX**.

Bitfields must be **AGGREGATE** declaration members. They cannot be scalar items, objects of **ADDRESS** declarations, parameters, or return data types of entries. The following is an example of the **BITFIELD** keyword used in an **AGGREGATE** declaration that specifies a structure with two bitfield members:

```
AGGREGATE flags STRUCTURE PREFIX tst_;
    resolved BITFIELD MASK SIGNED;
    spare_bits BITFIELD LENGTH 5;
END;
```

The declaration of **resolved** in the previous example results in two declarations in the output file: a declaration for the bitfield itself and a declaration of a constant mask whose value is 1. Because the **PREFIX** option is specified for this aggregate, the source output file identifiers produced for this declaration are **tst\_v\_resolved** and **tst\_m\_resolved**, where the tag **v\_** indicates the bitfield variable and the tag **m\_** indicates the mask. Prefixes and tags are described in more detail in Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49.

### 3.2.3.4 BOOLEAN Data Type

The **BOOLEAN** keyword specifies a Boolean data type that is a one-byte field that can have one of two values, 0 or 1. The **BOOLEAN** data type has the following syntax:

```
BOOLEAN
BOOLEAN
```

Produces a Boolean variable in the output file.

The following is an example of the **BOOLEAN** keyword used in an **ITEM** declaration:

```
ITEM true BOOLEAN;
```



### 3.2.3.5 CHARACTER Data Type

The **CHARACTER** keyword declares a character string of a given length. The **CHARACTER** data type has the following syntax:

```
CHARACTER [LENGTH {n}] [VARYING]
          [      {*}]
```

[LENGTH n]

Is the length of the character string. The length can be specified using any valid OpenSDL expression. If no length is specified, OpenSDL uses a default length of 1 character. You can specify a character string of unknown length using **LENGTH \***. An unknown length character string can be specified only for parameter types. For example:

```
ENTRY LIB$ROUTINE PARAMETER(CHARACTER LENGTH * NAMED foo);
```

[VARYING]

Is a keyword indicating that the identifier represents a varying-length character string (for languages that support this data type). In a varying-length character string, the first word of the string contains its current length; its declared length is the maximum length that it can have.

The following is an example of the **CHARACTER** data type used in an **AGGREGATE** declaration. The aggregate **msg\_buffer** contains a character-string member named **message\_text** with a length of 256 characters.

```
AGGREGATE msg_buffer STRUCTURE;
    message_text CHARACTER LENGTH 256 VARYING;
    severity WORD;
END;
```

### 3.2.3.6 COMPLEX Data Types

The **COMPLEX** keyword immediately follows any of the four floating-point data type keywords to specify any of the seven **COMPLEX** data types. The **COMPLEX** data types have the following syntax:

```
D_FLOATING COMPLEX
F_FLOATING COMPLEX
G_FLOATING COMPLEX
H_FLOATING COMPLEX
S_FLOATING COMPLEX
T_FLOATING COMPLEX
X_FLOATING COMPLEX
```

The following is an example of the **T\_FLOATING COMPLEX** data type used in an **ITEM** declaration:

```
ITEM foo T_FLOATING COMPLEX;
```

Section 3.2.3.8 [Floating Point Data Types], page 60, describes floating-point data types.

### 3.2.3.7 DECIMAL Data Type

The **DECIMAL** keyword specifies a packed decimal data type and the size of the data type. The **DECIMAL** data type has the following syntax:

```
DECIMAL PRECISION (precision,scale)
```

```
PRECISION (precision,scale)
```

Is the fixed-point decimal members precision and scale, respectively. Precision is the total number of decimal digits, and scale is the number of fractional digits.

Both precision and scale must be specified using valid OpenSDL expressions. The following is an example of the **DECIMAL** keyword used in an **ITEM** declaration that indicates a packed decimal data type consisting of three decimal digits, two of which are fractional:

```
ITEM percentage DECIMAL PRECISION (3,2);
```

### 3.2.3.8 Floating Point Data Types

The floating-point keywords, **D\_FLOATING**, **F\_FLOATING**, **G\_FLOATING**, **H\_FLOATING**, **S\_FLOATING**, **T\_FLOATING**, and **X\_FLOATING**, declare storage units for single, double, and long-double floating-point data, respectively. The floating-point data types have the following syntax:

```
D_FLOATING
F_FLOATING
G_FLOATING
H_FLOATING
S_FLOATING
T_FLOATING
X_FLOATING
```

**Note:** **F\_FLOATING**, **D\_FLOATING**, **G\_FLOATING**, and **H\_FLOATING** are OpenVMS VAX and OpenVMS Alpha floating-point formats. OpenSDL does not support these formats, but assumes the format of the underlying operating system and hardware. In order to be able to be backward compatible with earlier versions of this software, the following equivalences have been established:

- **F\_FLOATING** is the same as **T\_FLOATING**
- **D\_FLOATING** and **G\_FLOATING** are the same as **S\_FLOATING**
- **H\_FLOATING** is the same as **X\_FLOATING**

The following is an example of the **T\_FLOATING** data type used in an **ITEM** declaration:

```
ITEM foo T_FLOATING;
```

Section 3.2.3.6 [COMPLEX Data Types], page 59, describes **COMPLEX** data types.

### 3.2.3.9 Integer Data Types

The keywords **BYTE**, **WORD**, **LONGWORD**, **QUADWORD**, and **OCTAWORD** declare storage units of 8, 16, 32, 64, and 128 bits, respectively, to represent signed integer data. You may also specify the keyword **UNSIGNED** with any of these data types to indicate unsigned integer data.

It is also possible to specify the keyword **SIGNED**, which is the default if nothing is specified.

The keywords **INTEGER\_BYTE**, **INTEGER\_WORD**, **INTEGER\_LONG** and **INTEGER\_QUAD** are synonyms for **BYTE**, **WORD**, **LONGWORD**, **QUADWORD**, respectively, although some back ends may

treat `INTEGER_QUAD` and `QUADWORD` differently. The keyword `INTEGER` is also synonym for `LONGWORD` and `INTEGER_LONG`. The keywords `INTEGER_HW` and `HARDWARE_INTEGER` describe integer data types whose size depends on the underlying hardware. If the qualifier `-32` is specified, they are 4 bytes wide, and if `-64` is specified, they are 8 bytes wide.

The integer data types have the following syntax:

<code>BYTE</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>INTEGER_BYTE</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>WORD</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>INTEGER_WORD</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>LONGWORD</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>INTEGER_LONG</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>INTEGER</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>QUADWORD</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>INTEGER_QUAD</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>OCTAWORD</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>INTEGER_HW</code>	<code>[ UNSIGNED   SIGNED ]</code>
<code>HARDWARE_INTEGER</code>	<code>[ UNSIGNED   SIGNED ]</code>

The following are examples of the `LONGWORD` and `BYTE` data types used in an `ITEM` declaration:

```
ITEM foo LONGWORD UNSIGNED;
ITEM bar BYTE;
```

### 3.3 Expressions

An OpenSDL expression evaluates to an arithmetic value and can consist of any of the following syntax elements:

- Numeric values are, by default, expressed in decimal notation. You can override this default by preceding a constant with one of the prefixes in the following table.

Prefix	Interpretation	Valid Characters
<code>%X</code>	Hexadecimal	0-9, A-F, a-f
<code>%O</code>	Octal	0-7
<code>%B</code>	Binary	0 and 1
<code>%A<sup>1</sup></code>	ASCII value	Any ASCII character (see Appendix C [ASCII], page 107)

OpenSDL treats decimal constants as signed integer longword values.

You can also use a string of up to four characters as a numeric constant by enclosing the string in quotation marks (" "). OpenSDL inserts the ASCII value of each character into the byte field corresponding to that character's position in the string. If the string you specify has fewer than four characters, OpenSDL pads the string with the null character, which has the ASCII code of zero.

- Local symbols and output constants are assigned integer longword values that are available within the context (that is, during processing) of the input file.

<sup>1</sup> The `%A` operator takes the ASCII value of any ASCII character that follows it.

- Operators perform arithmetic and logical operations on numeric values, local symbols, and output constants. The following table lists the operators in order of precedence, with the operators of higher precedence listed first.

Operator	Meaning
<b>unary -</b>	Arithmetic negation
<b>*</b>	Arithmetic multiplication
<b>/</b>	Arithmetic division
<b>+</b>	Arithmetic addition
<b>-</b>	Arithmetic subtraction
<b>@</b>	Logical shift <b>x@y</b> shifts the value of <b>x</b> to the left <b>y</b> places; if <b>y</b> is negative, the value of <b>x</b> is shifted <b>y</b> places to the right
<b>&amp;</b>	Logical AND
<b> </b>	Logical OR
<b>~</b>	Logical NOT

- Offset symbols are used in expressions specified in **AGGREGATE** declarations:
  - The period (.) represents the current byte offset from the origin in an **AGGREGATE** declaration. If the **ORIGIN** option is specified, the value of the period is equal to the byte offset from the member specified using the **ORIGIN** option.
  - The colon (:) represents the current byte offset relative to the first member in an **AGGREGATE** declaration. The value is not affected by the presence of an **ORIGIN** option.
  - The circumflex (^) represents the current bit offset relative to the most recently declared aggregate or byte-aligned element. Section 4.5.8 [Using Offset Symbols], page 80, describes the use of offset symbols in **AGGREGATE** declarations in more detail.
- Parentheses group expressions to define the order of evaluation. Expressions within the innermost set of parentheses are evaluated first. The following is an example of an expression used in a **CONSTANT** declaration, which appears in the context of an **AGGREGATE** declaration:

```
CONSTANT foo EQUALS %Ag + 72 / (#abc * boo + .);
```

### 3.4 Local Comments

A comment that is local to the OpenSDL source file is not written to the output file. Local comments begin with the left brace ({) and extend to the end of the line. They can appear anywhere within the source file (not necessarily within a module) where white space (a space, tab, or carriage return) is allowed. The following is an example of a local comment:

```
{ Assigning the value 255 to #max_args.
```

### 3.5 Output Comments

There are two types of output comments, Line Comments and Block Comments.

### 3.5.1 Line Comments

Line comments appearing on lines by themselves are typically written to the output file as separate comment lines. Line comments appearing at the end of a line are output at the end of the corresponding target source line, if possible.

Line comments begin with a slash and an asterisk (/\*) and terminate at the end of the current line. They can appear in the following contexts:

- Outside **MODULE** declarations
- At the end of a line containing a declaration, that is, following the semicolon terminator(;;)
- On lines by themselves between member, **CONSTANT**, **ENTRY**, **ITEM**, and **AGGREGATE** declarations
- Between declarations with in an aggregate
- Following individual constant names with in a comma-delimited list of **CONSTANT** declarations

The following is an example of a line comment:

```
/* Get Job/Process Information System Service.
```

### 3.5.2 Block Comments

Block comments are those that appear as a large multi-line comment and are written to the output file as multiple comment lines.

Block comments begin with a slash and a plus sign (/+) and end with a slash and a minus sign (/−). Within these begin and end comments, lines beginning with a double slash (//) will be prefixed with a middle comment (if the target language supports it). These comments can appear anywhere Output Comments can appear.

The following are two examples of a block comments:

```

/+
// Get Job/Process Information System Service.
/−

/+
  Copyright 2018.
/−
```

## 3.6 INCLUDE Statement

The **INCLUDE** statement specifies that the contents of an external file are to be incorporated in the OpenSDL input file directly following the **INCLUDE** statement. The **INCLUDE** statement has the following syntax:

```

    INCLUDE "file-spec";
"file-spec"
```

Is any valid file specification enclosed in quotation marks (" "). An **INCLUDE** statement cannot appear embedded within an **AGGREGATE** declaration, but can appear anywhere else within the module. If a directory is not included in the file specification, the current default directory is used.

When OpenSDL encounters an `INCLUDE` statement, it stops reading from the current file and reads the statements in the included file. When it reaches the end of the included file, OpenSDL resumes translation with the source statement immediately following the `INCLUDE` statement.

## 3.7 Conditional OpenSDL Compilation

### 3.7.1 Conditional OpenSDL Compilation using the `IFLANGUAGE` statement

This feature allows a section of OpenSDL source code to be conditionally compiled, depending on whether output is being generated for a particular language or not.

