OSS Nokalva®

OSS® ASN.1 Compiler for C
Reference Manual

October 2015

for release 10

Every distributed copy of the OSS® ASN.1 Tools for C is associated with a specific license and related unique license number. That license determines, among other things, what functions of the OSS® ASN.1 Tools for C are available to you.

Credits and Trademarks

Readers' comments may be addressed to support@oss.com. OSS Nokalva, Inc., may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

PREFACE

About This Manual

This document serves as the reference manual for the OSS ASN.1 Tools.

OSS Nokalva, Inc., developed the OSS ASN.1 Tools for converting message definitions from ASN.1 to C, and for encoding, decoding and in general working with such messages. The OSS ASN.1 Tools implement the ASN.1 language as described by the following standards:

ITU-T Recommendation X.680 (2008) | ISO/IEC 8824-1:2008
Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation

ITU-T Recommendation X.681 (2008) | ISO/IEC 8824-2:2008
Information technology - Abstract Syntax Notation One (ASN.1): Information object specification

ITU-T Recommendation X.682 (2008) | ISO/IEC 8824-3:2008
Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification

ITU-T Recommendation X.683 (2008) | ISO/IEC 8824-4:2008
Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications

ITU-T Recommendation X.690 (2008) | ISO/IEC 8825-1:2008
Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

ITU-T Recommendation X.691 (2008) | ISO/IEC 8825-2:2008
Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)

ITU-T Recommendation X.693 | ISO/IEC 8825-4: 2008
Information technology - ASN.1 encoding rules: XML Encoding Rules (XER)

ITU-T Recommendation X.694 | ISO/IEC 8825-5: 2008
Information technology - ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1

ITU-T Recommendation X.696 | ISO/IEC 8825-7: 2014
Information technology - ASN.1 encoding rules: Specification of Octet Encoding Rules (OER)

CCITT Recommendation X.208 (1988) [1]
Specification of Abstract Syntax Notation One (ASN.1)

ISO/IEC 8824:1990
Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)

[1] In the few places where X.208 (1988) and ISO/IEC 8824:1990 are contradictory, the OSS® ASN.1 Tools implements ISO/IEC 8824:1990.

CCITT Recommendation X.209 (1988)
Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

ISO/IEC 8825:1990
Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

Related Documents and Manuals

For general information on ASN.1 and the OSS® ASN.1 Tools, see our website at (http://www.oss.com).

For information on how to install the OSS ASN.1 Tools, and for customized examples on how to use the toolkit on each supported platform, refer to the **OSS ASN.1 Tools for C Getting Started Manual.**

For more information about the functions available in the OSS run-time ASN.1 API, refer to the **OSS ASN.1/C Runtime API Reference Manual**.

For details about ASN.1 Studio, introduced in version 9.0, refer to the **ASN.1 Studio® Reference Manual**.

For more information about the functions available in the OSS Interpretive ASN.1 API, refer to the **OSS ASN.1 Tools for C IAAPI Reference Manual**.

For more information about the routines to parse and analyze PER encoded data, refer to the **OSS PER Encoding Analyzer API Manual**.

For details about the error messages issued by the OSS ASN.1/C compiler, refer to the **OSS ASN.1 Tools Compiler Error Reference Manual**.

For details about the error messages issued by the OSS ASN.1/C runtime functions, refer to the **OSS ASN.1 Tools Runtime Error Reference Manual**.

To learn about using the OSS OSAK Memory Manager, refer to **Using the OSS OSAK Memory Manager**.

Organization

This manual is organized as follows:

Chapter 1 introduces the OSS ASN.1 Tools.

Chapter 2 tells you where to look for installation information and complete examples that are customized for each supported platform.

Chapter 3 extensively describes the OSS ASN.1 compiler options available.

Chapter 4 gives details of the OSS ASN.1 compiler directives available for use.

Chapter 5 describes the role configuration files play in customizing the OSS ASN.1 compiler.

Chapter 6 notes some restrictions that the present OSS ASN.1 compiler places on Abstract Syntaxes and derived encodings/decodings.

Chapter 7 presents details about the C representations of ASN.1 notation that the present compiler uses.

Chapter 8 describes the OSS macro processor for backward compatibility purposes.

Appendix A discusses the role of OSS ASN.1 compiler configurations directives.

Appendix B contains some demonstrative examples.

Appendix C deals with the OSS BER Type-Length-Value Utility (`osstlv.exe`).

Appendix D sketches out the purpose and use of the version information utility (`osswhat.exe`).

Appendix E holds a glossary of commonly used terms in this document.

Appendix F gives information about contacting OSS Nokalva for obtaining help on both technical and non-technical matters.

Notational Conventions

In this manual, the following notational conventions are used:

Items enclosed in brackets, [ ], or angle brackets, < >, are optional.

The vertical bar, |, indicates alternative items.

Underlined items indicate defaults.

Items in **boldface type** must be entered as shown, while those in italics indicate that you must substitute the appropriate value.

An ellipsis, ..., indicates that multiple entries for the preceding item are allowed.

Table of Contents

1 Introduction

This chapter introduces the OSS ASN.1 Tools while the chapters that follow provide detailed information on the ASN.1 compiler. For details on installing the toolkit, see the **OSS ASN.1 Tools for C Getting Started Manual**.

1.1 Overview

The International Standards Organization (ISO), the International Electrotechnical Commission (IEC) and the International Telecommunications Union - Telecommunications Sector (ITU-T) (formerly known as the International Telegraph and Telephone Consultative Committee (CCITT)) have established **Abstract Syntax Notation One (ASN.1)** and its encoding rules as a standard for describing and encoding messages. **ASN.1** is a formal notation for abstractly describing data to be exchanged between distributed computer systems. **Encoding rules** are sets of rules used to transform data specified in the ASN.1 notation into a standard format that can be decoded by any system that has a decoder based on the same set of rules.

The **OSS ASN.1 Tools** is a toolkit developed by **OSS Nokalva** to support ASN.1 and its encoding rules. The toolkit supports the full ASN.1 syntax as described by the ASN.1:1990, and ASN.1:2008 standards and the following encoding rules**: Basic Encoding Rules (BER), Packed Encoding Rules (PER)** (aligned and unaligned), **Distinguished Encoding Rules (DER)**, **Canonical Encoding Rules (CER), XML Encoding Rules (Basic: XER, Canonical: CXER, and Extended: E-XER), OCTET Encoding Rules (OER), and Canonical OCTET Encoding Rules (C-OER)**. It completely shields the user from the intricacies of the encoding rules.

Notes:
1) If you would like to learn more about the Abstract Syntax Notation, its encoding rules, and the OSS ASN.1 Tools, you should note that OSS Nokalva offers **Professional ASN.1 Training Courses** (contact info@oss.com for more information).
2) If you would like personalized help in developing your ASN.1-based application or are facing technical difficulties implementing your ASN.1 specification, you can contact info@oss.com to request OSS Nokalva's **Professional ASN.1 Consulting Services** especially designed to help you produce a reliable product in a shorter timeframe.

1.2 Components of the OSS ASN.1 Tools package

The **OSS ASN.1 Tools** consists of the **OSS ASN.1 compiler** (**asn1**), several **encoder/decoders** (the SOED and TOED described below) which use BER, PER, CER, DER, XER, E-XER, CXER, OER, and C-OER to encode and decode application data, plus several other **utilities** to simplify and speed up your development process.

The **OSS ASN.1 compiler (asn1)** is a stand-alone program that takes as input one or more files which in turn can each contain one or more **ASN.1 modules**. The compiler verifies that the specification is valid, and can generate:

- diagnostic messages to help you pinpoint errors in your syntax
- a C language header file containing data structures to be included into your application program (see section 3.3.30)
- a control table for use by the space-optimized or a code file containing the time-optimized encoder/decoder (see sections 3.3.9 and 3.3.15)
- an input-module listing-file (see sections 3.3.45 and 3.3.48)
- a file that captures all on-screen compiler output (see section 3.3.23)
- a file that captures all input compiler directives and generates directives for name preservation (see sections 3.3.28 and 3.3.43)
- a file that contains compiler directives to ignore from input syntax (see section 4.4.1.7)

The OSS ASN.1 Tools for C on Windows and Linux are equipped with a graphical application, called **ASN.1 Studio**, that greatly facilitates ASN.1 compilation and assists in performing many other tasks. Refer to section 4 of the **Getting Started Manual** for an overview of ASN.1 Studio.

The **space-optimized encoder/decoder (SOED)** is table-driven and consists of the functions `ossEncode()` and `ossDecode()`. `ossEncode()` takes data placed by your application program into the compiler-generated C data structures and converts it to a string of bytes encoded according to BER, PER, CER, DER, XER, E-XER, CXER, OER, or C-OER. The other function, `ossDecode()`, takes a string of bytes that have been encoded according to BER, PER, CER, DER, XER, E-XER, CXER, OER, or C-OER and performs the inverse operation of the encoder, resulting in a C data structure that you can easily manipulate. The space-optimized encoder/decoder is so named because its emphasis is on minimizing use of memory, especially when the abstract syntax is large or complex. Both the encoder and decoder possess a simple yet flexible memory management interface, as well as a wealth of trace, error trapping, diagnostic and recovery capabilities. Figure 1.1 shows how the components of the OSS ASN.1 Tools are used to build an application that uses the space-optimized encoder/decoder.

Note: OSS Nokalva also separately provides (contact info@oss.com) a third type of table-driven encoder/decoder (called the Lean encoder/decoder (LED)) which has a speed comparable to the time-optimized encoder/decoder and a memory usage smaller than that of the space-optimized encoder/decoder. This encoder/decoder also has the same interface as the SOED and is well-suited for embedded systems with limited available memory running applications which require fast processing times. For further information, refer to the documentation of the `-lean` compiler option in section 3.3.44.

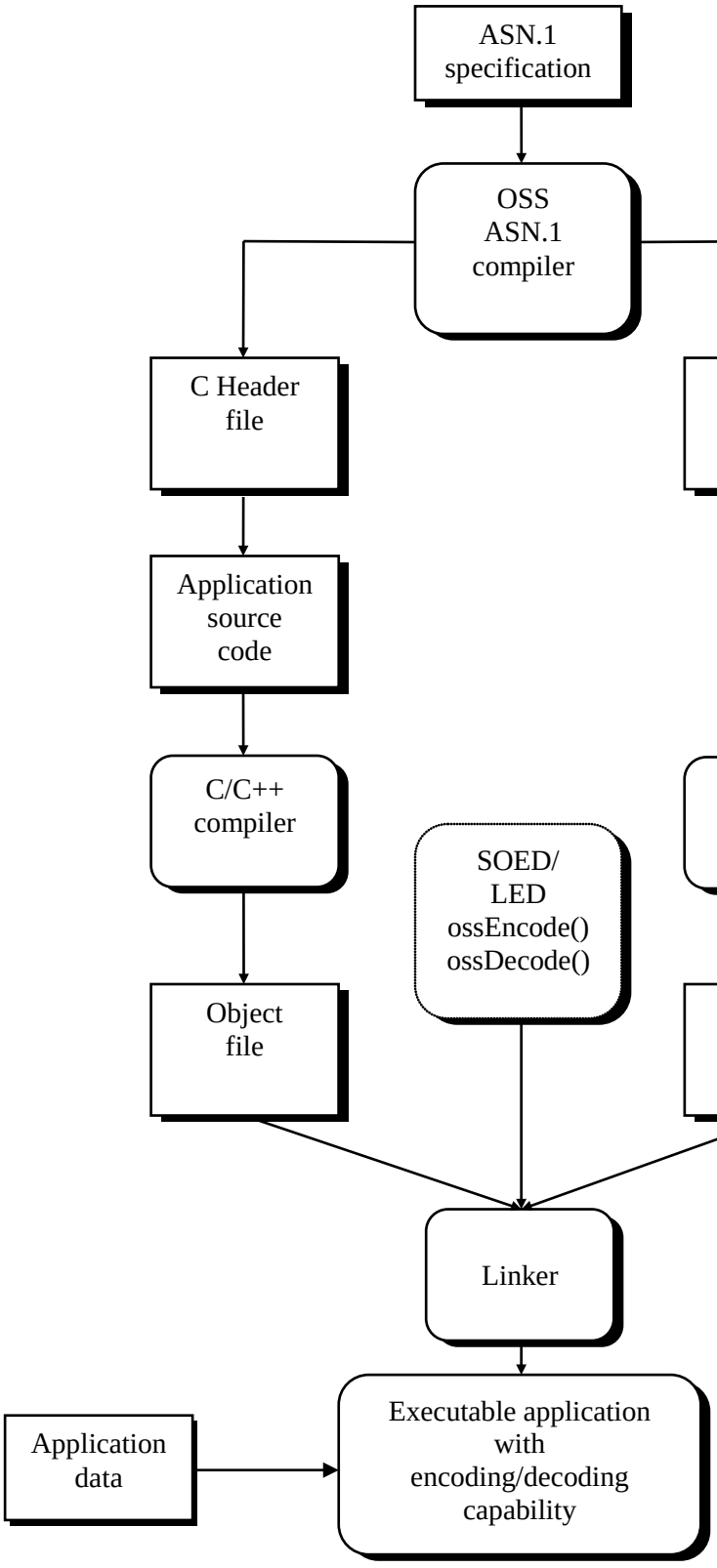


Figure 1.1: Building an application that uses the SOED or LED encoder/decoder

The **time-optimized encoder/decoder (TOED)** has a program call interface identical to the space-optimized encoder/decoder's and performs the same functions (except that it does not possess any trace capability). The main advantage that the TOED has over the SOED encoder/decoder is that it minimizes CPU utilization.

Figure 1.2 shows how the components of the OSS ASN.1 Tools are used to build an application that uses the time-optimized encoder/decoder.

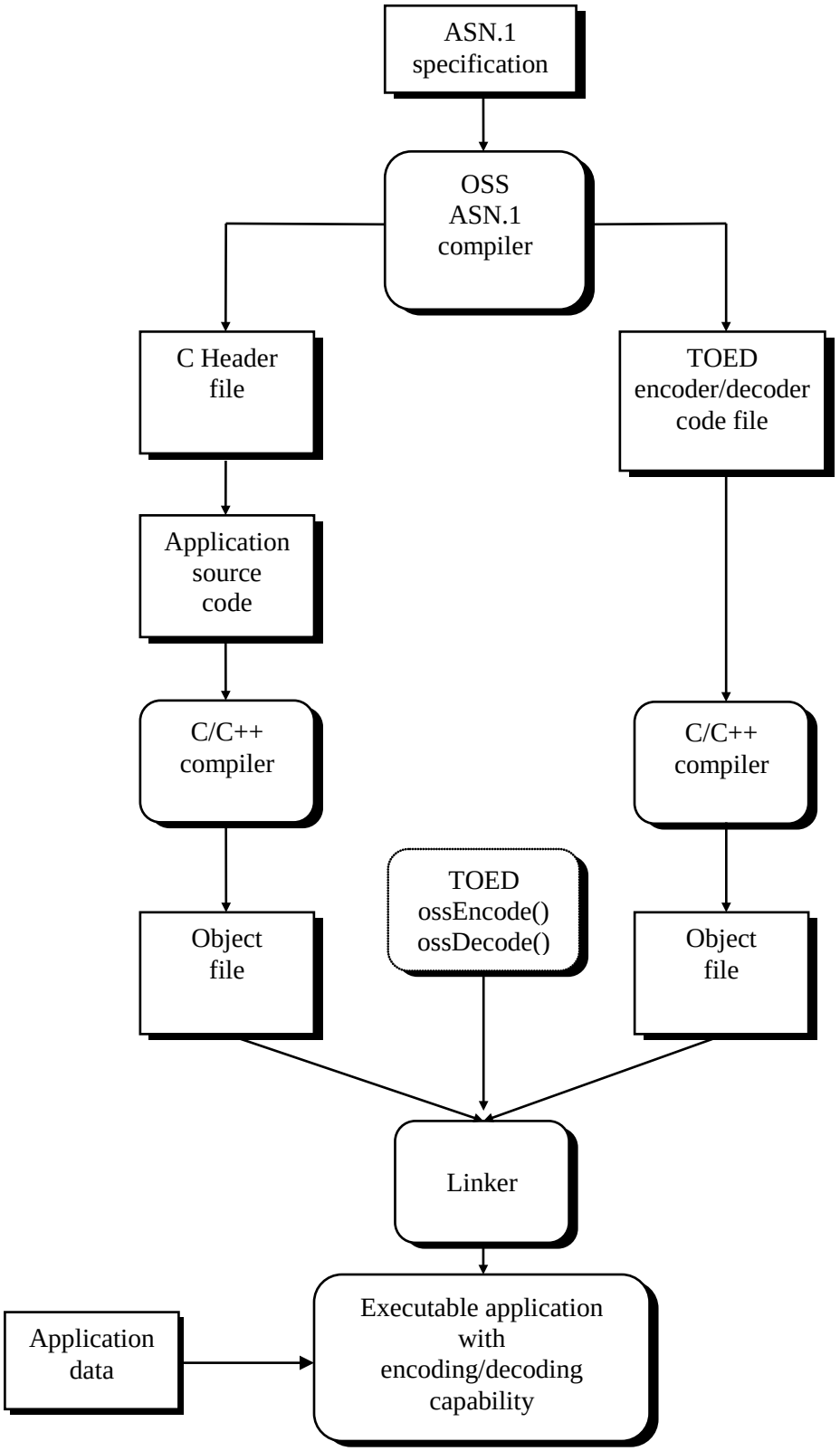


Figure 1.2: Building an application that uses the TOED

All subsequent references to ‘encoder/decoder', unless qualified, apply equally to the space-optimized, Lean, and time-optimized encoder/decoder and for BER, CER, DER, PER (aligned and unaligned), XER, CXER, E-XER, OER, and C-OER variants.

The OSS ASN.1 Tools contain many other functions; for a description of the complete list of functions, refer to the **OSS ASN.1/C Runtime API Reference Manual**.

1.3 OSS ASN.1 Tools shipments

The OSS ASN.1 Tools is packaged in four ways for shipment: compiler-only, target (run-time only), development (full), or tuned tools shipment.

The **compiler-only shipment** includes the OSS ASN.1 compiler only; since the run-time libraries are missing, this shipment cannot produce an executable program. It can be used to compile your ASN.1 specification for use on the same platform as that on which the compiler is running or to **cross-compile** (compiling **ASN.1 specifications** on a particular computer platform to obtain `.c` and `.h` files for use on a different platform) to any other platform on which you are licensed to use the run-time libraries.