The syntax for conditional compilation has the format:

```
IFLANGUAGE language-name [ language-name ... ] ;
.
.
.
[ELSE ;
.
.
.
]
END_IFLANGUAGE [ language-name [ language-name ... ] ] ;
```

Note the following:

- The three keywords may appear wherever a statement is valid.
- Conditional compilation statements may not be nested.
- The list of language-names on the `END_IFLANGUAGE` is optional, but if it is included, it must match the list on the `IFLANGUAGE` statement. The languages need not necessarily appear in the same order.
- Language names may not be abbreviated.
- The validity of language names is not checked. This is in keeping with the philosophy of OpenSDL that new back ends may be added without changes to the front end.
- A comment on the same line as the `IF_LANGUAGE` statement is only output for languages which satisfy the condition. A comment on the same line as the `END_IFLANGUAGE` statement is always output, as this is considered to be outside the body of the conditional.

In the following example, OpenSDL generates a translation of the `ITEM` statement if output is being generated for Pascal, Ada, or FORTRAN. For other languages, OpenSDL does not generate a translation of the `ITEM` statement.

```
IFLANGUAGE PASCAL ADA FORTRAN;
    ITEM foo LONGWORD;
END_IFLANGUAGE PASCAL ADA FORTRAN;
```

### 3.7.2 Conditional OpenSDL Compilation using the IFSYMBOL statement

This feature allows a section of OpenSDL source code to be conditionally compiled, depending on symbols specified with the command line qualifier `--symbol`. The syntax for conditional compilation has the format:

```

IFSYMBOL symbol-name ;
    .
    .
    .
[ ELSE_IFSYMBOL symbol-name ;
    .
    .
    .
]
[ELSE ;
    .
    .
    .
]
END_IFSYMBOL ;

```

Note the following:

- The four keywords may appear wherever a statement is valid.
- Conditional compilation statements may not be nested.
- A comment on the same line as the `IFSYMBOL` statement is only output if this symbol is specified on the command line. A comment on the same line as the `END_IFSYMBOL` statement is always output, as this is considered to be outside the body of the conditional.

With the following example:

```

IFSYMBOL s1;
    <sdl code 1>
END_IFSYMBOL;
IFSYMBOL s2;
    <sdl code 2>
ELSE_IFSYMBOL s3;
    <sdl code 3>
ELSE;
    <sdl code 4>
END_IFSYMBOL;

```

Example 3.4: Conditional Compilation Example

```
$ openssl --lang:cc --symbol=s1:0 --symbol:s2=0 --symbol=s3:0 <file-spec>
```

produces

```
    <sdl code 4>
```

```
$ openssl --lang:cc --symbol=s1:1 --symbol:s2=0 --symbol=s3:0 <file-spec>
```

```

produces
    <sdl code 1>
    <sdl code 4>
$ openssl --lang:cc --symbol=s1:1 --symbol:s2=1 --symbol=s3:0 <file-spec>
produces
    <sdl code 1>
    <sdl code 2>
$ openssl --lang:cc --symbol=s1:1 --symbol:s2=0 --symbol=s3:1 <file-spec>
produces
    <sdl code 1>
    <sdl code 3>

```

### 3.8 Text Pass-through

Text pass-through allows literal text to be passed through to the output language file without translation. It is normally used in conjunction with conditional compilation for a specific target language. The purpose is to allow language-specific constructs, which cannot be represented in OpenSDL, to be emitted.

The syntax for text pass-through is:

```

LITERAL;
Any number of lines to be passed directly to
the output stream without translation
END_LITERAL;

```

Note the following:

- The keywords `LITERAL` and `END_LITERAL` may appear wherever a statement is valid.
- The keyword `END_LITERAL` terminates a literal construct, and therefore cannot be included in a line of literal text. Any text preceding `END_LITERAL` on the same line is output as a line of literal text.

Literal text is processed identically in all back ends. The literal text is merely written directly to the output file.

The following is an example of text pass-through for the C/C++ language.

```

IFLANGUAGE CC
LITERAL
#define ctext "This appears in C language output only"
END_LITERAL
END_IFLANGUAGE CC

```

Literal text within an `AGGREGATE` that contains a member definition, will not be taking into account when calculating offsets. It is recommended to use conditional compilation Section 3.7 [Conditional OpenSDL Compilation], page 64, around text pass-through and include a local variable Section 3.1.1 [Local Symbol Names], page 42, to account for the added offset and then add this local variable when using the offset after the text pass-through. Also, do not forget to define the local variable set to zero when the conditional code is not included, so that there an error message will not be generated as the result of the missing variable.



### 3.9 DECLARE Statement

The **DECLARE** statement allows you to declare a data item of a type that you define, which may be unknown in the current OpenSDL compilation. When you use **DECLARE**, the type is made known when the target language source is compiled.

The **DECLARE** statement uses the **SIZEOF** clause to allow you to specify the size of the user-defined data type being declared. The parameter you specify in the **SIZEOF** clause may be a built-in OpenSDL data type, a user-defined type defined in the current OpenSDL compilation, or an expression.

The **DECLARE** statement has the following syntax:

```
DECLARE user-type SIZEOF { data-type }      [ PREFIX prefix-string ]
                        { user-type }      [ TAG tag-string ]
                        { ( expression ) }
```

**user-type**

Represents the unknown data type name you wish to declare.

**SIZEOF**

A clause that must be appended to user-type. The **SIZEOF** clause may be specified in several ways, as shown, to indicate the size of the user-defined type being declared.

**data-type**

Represents either a built-in OpenSDL data type, a user-defined type that is known at OpenSDL compile time, or a data type that has been sized by a previous **SIZEOF** clause.

**user-type**

Represents a user-defined data type.

**( expression )**

Represents an expression. If specified, the expression must specify the number of bytes to be reserved for this data type. If the data type is dimensioned, the **SIZEOF** clause must specify the size of a single element. When you specify an expression, always enclose it in parentheses, as shown in this syntax.

Notes:

1. **DECLARE** identifies the size of the data type when included in an **AGGREGATE** declaration. (OpenSDL needs to know its equivalent predefined type so that the correct default tag letter, if required, can be output.)
2. You may declare a user-defined data type more than once (either explicitly or implicitly), but any subsequent declaration must match the first.
3. The default tag letter for the unknown variable being sized is derived from the type specified in the **SIZEOF** clause. If you use an expression in place of a data type to reserve a fixed number of bytes, the default tag letter is T. You may override the default tag by using an explicit **TAG** option.
4. **SIZEOF** clauses cannot be nested.
5. You cannot qualify a reference to the name of a previously-declared aggregate using the **SIZEOF** clause.

6. Although `DECLARE` statements and implicit `SIZEOF` declarations appear in the output tree for use by the back ends, these do not result in specific generated code.
7. Do not use the `SIZEOF` clause for data types that are aggregate names. Also, do not nest `SIZEOF` clauses where the syntax would otherwise allow aggregate names. For example, the following statements are valid:

```
DECLARE type SIZEOF ADDRESS (CHARACTER);
ITEM type ADDRESS (bar SIZEOF LONGWORD);
```

However, the following statement generates an error message:

```
DECLARE type SIZEOF ADDRESS (bar SIZEOF LONGWORD);
```

## Examples

A database contains information on a number of forests in a region, each of which contains a number of different types of trees. The definition of the tree structure is held in a different OpenSDL file from the other definitions. When the second OpenSDL file is compiled, the composition of the tree structure is unknown — the definitions will only come together when the output files are included in a compilation in the target language. The following shows the two OpenSDL files and the corresponding output in C/C++.

tree1.sdl:

```
AGGREGATE tree STRUCTURE TYPEDEF;
    flink ADDRESS (tree);
    blink ADDRESS (tree);
    height LONGWORD;
    age LONGWORD;
END;
```

tree2.sdl:

```
DECLARE tree SIZEOF (16);
AGGREGATE forest STRUCTURE TYPEDEF;
    oak tree;
    ash tree;
    elm tree;
    conifers tree DIMENSION 6;
END;
ITEM tree_pointer ADDRESS (tree);
ITEM tree_storage tree DIMENSION 1000;
ITEM region forest DIMENSION 4; {No SIZEOF, since forest defined here }
```

tree1.h:

```
typedef struct _tree
{
    _tree *flink;
    _tree *blink;
    long int height;
    long int age;
} tree;
```

tree2.h:

```
typedef struct _forest
```

```
{
    tree oak;
    tree ash;
    tree elm;
    tree conifers[6];
} forest;
tree *tree_pointer;
tree tree_storage[1000];
forest region[4];
```

Note that the `sizeof` information is discarded by C, but is used in offset calculations.



## 4 OpenSDL Declarations

This chapter describes the function and format of each of the following OpenSDL declarations:

- **MODULE** declaration
- **ITEM** declaration
- **AGGREGATE** and subaggregate declarations
- **CONSTANT** declaration
- **ENTRY** declaration

OpenSDL declarations are composed of the language elements described in Chapter 3 [Elements], page 41. The output generated by each OpenSDL declaration depends on which back end is used for the translation.

Online examples of output files for each language are available by processing the OpenSDL source file `/usr/lib/opensdl/examples/example.sdl`. Appendix B [Summaries], page 103, provides translation summaries for each output language.

### MODULE Declaration

The following sections describe the function and format of a **MODULE** declaration.

#### 4.1 MODULE Description

A **MODULE** declaration groups all related symbols and data structures. All declarations (other than local symbol assignments and local and output comments) must occur within a module, which is delimited by the **MODULE** and **END\_MODULE** keywords. An OpenSDL source file may contain multiple **MODULE** declarations, but modules may not be nested. You must specify a module name on the **MODULE** declaration; this name corresponds to the name of a macro or module in a given output language. (Section 3.1 [Names], page 41, describes the syntax for names.) For example:

```
MODULE _moddef;
```

This declaration generates the beginning of the declaration for the macro or module named `_moddef`.

You can use the **IDENT** option on a **MODULE** declaration to pass a version number or other information that must be enclosed in quotation marks (" ") to the output file. For example:

```
MODULE params IDENT "V2.0";
```

You can optionally specify the module name on the **END\_MODULE** statement; this is particularly useful if you place more than one **MODULE** declaration in the same OpenSDL source file. The module name is case-sensitive and must match exactly (in case) the name specified on the **MODULE** declaration.

## 4.2 MODULE Format

A MODULE declaration has the following syntax:

```
MODULE module-name [ IDENT "ident-string" ];
    [ module-body ];
    .
    .
    .
END_MODULE [ module-name ];
```

**MODULE module-name**

Specifies any valid OpenSDL identifier you want to use to identify the module.

**[IDENT "ident-string"]**

Specifies any valid OpenSDL identifier, or a string of any characters, that must be enclosed in quotation marks (" ") and that either further identifies the module or is a version number.

**[module-body]**

Is one or more of the following:

- ITEM declaration
- AGGREGATE declaration
- CONSTANT declaration
- ENTRY declaration
- Local symbol assignment
- Local or output comment
- INCLUDE statement
- DECLARE statement
- IFLANGUAGE construct
- IFSYMBOL construct
- LITERAL construct

**END\_MODULE [module-name]**

Marks the end of the MODULE declaration. The module-name, if specified, must match the name on the most recently specified MODULE declaration.

## ITEM Declaration

The following sections describe the function and format of an `ITEM` declaration.

### 4.3 ITEM Description

An `ITEM` declaration defines scalar items and single-dimensional arrays of scalar items that are not members of aggregates. You must specify an item name and data type on the `ITEM` declaration. For example:

```
ITEM block_list_id WORD;
```

This declaration specifies the scalar item `block_list_id` of data type `WORD`.

### 4.4 ITEM Format

An `ITEM` declaration has the following syntax:

```
ITEM item-name { data-type }          [ COMMON ]
                        { user-type sizeopt } [ GLOBAL ]
                                                [ TYPEDEF ]

                                                [ BASEALIGN basealign-option ]

                                                [ DIMENSION [ lbound: ] hbound ]

                                                [ PREFIX prefix-string ]

                                                [ TAG tag-string ];
```

**ITEM item-name**

Specifies any valid OpenSDL name used to identify the item.

**data-type**

Is any valid OpenSDL data type (see Section 3.2.3 [Data Type Keywords], page 57).

**user\_type sizeopt**

Is a user-defined data type using the `DECLARE` statements `SIZEOF` clause, shown and described in Section 3.9 [DECLARE Statement], page 67.

[ COMMON ]

[ GLOBAL ]

[ TYPEDEF ]

Is the storage class of the item, if other than the default (`BASED`) storage class (see Section 3.2.2.5 [Storage Class Keywords], page 54).

[ BASEALIGN basealign-option ]

`BASEALIGN` specifies the alignment of an `ITEM`. `basealign-option` is either an integer expression in parentheses or the name of a data type. `BASEALIGN` aligns the item on a multiple of the value of the `basealign-option`.

[ DIMENSION [ lbound: ] hbound ]

Specifies that the item is an array. If a single value is specified, that value indicates the number of elements in the array. Otherwise, `lbound` and `hbound` represent lower and

upper bounds of the array, respectively (see Section 3.2.2.8 [DIMENSION Keyword], page 56).

[ PREFIX *prefix-string* ]

Specifies a user-defined prefix that becomes part of the identifier. It can be any valid OpenSDL name with 0 to 32 characters, may or may not be enclosed in quotation marks ( " ") and may be null (see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49).

[ TAG *tag-string* ]

Specifies the user-defined tag that follows the prefix used in forming the identifier. It can be any valid OpenSDL name with 0 to 32 alphanumeric characters; the tag must be enclosed in quotation marks ( " ") if it begins with a numeric character (see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49).

## **AGGREGATE Declaration**

The following sections describe the function and format of **AGGREGATE** and subaggregate declarations.



## 4.5 AGGREGATE Description

An **AGGREGATE** declaration defines nonscalar items and dimensional arrays of nonscalar items. An aggregate must contain at least one member declaration. The body of an aggregate can also contain local symbol assignments, **CONSTANT** declarations, and subaggregate declarations.

You must specify a valid aggregate name on the **AGGREGATE** declaration. For example:

```
AGGREGATE dcb STRUCTURE;
    type CHARACTER;
    size WORD;
    next ADDRESS;
END dcb;
```

You must terminate an **AGGREGATE** declaration with the **END** statement. You can optionally specify the aggregate name on the **END** statement. The aggregate name is case-sensitive and must match exactly (in case) the name specified on the **AGGREGATE** declaration.

An aggregate that is the target of an **ADDRESS** declaration must have the default (**BASED**) storage class.

### 4.5.1 Subaggregate Declaration

Subaggregates are declared using the keyword **STRUCTURE** or **UNION**. For example:

```
AGGREGATE tree_node STRUCTURE;
    opcode WORD;
    lang_bits UNION;
        pli_bits STRUCTURE;
            resolved BITFIELD;
            psv BITFIELD;
            mark1 BITFIELD;
            spare_bits BITFIELD LENGTH 5;
        END pli_bits;
        c_bits STRUCTURE;
            value_variable_size BITFIELD;
            psv BITFIELD;
            expanded BITFIELD;
            resolved BITFIELD;
            reduced BITFIELD;
            spare_bits BITFIELD LENGTH 3;
        END c_bits;
    END lang_bits;
END tree_node;
```

Example 4.1: Subaggregate Example

In this example, the structures **pli\_bits** and **c\_bits** are both subaggregates of the **union lang\_bits**. Because **lang\_bits** is a **union**, **c\_bits** and **pli\_bits** occupy the same storage.

The **COMMON**, **GLOBAL**, **BASED** *pointer-name*, and **ORIGIN** options are invalid for a subaggregate. All other **AGGREGATE** options are valid.

### 4.5.2 STRUCTURE Declaration

STRUCTURE declarations produce aggregate or subaggregate declarations that are structures. The members are not overlaid; each member has a unique offset from the beginning of the structure, which means that members occupy consecutive storage locations.

The following shows the syntax of a first-level STRUCTURE declaration:

```
AGGREGATE aggregate-name STRUCTURE [ options ];
```

The following shows the syntax of a STRUCTURE declaration as an aggregate member, that is, a subaggregate STRUCTURE declaration:

```
member-name STRUCTURE [ options ];
```

### 4.5.3 UNION Declaration

UNION declarations produce aggregate or subaggregate declarations that are unions. The first-level members are overlaid, which means that they occupy the same storage location. Each first-level member begins at the beginning of the union and, thus, has an offset of zero. A UNION declaration lets you represent the same storage location using different names and different data types.

The following shows the syntax of a first-level UNION declaration:

```
AGGREGATE aggregate-name UNION [ options ];
```

The following shows the syntax of a UNION declaration as an aggregate member, that is, a subaggregate UNION declaration:

```
member-name UNION [ options ]
```

### 4.5.4 Implicit Union Declarations

You may specify data types on an AGGREGATE declaration to cause the AGGREGATE declaration to become an implicit union declaration. An implicit union declaration has these features:

- Gives OpenSDL the ability to detect subfield overflow without the need to define extraneous STRUCTURE and UNION declaration names.
- Makes user-defined fill fields unnecessary. This feature is most useful for data structures containing substructures that are required to begin at fixed offsets.

The following OpenSDL source code shows the syntax of an implicit union (structure B defines the implicit union declaration):

```
AGGREGATE A STRUCTURE;
  B STRUCTURE LONGWORD;
    bit_string1 BITFIELD LENGTH 1;
    bit_string2 BITFIELD LENGTH 4;
  END B;
  last_item WORD;
END A;
```

Example 4.2: Implicit Union Example

This implicit union declaration would be more cumbersome if represented as shown in the following STRUCTURE subaggregate:

```
AGGREGATE A STRUCTURE;
```

```

X UNION ;
  B LONGWORD;
  Y STRUCTURE ;
    bit_string1 BITFIELD LENGTH 1;
    bit_string2 BITFIELD LENGTH 4;
  END Y;
END X;
  last_item WORD;
END A;

```

In the previous example, the names `X` and `Y` become OpenSDL generated filler names in the implicit union case.

You do not have to define the union `X` and the structure `Y` if an implicit union declaration is used. By giving the structure `Y` a type, OpenSDL creates a union of field `Y` with the specified type overlaid with a structure containing the fields in `Y`. In some language translations, such as BLISS and MACRO, the OpenSDL-generated union and structure (`X` and `Y` above) can be suppressed in the output because they are considered extraneous fields. In C, it is necessary to use the OpenSDL-generated union and structure to generate the correct offsets within a structure. Because the union is of the length specified in the structure type, no filler is necessary to ensure that subsequent fields (for example, `last_item`) are at the correct offset. If the fields of a structure extend past the size specified, OpenSDL flags the overflow. The C/C++ translation for the previous implicit union declaration example is as follows:

```

struct A
{
    union B
    {
        uint32_t bit_string1 : 1;
        uint32_t bit_string : 4;
    };
    int16_t last_time;
};

```

The OpenSDL source file `/usr/lib/opensdl/examples/example.sdl` shows an example of an implicit union declaration.

### 4.5.5 Implicit Union Declarations with the Optional DIMENSION

Keyword

The following is an example of an OpenSDL structure defined with a data type and the `DIMENSION` keyword:

```

AGGREGATE fid STRUCTURE WORD DIMENSION 3;
  first WORD;
  second WORD;
  third WORD;
END fid;

```

Example 4.3: `DIMENSION` Example

In the previous example, a single structure is overlaid by an array of elements of the type specified by the structure type.

The `fid` structure is a three-word field that can be addressed as a single field or by each individual word so that it is easily defined in OpenSDL as represented above. The following is the C/C++ translation for the OpenSDL source code in the previous example:

```
union fid[3]
{
    int16_t first;
    int16_t second;
    int16_t third;
};
```

#### Example 4.4: DIMENSION Output

The implicit union declaration allows structures to grow without the need to modify any trailing fillers. OpenSDL detects any overflow that may occur if the structure grows past the size of its data type. The size of the aggregate (in bytes) is equal to the size of the data type (in bytes) multiplied by the upper dimension (if any). If the size of the aggregate is greater than the sum of the size of all its members, OpenSDL still translates the declaration. However, if the size of the members exceeds the size of the aggregate, OpenSDL issues a message that has the following format:

```
%SDL-E-TOOMANYFIELDS, Structure fid has too many fields [Line ?]
```

### 4.5.6 Forcing Negative Offsets

The default origin of an aggregate is the beginning of the first aggregate member. You may specify the `ORIGIN` option on a level-1 `AGGREGATE` declaration to indicate that the origin is located at the beginning of any aggregate member. The resulting declaration forces all aggregate members declared before the specified origin to be located at negative offsets from the origin. For example:

```
AGGREGATE nodes STRUCTURE ORIGIN qflink;
    flink ADDRESS;
    blink ADDRESS;
    qflink ADDRESS;
    qblink ADDRESS;
END;
```

#### Example 4.5: ORIGIN Example

This declaration defines the origin of the structure `nodes` to be at the member `qflink`, so you may address `flink` and `blink` as negative offsets from `qflink`. Specifying an origin does not change the values of the current bit and byte offset symbols (`^` and `:`). These are always calculated as being relative to the beginning of the aggregate.

### 4.5.7 Forcing Data Alignment

OpenSDL forces all `AGGREGATE` and subaggregate declarations to begin on byte boundaries. Thus, if an aggregate or subaggregate ends with `BITFIELD` declarations that do not end on byte boundaries, OpenSDL ensures that the next aggregate begins on a byte boundary by supplying `BITFIELD` declarations as fillers, if necessary.

When OpenSDL adds BITFIELD declarations, it determines the length required and provides a unique name of the form `string_V_FILL_n`. The string is the prefix supplied in the AGGREGATE declaration, or the aggregate name if no prefix was supplied. Within an aggregate, `n` begins at 0 and is incremented for each filler required.

The subaggregate declarations shown in the following example declare filler bitfields to force byte alignment at the end of each subaggregate; this programming practice makes it unnecessary for OpenSDL to perform the alignment.

```

AGGREGATE tree_node STRUCTURE;
  opcode WORD;
  lang_bits UNION;
    pli_bits STRUCTURE;
      resolved BITFIELD;
      psv BITFIELD;
      mark1 BITFIELD;
      spare_bits BITFIELD LENGTH 5;
    END pli_bits;
    c_bits STRUCTURE;
      value_variable_size BITFIELD;
      psv BITFIELD;
      expanded BITFIELD;
      resolved BITFIELD;
      reduced BITFIELD;
      spare_bits BITFIELD LENGTH 3;
    END c_bits;
  END lang_bits;
END tree_node;

```

#### Example 4.6: Alignment Example

The current bit offset is set at 0 at the beginning of each aggregate and is incremented by the bit lengths of each structure member in the aggregate at that level. In the following example, OpenSDL forces the structure flags to be aligned on a byte boundary.

```

AGGREGATE dcb STRUCTURE PREFIX dcb_;
  .
  .
  .
  uflags STRUCTURE;
    context BITFIELD LENGTH 3;
    local BITFIELD;
  END uflags;
  flags STRUCTURE;
    extern BITFIELD;
    relo BITFIELD;
  END flags;

```

The C/C++ translation of the OpenSDL AGGREGATE declaration in the previous example is as follows:

```

struct dcb_r_dcb

```

```

{
    struct dcb_r_uflags
    {
        uint8_t dcb_v_context : 3;
        uint8_t dcb_v_local : 1;
        uint8_t dcb_v_fill_0 : 4;
    };
    struct dcb_r_flags
    {
        uint8_t dcb_v_extern : 1;
        uint8_t dcb_v_relo : 1;
        uint8_t dcb_v_fill_1 : 6;
    }
};

```

The bit offsets of the members of the structures in the previous OpenSDL example are shown in the following table.

Member	Bit Offset
dcb_v_context	0
dcb_v_local	3
dcb_v_fill_0	4
dcb_v_extern	0
dcb_v_relo	1
dcb_v_fill_1	2

#### 4.5.8 Using Offset Symbols

The period (.) represents the current byte offset from the origin in an **AGGREGATE** declaration. If the **ORIGIN** option is specified, the value of the period is equal to the byte offset from the member specified using the **ORIGIN** option. The current byte offset is useful for capturing the size of an aggregate or a portion of it. For example, in the declaration of a variable-length data structure, you can capture the size of the fixed-length portion.

The colon (:) represents the current byte offset relative to the first member in an **AGGREGATE** declaration. The value is not affected by the presence of an **ORIGIN** option.

The circumflex (^) represents the current bit offset relative to the most recently declared aggregate or byte-aligned element.