The **target shipment** includes the OSS run-time libraries only; the OSS ASN.1 compiler is missing, thus this shipment cannot compile your ASN.1 specification. It can only be used for run-time support on the **target platform** where your application that uses the encoder/decoder will execute.

The **development shipment** is a combination of the compiler-only and target shipments. It includes both the OSS ASN.1 compiler and the run-time libraries. The development shipment can be used to compile your ASN.1 specification for use on the same type of platform as that on which the compiler is running, or to cross-compile to any other platform on which you are licensed to use the run-time libraries. Since the run-time libraries are included, the development shipment can also be used for run-time support on the target platform where your application that uses the encoder/decoder will execute.

The **tuned tools shipment** is an economic alternative to the development shipment. This shipment includes no ASN.1 compiler; rather, it contains custom-built header files (containing ASN.1 data structure representations) and encoding/decoding runtime routines specific to the ASN.1 specification that you are using and to the platform that you are running on. Since only necessary functions are included, the runtime libraries sent with this shipment will lead to a significantly smaller application object code size. For more information about the tuned tools shipment, contact info@oss.com.

More information on the first three shipments listed above is given in the **OSS ASN.1 Tools Getting Started Manual**, which gives details about the contents of the OSS ASN.1 Tools package for the different platforms on which the OSS ASN.1 Tools can be used.

1.4 Product foundation

Development specifications are based upon the latest ASN.1 standard, viz.: ITU-T Rec. X.680 | ISO/IEC 8824-1, ITU-T Rec. X.681 | ISO/IEC 8824-2, ITU-T Rec. X.682 | ISO/IEC 8824-3, ITU-T Rec. X.683 | ISO/IEC 8824-4, ITU-T Rec. X.690 | ISO/IEC 8825-1, ITU-T Rec. X.691 | ISO/IEC 8825-2, ITU-T Rec. X.693 | ISO/IEC 8825-4, ITU-T Rec. X.694 | ISO/IEC 8825-5, and ITU-T Rec. X.696 | ISO/IEC 8825-7.

The toolkit is also backward compatible with the ASN.1:1990 standard as defined by the following standards: CCITT Rec. X.208 | ISO/IEC 8824 (1990), CCITT Rec. X.209 | ISO/IEC 8825 (1990), the corrigenda described by ISO/IEC JTC1/SC21 N5901 (April 1991), and the NIST Stable Implementation Agreements (1991).

Complete support is provided for the full **ASN.1:1990** and **ASN.1:2008** syntaxes; the ASN.1 type, subtype, value, information object class, parameterization, and macro notations as well as the following encoding rules: BER, CER, DER, PER (aligned and unaligned), XER, CXER, E-XER, OER, and C-OER are supported as specified in the international standards.

A few minor restrictions are imposed on the ASN.1 grammar as practical limitations (e.g., the maximum length of a line is 4096 bytes) or to circumvent known defects in the macro notation. Restrictions on the ASN.1 Macro Notation are listed in section 8.3 and all other restrictions are listed in chapter 6.

1.5 Customizing the OSS ASN.1 Tools to meet your needs

Several product customization features are built into the OSS ASN.1 Tools to tailor the output to suit your needs. These features include command-line options, compiler directives, and encoder/decoder flags. For example, the OSS ASN.1 compiler directives allow you to choose among various C language representations available for an ASN.1 type; thus, you can choose to have an ASN.1 character string type represented either as a length-prefixed or null-terminated character string.

1.6 Differences in Sample Code

The many examples in this manual assume that the size of a `short int` is two bytes, an `int` and a `long int` are both four bytes, and a `long long int`, if defined, is eight bytes. Integers larger than eight bytes can be stored in the platform-independent HUGE representation (see sections 4.4.1.3 & 4.4.2.24), which allows a maximum integer size of 4096 bytes.

2 Getting started

See the **OSS ASN.1 Tools for C Getting Started Manual** for instructions on how to install the OSS ASN.1 Tools and for customized examples on how the toolkit may be used on the supported platforms.

3 Using the OSS ASN.1 compiler and its options

3.1 The Command-line

The command-line syntax described in this chapter is used on most platforms on which the compiler runs. This syntax may differ on some systems, such as IBM MVS and Tandem NonStop OS. For such systems, see the **OSS ASN.1 Tools for C Getting Started Manual** for details on how the syntax differs.

3.1.1 Introduction

The compiler options allow you greater flexibility over the behavior of the **OSS ASN.1 compiler**. The following sections describe the command-line syntax used by the OSS ASN.1 compiler. Note that OSS provides the ASN.1 Studio graphical user interface (GUI) on selected platforms. ASN.1 Studio graphically represents the compiler options, calls the ASN.1 compiler and assists in performing many other tasks. For more information about the ASN.1 Studio interface, refer to section 3.2.

IMPORTANT NOTE: Starting with version 9.0, OSS has changed the default behavior of the OSS ASN.1 compiler. Now, by default, the OSS ASN.1 compiler (`asn1`) generates the time-optimized encoder/decoder (TOED) instead of the space-optimized control file (SOED).
The command line:

```
asn1
```

is now equivalent to the command line:

```
asn1 –toed
```

3.1.2 Command-line syntax

Command-line keyword options are prefixed by a dash (`-`). They may be shortened to the fewest number of characters needed to differentiate them from other keywords. Command-line keyword options are not case-sensitive. Thus, they may be specified in all uppercase, all lowercase, or mixed case.

Some keywords accept operands; keyword operands are not prefixed with a dash.

If two command-line options contradict each other (e.g., both `-headerFile` and `-noHeaderFile` are specified), the rightmost option is used.

3.1.3 Examples of compiler invocation

The following compiler invocations show alternative ways of specifying the input filename. Both invocations will compile the contents of `input1.asn`.

```
asn1 input1

asn1 input1.asn
```

The following compiler invocations show alternative ways of specifying the `-noWarning` command-line option. Both invocations will suppress all warning messages while compiling the contents of `input1.asn`. Notice that the second invocation uses an abbreviated form of the command-line option.

```
asn1 input1 -noWarning

asn1 input1 -now
```

The following compiler invocations show two different ways of specifying the command-line option to produce an output listing file. The first invocation creates an output listing file named `input2.lst` after compiling the contents of `input1.asn` and `input2.asn` (By default, output filenames are derived from the last input filename.) while the second invocation creates an output listing file named `listFile.lst` after compiling the contents of `input1.asn` and `input2.asn`

```
asn1 input1 input2 -listingFile

asn1 input1 input2 -listingFile listFile
```

Note that the last command-line option, `listFile`, is not prefixed with a dash (`-`) since it is an operand to the `-listingFile` keyword preceding it.

3.1.3.1 Specifying input files

The following are important notes about specifying input filenames on the command line:

1) Items on the command line that do not start with a hyphen and are not operands to keyword options are considered to be input filenames.

2) The default file extension for input files is `.asn` and if the filename contains no extension, an extension of `.asn` will be added to it before attempting to open the file. (To specify a filename without an extension, you should place a single `'.'` at the end of the filename prefix. You may also specify filenames with embedded white-space in them by enclosing the entire filename in double quotes (e.g., `"my syntax definitions.asn"`))

3) The optional cross-compiling configuration **asn1dflt** file (see section 5.1) must be the first file listed on the command-line.

4) A file that contains macro definitions must be specified prior to any files that reference such definitions (especially when the references include a macro argument).

5) Similar to most language compilers, the OSS ASN.1 compiler expects input files to be plain text files.

3.2 ASN.1 Studio: an IDE for ASN.1 application development

Starting with version 9.0, the new ASN.1 Studio graphical application is available to ASN.1/C compiler users on Windows and Linux. ASN.1 Studio may be regarded as a Graphical User Interface (GUI) to the ASN.1 compiler, but it can do much more. In fact, this is an Integrated Development Environment (IDE) that supports the entire lifecycle of your ASN.1 application, from writing or combining ASN.1 syntax files to producing data for testing. You can even use ASN.1 Studio after deploying your application to analyze real field data.

The following functions are available in ASN.1 Studio:

ASN.1 compiling

- *Create* ASN.1 Studio *project files* with the `.a1sproj` extension. ASN.1 Studio projects provide a visual representation of the files and compiler options you are using. The `New Project Wizard` simplifies project creation. You can work with known ASN.1 Standards and reuse project settings and files from existing projects.
- *Import* ASN.1 Studio *project files* from command line files or from old ASN.1/C compiler GUI project formats.
- *Create*, *modify*, and *view ASN.1 files* in a text editor that is specially designed for editing ASN.1 syntaxes, with highlighting and auto-completion of ASN.1 reserved words.
- *Import ASN.1 files* from Microsoft Word (.doc) files. The ASN.1 must be specially separated from other text in the document, as is done in some standards.
- *Invoke* any of the installed versions of *ASN.1 compilers* for C and other languages (C++, Java, C#) for which a license is available. When invoking old compilers, unsupported options are filtered out to avoid compilation errors.
- *Maintain* a list of *Generated Files* produced by the compiler. ASN.1 Studio indicates when these files become outdated relative to the input ASN.1 files and compiler options.
- *Export* any ASN.1 Studio *project file* to an ASN.1/C compiler command file with the `.cmd` extension. Then you can invoke the compiler from the command line and pass the name of the exported file as a parameter to the `-command` compiler option. This can be helpful if you are using the command line ASN.1/C compiler in an automated build environment.

Working with ASN.1 encoded data

- Graphically *create* any arbitrary *ASN.1 value* for any valid PDU type.
- Use a tree-like view to graphically *display* and *modify values* of any field in a message and *save* the resultant value to an *encoding* file using any of the available *encoding rules*.
- *View* and *edit* binary encoded messages in hexadecimal format and use a *TLV* (tag-length-value) mode for *BER-based encoded messages*.
- *View* and *edit XML-based* encoded messages in a *text editor*.
- *View* the details of *PER encodings* in a convenient *tree-like form*, collapse and expand the details of nested encodings in the `Encoding Viewer`.

Building different types of ASN.1 applications instantly

- *Export* any ASN.1 Studio project to a *Microsoft Visual Studio project* on Windows.
- *Export* any ASN.1 Studio project to a Microsoft `nmake` *makefile* on Windows or to a `gmake` *makefile* on Linux.
- An exported makefile or project can build an *executable program* linked with any version of the OSS ASN.1 runtime libraries (/MT, /MD or DLL on Windows; static, static position independent or shared on Linux), a *static runtime library*, or a *shared library* (*DLL* on Windows).
- Any type of OSS runtime (*SOED*, *TOED*, or *LED*) may be used by an exported application, depending on the compiler options specified in the project.
- The `Project Export Wizard` can generate a file with a placeholder `main()` or `DllMain()` function and add it to the project.
- Exported projects or makefiles support both *Debug* and *Release* configurations.
- For those using a Qt `qmake` utility as a build tool, a `.pro` file may be produced.

ASN.1 Studio automatically checks for available software updates and notifies the user when updates are available.

Section 4 of the **Getting Started Manual** gives brief instructions to help you get started using ASN.1 Studio. Consider it a short introduction; your primary source of information is the *ASN.1 Studio help file*, which is invoked by selecting **Help → Contents** or **F1**.

Section 3.3 of this manual provides a cross reference of the ASN.1 Studio controls and the command line options. You can also mouse over the ASN.1 Studio controls in a *Project Settings dialog* to see the compiler options represented by those controls.

3.3 Compiler options

This section outlines for each compiler command-line option:

- its purpose,
- its operands (if any),
- the compiler components to which the option applies,
- the default value for this option. (**Note:** Default command-line options have underlined headings)

For those command-line options that accept a filename operand (e.g., `-codeFile`, `-controlFile`, `-headerFile`, `-listingFile`, `-modListingFile`, and `-output`) specifying a filename operand of only a dash (`-`) will cause the output to be written to the standard output.

Compiler options listed by category

The ASN.1 compiler's options can be divided into the following categories:
(Note that an option may fall into more than one category.)

C file content controls:

```
-codeFile, -constraints, -controlFile, -debug,
-decodeOnly / -encodeOnly, -lean, -soedFile, -test,
-toedFile, -uniquePDU, -userConstraints
```

Encoding rules specification controls:

```
-ber / -cer / -der /-per / -uper / -xer / -cxer / -exer / -oer / -coer
```

Header file content/arrangement controls:

```
-ANSI, -C / -C++, -comments / -noComments, -CstyleComments, -defines /
-noDefines, -dualHeader, -pdusForContainingTypes, -pdusForOpenTypes,
-shippable, -sort, -splitHeader, -splitForSharing, -CStyleComments
```

Helper names, macros and API controls:

```
-helperNames / noHelperNames, -helperAPI / -noHelperAPI,
-helperMacros / -noHelperMacros, -helperListAPI / noHelperListAPI,
-hdebug, -exportDllAPI, -helperDefineNames, -helperEnumNames
```

Miscellaneous options:

```
-allow, -assignments, -commandFile, -exportDllData, -help,
-ignoreIncompleteItems, -ignoreRedefinedAssignments,
-relaxedMode / -norelaxedMode, -relaySafe, -root,
-sampleCode / -noSampleCode
```

Performance and efficiency options:

```
-autoEncDec, -compress, -enablePartialDecode, -lean, -minimize,
-noConstraints, -noDebug, -partialDecodeOnly
-restrictedConstraintChecking
```

Compiler options

Specification of output files:

```
-codeFile, -controlFile, -dtd, -errorFile, -genDirectives,
-headerFile, -keepNames, -listingFile, -modListingFile,
-output, -soedFile, -toedFile, -xsl
```

Standard output controls:

```
-designerWarnings, -ignoreError, -ignoreSuppress,
-informatoryMessages, -messageFormat, -suppress, -uniquePDU,
-verbose, -warningMessages
```

Syntax-checking (with no output-file generation) controls:

```
-pedantic, -syntaxOnly
```

Type representation controls:

```
-charIntegers, -extendOpenType
```

Valuereference controls:

```
-noStaticValues, -valueRef
```

Variable name manipulation aides:

```
-externalName, -prefix, -reservedWords, -shortenNames,
-useQualifiedNames
```

Version compatibility aides:

```
-1990 / -2008, -compat, -genDirectives, -keepNames
```

Compiler options quick reference table

Compiler Option	Brief description	Section
`-1990`	specifies the **ASN.1:1990** syntax is used in the input	3.3.1
`-2008`	specifies the **ASN.1:2008** syntax is used in the input	3.3.1
`-allow`	to retain compiler behavior from previous versions	3.3.2
`-ANSI`	specifies that the C compiler being used is fully ANSI compliant	3.3.3
`-assignments`	instructs the ASN.1 compiler to process type and value assignments found outside of defined input modules	3.3.4
`-autoEncDec`	allows open types to be automatically encoded/decoded even if runtime constraint checking is disabled	3.3.5
`-ber`	specifies that the default encoding rule is BER	3.3.6
`-C`	specifies the generation of header files for use with a C compiler	3.3.7
`-C++`	specifies the generation of header files for use with a C++ compiler	3.3.7
`-cer`	specifies that the default encoding rule is CER	3.3.6
`-charIntegers`	instructs the ASN.1 compiler to generate an `unsigned char` or a `signed char` for small INTEGER type variables	3.3.8
`-codeFile`	generate an output C file containing the time-optimized encoder / decoder.	3.3.9
`-coer`	specifies that the default encoding rule is C-OER	3.3.6
`-commandFile`	specifies the name of the text-file containing command-line options for the current ASN.1 compile.	3.3.10
`-comments`	instructs the ASN.1 compiler to transfer ASN.1 comments to the header file as C/C++ comments	3.3.11
`-compat`	specifies the generation of header files compatible with those generated by previous versions of the OSS ASN.1 compiler.	3.3.12
`-compress`	specifies that encoding compression should be available at runtime	3.3.13
`-constraints`	specifies the generation of information in the control table for runtime constraint checking	3.3.14
`-controlFile`	informs the ASN.1 compiler to generate a control table for use by the space-optimized encoder/decoder.	3.3.15
`-CStyleComments`	specifies the transfer of all ASN.1 comments into the header file using C-language style comments	3.3.16
`-cxer`	specifies that the default encoding rule is Canonical XER	3.3.6
`-debug`	specifies the generation of debugging information into the C file which allows the run-time to trace PDUs	3.3.17
`-decodeOnly`	specifies the generation of only decoding routines for the time-optimized encoder/decoder	3.3.18
`-defines`	specifies generation of `#defined` constants into the header file	3.3.19
`-der`	specifies that the default encoding rule is DER	3.3.6
`-designerWarnings`	option instructs the ASN.1 compiler to generate extra warning messages for protocol designers	3.3.20