The following example shows the use of the byte offset symbols.



```

[ TAG tag-string ]

[ ORIGIN member-name ]

[ FILL ];

aggregate-body
.
.
.
END [ aggregate-name ];

```

**AGGREGATE aggregate-name**

Specifies any valid OpenSDL name used to identify the aggregate.

{ STRUCTURE } { UNION }

Is the type of aggregate (see Section 4.5 [AGGREGATE Description], page 75).

[ data-type ]

If specified, causes the **AGGREGATE** declaration to become an implicit union declaration (see Section 4.5 [AGGREGATE Description], page 75).

[ COMMON ]

[ GLOBAL ]

[ BASED pointer-name ]

[ TYPEDEF ]

Is the storage class of the aggregate, if other than the default (**BASED**) (see Section 3.2.2.5 [Storage Class Keywords], page 54). If an aggregate is the object of an **ADDRESS** declaration, it must have either the default or the **BASED pointer-name** storage class.

[ ALIGN ]

[ NOALIGN ]

[ BASEALIGN basealign-option ]

The **ALIGN** and **NOALIGN** keywords can be used to align (or de-align) the members of an aggregate on their natural boundary. The **BASEALIGN** keyword ensures that the size of an aggregate is a multiple of the given alignment. The **BASEALIGN** keyword therefore takes an argument, which specifies the alignment, either an expression in parenthesis or the name of a data type.

For example:

```

AGGREGATE MyStruct STRUCTURE ALIGN;

```

This aligns every member in the structure.

```

AGGREGATE MyStruct STRUCTURE NOALIGN;

```

No action will be taken to ensure that the members of the aggregate will be aligned.

```

AGGREGATE MyStruct STRUCTURE BASEALIGN (8);

```

The aggregate will be padded, so that in an array of elements of this aggregate, all elements will have a size that is a multiple of the given alignment (256, 28). Alignment



attributes on aggregates can be partially overridden by specifying alignment attributes on the members of the aggregate. See also Section 3.2.2.4 [Alignment Keywords], page 53.

[ DIMENSION [ lbound: ] hbound ]

Specifies that the aggregate is an array. If a single value is specified, that value indicates the number of elements in the array. Otherwise, lbound and hbound represent lower and upper bounds of the array, respectively.

[ MARKER marker-string ]

Specifies the prefix used to form the aggregate name. It may be any valid OpenSDL name with 0 to 32 characters, may or may not be enclosed in quotation marks ( " ") and may be null.

[ PREFIX prefix-string ]

Specifies the prefix used in forming the names of aggregate members. It may be any valid OpenSDL name with 0 to 32 characters, may or may not be enclosed in quotation marks ( " ") and may be null (see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49).

[ TAG tag-string ]

Specifies the tag used to form the aggregate name. The tag is appended to the prefix, if a prefix was specified. It can have 0 to 32 alphabetic or numeric characters; the tag must be enclosed in quotation marks ( " ") if it begins with a numeric character (see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49).

[ ORIGIN member-name ]

Specifies the name of a member of this aggregate that is to be used as the origin of the aggregate.

Member declarations produce declarations of the members of an aggregate and have the following syntax:

```
member-name { data-type }      [ ALIGN ]
                { aggregate-name }  [ NOALIGN ]
                { user-type sizeopt } [ BASEALIGN basealign-option ]

                                [ DIMENSION [ lbound: ] hbound ]

                                [ PREFIX prefix-string ]

                                [ TAG tag-string ]

                                [ FILL ];
```

**member-name**

Is any valid OpenSDL name used to identify the member.

**{ data-type }**

Is any valid OpenSDL data type (see Section 3.2.3 [Data Type Keywords], page 57).

**{ aggregate-name }**

Is the name of the previously declared aggregate to be used as a type name. The name must be the full (OpenSDL-expanded) aggregate name, including the prefix and tag.

**{ user-type sizeopt }**

Is a user-defined variable using the DECLARE statements SIZEOF clause, shown and described in Section 3.9 [DECLARE Statement], page 67.

**[ ALIGN ]**

**[ NOALIGN ]**

**[ BASEALIGN basealign-option ]**

Alignment attributes on a structure can be overridden with alignment attributes on a member declaration. For example:

```
StructMember1 LONGWORD SIGNED ALIGN;
```

This ensures that the member is aligned, even if it is within an aggregate that is not aligned.

```
StructMember1 LONGWORD SIGNED NOALIGN;
```

No action will be taken to ensure that this member will be aligned.

```
StructMember1 LONGWORD SIGNED BASEALIGN (4)
```

Here, **StructMember1** will have an offset that is a multiple of 16 (24).

**basealign-option** can either be an expression in parentheses or the name of a data type. See also Section 3.2.2.4 [Alignment Keywords], page 53.

**[ DIMENSION [ lbound: ] hbound ]**

Specifies that the member is an array. If a single value is specified, that value indicates the number of elements in the array. Otherwise, **lbound** and **hbound** represent lower and upper bounds of the array, respectively (see Section 3.2.2.8 [DIMENSION Keyword], page 56).

**[ PREFIX prefix-string ]**

Specifies the prefix used to form the member name. For subaggregates, the prefix is used to form the names of subaggregate members. It may be any valid OpenSDL name with 0 to 32 characters, may or may not be enclosed in quotation marks (" ") and may be null (see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49).

**[ TAG tag-string ]**

Specifies the tag used to form the member name. The tag is appended to the prefix, if a prefix was specified. It can have 0 to 32 alphabetic or numeric characters; the tag must be enclosed in quotation marks (" ") if it begins with a numeric character (see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49).

**[ FILL ]**

Indicates that the specified member or aggregate occurs only as a fill to force byte alignment on the following member or aggregate, respectively. In some languages, filler member and aggregate declarations do not appear in the output file.

**aggregate-body**

Is one or more of the following:

- Member declaration

- Subaggregate declaration
- CONSTANT declaration
- Local symbol assignment

END [ **aggregate-name** ]

Marks the end of the **AGGREGATE** or subaggregate declaration. The **aggregate-name**, if specified, must match the name on the most recently specified **AGGREGATE** or subaggregate declaration.

## CONSTANT Declaration

The following sections describe the function and format of a **CONSTANT** declaration.

### 4.7 CONSTANT Description

A **CONSTANT** declaration generates a list of one or more named constants in the output file. You may specify a valid constant name or names and the constant values to be assigned to them. For example:

```
CONSTANT block_node_size EQUALS 24;
```

```
CONSTANT Strcon EQUALS STRING "This is a string constant" PREFIX Jg_;
```

The first declaration creates the named constant `block_node_size` and assigns it the value 24. The second declaration creates the named string constant `Jg_K_Strcon` and assigns it the specified value.

The values of both declared constants (except string constants) and local symbols may be used in OpenSDL expressions. However, there is an important difference between declared constants and local symbols: declared constants are translated directly to the output file, whereas local symbols are not passed directly to the output file. For example, you can define the local symbol `#block_size` as follows:

```
#block_size = 24;
```

A subsequent **CONSTANT** declaration may refer to `#block_size` and use the value 24, as follows:

```
CONSTANT block_node_size EQUALS #block_size;
```

**CONSTANT** declarations (except string constants) can also be specified in a comma-delimited list, as follows:

```
CONSTANT
    xyz EQUALS 10,
    alpha EQUALS 0,
    noname EQUALS 63;
```

To specify related constants with the same or incremental values, use a **CONSTANT** declaration with the **INCREMENT** option. In this form, the **EQUALS** expression gives the value to be assigned to the first named constant; values for subsequent constants are derived by incrementing the first value by the specified increment and assigning the result to the next name in the list. For example:

```
CONSTANT (
    bits,
    bytes,
    words,
    longs,
    quads,
    octas
) EQUALS 0 INCREMENT 1 PREFIX ctx_;
```

When OpenSDL assigns incremental values, it loops until values are assigned to all the names in a list. If there is no **INCREMENT** clause, the increment value is 0; thus, the same

initial value is assigned to all the names. If names are omitted from a comma-delimited list, OpenSDL reserves the numbers that would be assigned to names in those positions. This lets you reserve numeric values for later assignment of names. For example:

```
CONSTANT
    (bad_block,bad_data,,,,
     overlay,rewrite) EQUALS 0 INCREMENT 4;
```

In the previous example, OpenSDL assigns the values 0 and 4 to the names `bad_block` and `bad_data`, reserves the values 8, 12, and 16, and assigns the values 20 and 24 to the names `overlay` and `rewrite`, respectively.

The `COUNTER` option saves the last assigned value in a specified local symbol for subsequent use. For example:

```
CONSTANT (pli,c,bliss,macro)
    EQUALS 4 INCREMENT 4 PREFIX lang_
    COUNTER #lang;

CONSTANT (basic,pascal,fortran)
    EQUALS #lang + 4 INCREMENT 4 PREFIX lang_;
```

The following table shows the constant names produced by these two declarations.

Constant Name	Value
<code>lang_k_pli</code>	4
<code>lang_k_c</code>	8
<code>lang_k_bliss</code>	12
<code>lang_k_macro</code>	16
<code>lang_k_basic</code>	20
<code>lang_k_pascal</code>	24
<code>lang_k_fortran</code>	28

You can comment individual declarations in a `CONSTANT` declaration list. For example:

```
CONSTANT(
    pli, /* PL/I
    c, /* C
    macro /* MACRO-32
    ) EQUALS 4 INCREMENT 4 PREFIX lang_;
```

## 4.8 CONSTANT Format

A CONSTANT declaration has the following syntax:

```

CONSTANT constant-name constant-class
    constant-class = { EQUALS expression numeric-options }
                    { EQUALS STRING string string-options }

    numeric-options = [ PREFIX prefix-string ]

                      [ TAG tag-string ]

                      [ COUNTER #local-name ]

                      [ TYPENAME type-name ]

                      [ RADIX radix-name ]

                      [ ENUMERATE enum-name ];

    string-options = [ PREFIX prefix-string ]

                     [ TAG tag-string ] ;

CONSTANT (constant-name,...) EQUALS expression

                      [ INCREMENT expression ]

                      [ PREFIX prefix-string ]

                      [ TAG tag-string ]

                      [ COUNTER #local-name ]

                      [ TYPENAME type-name ]

                      [ ENUMERATE enum-name ];

CONSTANT (constant-name,...) EQUALS expression,
.
.
.
;

```

CONSTANT constant-name

Specifies any valid OpenSDL name used to identify the constant. When more than one name is specified, separate the names with commas and enclose the list in parentheses.

**EQUALS expression****EQUALS STRING string**

Specifies the value to be assigned to the constant.

**[ PREFIX prefix-string ]**

Specifies the prefix used to form the constant name. It may be any valid OpenSDL name with 0 to 32 characters, may or may not be enclosed in quotation marks (" ") and may be null (see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49).

**[ TAG tag-string ]**

Specifies the tag used to form the constant name. The tag is appended to the prefix, if a prefix was specified. It can have 0 to 32 alphabetic or numeric characters; the tag must be enclosed in quotation marks (" ") if it begins with a numeric character (see Section 3.2.2.2 [PREFIX MARKER and TAG Keywords], page 49).

**[ COUNTER #local-name ]**

Specifies the local symbol assigned to the last value that is assigned to a constant in the list.

**[ TYPENAME type-name ]**

Specifies a named data type that is not an OpenSDL data type.

**[ RADIX radix-name ]**

Specifies how the number should be written out to the output file. It can take one of three possible values:

<b>radix-name</b>	<b>Description</b>
DEC	Write the value out in decimal format (this is the default) <b>#define constant-name 123</b>
OCT	Write the value out in octal format <b>#define constant-name %0175</b>
HEX	Write the value out in hexadecimal format <b>#define constant-name %X7D</b>

**(constant-name,...)**

Is a list of valid names. A **constant-name** in this list can be null, though the list itself cannot be null. If a member of the list is null, the corresponding value is reserved. If **INCREMENT** is not specified, all names are assigned the value specified in the **EQUALS** expression.

**[ INCREMENT expression ]**

Specifies the value to be added to the **EQUALS** expression for each iteration of OpenSDLs generation of named **CONSTANT** declarations. It must be a valid OpenSDL integer expression. OpenSDL assigns the value of the **EQUALS** expression to the first constant name; the value of the **INCREMENT** expression is added to the **EQUALS** expression and assigned to the next constant name.

**[ ENUMERATE enum-name ]**

Rather than a constant being created, an enumeration is created instead, if the output language supports enumerations.

## **ENTRY Declaration**

The following sections describe the function and format of the **ENTRY** declaration.



## 4.9 ENTRY Description

An **ENTRY** declaration produces an external procedure or function declaration in the output file. You must specify a valid entry name on the **ENTRY** declaration. You may also specify any or all of the **ENTRY** declaration options described in Section 3.2 [Keywords Used in Declarations], page 43, in Section 3.2.2 [Declaration Modifier Keywords], page 44, and shown within the context of an **ENTRY** declaration in Section 4.10 [ENTRY Format], page 91.

The following example shows the **ENTRY** declaration for the OpenVMS system service **SYS\$GETJPI**:

```
ENTRY SYS$GETJPI ALIAS $GETJPI PARAMETER (
    LONGWORD UNSIGNED VALUE NAMED EFN DEFAULT 0 TYPENAME EFNUM,
    LONGWORD UNSIGNED NAMED PIDADR IN OUT DEFAULT 0 TYPENAME PROCID,
    CHARACTER REFERENCE NAMED PRCNAM IN DEFAULT 0 TYPENAME PROCNAME,
    ANY NAMED ITMLST IN TYPENAME ITEMLIST,
    QUADWORD UNSIGNED NAMED IOSB OUT DEFAULT 0 TYPENAME IOSB,
    ADDRESS(ENTRY) NAMED ASTADR DEFAULT 0 TYPENAME ASTADR,
    LONGWORD UNSIGNED VALUE NAMED ASTPRM DEFAULT 0 TYPENAME USERPARM
) RETURNS LONGWORD TYPENAME CONDVALU;
```

## 4.10 ENTRY Format

An **ENTRY** declaration has the following syntax:

```
ENTRY entry-name [ ALIAS internal-name ]

[ PARAMETER (param-desc,...) ]

[ LINKAGE link-name ];

[ VARIABLE ]

[ RETURNS return-data-type [ NAMED param-name ] ]

[ TYPENAME type-name ];
```

**ENTRY entry-name**

Specifies any valid OpenSDL name used to name the external entry point.

**[ ALIAS internal-name ]**

Specifies an alternate internal name used to designate the entry point.

**[ PARAMETER (param-desc,...) ]**

Describes the parameters of the external entry, if any. Param-desc must be specified as follows:

{ data-type }	[ VALUE ]	[ IN ]
{ aggregate-name }	[ REFERENCE ]	[ OUT ]
		[ NAMED param-name ]
		[ DIMENSION expression ]
		[ DEFAULT constant-value ]

- [ TYPENAME *type-name* ]
- [ OPTIONAL ]
- [ LIST ]
- { *data-type* }
  - Is any valid OpenSDL data type.
- { *aggregate-name* }
  - Is the name of a previously declared aggregate that describes the data type of the parameter.
- [ VALUE ]
  - Indicates that the parameter is passed by immediate value.
- [ REFERENCE ]
  - Indicates that the parameter is passed by reference. REFERENCE is the default.
- [ IN ]
  - Indicates an input parameter. IN is the default.
- [ OUT ]
  - Indicates an output parameter.
- [ NAMED *param-name* ]
  - Specifies the parameter name.
- [ DIMENSION *expression* ]
  - Is the number of elements of an array parameter. If *expression* is an asterisk (\*), the number of dimensions depends on the dimensions of the actual parameter.
- [ DEFAULT *constant-value* ]
  - Specifies a default value for a parameter. In languages supporting this option, the omission of an actual parameter is allowed.
- [ TYPENAME *type-name* ]
  - Specifies a named data type that is not an OpenSDL data type.
- [ OPTIONAL ]
  - Specifies that the parameter may or may not appear in the sequence of (key-word) parameters for a call using the entry point name.
- [ LIST ]
  - Indicates that the routine may be called with one or more parameters of the type being described. LIST may be specified only for the last parameter.
  - NOTE:** All the PARAMETER options, if specified, must follow the data type or aggregate name in the declaration.
- [ LINKAGE *link-name* ]
  - Specifies (for MACRO only) that a special call macro (spelled *link-name*) will be used in the expansion of the entry macro.
  - NOTE:** Currently, a MACRO back-end does not exist for this implementation. This keyword is defined so that this software can successfully parse files originally created for the OpenVMS environment.

**[ VARIABLE ]**

Indicates that not all parameters are described; that is, the entry point has a variable number of parameters and not all corresponding arguments need be present in the argument list when the entry point is invoked. See also the description of the LIST parameter option.

**[ RETURNS return-data-type [ NAMED param-name ] ]**

Specifies the data type and, optionally, the name of the parameter returned by the external entry, if it is a function.

The VOID keyword cannot be used in a PARAMETER clause.

The syntax is:

```
return-data-type = { data-type }  
                  { user-type sizeopt }  
                  { VOID }
```

The argument `user-type sizeopt` specifies a user-defined type declared using the DECLARE statements SIZEOF clause, shown and described in Section 3.9 [DECLARE Statement], page 67.

The return type VOID indicates that the procedure returns no value.

**[ TYPENAME type-name ]**

Specifies a named data type for the return value that is not an OpenSDL data type.



## Appendix A OpenSDL Diagnostic Messages

This appendix summarizes the OpenSDL diagnostic messages. All messages indicating errors in OpenSDL syntax specify the line number in the OpenSDL source file at which the error occurred.

### **ABORT, Fatal internal error. unable to continue execution**

**Fatal:** An internal error has occurred.

**User Action:** Please report the problem to the GitHub Issues for this utility.

### **ADROJBAS, Address object <object-name> must have based storage class [Line n]**

**Error:** An address item is pointing to an aggregate that is not based.

**User Action:** Change the storage class of the aggregate to `BASED`.

### **BADNODETYPE, internal node type is unknown for language <language-name>**

**Warning:** A language backend has encountered a node type in the parsed OpenSDL input that reflects an OpenSDL element which the language does not support.

**User Action:** Depending on your needs, ignore the warning or change the OpenSDL input file.

### **BASEALIGN, Invalid expression with BASEALIGN option. Value must be in range 0 to 124. <basealign-parameter> [Line n]**

**Error:** The value for the `--align` qualifier is smaller than 0 or larger than 124.

**User Action:** Use a value in the range [0 ... 124].

### **BUGCHECK, Internal consistency failure [Line n] - please submit a bug report**

**Fatal:** OpenSDL has detected an internal error or inconsistency.

**User Action:** Please submit a bug report to GitHub Repository.

### **BYTSIZ, Aggregate <aggr-name> must be integral byte size [Line n]**

**Error:** An aggregate is required to be a specific number of bytes long.

**User Action:** Add one or more members to the aggregate with the `FILL` option to correct the size, in bytes, of the indicated `AGGREGATE`, and invoke OpenSDL again.

### **DIMENSIONSTAR, DIMENSION \* for MEMBER <"member-name"> has no known discriminant [Line n]**

**Warning:** The use of `"DIMENSION *"` within an aggregate is not allowed.

**User Action:** Use fixed values for `"DIMENSION"` within all aggregate.

### **DUPCONATT, Item <item-name> has duplicate or conflicting attributes [Line n]**

**Error:** A declaration contains keywords that are not compatible.

**User Action:** Verify the syntax of the OpenSDL declaration, correct the declaration, and invoke OpenSDL again.

### **DUPLANG, Language name <language> appears more than once on the command line**

**Error:** The indicated language was detected on the command-line more than once.

**User Action:** Remove the duplicate languages from the command line, and invoke OpenSDL again.

### **DUPLISTQUAL, '-list' qualifier specified more than once on the command line**

**Error:** Multiple `--list` qualifiers were detected on the command-line.

**User Action:** Remove the duplicate qualifiers from the command line, and invoke OpenSDL again.

**ERREXIT, Error exit**

**Fatal:** Previous errors prevent continuation.

**User Action:** Correct the errors and invoke OpenSDL again.

**FILLNEGLN, Fill item <item-name> has a negative length [Line n]**

**Warning:** The FILL is generating a negative fill length. The generated output may not be as expected or desired.

**User Action:** Correct the waraning and invoke OpenSDL again.

**FILLZEROLEN, Fill item <item-name> has a zero length [Line n]**

**Informational:** The FILL is generating a zero fill length. The generated output may not be as expected or desired.

**User Action:** Correct the waraning and invoke OpenSDL again.

**FIXUP, Temporary hardcoded list used to discriminate for MEMBER <"member-name"> [Line n]**

**Informational:** From the comment in the Ada backend: This is informational, with the first line otherwise being identical to a warning, to aid diagnosing problems where someone makes a copy of a construct on our hardcoded list without changing OpenSDL.

This message is issued together with the DIMENSIONSTAR diagnostic.

**User Action:** See DIMENSIONSTAR.

**IDENTGTR31, SDL-generated identifier longer than 31 characters exceeds capacity of <language-name> compiler [Line n]**

**Warning:** The Pascal backend appends "\$TYPE" to data types that contain "DEF". The resulting name then can exceed 31 characters, the maximum length of Pascal type names.

**User Action:** Dont use "DEF" in your type names and/or shorten your type names.

**ILLFORWREF, Illegal forward reference for output language <language> [Line n]**

**Error:** The specified output language does not allow forward referencing, or the language does not allow forward referencing in this context.

**User Action:** Correct or remove the forward reference.

**IMMGTR32, Cannot pass values larger than 32 bits by immediate mechanism [Line n]**

**Warning:** Using VALUE is invalid for this parameter because its size is greater than that of a longword. A reasonable translation was attempted, however.

**User Action:** Verify that the output file contains a satisfactory translation of the parameter description in your OpenSDL source file.

**INCDEFSTRUC, Incompletely defined structure – <structure-name> [Line n]**

**Error:** A structure name has been referenced before the structure has been completely defined.

**User Action:** Remove the reference and invoke OpenSDL again.

**INFILOPN, Unable to open input file <file-spec>**

**Fatal:** OpenSDL cannot locate or open the OpenSDL source file.

**User Action:** Verify that you correctly specified the name of the source file.

**INFILSDI, File format error reading intermediate file file-spec. Possible version mismatch**

**Error:** An intermediate file (.sdi) could not be read.

**User Action:** READ is not supported by this implementation. Therefore, this error is not emitted by OpenSDL.

**INTOVF, Integer overflow in expression [Line n]**

**Error:** Evaluation of an OpenSDL expression resulted in a value that does not fit in a quadword.

**User Action:** Correct the expression.

**INVACTSTA, Invalid action for internal state [Line n]**

**Error:** The parsing of the input file detected an unsupported action in the current state of parsing.

**User Action:** If there is a coding error at the line indicated, then correct the error, and invoke OpenSDL again. If there is no coding error, please submit a support request.

**INVAGGRNAM, Invalid or no aggregate name specified**

**Error:** The id provided on the AGGREGATE is either invalid or missing.

**User Action:** Correct the error, and invoke OpenSDL again.

**INVALIGN, Illegal value for --align qualifier in command line**

**Error:** The value of the --align qualifier is not a positive number.

**User Action:** Use a positive number as value for the --align qualifier.

**INVBITFLD, Invalid bitfield <bitfield-name> – bitfields> must be aggregate members [Line n]**

**Error:** Bit fields must be members of aggregates. They cannot be scalar items.

**User Action:** Incorporate the BITFIELD declaration in an aggregate.

**INVCONDST, Invalid condition state [Line %d]**

**Error:** An invalid condition state was detected.

**User Action:** Correct the condition, and invoke OpenSDL again.

**INVDECL, Invalid DECLARE for type <user-defined-name> [Line n]**

**Error:** A DECLARE statement refers to a user defined data type that is invalid.

**User Action:** Correct the DECLARE statement, and invoke OpenSDL again.

**INVENUMNAM, Invalid or no enumeration name specified**

**Error:** The id provided on the CONSTANT...ENUMERATE is either invalid or missing.

**User Action:** Correct the error, and invoke OpenSDL again.

**INVEXPR, Invalid expression – cannot be resolved to a constant as required, <name-of-defined-item> [Line n]**

**Error:** A non-constant expression has been used in a context which requires a constant expression.

**User Action:** Use a constant expression.

**INVFLDSIZ, Item <item-name> has bit field or offset length greater than 32 or 64 [Line n]**

**Error:** OpenSDL cannot generate bit fields larger than 32 or 64 bits or cannot generate the proper bit mask.

**User Action:** Verify the BITFIELD declaration and correct it. If the BITFIELD declaration occurs within an aggregate and you specify the MASK option, verify that the bit offset of the start of the declaration plus the bit field size does not exceed 32 bits (when using -32) or 64 bits (when using -64).