Compiler options

`-dtd`	instructs ASN.1 compiler to generate XML data type definitions for input PDUs	3.3.21
`-dualHeader`	instructs the ASN.1 compiler to generate one header file which can be included in both C and C++ applications	3.3.22
`-enablePartialDecode`	instructs the compiler to enable partial decoding in addition to standard decoding	3.3.22
`-encodeOnly`	specifies the generation of only encoding routines for the time-optimized encoder/decoder	3.3.18
`-errorFile`	specifies the redirection of all ASN.1 compiler messages to the named error file	3.3.23
`-exer`	specifies that the default encoding rule is Extended XER	3.3.6
`-exportDllAPI`	instructs the ASN.1 compiler to export helper list API functions from a control table or a codefile DLL	3.3.24
`-exportDllData`	instructs the ASN.1 compiler to export initialized values from a control table or codefile DLL	3.3.25
`-extendOpenType`	specifies the generation of an extra field called `userfield` of type `void *` in the `OpenType` structure	3.3.26
`-externalName`	allows the renaming of the external variable that references the space-optimized encoder/decoder control table or the time-optimized encoder/decoder entry points	3.3.27
`-genDirectives`	instructs the ASN.1 compiler to generate a `.gen` file that captures all the directives from the ASN.1 input syntax as well as the directives for names fabricated by the compiler	3.3.28
`-headerFile`	allows the generation of a header file which represents the ASN.1 root module in C	3.3.30
`-hdebug`	instructs the ASN.1 compiler to generate compile-time type checking diagnostic inside helper macros	3.3.29
`-help`	specifies the display of a synopsis of the command-line syntax	3.3.31
`-helperAPI`	turns on a group of ASN.1 compiler options: -helperNames, -helperMacros, and -helperListAPI	3.3.32
`-helperDefineNames`	instructs the compiler to generate context-based names for #define constants used for bit-masks, named bits and numbers	3.3.33
`-helperEnumNames`	instructs the compiler to generate context-based names for enumerators	3.3.34
`-helperListAPI`	instructs the ASN.1 compiler to generate a set of list manipulation API functions for each SET OF / SEQUENCE OF type specified in the input syntax	3.3.35
`-helperMacros`	instructs the ASN.1 compiler to generate helper macros for each generated structure	3.3.36
`-helperNames`	instructs the ASN.1 compiler to generate context-based names for built-in ASN.1 types whose C representations are not simple types	3.3.37
`-ignoreError`	instructs the ASN.1 compiler to treat a certain error as a mere warning instead of an error	3.3.38
`-ignoreIncompleteItems`	instructs the compiler to ignore types and values that directly or indirectly reference types or values whose definitions were not found in the input ASN.1 modules	3.3.39
`-ignoreRedefinedAssignments`		

	instructs the compiler to ignore redefined ASN.1 definitions of types and values	3.3.40
-ignoreSuppress	instructs the compiler to ignore all OSS.SUPPRESS directives	3.3.41
-informatoryMessages	allows the generation of all compiler informatory messages	3.3.42
-keepNames	causes the generation of a .gen file containing directives to preserve variable names in the header file for future compatibility	3.3.43
-lean	instructs the ASN.1 compiler to generate output files for use with the fast and efficient Lean encoder/decoder (LED)	3.3.44
-listingFile	specifies the generation of an ASN.1 composite listing of the root module(s)	3.3.45
-messageFormat	controls the format of the error/warning/informatory messages used by the ASN.1 compiler	3.3.46
-minimize	specifies the automatic selection of command-line options which results in the most compact control table	3.3.47
-modListingFile	tells the compiler to produce a composite module listing file	3.3.48
-noConstraints	tells compiler not to generate information in the control table for runtime constraint checking	3.3.14
-noControlFile	instructs the ASN.1 compiler not to generate an output .c file	3.3.15
-noComments	tells the ASN.1 compiler not to transfer ASN.1 comments to the header file	3.3.11
-noDebug	instructs the ASN.1 compiler not to generate debugging information into the output C file	3.3.17
-noDefines	specifies that #defined constants should not be generated into the header file	3.3.19
-noHeaderFile	instructs the ASN.1 compiler not to generate an output .h file	3.3.30
-noHelperAPI	turns off ASN.1 compiler options -helperNames, -helperMacros, and -helperListAPI	3.3.32
-noHelperListAPI	instructs the ASN.1 compiler to not generate a set of list manipulation API functions for each SET OF / SEQUENCE OF type specified in the input syntax	3.3.35
-noHelperMacros	instructs the ASN.1 compiler to not generate helper macros for each generated structure	3.3.36
-noHelperNames	instructs the ASN.1 compiler to use old name generation convention for built-in ASN.1 types whose C representations are not simple types	3.3.37
-noInformatoryMessages	suppressed the generation of all compiler informatory messages	3.3.42
-noListingFile	specifies the suppression of an ASN.1 composite listing of the root module(s)	3.3.45
-noModListingFile	tells the compiler not to produce a composite module listing file	3.3.48
-noOutput	specifies that output files should not be generated	3.3.51
-noPdusForContainingTypes	instructs the ASN.1 compiler to not treat types referenced by contents constraints as PDUs	3.3.52
-noPdusForOpenTypes	instructs the ASN.1 compiler to not treat open types as PDUs	3.3.53

Compiler options

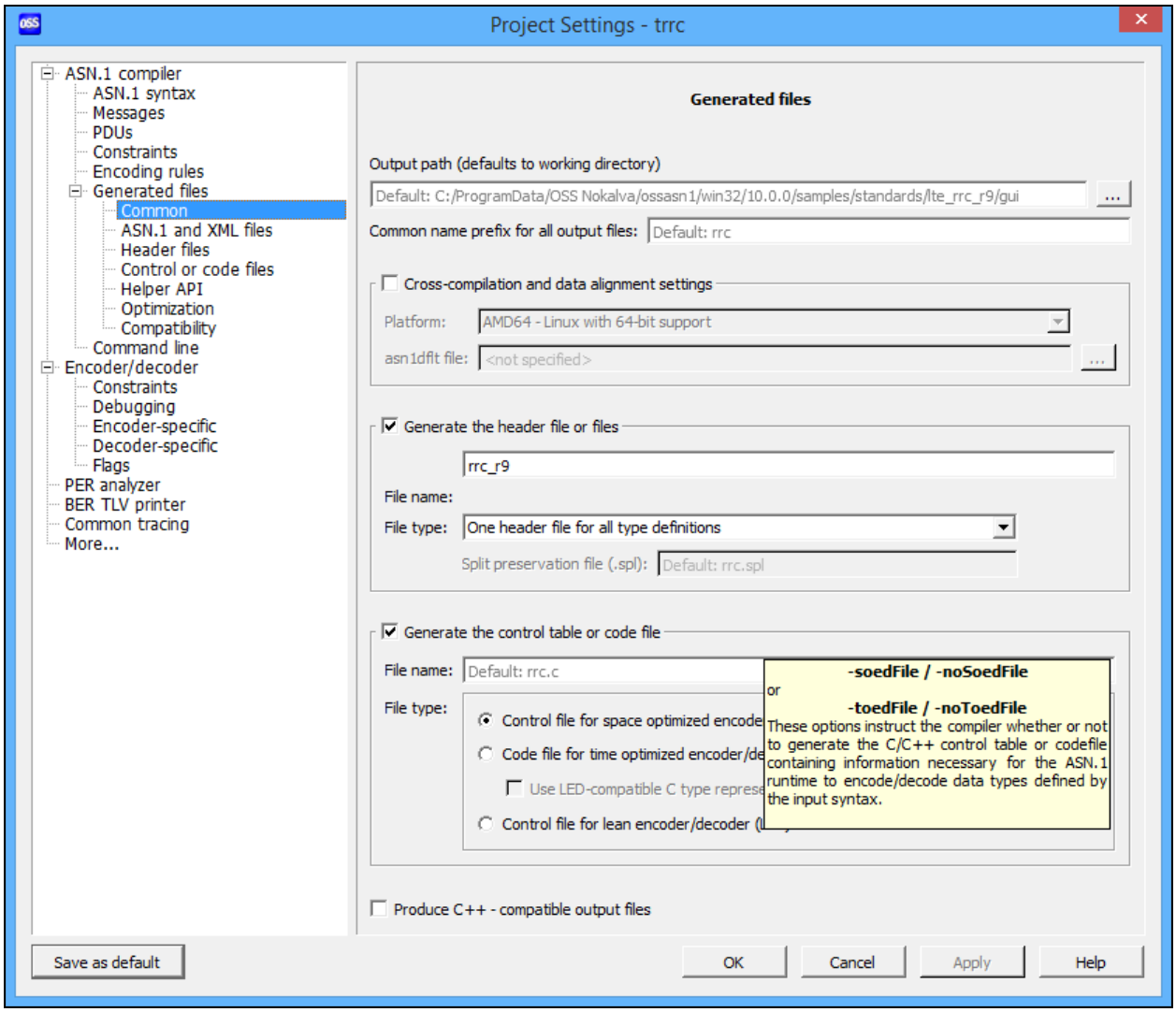
-noRelaxedMode	prevents the automatic selection of command-line options that would result in relaxed compiler behavior	3.3.56
-noRelaySafe	disables the use of relay-safe decoding and re-encoding of messages with extension addition fields	3.3.58
-noSampleCode	instructs the ASN.1 compiler not to generate sample code	3.3.62
-noShortenNames	specifies that long variable names should not be shortened	3.3.64
-noSignalHandler	specifies that the OSS Compile-and-Go-Library (CAGL) should not install signal handlers (this option is not recognized by the OSS ASN.1 compiler)	3.3.49
-noSoedFile	instructs the ASN.1 compiler not to generate an output .c file	3.3.65
-noSort	indicates that variables and structures in the header file should be put in the same order as they appear in the ASN.1 input	3.3.66
-noStaticValues	specifies that the ASN.1 compiler should not statically initialize any variable generated from the ASN.1 value notation	3.3.50
-noToedFile	instructs the ASN.1 compiler not to generate an output .c file	3.3.72
-noUniquePDU	specifies that protocol data unit (PDU) tags need not be unique.	3.3.73
-noUserConstraints	specifies the suppression of user-defined constraints from the control table and constraint-checking function prototypes from the header file	3.3.75
-noValueRef	suppresses IAAPI valuereference information from the control table	3.3.77
-noVerbose	suppresses compiler status messages	3.3.78
-noWarningMessages	suppresses the generation of all compiler warning and informatory messages	3.3.79
-oer	specifies that the default encoding rule is OER	3.3.6
-output	specifies that output files should be generated and allows the custom-renaming of output files	3.3.52
-partialDecodeOnly	instructs the compiler to replace standard decoding with partial decoding	3.3.53
-pdusForContainingTypes	instructs the ASN.1 compiler to treat types referenced by contents constraints as PDUs	3.3.52
-pdusForOpenTypes	instructs the ASN.1 compiler to ignore the local and global OSS.NOPDU directive and to treat open types created by the use of the ASN1.DeferDecoding/OSS.ENCODABLE directive or created for types with component relation constraint as PDUs	3.3.53
-pedantic	instructs the compiler to perform a thorough syntax-check of the entire abstract syntax without generating output files	3.3.54
-per	specifies that the default encoding rule is ALIGNED PER	3.3.6
-prefix	specifies one or more characters to prefix to all generated names whose scope is global in your C program file.	3.3.55
-relaxedMode	specifies the automatic selection of command-line options which results in relaxed compiler behavior	3.3.56
-relaxPerToedIntegerRangeConstraint	instructs the compiler to generate TOED code so as to permit the user to deliberately force the encoding of a disallowed, but possible, value	3.3.57

`-relaySafe`	specifies the use of relay-safe decoding and re-encoding of messages with extension addition fields	3.3.58
`-reservedWords`	instructs the compiler to treat the words specified in the option as reserved words, which should always be mangled in the generated files	3.3.59
`-restrictedConstraintChecking`	don't ignore the `OSS.NoConstrain` directive even when the explicit -constraints options is specified	3.3.60
`-root`	instructs the compiler to treat all input modules as if they were root modules	3.3.61
`-sampleCode`	instructs the ASN.1 compiler to generate sample code to demonstrate how to initialize, encode, decode and traverse values of PDU types defined in the input ASN.1 syntax	3.3.62
`-shippable`	instructs the ASN.1 compiler to generate a special version of the header file suitable for distribution in a toolkit	3.3.63
`-shortenNames`	specifies the shortening of variable names generated by the ASN.1 compiler to 31 characters	3.3.64
`-soedFile`	Informs the ASN.1 compiler to generate a control table for use by the table-driven encoders/decoders (Space-optimized and Lean encoder/decoder)	3.3.65
`-sort`	indicates that variables and structures in the header file should be put in an order acceptable to a standard C compiler	3.3.66
`-splitForSharing`	instructs the ASN.1 compiler to generate multiple autonomous header files from a single ASN.1 specification	3.3.67
`-splitHeaders`	instructs the ASN.1 compiler to generate multiple interdependent header files for a single ASN.1 specification	3.3.68
`-suppress`	instructs the compiler to suppress a specific warning or informatory message	3.3.69
`-syntaxOnly`	instructs the ASN.1 compiler to only do a syntax check on the ASN.1 input without generating output files	3.3.70
`-test`	instructs the ASN.1 compiler to generate a `main()` routine which calls the function `ossTest()`	3.3.71
`-toedFile`	generate an output .c file containing the time-optimized encoder/decoder	3.3.72
`-uniquePDU`	specifies that protocol data unit (PDU) tags must be unique.	3.3.73
`-uper`	specifies that the default encoding rule is UNALIGNED PER	3.3.6
`-useQualifiedNames`	instructs the ASN.1 compiler to to add a prefix, depending on the type name, to enumerators generated for ENUMERATED types, and to constants generated for named numbers of INTEGER types and named bits of BIT STRING types.	3.3.74
`-userConstraints`	specifies the generation of user-defined constraints into the control table and constraint-checking function prototypes in the header file.	3.3.75
`-useXMLNames`	causes generated names in the output header and C file to be the same as the names used for the components of the corresponding E-XER encoding	3.3.76
`-valueRefs`	specifies the generation of IAAPI valuereference information in the control table	3.3.77

Compiler options

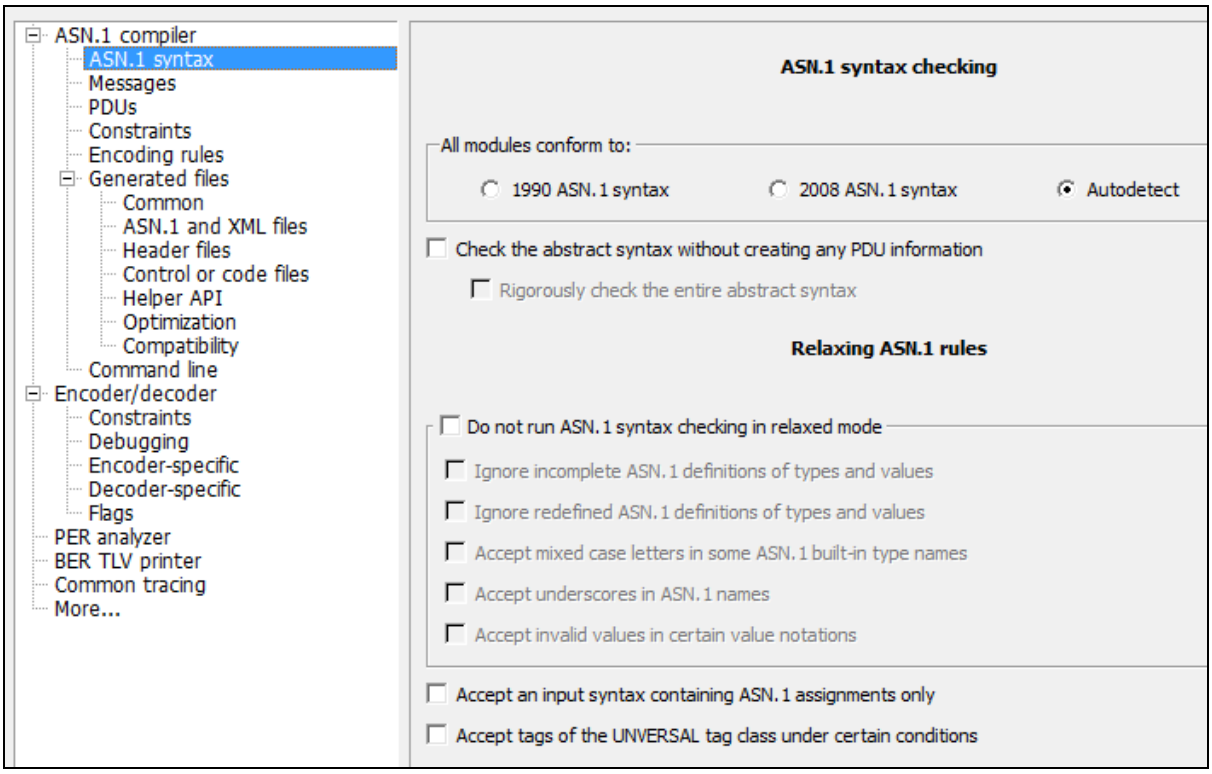
`-verbose`	instructs the ASN.1 compiler to issue status messages	3.3.78
`-warningMessages`	allows the generation of all compiler warning and informatory messages.	3.3.79
`-xsl`	instructs ASN.1 compiler to generate XML stylesheets for input PDUs	3.3.80
`-xer`	specifies that the default encoding rule is Basic XER	3.3.6

On platforms where the ASN.1 compiler is used with the ASN.1 Studio GUI (Graphical User Interface), all compiler options can be specified in a Project Settings dialog. Below are shown the dialog pages and a list of compiler options represented by each page.



Compiler options

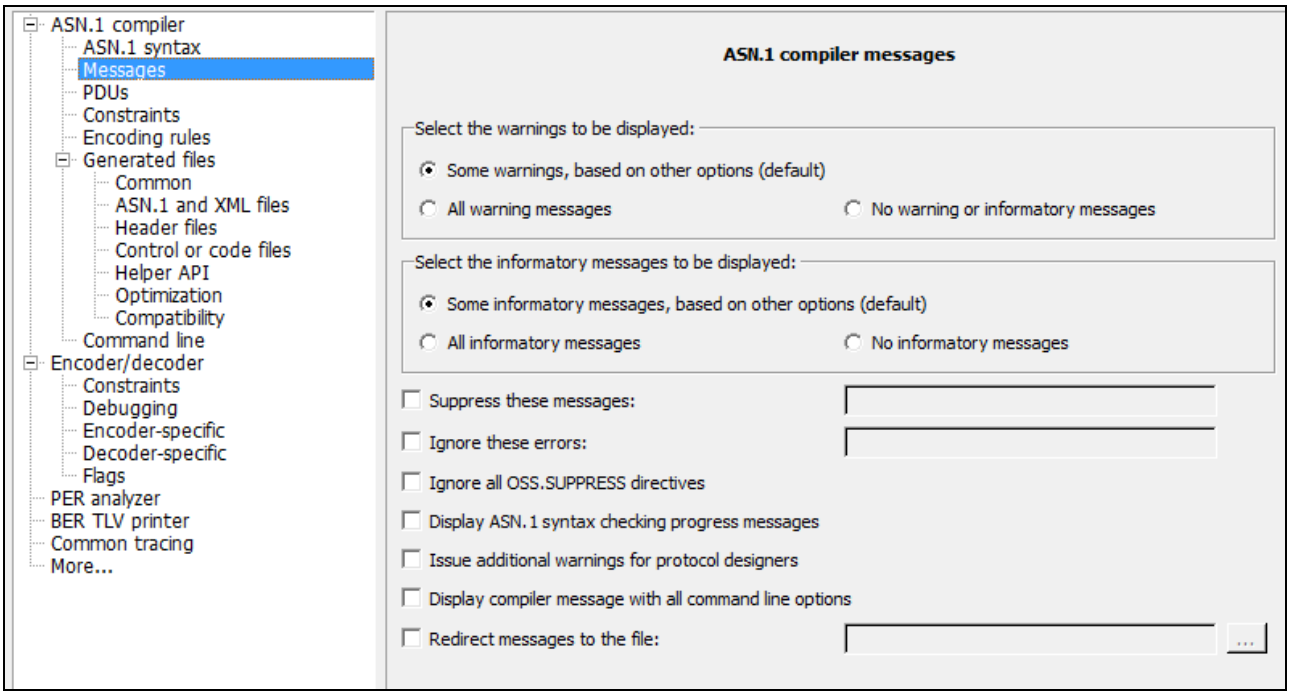
ASN.1 syntax



Options on this page:

`-1990` / `-2008` (see 3.3.1)
`-syntaxOnly` (see 3.3.70)
`-pedantic` (see 3.3.54)
`-relaxedMode/-norelaxedMode` (see 3.3.56)
`-ignoreIncompleteItems` (see 3.3.39)
`-ignoreRedefinedAssignments` (see 3.3.40)
`-allow MixedCaseForSomeBuiltInTypesNames` (see 3.3.2)
`-allow UnderscoresInAsn1Names` (see 3.3.2)
`-allow BadValues` (see 3.3.2)
`-assignments` (see 3.3.4)
`-allow universaltags` (see 3.3.2)

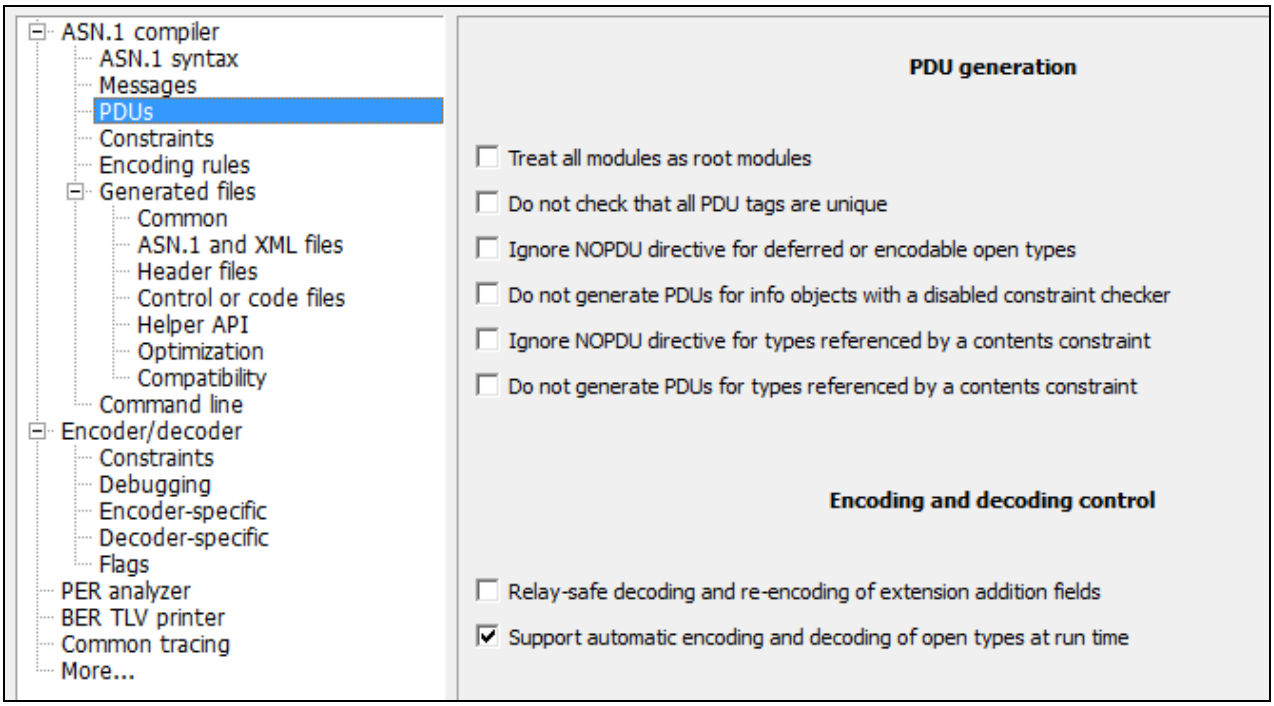
Messages



Options on this page:

`-warningMessages` / `-noWarningMessages` (see 3.3.79)
`-informatoryMessages/-noInformatoryMessages` (see 3.3.42)
`-suppress` (see 3.3.69)
`-ignoreError` (see 3.3.38)
`-ignoreSuppress` (see 3.3.41)
`-verbose/-noVerbose` (see 3.3.78)
`-designerWarnings` (see 3.3.20)
`-errorFile` (see 3.3.23)

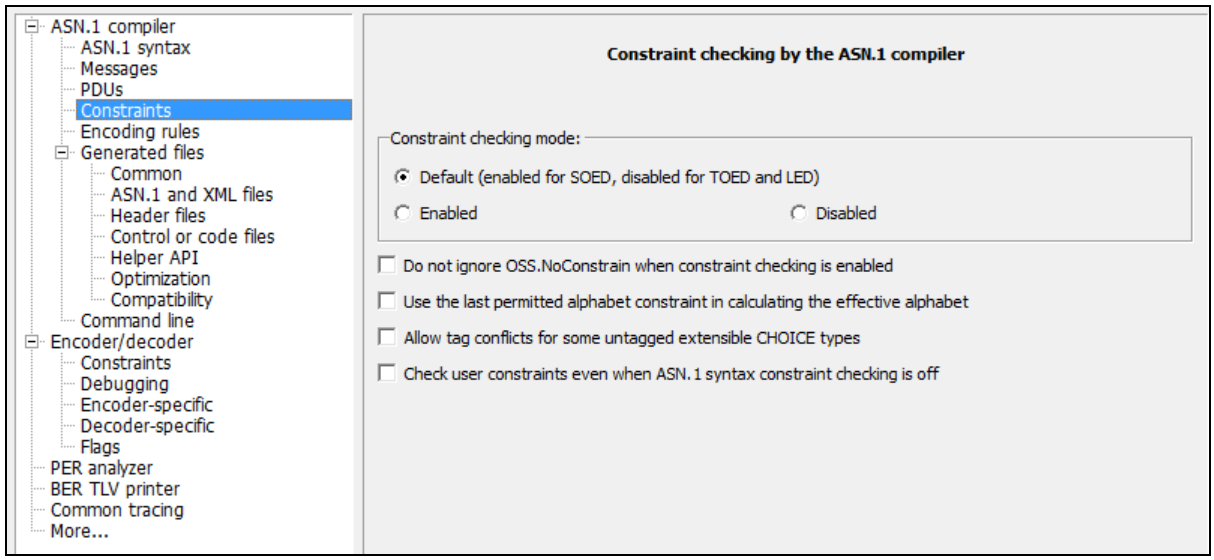
PDUs



Options on this page:

`-root` (see 3.3.61)
`-uniquePDU/-noUniquePDU` (see 3.3.73)
`-pdusForOpenTypes/-noPdusForOpenTypes` (see 3.3.52)
`-pdusForContainingTypes/-noPdusForContainingTypes` (see 3.3.52)
`-relaySafe/-noRelaySafe` (see 3.3.58)
`-autoEncDec` (see 3.3.5)

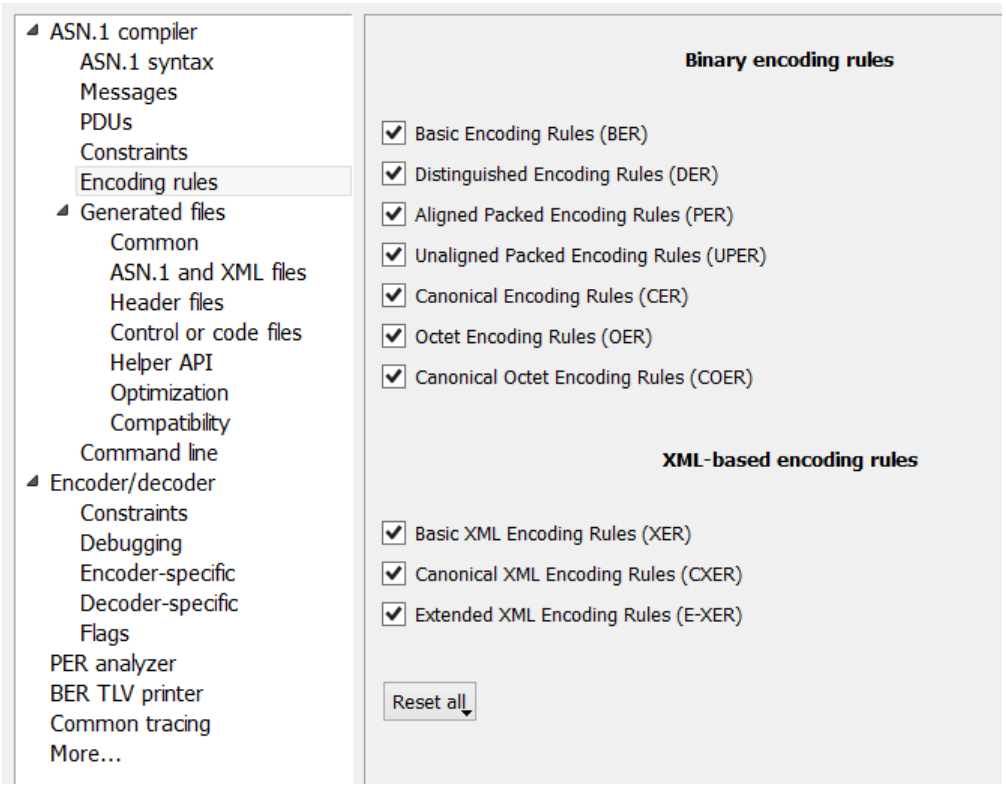
Constraints



Options on this page:

`-constraints/-noConstraints` (see 3.3.14)
`-restrictedConstraintChecking` (see 3.3.60)
`-allow BadSerialFROM` (see 3.3.2)
`-allow Bad6.0Extensions` (see 3.3.2)
`-userConstraints` (see 3.3.75)

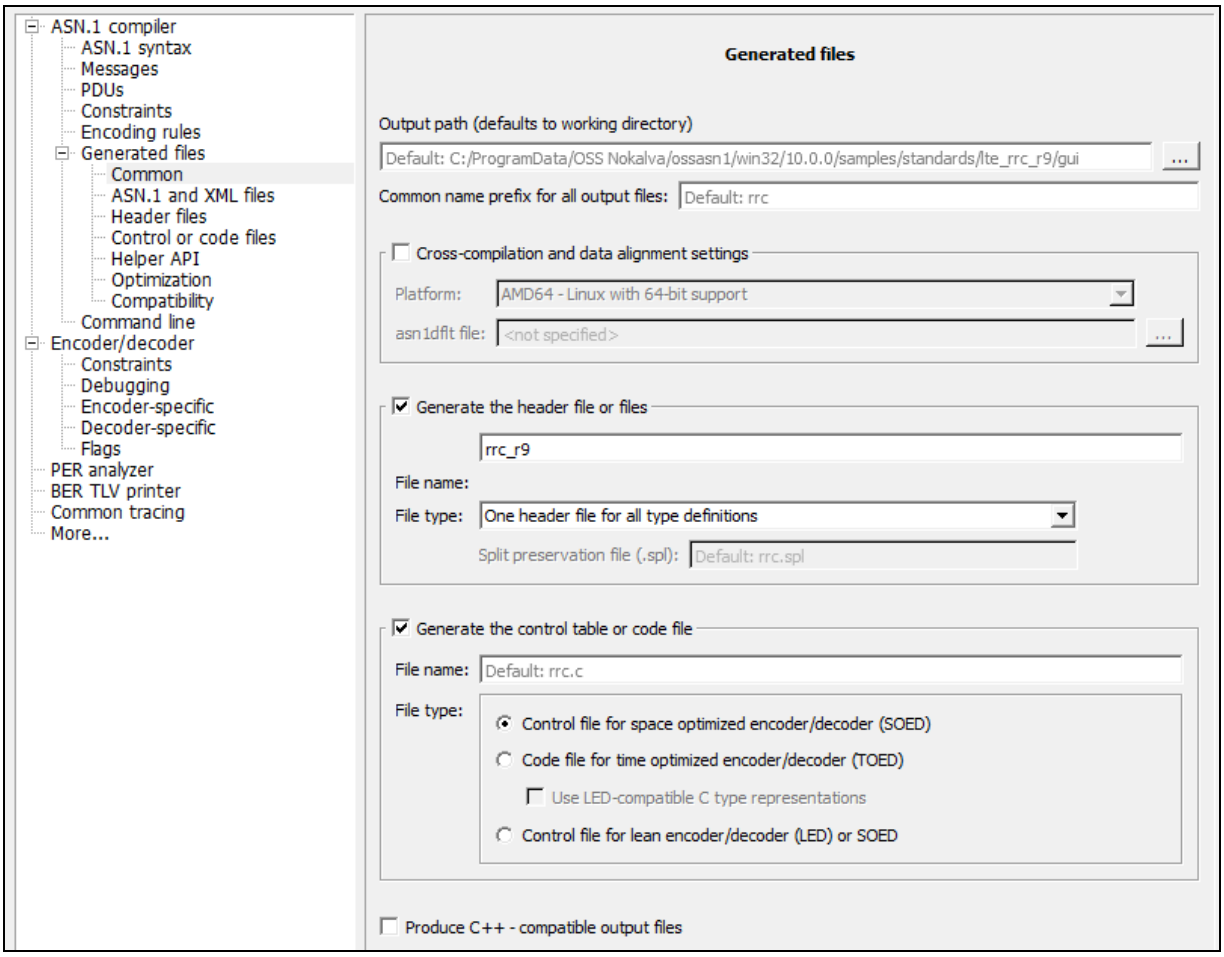
Encoding rules



Options on this page:

`-ber/-der/-per/-uper/-cer/-oer/-coer/-xer/-cxer/-exer` (see 3.3.6)

Generated files / Common

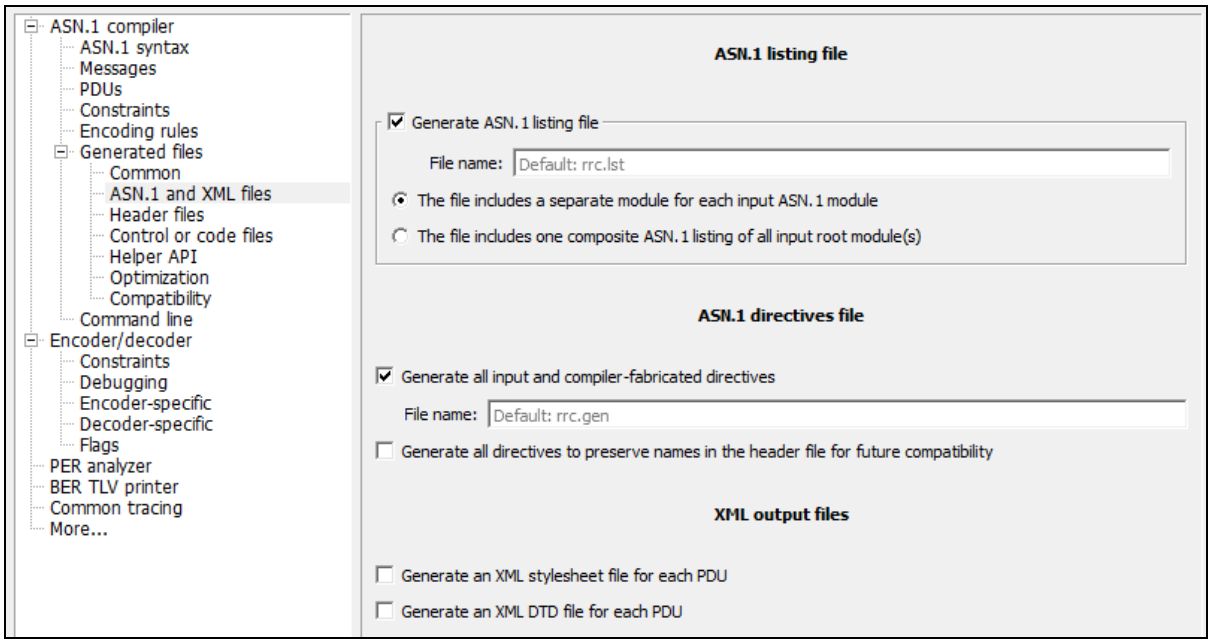


Options on this page:

`-output` (see 3.3.51)
`-headerFile/-noHeaderFile` (see 3.3.30)
`-splitHeaders` (see 3.3.68)
`-splitForSharing` (see 3.3.67)
`-shippable` (see 3.3.63)
`-controlFile/-noControlFile` (see 3.3.15)
`-soedFile/-noSoedFile` (see 3.3.65)
`-codeFile/-noCodeFile` (see 3.3.9)
`-toedFile/-noToedFile` (see 3.3.72)
`-lean` (see 3.3.44)
`-C/-C++` (see 3.3.7)

Compiler options

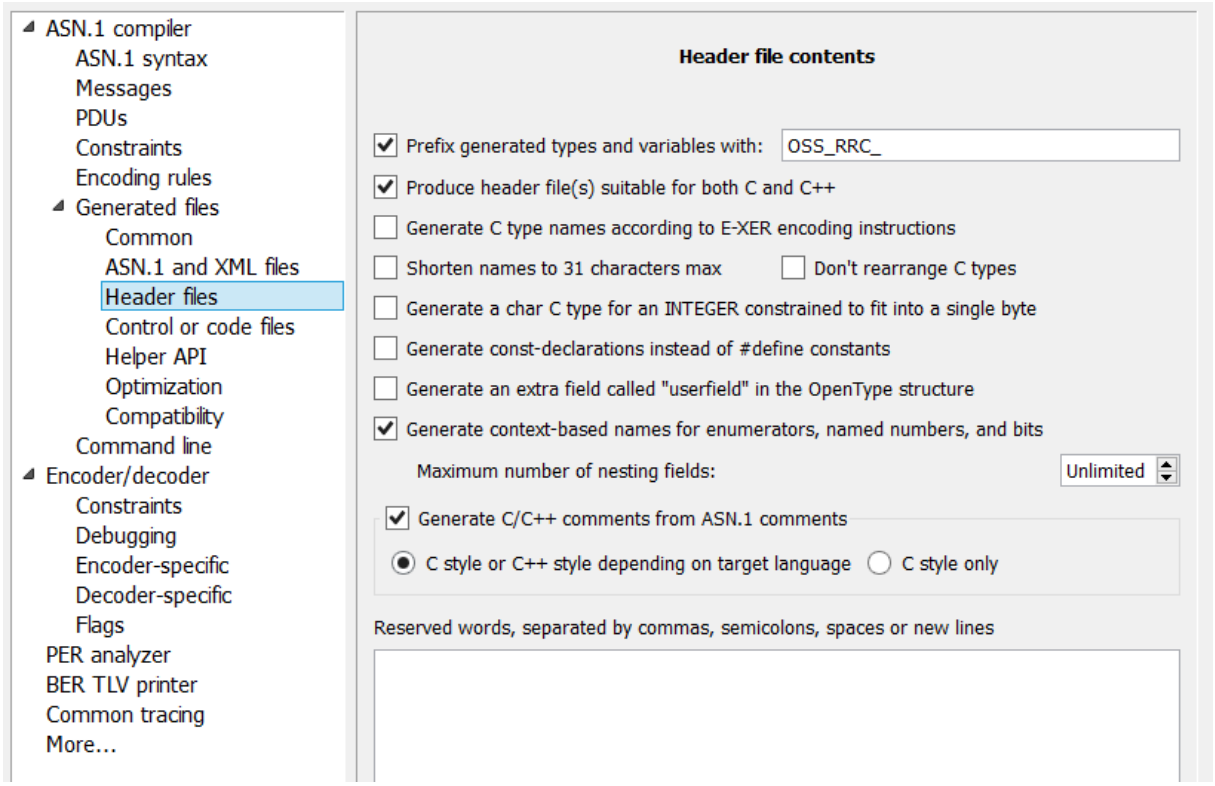
Generated files / ASN.1 and XML files



Options on this page:

`-modListingFile` (see 3.3.48)
`-listingFile` (see 3.3.45)
`-genDirectives` (see 3.3.28)
`-keepNames` (see 3.3.43)
`-xsl` (see 3.3.80)
`-dtd` (see 3.3.21)

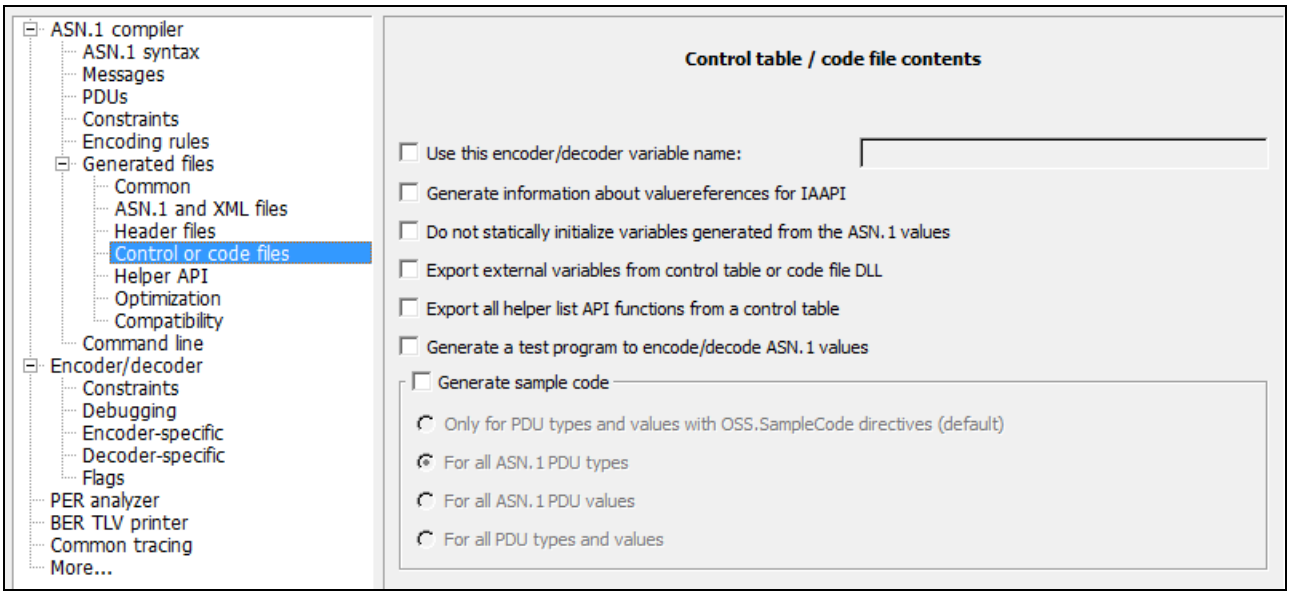
Generated files / Header files



Options on this page:

`-prefix` (see 3.3.55)
`-dualHeader` (see 3.3.22)
`-useXMLNames` (see 3.3.76)
`-shortenNames/-noShortenNames` (see 3.3.64)
`-sort/-noSort` (see 3.3.66)
`-charIntegers` (see 3.3.8)
`-defines/-noDefines` (see 3.3.19)
`-extendOpenType` (see 3.3.26)
`-useQualifiedNames` (see 3.3.74)
`-comments/-noComments` (see 3.3.11)
`-CStyleComments` (see 3.3.16)
`-reservedWords` (see 3.3.59)

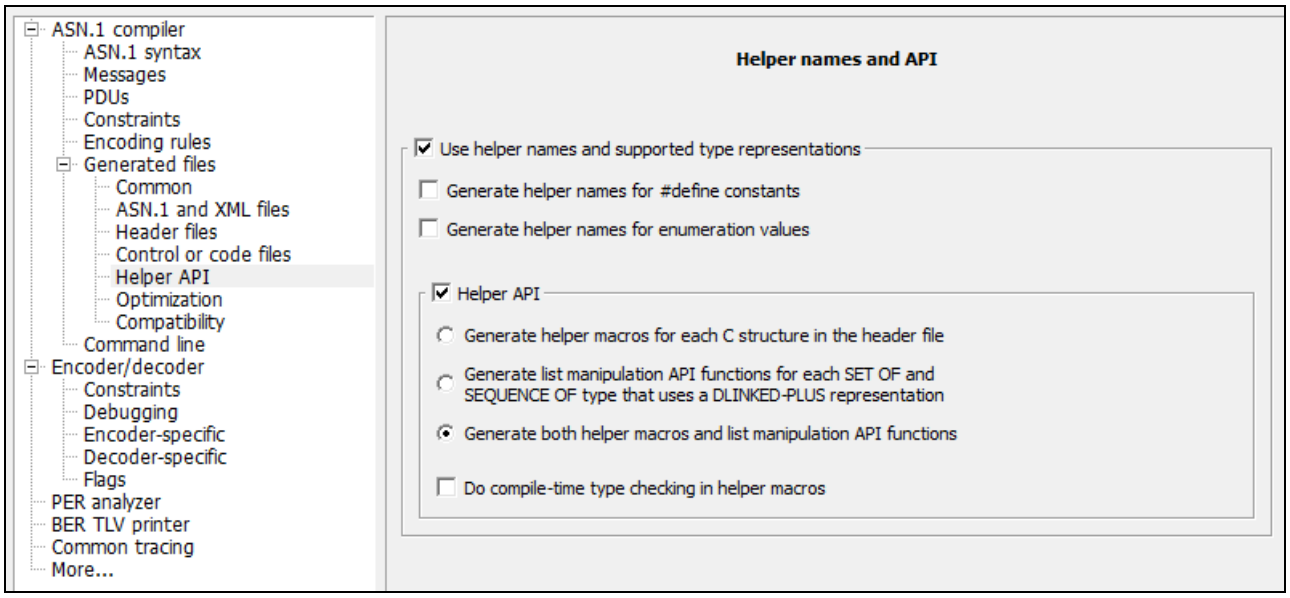
Generated files / Control or code files



Options on this page:

`-externalName` (see 3.3.27)
`-valueRefs/-noValueRefs` (see 3.3.77)
`-noStaticValues` (see 3.3.50)
`-exportDllData` (see 3.3.25)
`-exportDllAPI` (see 3.3.24)
`-test` (see 3.3.71)
`-sampleCode/-noSampleCode` (see 3.3.62)

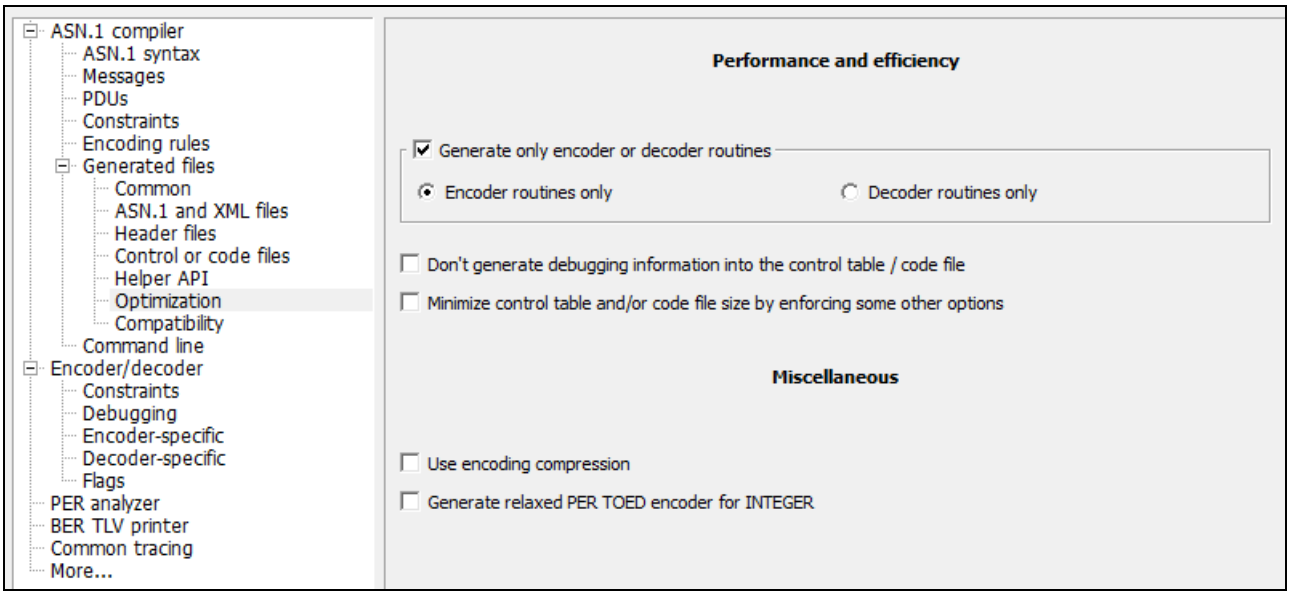
Generated files / Helper API



Options on this page:

`-helperNames/-noHelperNames` (see 3.3.37)
`-helperDefineNames` (see 3.3.33)
`-helperEnumNames` (see 3.3.34)
`-helperMacros/-noHelperMacros` (see 3.3.36)
`-helperListAPI/-noHelperListAPI` (see 3.3.35)
`-helperAPI/-noHelperAPI` (see 3.3.32)
`-hdebug` (see 3.3.29)

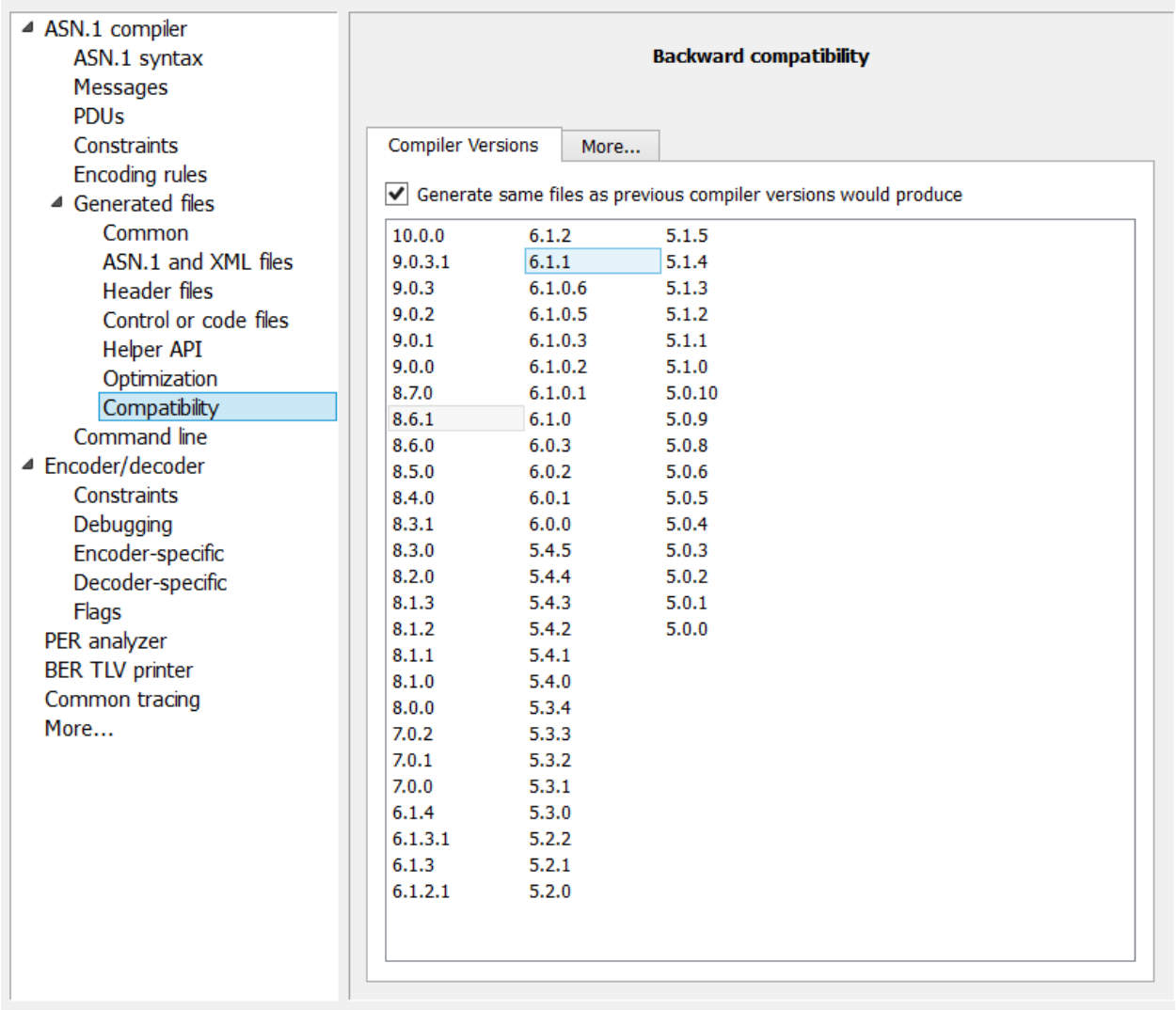
Generated files / Optimization



Options on this page:

`-encodeOnly/-decodeOnly` (see 3.3.18)
`-debug/-noDebug` (see 3.3.17)
`-minimize` (see 3.3.47)
`-compress` (see 3.3.13)
`-relaxPerToedIntegerRangeConstraint` (see 3.3.57)

Generated files / Compatibility



Options on this page:

`-compat` (see 3.3.12, 3.5)

Additional compiler options can be specified directly using the **Additional options** text field on the **Command line** page of a dialog. We do not recommend that you use this field to specify any compiler options that are represented by individual controls on the pages shown above.

The OSS ASN.1 compiler command-line options are listed on the following pages in alphabetical and numerical order.

3.3.1 -1990 / -2008

These options specify the version of ASN.1 (**ASN.1:1990** or **ASN1:2008**) used in the input ASN.1 specification file(s).

The `-1990` option specifies that the ASN.1 specification contains only the 1990 syntax. The compiler provides full support for the 1990 version of ASN.1, including support for the macro notation, ANY DEFINED BY and named types that have no identifiers.

The `-2008` option (added in version 9.0) specifies that the ASN.1 specification contains 2008 syntax. (The `-1993`, `-1994`, `-1997`, `-2002` compiler options are available as equivalents of `-2008`.) The compiler provides full support for the 2008 version of ASN.1, including automatic tagging, information object class, component relation constraints, table constraints, user-defined constraints, parameterization, version brackets, the enhanced capabilities of the subtype notation, TIME, OID-IRI and RELATIVE-OID-IRI types.

By default, the compiler starts out in a neutral mode expecting either the 1990 or 2008 version. Upon detecting notation that is unique to **ASN.1:1990** or **ASN.1:2008**, the compiler latches to that version and issues messages for any violation of that version. However, allowing the compiler to start in neutral mode can sometimes lead to a longer ASN.1 specification debugging cycle (especially when creating new ASN.1 specifications).

The ASN.1 compiler allows the mixing of the 1990 and 2008 syntaxes, but it warns of such usage since this is not allowed by X.680 (2008) ISO/IEC 8824-1:2008. When ASN.1:2008 syntax is used, the compiler complains about the following:

- all components of a SET, SEQUENCE or CHOICE that do not have an identifier,
- a colon (`:`) not included in the open type value or CHOICE value,
- a valuereference appearing anywhere in front of the keyword DEFINITIONS that signals the start of an ASN.1 module. The compiler does not allow a valuereference (defined value) to appear as or within the object identifier that is used in defining an ASN.1 module.

3.3.2 -allow

The new `-allow <argument>` compiler option has been added. It should be used to retain the old (incorrect) compiler behavior specified by the parameter. Do not use it unless absolutely necessary.

`-allow <BadCase>`

This option retains the incorrect ASN.1 compiler behavior as regards the case specified by the argument to make it compatible with previous versions of the compiler.

The option takes a single operand and may be specified more than once if there is a need to specify multiple compatibility options. Example:

```
asn1 foo.asn -allow BadValues -allow BadSerialFROM
```

The arguments that can be specified with the -allow option are discussed below.

`-allow BadValues`

This option is implied via the default `-relaxedMode` option.

This option instructs the ASN.1 compiler not to abort the current compile upon finding a content error in the value notation of restricted character strings (i.e. NumericString, PrintableString, and VisibleString) the OBJECT IDENTIFIER type, and extensible types with missing mandatory extension additions. For example, the following value notation will cause the ASN.1 compiler to exit with an error message, unless the `-allow BadValues` compiler option is specified:

```
myPhoneNumber NumericString ::= “(800) 555-1212 EXT-6”
myObjectID OBJECT IDENTIFIER ::= {0 26}
```

Note that according to the ASN.1 standard, the NumericString type only allows numbers (0-9) and white space in its contents while the OBJECT IDENTIFIER type does not allow a value larger than 4 for its second node[2].

Also, if the `-allow BadValues` option is present, the compiler does not truncate values of

- Restricted string, BIT STRING, and OCTET STRING types with extensible size constraints when the UNBOUNDED or NULLTERM representation is used.
- SEQUENCE OF and SET OF types, unless the ARRAY representation is used.

You may also instruct the ASN.1 compiler to treat such errors as mere warnings by using the `-ignoreError` compiler option (see section 3.3.38).

This option also instructs the ASN.1 compiler to issue the warning message A1007W, instead of the error message A1006E in some cases of type-value mismatch in the assignments of values of ENUMERATION, SET, SEQUENCE or CHOICE types. For ENUMERATION types, the option allows a difference in the named values set. For SET, SEQUENCE and CHOICE types, it allows a difference in the tags of the fields and allows a difference in position of the extension marker.

`-allow BadSerialFROM`

This retains the ASN.1 compiler behavior prior to 7.0 as regards handling of a serial application of a permitted alphabet constraint to a type: where only the last such constraint is taken into account when an effective permitted alphabet for the type is calculated.

Example:

```
M DEFINITIONS ::= BEGIN
    S ::= PrintableString (FROM("ABC"))  (FROM("DEF"))
END
```

[2] Note: if you specify a value greater than 39 for the second node of an OBJECT IDENTIFIER value and the first node is 0 or 1, an undecipherable encoding will result. You are responsible for any interoperability problems which occur due to use of the `-allow BadValues` option.

Since version 7.0, the compiler normally issues an error for the example because the permitted alphabet is empty. However, if `-allow BadSerialFROM` is specified no error is issued and the alphabet "DEF" is used for S.

`-allow Bad6.0Extensions`

This instructs the ASN.1 compiler not to report a tag conflict in some cases of untagged extensible CHOICE inside an extensible SEQUENCE.

Example:

```
M DEFINITIONS EXPLICIT TAGS ::= BEGIN
    S ::= SEQUENCE {
        a INTEGER,
        c C OPTIONAL,
        ...,
        e BMPString,
        ...
    }

    C ::= CHOICE {
        c1 [0] INTEGER,
        c2 [1] IA5String,
        ...
    }
END
```

If `-allow Bad6.0Extensions` is not specified the compiler reports the error:

```
"test.asn", line 4 (M): A0554E: Extension marks conflict in type S:
extension marker and element 'c' (line 4).
```

`-allow MixedCaseForSomeBuiltInTypesNames`

This option is implied via the default `-relaxedMode` option.

This option instructs the compiler to accept mixed case names for some ASN.1 built-in types, if types with such names are not already defined in the ASN.1 input. Currently this option supports mixed case names for the following built-in types:

- INTEGER without named numbers
- BOOLEAN
- NULL
- The restricted character string types (UTF8String, NumericString, PrintableString, TeletexString, T61String, VideotexString, IA5String, GraphicString, VisibleString, ISO646String, GeneralString, UniversalString and BMPString)
- The time types (TIME, DATE, TIME-OF-DAY, DATE-TIME and DURATION)
- GeneralizedTime and UTCTime
- OID-IRI and RELATIVE-OID-IRI
- ObjectDescriptor

Example:

```
M DEFINITIONS ::=  BEGIN
 Type ::= IA5STRING
 value Type ::= "abc"
END
```

Normally this is invalid ASN.1 syntax since 'IA5STRING' is used but is not defined.

If `-allow MixedCaseForSomeBuiltInTypesNames` is not specified the compiler reports the following errors:

```
"test.asn", line 3 (M): A0256W: 'IA5STRING' is referenced, but is
not defined.

"test.asn", line 3 (M): A0052E: 'IA5STRING' is not defined.
```

The `-allow MixedCaseForSomeBuiltInTypesNames` option allows you to compile such an ASN.1 syntax; the compiler will assume that the 'IA5STRING' is a misspelling of the built-in type 'IA5String':

```
"test.asn", line 3 (M): A1277W: 'IA5STRING' is assumed to be a
misspelling of the built-in type 'IA5String'.
```

`-allow UnderscoresInAsn1Names`

This option is implied via the default `-relaxedMode` option.

This option instructs the ASN.1 compiler to accept ASN.1 names that contain the underscore character ('_'), which is normally prohibited. When this option is specified, the ASN.1 compiler silently accepts any number of underscores in any part of an ASN.1 name (modulereference, typereference, valuereference, identifier, etc.) except for its initial character.

Example:

```
M DEFINITIONS ::=
BEGIN
  Some_Type ::=INTEGER
END
```

If `-allow UnderscoresInAsn1Names` is not in effect the compiler reports the following error:

```
"test.asn", line 3 (M): A1271E: The low line character ('_') is not
among the permitted characters for ASN.1 names. Please use the
'-allow UnderscoresInAsn1Names' command line option to force the
compiler to accept such invalid names.
     Some_Type ::= INTEGER
         ^

"test.asn", line 3 (M): A0120E: Parsing error: expecting '::=', '{'
or an ASN.1 Type but found illegal character.
     Some_Type ::= INTEGER
         ^
```

`-allow universaltags`

The option instructs the ASN.1 compiler to accept tags of the UNIVERSAL tag class, which are normally prohibited.

The use of the UNIVERSAL tag class is allowed for
a) IMPLICIT tags whose number fully corresponds to a builtin type tag;
b) CHARACTER STRING and OCTET STRING types with a specified tag number corresponding to one of the restricted character string types.

Otherwise, a UNIVERSAL tag is rejected.

Using the UNIVERSAL tag for the CHARACTER STRING and OCTET STRING types results in redefinition of the type by the compiler to some restricted character string type according to its tag number.

For example, the type `CS-U8` in the below specification is treated as a UTF8String and the following warning message is issued:

```
"A1163W: OSS has relaxed the standards to allow the definition
of a type with the tag [UNIVERSAL 12] resulting in
'CHARACTER STRING' to be redefined to 'UTF8String'.
This is normally an invalid ASN.1."

M DEFINITIONS IMPLICIT TAGS ::= BEGIN
        B-ignore ::= [UNIVERSAL 1] IMPLICIT BOOLEAN

        CS-ignore ::= [UNIVERSAL 29] CHARACTER STRING
        OS-ignore ::= [UNIVERSAL 4] OCTET STRING

        GT-ignore ::= [UNIVERSAL 24] GeneralizedTime
        UTCT-ignore ::= [UNIVERSAL 23] UTCTime
        U8-ignore ::= [UNIVERSAL 12] UTF8String
        BMP-ignore ::= [UNIVERSAL 30] BMPString

        CS-U8 ::= [UNIVERSAL 12] CHARACTER STRING
        OS-U8 ::= [UNIVERSAL 12] OCTET STRING

        CS-BMP ::= [UNIVERSAL 30] CHARACTER STRING
        OS-BMP ::= [UNIVERSAL 30] OCTET STRING
END
```

By default the option is enabled in trial versions of the ASN.1 compiler.

3.3.3 -ANSI

The `-ANSI` option specifies that the C compiler being used is fully ANSI compliant. The -ANSI option is always assumed while running the ASN.1 compiler.

By default, the `-ANSI` option is used.

3.3.4 -assignments

The `-assignment` option (added in version 6.0) instructs the ASN.1 compiler to process type and value assignments found outside of defined input modules.

When the `-assignments` option is specified, you may input to the ASN.1 compiler syntaxes such as the following:

```
Module1 DEFINITIONS ::= BEGIN
   MyInt ::= INTEGER
END

ExtraModularType ::= INTEGER

Module2 DEFINITIONS ::= BEGIN
   MyBit ::= INTEGER
END
```

Normally, the ASN.1 compiler will issue an error message upon finding a type or value assignment (such as `ExtraModularType` above) outside of a containing BEGIN-END ASN.1 module container.

By default, this option is turned off.

3.3.5 -autoEncDec

The `-autoEncDec` option allows open types to be automatically encoded/decoded even if runtime constraint checking is disabled. [Encoding/decoding while constraint checking is disabled is faster and requires fewer system resources than when constraint checking is enabled.]

If the `-autoEncDec` option is not specified, the automatic encoding/decoding of open types will not be permitted when constraint checking is turned off (e.g., when the `-noConstraints` option is specified or the OSS.NoConstrain directive is used) even if the `AUTOMATIC_ENCDEC` runtime flag is specified (see the `ossSetFlags()` function in the **OSS ASN.1/C API Reference Manual**).

Also note that if you specify this option, you will be able to automatically encode open types that do not have any component relation constraint defined for them. However, automatically decoding such ambiguous open types is not physically possible.

Note: Starting with version 5.3.0, the time-optimized encoder decoder also supports the automatic encoding/decoding of open types. To enable this support, you must ASN.1-compile your input specification with both the `-autoEncDec` and `-codeFile` options issued. Additionally, you must set the `AUTOMATIC_ENCDEC` flag (e.g., via `ossSetFlags()`) at runtime.

By default, the `-autoEncDec` option is turned off.

Related options:

`-constraints` / `-noConstraints`

3.3.6 -ber / -der / -cer / -per / -uper / -xer / -cxer / -exer / -oer / -coer

These options specify the encoding rules available at run-time. The OSS ASN.1 Tools supports BER (`-ber`), DER (`-der`), CER (`-cer`, added in version 6.0), ALIGNED PER (`-per`), UNALIGNED PER (`-uper`), Basic XER (`-xer`, added in version 6.0), Canonical XER (`-cxer`, added in version 6.0) encoding rules, Extended XER (`-exer`, added in version 7.0), Octet (`-oer`, added in version 10.0) encoding rules, and Canonical Octet (`-coer`, added in version 10.1) encoding rules. At the current time, only the Space-Optimized Encoder/Decoder (SOED) supports CER.

These options have the following precedence: (1) `-ber`, (2) `-per`, (3) `-der`, (4) `-uper`, (5) `-xer`, (6) `-cxer`, (7) `-cer`, (8) `–exer`, (9) `–oer`, and (10) `–coer`. So, if more than one option is specified, the one that is first in this list will be used by default. For example, if you specify `-der -per -xer`, the PER encoding rules will be used by default, though you will be able to switch to the DER and XER encoding rules at runtime.

To switch to a different set of available encoding rules at runtime, you can simply call the `ossSetEncodingRules()` as shown by the code excerpt below:

```
ossSetEncodingRules(world, OSS_DER);    /* switch to DER */
```

For more details on the function `ossSetEncodingRules()`, see **The OSS ASN.1/C API Reference Manual**.

The specific encoding rules available at runtime when one of these command-line options is specified varies according to the encoder/decoder library in use. This is summarized in the table below:

Option / Runtime	-ber	-per	-der	-uper	-xer	-cxer	-cer	-exer	-oer	-coer
SOED	BER	PER UPER	DER BER	UPER PER	XER	XER CXER	CER BER	E-XER	OER	C-OER OER
TOED	BER	PER	DER BER	UPER	XER	CXER	N/A	E-XER	OER	C-OER
LED	BER	PER UPER	DER BER	UPER PER	XER	XER CXER	N/A	E-XER	OER	OER C-OER

The `-ber` option specifies that the Basic Encoding Rules should be available during runtime.
The `-der` option specifies that the Distinguished Encoding Rules should be available during runtime.
The `-cer` option (added in version 6.0) specifies that the Canonical Encoding Rules should be available during runtime.
The `-per` option specifies that the Packed Encoding Rules (aligned version) should be available during runtime.
The `-uper` option specifies that the Packed Encoding Rules (unaligned version) should be available during runtime.
The `-xer` option (added in version 6.0) specifies that the XML Encoding Rules (basic variant) should be available during runtime.
The `-cxer` option (added in version 6.0) specifies that the XML Encoding Rules (canonical variant) should be available during runtime.
The `-exer` option (added in version 7.0) specifies that the Extended XML Encoding Rules should be available during runtime.
The `-oer` option (added in version 10.0) specifies that the Octet Encoding Rules should be available during runtime.
The `-coer` option (added in version 10.1) specifies that the Canonical Octet Encoding Rules should be available during runtime.

When the `-xer`, `-cxer` or `-exer` option is specified, the `-debug` option is also forced to be on. The `-noDebug` option along with the `-xer`, `-cxer` or `-exer` option has no effect. Also note that the header file generated when these options are specified is identical (below the initial comments) to the one generated without these options.

Note: If you want to be able to switch between all ten encoding rules: (1) BER, (2) CER, (3) DER, (4) ALIGNED PER, (5) UNALIGNED PER, (6) Basic XER, (7) Canonical XER, (8) Extended XER, (9) OER, and (10) C-OER at run-time, you should specify all of the relevant options together (e.g., `-ber`, `-cer`, `-der`, `-per`, `-uper`, `-xer`, `-cxer`, `-exer`, `-oer`, and `-coer`). Then you can use the `ossSetEncodingRules()` function to switch to one of these encoding rules before calling either the encoder or decoder.

By default the `-ber` option is implied. If the `-ber` option is specified or `implied` (default) and you call the encoder or decoder without first calling `ossSetEncodingRules()`, the encoding rules used will be BER. You can set any of the encoding rules to be the default by specifying only one of the `-ber, -cer, -per, -der, -uper, -xer, -cxer, –exer, –oer,` or `–coer` options.

3.3.7 -C / -C++

These options specify whether the header files and control-table/code-file generated are intended for inclusion in a C or C++ application. You should C-compile the generated files when the `-C` option is in use (`-C` is specified by default) and C++-compile the produced files when the `-C++` option is in use.

The generated C and C++ header files and `.c` files are similar, but constructs that are not valid in both languages are modified to avoid C/C++ compiler errors. For example if `class` is used as an identifier in the ASN.1 input, it is altered before being written to the header file if `-C++` is specified, since it is a C++ reserved word, but it is not altered for `-C` since it is not a C reserved word.

Note: For brevity, no other references are made to C++ in this manual. Though there may be differences in the generated header files, these differences are simple enough that they do not warrant special discussion.

Also the -c++ compiler option changes the default suffix of the generated codefile (control file) to `.cpp` (see `-soedFile/-toedFile/-output` options).

Note that the following list of reserved words is used by the ASN.1 compiler, regardless of whether C is the target language or C++:

```
auto, break, case, char, const, continue, default, do, double, else, enum,
extern, float, for, goto, if, int, long, main, register, return, short,
signed, sizeof, static, struct, switch, typedef, union, unsigned, void,
volatile, while
```

However, the following are also considered reserved words when `-C++` is specified:

```
asm, auto, bool, catch, class, const_cast, delete, dynamic_cast, explicit,
false, friend, inline, mutable, naked, namespace, new, operator, private,
protected, public, reinterpret_cast, static_cast, template, this, thread,
throw, true, try, typeid, typename, using, uuid, virtual, wchar_t, xalloc
```

Note that other steps besides reserved word conflict-resolution are also taken to make the output files compatible with a C/C++ compiler (depending on whether `-C` or `-C++` was specified).

By default, the `-C` option is implied.

Related options:

`-dualHeader`

3.3.8 -charIntegers

This option instructs the ASN.1 compiler to generate an `unsigned char` or a `signed char` for an INTEGER type if it is constrained to a range of values that is small enough to fit into a single byte. Table 3.1 shows the representation used for various ASN.1 types when the `-charIntegers` compiler option is specified.

ASN.1 Type	C Representation
A ::= INTEGER (0..7)	`typedef unsigned char A;`
B ::= INTEGER (0..255)	`typedef unsigned char B;`
C ::= INTEGER (0..256)	`typedef unsigned short C;`
D ::= INTEGER (-100..100)	`typedef signed char D;`
E ::= INTEGER (-1..127)	`typedef signed char E;`
F ::= INTEGER (-1..128)	`typedef short F;`

Table 3.1: - Some C representations of constrained INTEGER ASN.1 types

By default, this option is turned off.

3.3.9 -codeFile ***<CFileName>*** / -noCodefile

This option informs the ASN.1 compiler whether or not it should generate an output C file containing the time-optimized encoder/decoder.

The `-codeFile` keyword can be optionally followed by a name for the output C file. If no name is specified and the `-output` keyword is not used, the output C filename will default to the last command-line input filename, with a suffix of `.c` (or '`c`' on the Tandem NonStop OS) appended.

Note that use of this option affects the default for other compiler options such as `-constraints`.

The `-noCodeFile` option specifies that an output C file should not be generated. This option is useful when you want only a header file (and no `C` file) to be generated by the ASN.1 compiler.

By default, the `-codeFile` option is implied.

Note: The `-codeFile` and `-controlFile` options are mutually exclusive.

Related options:

`-controlFile, -soedFile, -toedFile`

3.3.10 -commandFile ***commandFileName***

This option specifies the name of the text-file containing command-line options for the current compile.

This is especially useful on systems (e.g., MS-DOS) that do not permit long command lines or do not support command-line continuation characters. The content of the specified file is treated as if it were put on the command line. For example, to compile the file, `input.asn`, using the command-line options contained in the text file `tenLinesOfOptions.opt`, we would issue the command:

```
asn1 input.asn -commandFile tenLinesOfOptions.opt
```

Command files may contain comments. The "--" characters specify the beginning of a comment. They can be placed anywhere on the line. The comment continues until either a subsequent "--" is reached or the line ends.

By default, this option is turned off.

3.3.11 -comments / -noComments

The `-comments` option specifies that ASN.1 comments specified in the input ASN.1 notation should be transferred to the generated header file using valid C/C++ comment constructs. Refer to section 7.10 for more details about this topic.

Related options:

`-CStyleComments`

3.3.12 -compat ***VersionNumber* or *Option***

This option specifies the generation of header and `.c` files compatible with those generated by previous versions of the OSS ASN.1 compiler.

This option takes a single operand or flag, and can be specified more than once to specify multiple compatibility options, if needed. Example:

```
asn1 foo.asn -compat v3.6 -compat noUInt
```

The arguments that can be specified with the `-compat` option are discussed in section 3.5. As of version 6.1.4, the `-compat` option now accepts any valid version number, not just the ones listed in section 3.5.

Note: OSS does not guarantee compatibility between different versions of the ASN.1 Tools released more than 10 years apart.

By default, compiler-generated files are compatible with the most recent version of the ASN.1 compiler executable.

3.3.13 -compress

The `-compress` option (added in version 7.0) specifies that compression should be available during runtime. Compression is especially useful for text-based encoding rules such as XER/E-XER but can be used for all of the available encoding rules (e.g., BER/DER). By default, the OSS libraries use the common zlib/gzip algorithm for compression, but you can also substitute your own compression routines (i.e. via the `ossSetCompressDecompressFunctions()` and `ossGetCompressDecompressFunctions()` routines). You may refer to the **OSS API/C Runtime Manual** for more details on how to use compression.

During runtime, you will also need to specify the `USE_COMPRESSION` flag (e.g., with the `ossSetFlags()` function) before calling the encoder/decoder for the types for which compression is desired or expected.

By default, this option is turned off.

Notes:
1) Currently, only the Space-optimized encoder/decoder supports compression.
2) You may only employ compression when the default memory manager is in use.
3) The special handling of ASN.1 types marked with the OSS.NOCOPY or ASN1.DeferDecoding directives is disabled in the decoder because these directives require uncompressed encoded data to be available immediately after decoding.