**INVLISTOPT, Invalid use of LIST attribute – LIST may only appear on the last parameter.**  
entry-name [Line n]

**Error:** The LIST attribute appears on a parameter other than the last.

**User Action:** Remove the LIST attribute.

**INVNAME, Item name is invalid**

**Error:** The item name contains illegal characters or is specified in an illegal context.

**User Action:** Correct or relocate the item name.

**INVOUT, Invalid attributes for output language <language> [Line n]**

**Error:** An OpenSDL construct or data type is invalid for the specified target language.

**User Action:** Determine whether you specified the data type or OpenSDL declaration correctly, or whether you may be requesting language output that you do not require. Either correct the declaration or reissue the OpenSDL command so that the indicated language output routine does not execute.

**INVPARMTYP, Invalid parameter type for language <language> [Line n]**

**Error:** A parameter specification is illegal for the specified language.

**User Action:** Modify the parameter specification and invoke OpenSDL again.

**INVQUAL, Invalid qualifier, <qualifier>, specified on the command line**

**Error:** An invalid or unrecognized qualifier was specified on the command line.

**User Action:** Either correct or remove the qualifier, and invoke OpenSDL again.

**INVREQPARAM, Required parameter encountered after an optional parameter <parameter-name>**

**Error:** Required parameters must not follow optional parameters.

**User Action:** Correct the error and invoke OpenSDL again.

**INVSHRIMG, Shareable image not found specified-language**

**Error:** OpenSDL does not know about the specified language.

**User Action:** Verify that the language is supported.

**INVSYMDEF, Invalid symbol <symbol-name> specified in --symbol qualifier**

**Error:** The value of the --symbol qualifier is not correct. The --symbol qualifier expects a single symbol definition in the form symbol-name=value. The error message indicates a missing value.

**User Action:** Correct the syntax of the --symbol qualifier.

**INVUNKLEN, Unknown length attribute valid only for parameter type [Line n]**

**Error:** CHARACTER LENGTH \* is only allowed in a parameter description. Specifying an unknown length for an ITEM or AGGREGATE member is an error.

**User Action:** Remove the LENGTH specification or replace the "\*" with a valid expression.

**LANGDUP, Language name <language-name> appears more than once in list [Line n]**

**Warning:** In an IFLANGUAGE or END\_IFLANGUAGE statement, the name of a language appears twice.

**User Action:** Remove the duplicate language name.

**LANGMATCH, Language <language-name> does not appear in list of matching IF statement [Line n]**



**Warning:** The list of language specified after the `END_IFLANGUAGE` keyword does not match the list of languages specified after the corresponding `IFLANGUAGE` keyword.

**User Action:** Correct the language list.

**LANGMISS, Language <language-name> in list of matching IF statement missing from END list [Line n]**

**Warning:** In an `END_IFLANGUAGE` statement, one of the languages from the `IFLANGUAGE` statement is missing.

**User Action:** Add the missing language to the `END_IFLANGUAGE` statement or remove all languages from the `END_IFLANGUAGE` statement.

**LISFILOPN, Unable to open listing file <file-spec>**

**Error:** OpenSDL cannot open the indicated listing file.

**User Action:** Verify that you have write access to the directory to which the OpenSDL listing file is directed.

**MATCHEND, End name does not match declaration name <name> [Line n]**

**Warning:** The name specified on the `END_MODULE` or `END` delimiter does not match the most recent module name or aggregate name.

**User Action:** Verify that the spelling of the names specified on the `END` and `END_MODULE` delimiters match. Check whether you have illegally nested `MODULE` declarations.

This is only a warning message, but it may indicate an error.

**MULTDEFSYM, Multiply defined symbol – <symbol-name> [Line n]**

**Error:** A structure contains a duplicate symbol name.

**User Action:** Remove the duplicate name and invoke OpenSDL again.

**NAMTRUNC, Generated name too long - truncated to 64 characters <name-to-be-truncated>**

**Warning:** OpenSDL appends various prefixes and suffixes to names. The resulting name then can exceed 64 characters, the maximum length.

**User Action:** Shorten your names.

**NEGORIGIN, Aggregate <aggregate-name> has a negative origin - negative offset elements will be ignored [Line n]**

**Informational:** The `ORIGIN` attribute defines a member as origin which is not at the beginning of the aggregate.

**User Action:** Specify the first member of the aggregate as origin.

**NOCOPYFIL, No copyright input file**

**Error:** The `--copy` command line qualifier was specified on the command line, but OpenSDL could not open the `copyright.sdl` file.

**User Action:** Verify that the `copyright.sdl` is located in the `/usr/lib/opensdl` directory and can be opened for reading. If the file is not there, or cannot be opened for reading, then either correct the error or remove the `--copy` qualifier, and invoke OpenSDL again.

**NOINPFIL, No input file specified**

**Error:** OpenSDL needs an input file for its processing and one was not specified on the command line.

**User Action:** Add the input file to the command line when invoking OpenSDL again.

**NOOUTPUT, No language output produced**

**Warning:** There were too many errors, or fatal errors, which prevented OpenSDL from generating any output files.

**User Action:** Correct the errors indicated by the accompanying messages.

**NULLSTRUC, Null structure <structure-name> has no members [Line n]**

**Error:** An AGGREGATE or subaggregate declaration did not have any members.

**User Action:** Verify that the AGGREGATE or subaggregate declaration is\ correctly positioned in the file.

**OFFSETEXPR, Offset or origin relative expression involves a forward or circular reference. <member-name> [Line n]**

**Warning:** The offset or origin cannot be calculated.

**User Action:** Correct the error and invoke OpenSDL again.

**OUTFILOPN, Unable to open output file <file-spec>**

**Error:** OpenSDL cannot locate or open an OpenSDL output file.

**User Action:** Verify that you correctly specified the name of the source file.

**PARSEERR, Parse error, <parse-string>, detected**

**Error:** An error occurred while parsing the input file.

**User Action:** Correct the error, and invoke OpenSDL again.

**POSSCIRC, Possible circular definition for type <type-name> [Line n]**

**Informational:** The definition for the type cannot be processed.

**User Action:** Correct the error and invoke OpenSDL again.

**REVCHECK, Front-end / back-end version mismatch. Check installation.**

**Fatal:** A language backend has a different version than the calling frontend.

**User Action:** This error is not emitted by this implementation. It is here for consistency with VSI SDL.

**SIZENEST, Illegal nesting of SIZEOF clauses (Item <item-name>) [Line n]**

**Error:** SIZEOF clauses cannot be nested.

**User Action:** Remove the nested SIZEOF clause.

**SIZEQUAL, Item <item-name>, an aggregate, cannot be qualified by SIZEOF [Line n]**

**Error:** The SIZEOF clause is not allowed in this context.

**User Action:** Remove the SIZEOF clause.

**SIZEREDF, Size or type of item <item-name> redefined [Line n]**

**Error:** OpenSDL has detected a redefinition of the size or data type of the specified item.

**User Action:** Remove the clause causing the redefinition.

**STRINGCONST, String constant <item-name> used in arithmetic expression [Line n]**

**Error:** A reference to a string constant is not allowed in the context of an arithmetic expression.

**User Action:** Remove the string constant reference.

**SYMALRDEF, Symbol <symbol-name> was already defined in command line**

**Error:** The value of the `--symbol` qualifier with the same symbol name is repeated more than once.

**User Action:** Remove the duplicate name.

**SYMNOTDEF, Symbol <symbol-name> was not defined in command line, value zero assumed [Line n]**

**Warning:** A symbol is used in an `IFSYMBOL` or `ELSE_IFSYMBOL` statement, that has not been defined using the `--symbol` qualifier.

**User Action:** Define the symbol using the `--symbol` qualifier.

**SYMTABOVR, Symbol table overflow**

**Fatal:** OpenSDL exceeded its symbol table space.

**User Action:** Reduce the size or complexity of the OpenSDL source file; if possible, separate the file into several different files or modules.

**SYNTAXERR, Syntax error [Line n]**

**Error:** The OpenSDL translator detected a syntax error. This message is accompanied by a message indicating the type of error and tells you what type of token or keyword OpenSDL expected but did not find.

**User Action:** Determine the syntax error from the accompanying message and correct it.

**TOKOVF, Token exceeds maximum size of <maximum-token-length> [Line n]**

**Error:** A line in the OpenSDL source file is longer than the maximum length.

**User Action:** Shorten the offending line.

**TOOMANYFIELDS, Structure <structure-name> has too many fields [Line n]**

**Error:** The structure has too many fields.

**User Action:** Simplify the structure.

**TYPNAM, Aggregate type name not supported [Line n]**

**Warning:** An illegal aggregate name has been used.

**User Action:** Choose another aggregate name.

**TYPNOTSUP, Output language does not support data type <data-type-name> [Line n]**

**Warning:** The specified data type is not supported by the output language. A reasonable translation was attempted, however.

**User Action:** Verify that the output file contains a satisfactory translation of the data type you specified in your OpenSDL source file.

**UNALIGNED, <member-name> does not align on its natural boundary [Line n]**

**Warning:** A member does not fall on its natural alignment (if `--check` is present on the command line) or on the alignment specified with the `--align` qualifier.

**User Action:** Check the layout of the aggregate in question.

**UNDEFCON, Undefined constant name <constant-name> used in expression [Line n]**

**Error:** In the definition of a `CONSTANT`, an undefined name has been used.

**User Action:** Verify the spelling of the name.

**UNDEFFIL, Unable to open include file <file-name> [Line n]**

**Error:** A file to be `INCLUDED` could not be opened.

**User Action:** Check the spelling of the file name, existence and protection of the file.

**UNDEFORG, Definition of ORIGIN name <member-name> not found in aggregate [Line n]**

**Error:** The member used as argument to the ORIGIN attribute was not found within the aggregate.

**User Action:** Verify the spelling of the member name.

**UNDEFSYM, Undefined local symbol <symbol-name> used in expression [Line n]**

**Error:** A name preceded by a pound sign (#) is not defined.

**User Action:** Verify that the local symbol name is spelled correctly and that it appears before its reference in the OpenSDL source file.

**UNDEFUSER, Undefined user type name <type-name> referenced [Line n]**

**Error:** A DECLARE statement refers to a data type that is neither a built-in nor a known user defined data type.

**User Action:** Check the spelling of the data type referenced.

**UNKNCOSTTYP, Unknown constant type <type-value> [Line n]**

**Error:** A CONSTANT can only be a value or a string (STRING).

**User Action:** Correct the error, and invoke OpenSDL again.

**UNKRADIX, Unknown radix <radix-value> [Line n]**

**Error:** A value for the RADIX keyword is not known.

**User Action:** The qualifier for the RADIX keyword can only be DEC (for decimal), OCT (for octal), or HEX (for hexadecimal). Supply the correct qualifier on the indicated line, and invoke OpenSDL again.

**UNKOPTION, Unknown option specified [Line %d]**

**Error:** An unrecognized option was specified in the OpenSDL file.

**User Action:** Correct the error, and invoke OpenSDL again.

**WARNEXIT, Warning exit**

**Warning:** A warning message has been issued.

**User Action:** Output can be compiled, but the results may be unexpected.

**ZERODIV, Zero divide in expression [Line n]**

**Error:** An expression specified in an OpenSDL declaration resulted in a divide-by-zero exception condition.

**User Action:** Verify the expression and correct it.

**ZEROLEN, Item <item-name> has 0 or negative length [Line n]**

**Warning:** A BITFIELD or CHARACTER declaration or a DIMENSION option specified a length of 0 or less.

**User Action:** Correct the declaration. If the length or bound value was specified using an OpenSDL expression, verify the local symbol values and the results of arithmetic operations in the expression, if any.

## Appendix B OpenSSL Language Translation Summaries

This appendix shows the translation summaries of OpenSSL language elements to their corresponding output in each of the following supported languages:

- C

### C/C++ Translation Summary

The following table shows the OpenSSL to C/C++ language translation summary.