3.3.14 -constraints / -noConstraints

These options instruct the compiler whether or not to generate information in the control table for runtime constraint checking.

The `-constraints` keyword specifies the generation of information into the control table for constraint checking at run-time. This is **the default** for the space-optimized (SOED) encoder/decoder.

The `-noConstraints` option specifies that no information should be generated into the control table for enforcing constraints at run-time; this results in a smaller control table and slightly faster execution. This is **the default** for the time-optimized encoder/decoder (TOED) and Lean encoder/decoder. Also note that when the `-noConstraints` option is used, user-defined constraint checking functions are not called.

If the `-constraints` option is specified or implied , you can disable run-time constraint checking by specifying the encoder/decoder flag `NOCONSTRAIN`.

Note: If neither the `-constraints` nor the `-noConstraints` ASN.1 compiler options are explicitly specified, the time-optimized decoder (TOED) checks to see if the encoded size of a size constrained UNBOUNDED string exceeds the upper bound and reports an error if so. The rationale for this checking is to prevent the allocation of a possibly huge amount of memory that corresponds to an incorrectly encoded length while avoiding the performance penalty of full constraint checking.

Related options and directives:
`-restrictedConstraintChecking` (see section 3.3.60),
`OSS.NoConstrain` (see section 4.4.2.28)

3.3.15 -controlFile ***<CFileName>*** / -noControlFile

This option informs the ASN.1 compiler to generate a control table for use by the space-optimized encoder/decoder.

The `-controlFile` option specifies the generation of the space-optimized encoder/decoder control table in the output C file. It may optionally be followed by an operand to explicitly name the output C file. If the operand is absent and the `-output` keyword is not used, the name of the output C file will default to the last command-line input filename, with a suffix of `.c` (or '`c`' on the Tandem NonStop OS) appended.

The `-noControlFile` option specifies that a control table C file should not be generated. This option is useful when you only want a header file (and no `C` file) to be generated by the ASN.1 compiler.

By default, both of these options are turned off. The `-codeFile` option is implied.

Note: The `-codeFile` and `-controlFile` options are mutually exclusive.

Related options:
`-codeFile, -soedFile, -toedFile`

3.3.16 -CStyleComments

The `-CStyleComments` option instructs the ASN.1 compiler to transfer input ASN.1 comments to the generated header file but using C-language comment tokens (i.e. `/*` *comment text* `*/`) instead of C++ comment tokens (i.e. `//` *comment*) for single line comments even when the `-c++` option is specified.

This option is for use along with the `-c++` option and useful for those that would like the ASN.1-compiler generated files to be compilable in both a C and C++ compiler.

Note that you may completely turn off the transferring of ASN.1 comments to the header file by specifying the `-compat noASN.1Comments` option (see section 3.5.35).

By default, single-line ASN.1 comments are transferred using C++-style comments (i.e. `//` *comment*) when the `-c++` option is in use.

Related options:
`-C/-C++, -comments/-nocomments`

3.3.17 -debug / -noDebug

This option allows/suppresses the generation of debugging information into the C file, which the encoder/decoder can access and print if it is in debug mode or if it encounters an error. This allows the encoder/decoder to produce more descriptive messages that include the names of the ASN.1 identifiers and DefinedTypes.

By default, the `-debug` option is implied unless the `-soed` and `-lean` options are specified (i.e., the generation is for the Lean encoder/decoder).

Notes:
1) When using the time-optimized encoder/decoder (`-codeFile`), you must specify the C compiler option `-DOSSDEBUG=2` to have the encoder produce more descriptive error messages.
2) If you specify the `-xer` option, the `-debug` option is forced to be on and `-noDebug` is ignored.

3.3.18 -decodeOnly / -encodeOnly

These options specify the generation of only decoding or encoding routines for the time-optimized encoder/decoder. This option has no effect when the space-optimized encoder/decoder is in use (and `-xer` / `-cxer` are not specified), since the information generated into the control table is used both for encoding and decoding. However if you specify the `-xer` (or `-cxer`) option, then you may also specify `-decodeOnly` / `-encodeOnly` and that will reduce the amount of space-optimized library code linked to your final application.

The `-decodeOnly` option specifies the generation of decoder routines only while the `-encodeOnly` option specifies the generation of encoder routines only.

By default, encoder and decoder routines are both generated.

3.3.19 -defines / -noDefines

The `-defines / -noDefines` options instruct the ASN.1 compiler whether or not to generate `#defined` constants into the header file. If the `-noDefines` option is specified, `#defined` constants are not generated and are replaced with `const` data types. For example if `-noDefines` is specified, the generated header will contain:

```
extern const unsigned int     MyASN1Type_PDU;
```

instead of:

```
#define MyASN1Type_PDU 1
```

The `-defines` option turns off the behavior specified by `-noDefines`.

By default, the `-defines` option is implied.

3.3.20 -designerWarnings

The `-designerWarnings` options instructs the ASN.1 compiler to generate extra warning messages for protocol designers writing ASN.1 specifications. Specifically, warning messages are issued if one of the following oddities in the ASN.1 notation are found:

- The `-per` or `-uper` option is specified, but the compiler finds a SET, SEQUENCE, or CHOICE type containing only a single component and no extension marker (i.e., ellipsis (`...`)). When using PER, this will make adding fields in the future impossible, and having a structured type with only one field is inefficient.
- The `-per` or `-uper` option is specified and there is more than one unreferenced type in the root module, which is the last module in the input. This can cause problems determining the PDU number when decoding, as PER encodings have no tag information. In this case, the application developer must explicitly pass the desired PDU number to the decoder.
- A defined object set is missing in a table constraint when the ObjectSetFromObject notation is used for dummy information object sets.
- An information object CLASS is defined using the WITH SYNTAX notation, but an optional element is not enclosed in square brackets ('`[]`')
- A contents constraint includes only the ENCODED BY clause without a containing type defined, and automatic encoding/decoding is enabled.
- A WITH COMPONENTS clause that has the OPTIONAL or ABSENT keyword applied to a type defined with a DEFAULT value is present in the input ASN.1 specification.

By default, the `-designerWarnings` option is turned off.

3.3.21 -dtd

The `-dtd` option allows you to generate multiple data type definitions (DTD) files one for each PDU. Normally, no separate DTD file is produced even when the XER encoding rules are requested.

The produced data definition files will have a filename extension of `.dtd` and a filename prefix identical to the name generated for the PDU that they correspond to. You can specify a different filename prefix and extension by using the OSS.DTD compiler directive (refer to section 4.4.2.12).

The `-output` <*output-pathname*> and `-prefix` <*prefix-for-identifiers*> options affect the name and location of the produced `.dtd` file. Additionally, the ASN1.Nickname directive also has effect on the name of the produced `.dtd` file.

If you want `ossEncode()` to generate a reference to your data definition file when producing its XML output for a particular PDU, you should call the `ossSetXmlDTD()` function prior to invoking the encoder.

By default, no XML data definition files are generated by the ASN.1 compiler

Related options and directives:

`-output` (see section 3.3.51), `-prefix` (see section 3.3.55),
OSS.DTD (see section 4.4.2.12), ASN1.Nickname (see section 4.4.1.4)

3.3.22 -dualHeader

The `-dualHeader` option instructs the ASN.1 compiler to generate one header file which can be included in both C and C++ applications. This is achieved by generating preprocessor directives:

```
#ifdef __cplusplus
   /* C++ code */
#else
   /* C code */
#endif
```

If you wish to use the C++ code instead of the code for C, you would simply specify the `-D__cplusplus` command-line option when C++-compiling your application.

Note however, that if you wish to compile the produced `.c` file in a C++ compiler, you should also specify the `-c++` option (see section 3.3.7) along with the `-dualHeader` option.

By default, the `-dualHeader` option is turned off.

Related options:

`-C/-C++`

3.3.23 -enablePartialDecode

The `–enablePartialDecode` option instructs the ASN.1 compiler to enable partial decoding in addition to standard decoding. Encoding is not affected. Partial decoding is only supported by the TOED runtime. That is, `-enablePartialDecode` cannot be used together with the `-soedFile` or `-controlFile` option.

The `-enablePartialDecode` option is not supported for use with CXER or E-XER.

By default, the `-enablePartialDecode` option is turned off.

Related options and functions:

`-partialDecodeOnly`, `-toedFile`, `ossPartialDecode()` (see the ASN.1/C Runtime API Reference Manual, section 3.5.48)

3.3.24 -errorFile ***errorFilename***

This option specifies the redirection of all ASN.1 compiler messages to a named error file. This is useful on platforms (such as the Tandem NonStop OS) which do not have a simple mechanism for redirecting `stdout` and `stderr` to a file.

If an error is detected on the command line prior to the occurrence of the `-errorFile` option, the message will be written to `stdout` instead of to the error file. Thus, it is advisable that the `-errorFile` option be the first option on the command line (as shown below) so that all messages resulting from errors on the command line are written to the error file:

```
asn1 -err myerrs.err foo.asn
```

A suffix of `.err` (or '`e`' on the Tandem NonStop OS) is appended to the *errorFilename* if it doesn't already contain an extension. If the *errorFilename* has an extension of `.asn`, it is replaced with `.err`; otherwise, the *errorFilename* will keep the extension that you specify. Note that existing error-files with the same name as specified (i.e. *errorFilename*) will be overwritten by a new invocation of the compiler.

By default, this option is turned off.

3.3.25 -exportDllAPI

The `-exportDllAPI` option was introduced in version 8.0. This option allows users of the OSS ASN.1 Tools for Windows to have helper list API functions generated by the ASN.1 compiler explicitly exported from a control table or codefile DLL. This allows such functions to be available to user's application during runtime.

When this option is specified, each API function generated by the ASN.1 compiler gets an additional qualifier preceding the function declaration, `OSS_EXPORT_DLL_DATA`. This qualifier is a macro that exports the function from a control table or codefile compiled as a DLL.

In order to build a control table or codefile as a DLL so that API functions marked by `OSS_EXPORT_DLL_DATA` are exported from it, no special preparations required. Refer to `samples/advanced/control_table` for the exact build sequence.

In order to link an application with such a control table or codefile and to use the functions exported, one should redefine `OSS_EXPORT_DLL_DATA` as

```
#define OSS_EXPORT_DLL_DATA __declspec(dllimport)
```

before including the OSS-generated header file into the application code (assuming Microsoft Visual C is in use). This automatically makes the functions in question visible to the application.

For example, if the `-exportDllAPI` option is specified for the following ASN.1 specification together with `-helperListAPI` option:

```
M DEFINITIONS ::= BEGIN

S ::= SEQUENCE OF BIT STRING

END
```

Then generated list API functions will look like below:

```
/* creates an empty list */
OSS_EXPORT_DLL_DATA struct S * DLL_ENTRY oss_S(OssGlobal *_world);

/* creates a list node for the given value of 'struct _BitStr' */
OSS_EXPORT_DLL_DATA S_node * DLL_ENTRY oss_S_node(OssGlobal *_world, struct _BitStr *
_value);

/* appends a value of 'struct _BitStr' to the end of the list */
OSS_EXPORT_DLL_DATA S_node * DLL_ENTRY oss_S_append(OssGlobal *_world, struct S *
_list, struct _BitStr * _value);
…
```

By default, this option is turned off and list API functions are not exported from DLL control table or codefile.

Related options:

`-helperListAPI`

3.3.26 -exportDllData

The `-exportDllData` option allows users of the OSS ASN.1 Tools for Windows to have initialized values generated by the ASN.1 compiler explicitly exported from a control table or codefile DLL. This allows such values to be available to the user's application during runtime.

For example, if the `-exportDllData` option is specified for the following ASN.1 specification :

```
TEST DEFINITIONS ::= BEGIN
    PsType ::= SEQUENCE {
                    letters PrintableString (SIZE(1..5))
    }
    docName PsType ::= {
                letters "abcde"
    }
END
```

the ASN.1 compiler will generate at the bottom of the output header file:

```
OSS_EXPORT_DLL_DATA  PsType docName;
```

where the macro `OSS_EXPORT_DLL_DATA` exports the named variable from a control table or codefile compiled as a DLL.

By default, this option is turned off.

3.3.27 -extendOpenType

This option specifies the generation of an extra field called `userfield` of type `void *` in the `OpenType` structure. This field is available for use by you, but is ignored by the encoder/decoder.

By default, this option is turned off.

3.3.28 -externalName ***controlTableExternalName***

This option specifies the name of the external variable to be used for referencing the space-optimized encoder/decoder control table or the time-optimized encoder/decoder entry points. This variable is the last one generated in the `.c` and `.h` files.

By default, the external variable name will be the same as the last input filename on the command line (minus the `.asn` suffix).

3.3.29 -genDirectives ***<genDirFilename>***

This option instructs the ASN.1 compiler to generate a `.gen` file that captures all the directives from the ASN.1 input syntax as well as the directives for names fabricated by the compiler (e.g., `_setof1`). For more information on the ASN.1 compiler directives, see chapter 4.

The optional operand (`genDirFilename`) specifies the name of the `.gen` file to which all directives are written. If the filename is not specified, the ASN.1 compiler generates a filename from the last ASN.1 input file specified on the command line by substituting the suffix `.asn` with `.gen`.

By default, this option is turned off.

Related options:

`-keepNames`

3.3.30 -hdebug

The `-hdebug` option was introduced in version 8.0. This option instructs the ASN.1 compiler to generate compile-time type checking diagnostics in the generated helper macros. It may be used by the application at the development stage to enable compile time type checking in order to detect incorrect usage of the helper macros. It is not recommended to use the `-hdebug` option in a production environment because it slightly decreases the application's performance.

When the `-hdebug` option is present, each macro taking a structure and modifying its components is generated differently. Essentially, an extra local variable that is a pointer to the structure is declared by the macro, and the first input parameter of the macro is assigned to this variable thus guaranteeing compile-time type checking of this parameter by C compiler.

For example, for the ASN.1 notation below compiled with the `-helperMacro` option but without `-hdebug`:

```
M DEFINITIONS ::= BEGIN

S ::= SEQUENCE { i INTEGER OPTIONAL }

END
```

The following 'set' macro for the field 'i' is generated:

```
/* sets "i" field value */
#define oss_S_i_set(outp, i_) \
    { \
       (outp)->bit_mask |= i_present; \
       (outp)->i = (i_); \
    }
```

When `-hdebug is specified`, the same macro is produced differently:

```
/* sets "i" field value */
#define oss_S_i_set(outp, i_) \
    { \
       S *out = outp; \
       (out)->bit_mask |= i_present; \
       (out)->i = (i_); \
    }
```

Thus any attempt to use this macro to initialize a variable of type different from S even if it occasionally has a field 'i' will be tracked by the C compiler and a warning will be given.

By default, this option is turned off.

Related options:

`-helperMacros`

3.3.31 -headerFile ***<headerFileName>*** / -noHeaderFile

These options specify whether or not a header file should be generated to represent the input ASN.1 specification in C.

The `-headerFile` keyword may optionally be followed by an operand to explicitly name the output header file. If the operand is absent and the `-output` keyword is not used, the output header filename will default to the last command-line input filename, with a suffix of `.h` appended.

By default, the `-headerFile` option is implied.

3.3.32 -help

Refer to section 3.4, Command-line help, for more details about the `-help` option.

By default, command-line help is not displayed when input files are specified.

3.3.33 -helperAPI / -noHelperAPI

These options were introduced in version 8.0. They turn on or off a set of helper-related options:

- `-helperListAPI` / `-noHelperListAPI`
- `-helperMacros` / `-noHelperMacros`
- `-helperNames` / `-noHelperNames`

If the `-helperAPI` option is specified and one of the negative options from the above list is specified (for example, `-noHelperListAPI`), the latter one takes precedence. Accordingly, in the presence of `-noHelperAPI`, it is possible to specify any positive option from the above list (say `-helperNames`).

By default, the `-noHelperAPI` option is implied.

Related options:

```
-helperMacros / -noHelperMacros
-helperListAPI / -noHelperListAPI
-helperNames / -noHelperNames
```

3.3.34 -helperDefineNames

The `-helperDefineNames` option was introduced in versions 8.1.2/8.2. This option instructs the ASN.1 compiler to use context-based (helper) naming conventions for generated bit-masks with the suffixes "`_present`" and "`_chosen`", as well as for named bits and numbers. The option is supported only in the helper mode, that is, when the `-helperNames` option (see 3.3.37) is explicitly specified or is implied. Note that any of the `-helperAPI`, `-helperMacros`, or `-helperListAPI` options implies the `-helperNames` option.

Note: the `-helperDefineNames` option does not change user-defined names that were assigned, using such directives as `OSS.FIELDNAME`, `OSS.DefineName`, or `ASN1.Nickname`, to a field, named bit, or named number for which a #define constant is generated.