OpenSSL Declaration	C/C++ Output
MODULE name IDENT string	/** MODULE name IDENT string **/ #include <ctype.h> #include <stdint.h> #include <stdbool.h> #include <complex.h>
/* comment	/* comment */
/+	/*
//	*
/-	*/
CONSTANT	
EQUALS n;	#define x n
EQUALS STRING "s";	#define x "s"
ENTRY name	return-type name( ) return-type name(parameters...)
PARAMETER (type,...)	type param-name
ANY	void
IN	n/a
OUT	n/a
NAMED param-name	n/a
VALUE	n/a
REFERENCE	*type
DEFAULT n	n/a
LIST	n/a
OPTIONAL	n/a
TYPENAME type-name	n/a
RETURNS return-type	return-type name( ) return-type name(parameters...)
NAMED return-name	n/a
VARIABLE	n/a
ALIAS internal-name	n/a
LINKAGE	n/a
TYPENAME type-name	n/a
STRUCTURE	struct
UNION	union
name BYTE [SIGNED]	int8_t name
name BYTE UNSIGNED	uint8_t name

name INTEGER_BYTE [SIGNED]	int8_t name
name INTEGER_BYTE UNSIGNED	uint8_t name
name WORD [SIGNED]	int16_t int name
name WORD UNSIGNED	uint16_t int name
name INTEGER_WORD [SIGNED]	int16_t int name
name INTEGER_WORD UNSIGNED	uint16_t int name
name LONGWORD [SIGNED]	int32_t name
name LONGWORD UNSIGNED	uint32_t name
name INTEGER_LONG [SIGNED]	int32_t name
name INTEGER_LONG UNSIGNED	uint32_t name
name INTEGER [SIGNED]	int name
name INTEGER UNSIGNED	unsigned int name
name INTEGER_HW [SIGNED]	int32_t name (32-bit)
	int64_t name (64-bit)
name INTEGER_HW UNSIGNED	uint32_t name (32-bit)
	uint64_t name (64-bit)
name HARDWARE_INTEGER [SIGNED]	int32_t name (32-bit)
	int64_t name (64-bit)
name HARDWARE_INTEGER UNSIGNED	uint32_t name (32-bit)
	uint64_t name (64-bit)
name QUADWORD [SIGNED]	int64_t name
name QUADWORD UNSIGNED	uint64_t name
name INTEGER_QUAD [SIGNED]	int64_t name
name INTEGER_QUAD UNSIGNED	uint64_t name
name OCTAWORD [SIGNED]	__int128_t name
name OCTAWORD UNSIGNED	__uint128_t name
name D_FLOATING	double float name
name F_FLOATING	float name
name G_FLOATING	double float name
name H_FLOATING	long double float name
name S_FLOATING	double float name
name T_FLOATING	float name
name X_FLOATING	long double float name
name D_FLOATING COMPLEX	double float complex name
name F_FLOATING COMPLEX	float complex name
name G_FLOATING COMPLEX	double float complex name
name H_FLOATING COMPLEX	long double float complex name
name S_FLOATING COMPLEX	double float complex name
name T_FLOATING COMPLEX	float complex name
name X_FLOATING COMPLEX	long double float complex name
name DECIMAL PRECISION (p,q)	char name [p/2+1]
name BITFIELD	uint8_t name:1 (bitfields of 1 to 8 bits)
	uint16_t name : 1 (bitfields of 9 to 16 bits)
	uint32_t name : 1 (bitfields of 17 to 32 bits)
	uint64_t name : 1 (bitfields of 33 to 64 bits)
	__uint128_t name :1 (bitfields of 65 to 128 bits)

LENGTH n	uint8_t name:n
MASK	#define <pref>_m_name <hex-value>
SIGNED	int8_t name : 1 (bitfields of 1 to 8 bits)
	int16_t name : 1 (bitfields of 9 to 16 bits)
	int32_t name : 1 (bitfields of 17 to 32 bits)
	int64_t name : 1 (bitfields of 33 to 64 bits)
	__int128_t name : 1 (bitfields of 65 to 128 bits)
name CHARACTER	char name
LENGTH n	char name[n]
LENGTH *	n/a
VARYING	struct {short string_length; char string_text[n];} name; where n is the maximum length of the character string
name ADDRESS (object-type)	object-type *name
name POINTER (object-type)	object-type *name
name POINTER_LONG (object-type)	uint32_t
name POINTER_HW (object-type)	uint32_t name (32-bit)
	uint64_t name (64-bit)
name HARDWARE_ADDRESS (object-type)	uint32_t name (32-bit)
	uint64_t name (64-bit)
name POINTER_QUAD (object-type)	uint64_t name
name BOOLEAN	bool
name user-type-name	user-type-name name
Default storage class	struct or union with <member-name> as the tag, and no declared variable names
COMMON storage class	extern attribute
GLOBAL storage class	n/a
BASED pointer-name	A pointer will be generated for the based item
TYPDEF	For an AGGREGATE, a typedef struct is generated. In this case, a pre-tag is generated as well as the post-tag. The pre-tag consists of the structure named prefixed by an underscore (_). Any reference to the structure type within the definition (such as for forward and backward linkages) is also output with the underscore. For an ITEM, the TYPDEF keyword is generated, followed by the data type of the ITEM.
DIMENSION [lbound:]hbound	The declaration specifies an array of the number of elements specified with subscripts ranging from 0 to (hbound - lbound + 1)
ORIGIN member-name	n/a





## Appendix C ASCII Character Set

The following table shows the ASCII character set referred to in Section 3.3 [Expressions], page 61.

Character	ASCII Decimal	Hexadecimal
NUL	000	00
SOH	001	01
STX	002	02
ETX	003	03
EOT	004	04
ENQ	005	05
ACK	006	06
BEL	007	07
BS	008	08
HT	009	09
LF	010	0A
VT	011	0B
FF	012	0C
CR	013	0D
SO	014	0E
SI	015	0F
DLE	016	10
DC1	017	11
DC2	018	12
DC3	019	13
DC4	020	14
NAK	021	15
SYN	022	16
ETB	023	17
CAN	024	18
EM	025	19
SUB	026	1A
ESC	027	1B
FS	028	1C
GS	029	1D
RS	030	1E
US	031	1F
SP	032	20
!	033	21
"	034	22
#	035	23
\$	036	24
%	037	25
&	038	26
	039	27
(	040	28

)	041	29
*	042	2A
+	043	2B
,	044	2C
-	045	2D
.	046	2E
/	047	2F
0	048	30
1	049	31
2	050	32
3	051	33
4	052	34
5	053	35
6	054	36
7	055	37
8	056	38
9	057	39
:	058	3A
;	059	3B
<	060	3C
=	061	3D
>	062	3E
?	063	3F
@	064	40
A	065	41
B	066	42
C	067	43
D	068	44
E	069	45
F	070	46
G	071	47
H	072	48
I	073	49
J	074	4A
K	075	4B
L	076	4C
M	077	4D
N	078	4E
O	079	4F
P	080	50
Q	081	51
R	082	52
S	083	53
T	084	54
U	085	55
V	086	56
W	087	57

X	088	58
Y	089	59
Z	090	5A
[	091	5B
\	092	5C
]	093	5D
^	094	5E
-	095	5F
	096	60
a	097	61
b	098	62
c	099	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
{	123	7B
	124	7C
}	125	7D
~	126	7E
DEL	127	7F

The following table describes the ASCII character abbreviations used in the previous table.

<b>Characters</b>	<b>Description</b>	<b>Characters</b>	<b>Description</b>
NUL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC1	Device Control 1
STX	Start of Text	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3
EOT	End of Transmission	DC4	Device Control 4
ENQ	Enquiry	NAK	Negative Acknowledge
ACK	Acknowledge	SYN	Synchronous Idle
BEL	Bell	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tabulation	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertical Tab	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
SP	Space	DEL	Delete

# Index

—

-align qualifier . . . . . 37, 95, 97, 101  
 -check qualifier . . . . . 37, 101  
 -comments qualifier . . . . . 13, 37  
 -copy qualifier . . . . . 38  
 -32 qualifier . . . . . 37, 57, 61, 97  
 -64 qualifier . . . . . 37, 57, 61, 97

## A

Addition operator . . . . . 62, 87  
 ADDRESS data type . . . . . 57  
 ADDRESS declaration . . . . . 57, 75, 82, 105  
 ADDRESS default tag . . . . . 51  
 ADDRESS example . . 12, 17, 50, 56, 57, 68, 75, 78, 81  
 ADDRESS keyword . . 13, 27, 34, 50, 51, 57, 68, 75, 78, 82, 105  
 AGGREGATE declaration . . 13, 16, 43, 45, 49, 50, 52, 54, 55, 56, 58, 71, 72, 74, 75, 81  
 AGGREGATE example . . 12, 17, 50, 52, 53, 54, 55, 56, 57, 58, 59, 68, 75, 76, 77, 78, 79, 81  
 AGGREGATE keyword . . 9, 11, 13, 16, 25, 26, 27, 28, 29, 30, 31, 32, 41, 42, 43, 44, 45, 49, 50, 52, 54, 55, 56, 57, 58, 59, 62, 63, 66, 67, 68, 71, 72, 74, 75, 76, 78, 79, 80, 81, 82, 95, 97, 98, 99, 100, 101, 102, 105  
 AGGREGATE member . . . . . 11, 13, 27, 28, 29, 38, 42, 44, 45, 46, 49, 50, 53, 56, 57, 58, 59, 62, 63, 75, 76, 78, 79, 80, 82, 83, 85, 95, 96, 97, 98, 99, 100, 101, 102, 105  
 ALIAS example . . . . . 49, 91  
 ALIAS keyword . . . . . 48, 91, 103  
 ALIGN example . . . . . 53  
 ALIGN keyword . . . . . 38, 44, 53  
 Alignment . . . . 14, 37, 38, 43, 44, 45, 52, 53, 57, 73, 78, 79, 80, 81, 82, 83, 84, 95, 97, 101  
 Alignment example . . . . . 53, 54, 82, 84  
 Alignment keywords . . 43, 44, 53, 57, 73, 82, 83, 84, 95  
 AND operator . . . . . 62  
 ANY data type . . . . . 57  
 ANY example . . . . . 57, 91  
 ANY keyword . . . . . 57  
 Arrays . . . . . 9, 11, 12, 13, 17, 29, 30, 43, 56  
 ASCII character set . . . . . 107, 110  
 ASCII value operator . . . . . 61  
 Asterisk array dimension . . . . . 9, 92  
 Asterisk multiplication operator . . . . . 62

## B

BASEALIGN example . . . . . 54, 82, 84  
 BASEALIGN keyword . . 44, 53, 57, 73, 82, 84, 95  
 BASED keyword . . . . . 45, 54, 75, 82, 105  
 BASED storage class . . . . . 54, 73, 75, 95  
 Bit offset symbol . . . . . 62, 78, 80  
 Bit offset symbol example . . . . . 81  
 BITFIELD data type . . . . . 42, 51, 58  
 BITFIELD declaration . . . . . 52, 58, 97, 102, 105  
 BITFIELD example . . 52, 56, 58, 75, 76, 77, 79, 81  
 BITFIELD keyword . . . . . 51, 58, 78, 79, 105  
 Block comments . . . . . 63, 71  
 Block comments example . . . . . 63  
 BOOLEAN data type . . . . . 58  
 BOOLEAN declaration . . . . . 52  
 BOOLEAN keyword . . . . . 52, 58, 105  
 Byte offset symbols . . . . . 13, 62, 78, 80  
 Byte offset symbols example . . . . . 81  
 BYTE data type . . . . . 51, 60, 61, 103  
 BYTE declaration . . . . . 52  
 BYTE example . . . . . 61  
 BYTE keyword . . . . . 51, 52, 55, 60, 61, 103

## C

CHARACTER data type . . . . . 42, 51, 59  
 CHARACTER declaration . . . . . 102, 105  
 CHARACTER declarations . . . . . 59  
 CHARACTER default tag . . . . . 51  
 CHARACTER example . . 12, 17, 49, 52, 53, 54, 59, 75, 81, 91  
 CHARACTER keyword . . . . . 51, 98, 105  
 Command line qualifiers . . . . 10, 11, 12, 13, 36, 37, 38, 39, 49, 57, 61, 65, 95, 97, 98, 99, 101  
 Comments . . . . . 13, 19, 41, 62, 71  
 COMMON declaration . . . . . 55  
 COMMON keyword . . . . . 44, 45, 55, 75, 82, 105  
 COMMON storage class . . . . . 44, 55, 73  
 Compiling source code . . . . . 36  
 COMPLEX keyword . . . . . 51, 59, 104  
 Conditional compilation . . . . . 39, 41, 64, 66  
 Conditional compilation example . . . . . 65  
 CONSTANT declaraion . . . . . 21  
 CONSTANT declaration . . . . 13, 16, 33, 43, 46, 49, 50, 63, 71, 72, 75, 81, 86, 103  
 CONSTANT DECLARATION . . . . . 88  
 CONSTANT description . . . . . 86  
 CONSTANT example . . . 12, 17, 42, 52, 53, 56, 81, 86, 87  
 CONSTANT keyword . . . 11, 13, 16, 42, 43, 46, 49, 50, 51, 62, 63, 71, 72, 81, 85, 86, 88, 97, 101, 102, 103  
 Copyright file . . . . . 38, 99  
 COUNTER example . . . . . 87  
 COUNTER keyword . . . . . 46, 89

COUNTER option ..... 87  
 Creating a source file ..... 12

## D

D\_FLOATING\_COMPLEX data type ..... 51, 104  
 D\_FLOATING\_COMPLEX keyword ..... 51, 104  
 D\_FLOATING data type ..... 51, 60, 104  
 D\_FLOATING keyword ..... 51, 60, 104  
 D\_FLOATING\_COMPLEX data type ..... 59  
 Data alignment ..... 78, 82  
 Data structures ..... 11, 13, 71, 76  
 Data type, \_FLOATING ..... 51  
 Data type, ADDRESS ..... 13, 57  
 Data type, ANY ..... 57  
 Data type, BITFIELD ..... 42, 51, 58  
 Data type, BOOLEAN ..... 58  
 Data type, BYTE ..... 51, 60, 61, 103  
 Data type, C/C++ summaries ..... 103  
 Data type, CHARACTER ..... 42, 51, 59  
 Data type, COMPLEX ..... 59  
 Data type, D\_FLOATING ..... 51, 60, 104  
 Data type, D\_FLOATING\_COMPLEX ..... 51, 104  
 Data type, D\_FLOATING\_COMPLEX ..... 59  
 Data type, DECIMAL ..... 60  
 Data type, default tags ..... 51  
 Data type, F\_FLOATING ..... 51, 60, 104  
 Data type, F\_FLOATING\_COMPLEX ..... 51, 104  
 Data type, F\_FLOATING\_COMPLEX ..... 59  
 Data type, floating-point ..... 60  
 Data type, G\_FLOATING ..... 51, 60, 104  
 Data type, G\_FLOATING\_COMPLEX ..... 51, 104  
 Data type, G\_FLOATING\_COMPLEX ..... 59  
 Data type, H\_FLOATING ..... 51, 60, 104  
 Data type, H\_FLOATING\_COMPLEX ..... 51, 104  
 Data type, H\_FLOATING\_COMPLEX ..... 59  
 Data type, HARDWARE\_INTEGER ..... 61, 104  
 Data type, implicit union ..... 76, 77, 78  
 Data type, INTEGER ..... 61, 104  
 Data type, INTEGER\_BYTE ..... 61, 104  
 Data type, INTEGER\_HW ..... 61, 104  
 Data type, INTEGER\_LONG ..... 61, 104  
 Data type, INTEGER\_QUAD ..... 61, 104  
 Data type, INTEGER\_WORD ..... 61, 104  
 Data type, keywords ..... 43  
 Data type, LONGWORD ..... 60, 61, 104  
 Data type, OCTAWORD ..... 60, 61, 104  
 Data type, POINTER ..... 57  
 Data type, prefixes ..... 13  
 Data type, QUADWORD ..... 51, 60, 61, 104  
 Data type, RETURNS ..... 48  
 Data type, S\_FLOATING ..... 51, 60, 104  
 Data type, S\_FLOATING\_COMPLEX ..... 51, 104  
 Data type, S\_FLOATING\_COMPLEX ..... 59  
 Data type, summaries ..... 103  
 Data type, T\_FLOATING ..... 60, 104  
 Data type, T\_FLOATING\_COMPLEX ..... 51, 104  
 Data type, T\_FLOATING\_COMPLEX ..... 59

Data type, tag ..... 42, 50, 51, 67  
 Data type, tags ..... 13  
 Data type, TYPEDEF ..... 10, 55  
 Data type, TYPENAME ..... 47, 48, 49  
 Data type, UNION ..... 51, 76, 103  
 Data type, user defined ..... 47, 48, 55, 67  
 Data type, WORD ..... 60, 61, 104  
 Data type, X\_FLOATING ..... 51, 60, 104  
 Data type, X\_FLOATING\_COMPLEX ..... 51, 104  
 Data type, X\_FLOATING\_COMPLEX ..... 59  
 Data types ..... 57  
 Decimal constants ..... 61  
 DECIMAL data type ..... 60  
 DECIMAL declaration ..... 60  
 DECIMAL example ..... 60  
 DECIMAL keyword ..... 51, 104  
 Declaration keywords ..... 43  
 Declarations ..... 11, 13, 41, 71  
 Declarations keywords ..... 43  
 Declarations modifier keywords ..... 44  
 DECLARE example ..... 68  
 DECLARE SIZEOF clause ..... 67, 68, 73, 84, 93  
 DECLARE SIZEOF example ..... 68  
 DECLARE statement ... 41, 67, 68, 72, 73, 84, 93,  
 97, 102  
 Declared output constants ..... 13  
 Default storage class ..... 54, 105  
 DEFAULT declaration ..... 92, 103  
 DEFAULT example ..... 49, 91  
 DEFAULT keyword ..... 9, 47, 92, 103  
 DIMENSION declaration ..... 74, 83, 84, 92, 102  
 DIMENSION example ..... 12, 17, 56, 68, 77, 81  
 DIMENSION keyword ... 9, 29, 43, 44, 56, 74, 77,  
 83, 84, 92, 95, 96, 105  
 DIMENSION option ..... 13, 45, 47, 56, 57  
 Division operator ..... 62

## E

ELSE example ..... 65  
 ELSE statement ..... 64, 65  
 ELSE\_IFSYMBOL example ..... 65  
 ELSE\_IFSYMBOL statement ..... 65, 101  
 END keyword ... 11, 12, 16, 17, 43, 50, 52, 53, 54,  
 55, 56, 57, 58, 59, 68, 75, 76, 77, 78, 79, 81, 82, 99  
 END\_IFLANGUAGE example ..... 64, 66  
 END\_IFLANGUAGE statement ..... 64, 98, 99  
 END\_IFSYMBOL example ..... 65  
 END\_IFSYMBOL statement ..... 65  
 END\_LITERAL example ..... 66  
 END\_LITERAL keyword ..... 66  
 END\_MODULE keyword ... 11, 12, 13, 17, 43, 71,  
 72, 99  
 ENTRY declaration ..... 47, 71, 72, 90, 103  
 ENTRY example ..... 49, 57, 59, 91  
 ENTRY keyword ..... 11, 42, 43, 57, 63, 72, 103  
 ENTRY keywords ..... 48  
 EQUALS example .. 12, 17, 42, 52, 53, 62, 81, 86, 87

EQUALS expression ..... 12  
 EQUALS keyword ..... 42, 46, 86, 103  
 EQUALS STRING example ..... 86  
 EQUALS STRING keyword ..... 89, 103  
 Error messages ..... 95  
 Expressions ..... 62, 86

## F

F\_FLOATING COMPLEX data type ..... 51, 104  
 F\_FLOATING COMPLEX keyword ..... 51, 104  
 F\_FLOATING data type ..... 51, 60, 104  
 F\_FLOATING keyword ..... 51, 60, 104  
 F\_FLOATING\_COMPLEX data type ..... 59  
 Fatal errors ..... 95, 96, 100, 101  
 Fatal messages ..... 95  
 FILL keyword ..... 45, 82, 84, 95, 96  
 Fillers ..... 77, 78, 79, 81, 84  
 Fixed-point decimal data ..... 60  
 Fixed-point precision ..... 60  
 Floating-point keywords ..... 51, 60, 104  
 Forcing data alignment ..... 52, 78  
 Forcing negative offsets ..... 78

## G

G\_FLOATING COMPLEX data type ..... 51, 104  
 G\_FLOATING COMPLEX keyword ..... 51, 104  
 G\_FLOATING data type ..... 51, 60, 104  
 G\_FLOATING keyword ..... 51, 60, 104  
 G\_FLOATING\_COMPLEX data type ..... 59  
 GLOBAL keyword ..... 13, 44, 45  
 GLOBAL storage class ..... 13, 44, 55

## H

H\_FLOATING COMPLEX data type ..... 51, 104  
 H\_FLOATING COMPLEX keyword ..... 51, 104  
 H\_FLOATING data type ..... 51, 60, 104  
 H\_FLOATING keyword ..... 51, 60, 104  
 HARDWARE\_INTEGER data type ..... 61, 104  
 HARDWARE\_INTEGER keyword ..... 61, 104

## I

Implicit union declaration ..... 76, 77, 78, 82  
 Implicit union keyword ..... 57  
 Informational message ..... 95  
 INTEGER data type ..... 61, 104  
 INTEGER keyword ..... 61, 104  
 INTEGER\_BYTE data type ..... 61, 104  
 INTEGER\_BYTE keyword ..... 61, 104  
 INTEGER\_HW data type ..... 61, 104  
 INTEGER\_HW keyword ..... 61, 104  
 INTEGER\_LONG data type ..... 61, 104  
 INTEGER\_LONG keyword ..... 61, 104  
 INTEGER\_QUAD data type ..... 61, 104  
 INTEGER\_QUAD keyword ..... 61, 104

INTEGER\_WORD data type ..... 61, 104  
 INTEGER\_WORD keyword ..... 61, 104

## L

Line comments ..... 63, 71  
 Line comments example ..... 63  
 Local comments ..... 62, 71  
 Logical AND operator ..... 62  
 Logical NOT operator ..... 62  
 Logical OR operator ..... 62  
 Logical shift ..... 62  
 LONGWORD data type ..... 61, 104  
 LONGWORD example ..... 61  
 LONGWORD keyword ..... 61, 104

## M

Multiplication operator ..... 62

## N

Negate operator ..... 62  
 Not output comments ..... 62  
 NOT operator ..... 62

## O

OCTAWORD data type ..... 61, 104  
 OCTAWORD keyword ..... 61, 104  
 OR operator ..... 62  
 Output comments ..... 71  
 Output Comments ..... 62  
 Output comments example ..... 63

## Q

QUADWORD data type ..... 51, 60, 61, 104  
 QUADWORD example ..... 91  
 QUADWORD keyword ..... 60, 61, 104

## S

S\_FLOATING COMPLEX data type ..... 51, 104  
 S\_FLOATING COMPLEX keyword ..... 51, 104  
 S\_FLOATING data type ..... 51, 60, 104  
 S\_FLOATING keyword ..... 51, 60, 104  
 S\_FLOATING\_COMPLEX data type ..... 59  
 Shift operator ..... 62  
 SIGNED keyword ..... 103, 104  
 Subtraction operator ..... 62

**T**

T\_FLOATING COMPLEX data type..... 51, 104  
 T\_FLOATING COMPLEX keyword..... 51, 104  
 T\_FLOATING data type..... 51, 60, 104  
 T\_FLOATING keyword..... 51, 60, 104  
 T\_FLOATING\_COMPLEX data type..... 59

**U**

Unary operator..... 62  
 UNION data type..... 51, 76, 103  
 UNION declaration..... 43, 76  
 UNION example..... 75, 77, 79  
 UNION keyword..... 9, 43, 51, 76, 82, 103  
 UNSIGNED example..... 49, 54, 61, 91

UNSIGNED keyword..... 51, 60, 61, 103, 104  
 User specified names..... 46, 47, 48, 49

**W**

Warning messages..... 95  
 WORD data type..... 61, 104  
 WORD keyword..... 61, 104

**X**

X\_FLOATING COMPLEX data type..... 51, 104  
 X\_FLOATING COMPLEX keyword..... 51, 104  
 X\_FLOATING data type..... 51, 60, 104  
 X\_FLOATING keyword..... 51, 60, 104  
 X\_FLOATING\_COMPLEX data type..... 59