For example, if you compile the following ASN.1 syntax with the `-helperDefineNames` and `-helperNames` options:

1. Context-based names for #define constants used in bit-masks with a "`_present`" suffix will be generated for the following notation:

```
S ::= SEQUENCE {
   a [1] INTEGER OPTIONAL,
   b [2] INTEGER OPTIONAL,
   c [3] INTEGER --<FIELDNAME "my_new_name">-- OPTIONAL
}
```

C structure:

```
typedef struct S {
    unsigned char   bit_mask;
#       define      S_a_present 0x80
#       define      S_b_present 0x40
#       define      my_new_name_present 0x20
    int             a;  /* optional; set in bit_mask S_a_present if
                                                          present */
    int             b;  /* optional; set in bit_mask S_b_present if
                                                         present */
    int             my_new_name;  /* optional; set in bit_mask
                                     * my_new_name_present if present */
} S;
```

2. Context-based names for #define constants used in bit-masks with a "_chosen" suffix will be generated for the following notation:

```
C ::= CHOICE {
   c BOOLEAN,
   d REAL
}
```

C structure:

```
typedef struct C {
    unsigned short  choice;
#       define      C_c_chosen 1
#       define      C_d_chosen 2
    union {
        ossBoolean      c;  /* to choose, set choice to C_c_chosen */
        double          d;  /* to choose, set choice to C_d_chosen */
    } u;
} C;
```

3. Context-based names for #define constants generated for named bits will be generated from the following notation:

```
B ::= BIT STRING {bit1(1), bit2(2)}
```

C structure:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef _BitStr B;
#define                     B_bit1 0x40
```

```
#define                     B_bit1_byte 0
#define                     B_bit2 0x20
#define                     B_bit2_byte 0
```

4. Context-based names for #define constants generated for named numbers will be generated from the following notation:

```
I ::= INTEGER {int1(1), int2(2)}
```

C code:

```
typedef int             I;
#define                     I_int1 1
#define                     I_int2 2
```

By default, the `-helperDefineNames` option is turned off.

Related options:

`-helperAPI / -noHelperAPI`
`-helperMacros / -noHelperMacros`
`-helperListAPI / -noHelperListAPI`
`-helperNames / -noHelperNames`
`-helperEnumNames / -noHelperEnumNames`

3.3.35 -helperEnumNames

The `-helperEnumNames` option was introduced in versions 8.1.2/8.2. This option instructs the ASN.1 compiler to use context-based (helper) naming conventions for generated enumerators. The option is supported only in the helper mode, that is, when the `-helperNames` option (see 3.3.37) is explicitly specified or is implied. Note that any of the `-helperAPI`, `-helperMacros`, or `-helperListAPI` options implies the `-helperNames` option.

Note: the `-helperEnumNames` option does not change any user-defined names that were assigned to an enumerator using the ASN1.Nickname directive.

For example, the following ASN.1 syntax is compiled with the `-helperEnumNames` and `-helperNames` options:

```
--<ASN1.Nickname Mod.Enum.en3 "my_new_name">--

Enum ::= ENUMERATED {en1, en2, en3}
```

C structure:

```
typedef enum Enum {
    Enum_en1 = 0,
    Enum_en2 = 1,
    my_new_name = 2
} Enum;
```

By default, the `-helperEnumNames` option is turned off.

Related options:

```
-helperAPI / -noHelperAPI
-helperMacros / -noHelperMacros
-helperListAPI / -noHelperListAPI
-helperNames / -noHelperNames
-helperDefineNames / -noHelperDefineNames
```

3.3.36 -helperListAPI / -noHelperListAPI

The `-helperListAPI/-noHelperListAPI` options were introduced in version 8.0. They instruct the ASN.1 compiler to generate or not to generate a set of list manipulation API functions for each `SET OF` / `SEQUENCE OF` type specified in the input ASN.1 syntax that is represented as a `DLINKED-PLUS` C structure. If the `-helperListAPI` option is specified, it automatically turns on the `-helperNames` option and any `OSS.NoHelperListAPI` directives specified in the input syntax are ignored. In order to generate list manipulation API functions for selected `SET OF` or `SEQUENCE OF` types, use the global and/or the type-specific `OSS.HelperListAPI` and `OSS.NoHelperListAPI` directives and do not use the command-line options.

Generated API functions automate creation of a new list type, and addition/removal of list elements. They hide internals such as the exact operations performed to maintain a doubly-linked list of nodes in a consistent state. Additionally, if a memory handle is installed by a call to `ossCreateMemoryHandle()`, generated functions allocate output memory for the value using this handle.

List API functions are produced only for types that have the `DLINKED-PLUS` C representation. They are not produced for types that have the `UNBOUNDED` representation. Apply a `DLINKED-PLUS` directive either locally to such types or globally to the entire ASN.1 syntax to be able to generate and use helper list API functions for such types. For more information about C representations in helper mode, refer to 3.3.37.

For example, for the following ASN.1 syntax:

```
M DEFINITIONS ::= BEGIN

Names ::= SEQUENCE OF IA5String

END
```

The compiler produces the below functions:

```
/* ************************************** */
/* Helper List API functions for 'Names' list */
/* ************************************** */

/* creates an empty list */
struct Names * DLL_ENTRY oss_Names(OssGlobal *_world);

/* creates a list node for the given value of 'char *' */
Names_node * DLL_ENTRY oss_Names_node(OssGlobal *_world, char * _value);
```

```
/* appends a value of 'char *' to the end of the list */
Names_node * DLL_ENTRY oss_Names_append(OssGlobal *_world, struct Names * _list, char
* _value);

/* adds a value of 'char *' to the beginning of the list */
Names_node * DLL_ENTRY oss_Names_prepend(OssGlobal *_world, struct Names * _list, char
* _value);

/* adds a value of 'char *' to the certain position in the list */
Names_node * DLL_ENTRY oss_Names_insert(OssGlobal *_world, struct Names * _list,
Names_node * _prev, char * _value);

/* unlinks and frees node of list, doesn't free the node value */
char * DLL_ENTRY oss_Names_unlink(OssGlobal *_world, struct Names * _list, Names_node
* _node);
```

Using these functions it is possible to programmatically create list values as shown below:

```
Names n = oss_Names(world);

oss_Names_append(world, n, OSS_STR("Mary"));
oss_Names_prepend(world, n, OSS_STR("John"));
oss_Names_append(world, n, OSS_STR("Susan"));
```

Here `OSS_STR` is a macro copying a string to the new memory (defined in the shippable OSS header files). The resulting C value corresponds to the following ASN.1 value:

```
n Names ::= {"John", "Mary", "Susan"}
```

By default, the `-noHelperListAPI` option is implied.

Related options:

`-exportDllAPI`
`-helperNames / -noHelperNames`

3.3.37 -helperMacros / -noHelperMacros

The `-helperMacros/-noHelperMacros` options were introduced in version 8.0. They instruct the ASN.1 compiler to generate or not to generate helper macros for each C structure generated in the header file. If the `-helperMacros` option is specified, it automatically turns on the `-helperNames` option and any `OSS.NoHelperMacro` directives specified in the input syntax are ignored. In order to generate helper macros for selected ASN.1 types, use the global and/or the type-specific `OSS.HelperMacro` or `OSS.NoHelperMacro` directives and do not use the command-line options.

Generated macros automate creation of C structure values, and automate accessing and modifying of those values. They hide internals such as the use of a bit mask field to indicate the presence of optional fields of a `SEQUENCE`, the use of a `CHOICE` index, memory allocation and input data copying. In particular, when a memory handle is installed by a call to the `ossCreateMemoryHandle()` API function, all generated macros allocate memory from that handle.

The following kinds of macros are produced:

- macro having a suffix `_new` or `_new_pdu` is produced for each ASN.1 type that is either a PDU or referenced by pointer by some other type. Such a macro allocates a new empty value for the type (in some cases it also allocates another block of memory to store the value for 'length-value pair' structures such as `SEQUENCE OF` with the `UNBOUNDED` representation);
- macro having a suffix `_copy` or `_copy_pdu` takes an input value and its length (for strings and arrays) and creates a complete copy of this value in newly allocated memory; the input data is not referenced by the resulting value but its copy is created;
- macro having a suffix `_copy_nullterm` does the same as the previous one, but it calculates the length of the data by calling a system `strlen()` function; thus there is no need to pass the length parameter to the macro;
- `_setv` macro is produced for each length-value structure. It does not allocate memory but only sets the `length` and `value` fields of a given structure to certain values. Thus, the structure references input data of the macro after its execution;
- `_setv_nullterm` macro does the same as above, but the data length is calculated by a call to `strlen()` C function;
- `_<identifier_name>_get` macro is produced for each field of a `SEQUENCE` or `SET` and for each `CHOICE` alternative; it returns a value of the given field or alternative;
- `_<identifier_name>_set` macro for `SEQUENCE` or `SET` sets a field value taking into account the possible need to change the optional bit mask to indicate the presence of this field; for `CHOICE`, it sets the given `CHOICE` alternative value and modifies the choice index in accordance;
- `_<identifier_name>_is_present` macro is produced for each optional or default field of a `SET` or `SEQUENCE`. For optional or default field having a bit in the "optional" bit mask; it checks whether this bit is present or not; for optional or default fields that are referenced via a pointer it checks whether the pointer is non-NULL;
- `_<identifier_name>_omit` macro is produced for each optional field of a `SET` or `SEQUENCE` having a bit in the optional bit mask; it 'turns off' this bit in a bit mask;
- `_<identifier_name>_default` macro is produced for each field of a `SET` or `SEQUENCE` having a default value; it indicates that the default value for the field should be used;
- `_which` macro for `CHOICE` indicates which alternative is present;
- `_copy_encoded` and `_set_encoded` macros for open types and types with contents constraints perform logically the same actions (copying the encoded data or setting a pointer to it) that `_copy` and `_setv` macros perform for length-value types;
- `_copy_decoded` and `_set_decoded` macros for open types and types with contents constraints provide a decoded value of a type within the open type or contents constraint, either by copying it (`ossCpyValue` API function is called from within the `_copy_decoded` macro) or by setting a reference to an existing type value (`_set_decoded` macro does this).

For example, if you compile the following ASN.1 syntax with `-helperMacros`:

```
M DEFINITIONS ::= BEGIN

BTreeNode ::= SEQUENCE {
    name    IA5String,
    left    BTreeNode OPTIONAL,
    right   BTreeNode OPTIONAL
} --<PDU>--

END
```

The following C typedefs and helper macros are produced:

Compiler options

```
typedef struct BTreeNode {
    char            *name;
    struct BTreeNode *left;  /* NULL for not present */
    struct BTreeNode *right;  /* NULL for not present */
} BTreeNode;

/* allocates memory for an empty instance of the sequence type */
#define oss_BTreeNode_new(world) \
    (BTreeNode *)ossGetInitializedMemory(world, sizeof(BTreeNode))

/* allocates memory for BTreeNode_PDU */
#define oss_BTreeNode_new_pdu(world) \
    oss_BTreeNode_new(world)

/* gets "name" field value */
#define oss_BTreeNode_name_get(inp) \
    (inp)->name

/* sets "name" field value */
#define oss_BTreeNode_name_set(outp, name_) \
    (outp)->name = (name_)

/* allocates memory for a string of given length */
#define oss_BTreeNode_name_new(world, length_) \
    oss__CharStr_new(world, length_)

/* allocates memory and returns a copy of an input string */
#define oss_BTreeNode_name_copy(world, value_) \
    oss__CharStr_copy(world, value_)

/* gets "left" field value */
#define oss_BTreeNode_left_get(inp) \
    (inp)->left

/* sets "left" field value */
#define oss_BTreeNode_left_set(outp, left_) \
    (outp)->left = (left_)

/* checks if "left" field value is present */
#define oss_BTreeNode_left_is_present(inp) \
    ((inp)->left != NULL)

/* indicates that "left" field value is absent */
#define oss_BTreeNode_left_omit(outp) \
    (outp)->left = NULL

/* Macros for "left" field operate with 'BTreeNode' type; you may use macros
 * after this type declaration to create its values */

/* gets "right" field value */
#define oss_BTreeNode_right_get(inp) \
    (inp)->right

/* sets "right" field value */
#define oss_BTreeNode_right_set(outp, right_) \
    (outp)->right = (right_)

/* checks if "right" field value is present */
#define oss_BTreeNode_right_is_present(inp) \
    ((inp)->right != NULL)

/* indicates that "right" field value is absent */
#define oss_BTreeNode_right_omit(outp) \
    (outp)->right = NULL
```

```
/* Macros for "right" field operate with 'BTreeNode' type; you may use macros
 * after this type declaration to create its values */
```

In order to create the following value:

```
tree BTreeNode ::= {
    name "Root",
    left {
      name "l",
      right {
          name "lr"
      }
    },
    right {
      name "r"
    }
}
```

one may use these macros as shown below:

```
    BTreeNode *tree, *l, *r, *lr;

    tree = oss_BTreeNode_new(world);
    oss_BTreeNode_name_set(tree, oss_BTreeNode_name_copy(world, "Root"));
    oss_BTreeNode_left_set(tree, l = oss_BTreeNode_new(world));
        oss_BTreeNode_name_set(l, oss_BTreeNode_name_copy(world, "l"));
        oss_BTreeNode_right_set(l, lr = oss_BTreeNode_new(world));
            oss_BTreeNode_name_set(lr,
                                   oss_BTreeNode_name_copy(world, "lr"));
    oss_BTreeNode_right_set(tree, r = oss_BTreeNode_new(world));
        oss_BTreeNode_name_set(r, oss_BTreeNode_name_copy(world, "r"));
```

(it is possible to use the generic `OSS_STR(value)` macro defined in the shippable OSS header files instead of a type-specific `oss_BTreeNode_name_copy(world, value)` macro if there is an OSS environment variable named `world` in current scope).

By default, the `-noHelperMacros` option is implied.

Related options:

`-helperNames / -noHelperNames`

3.3.38 -helperNames / -noHelperNames

The `-helperNames/-noHelperNames` options were introduced in version 8.0. The `-helperNames` option instructs the ASN.1 compiler to use context-based (helper) naming conventions for generated C structures derived from built-in ASN.1 types. Any of the options `-helperAPI, -helperMacros, -helperListAPI` implies the `-helperNames` option.

The `-helperNames` option implies the following changes in the header file:

a) A limited set of C representations is supported for ASN.1 types. For more information on helper mode limitations, see section 7.2.2. The new DLINKED-PLUS representation is used for SET OF and SEQUENCE OF types with structured and pointered elements by default. The

UNBOUNDED representation is used for SET OF and SEQUENCE OF types with simple non-pointered elements (INTEGER without the HUGE directive, REAL with the DOUBLE directive, BOOLEAN, NULL and ENUMERATED).

b) All C structures that are derived from built-in ASN.1 types are extracted and have compiler-fabricated names.

c) Compiler-fabricated names for all structures that are not derived from built-in simple types are created based on the position of the built-in type within a complex user-defined type. Built-in simple types that are represented by a structure have simple intuitive names like "`_BitStr`" or "`_BmpStr`". For more information on context-based naming conventions, see section 8.2.2.

d) All references to structures within other C structures are generated as pointers. Note that the POINTER directive is supported only for a limited set of simple ASN.1 types. For more information on helper mode limitations, see section 7.2.2.

For example, if you compile the following ASN.1 syntax with the `-helperNames` option:

```
Module DEFINITIONS ::= BEGIN
S ::= SEQUENCE {
    a BMPString,
    b CHOICE {
        c BOOLEAN,
        d INTEGER
    }
}
END
```

the compiler produces the following structures in the `.h` file:

```
typedef struct _BmpStr {
    unsigned int    length;
    unsigned short  *value;
} _BmpStr;

typedef struct S {
    struct _BmpStr  *a;
    struct S_b      *b;
} S;

typedef struct S_b {
    unsigned short  choice;
#       define      c_chosen 1
#       define      d_chosen 2
    union {
        ossBoolean      c;  /* to choose, set choice to c_chosen */
        int             d;  /* to choose, set choice to d_chosen */
    } u;
} S_b;
```

By default, the `-noHelperNames` option is implied.

Related options:

`-helperAPI / -noHelperAPI`

```
-helperMacros / -noHelperMacros
-helperListAPI / -noHelperListAPI
```

3.3.39 -ignoreError *<errorNumber>*

The `-ignoreError` option instructs the ASN.1 compiler to treat certain message numbers (specified by the *`errorNumber`* parameter) as a warning message instead of an error message. This allows application developers to generate a header file and a control-table/code-file without correcting their ASN.1 specification. However, applications developers should be aware that the values generated into the control-table/code-file are not valid according to the ASN.1 standard.

Currently, only the following message numbers can be ignored: `49`, `76`, `77`, `305`, `319`, `811`, `832`, `1078`, `1130`, `1162`, `1207` and `1211`.

By default, all the above errors are ignored via the `-relaxedMode` option.

Related options:

```
-allow BadValues
-relaxedMode
```

3.3.40 -ignoreIncompleteItems

This option is implied via the default `-relaxedMode` option.

The `-ignoreIncompleteItems` option was introduced in versions 8.1.2/8.2. This option instructs the ASN.1 compiler to treat some errors related to undefined ASN.1 types and values as warnings, and to ignore incomplete ASN.1 definitions that directly or indirectly reference undefined ASN.1 types and/or values. This allows application developers to generate a header file and a control-table/code-file without correcting their ASN.1 specification, when undefined and incomplete ASN.1 items are not used by the application.

Note that ASN.1 types that are referenced only by ignored incomplete types are still not considered PDUs by the compiler, unless the global or local PDU directive is applied.

In order to see which incomplete items were ignored by the compiler, you may use the `-verbose` option in addition to `-ignoreIncompleteItems`.

For example, the type reference "A" references the type "B1", which is not defined:

```
M DEFINITIONS ::= BEGIN
   A ::= SEQUENCE {
      a1 A1,
      a2 A2,
      b B1
   }
   A1 ::= INTEGER
   A2 ::= BOOLEAN
   B ::= REAL
```

```
END
```

ASN.1 compiling without `-ignoreIncompleteItems` results in the following errors:

```
"test.asn", line 7 (M): A0256W: 'B1' is referenced, but is not defined.

"test.asn", line 7 (M): A0052E: 'B1' is not defined.

C0043I: 1 error message, 1 warning message and 0 informatory messages
issued.
```

ASN.1 compiling with `-ignoreIncompleteItems` and `-verbose`:

```
"test.asn", line 7 (M): A0256W: 'B1' is referenced, but is not defined.

"test.asn", line 4 (M): C1215W: The definition of the type identified by
the absolute reference 'M.A' is incomplete and will be ignored.

C0285I: Global checking abstract syntax.

C0286I: Generating header file.

C0287I: Generating control file.

C0043I: 0 error messages, 2 warning messages and 5 informatory messages
issued.
```

The generated header file includes:

```
#define          B_PDU 1

typedef int             A1;

typedef ossBoolean      A2;

typedef double          B;
```

Note that only one PDU is generated for type "B", which is not referenced by any other type in the input ASN.1 specification. Types "A1" and "A2" are referenced in the ignored incomplete type "A".

Related options:
`-verbose, -relaxedMode`

3.3.41 -ignoreRedefinedAssignments

This option is implied via the default `-relaxedMode` option.

This option instructs the ASN.1 compiler to treat some errors related to incorrectly redefined ASN.1 types and values as warnings, and to ignore all duplicate definitions within the same ASN.1 module.

For example, the type reference "`Type`" is defined twice within the same ASN.1 module:

```
M DEFINITIONS ::= BEGIN
   Type ::= INTEGER
   Type ::= INTEGER
END
```

This is normally invalid ASN.1. ASN.1 compiling without `-ignoreRedefinedAssignments` results in the following errors:

```
"test.asn", line 5 (M): A0146E: Type 'Type' is illegally redefined.
     Type ::= INTEGER
              ^

C0043I: 1 error message, 0 warning messages and 0 informatory messages
issued.
```

ASN.1 compiling with `-ignoreRedefinedAssignments`:

```
"test.asn", line 5 (M): A1270W: 'Type' has already been defined on line 3.
Its
redefinition is being ignored.

C0043I: 0 error messages, 1 warning message and 0 informatory messages
issued.
```

By default, this option is implied via the `-relaxedMode` option.

Related options:
`-relaxedMode / -noRelaxedMode`

3.3.42 -ignoreSuppress

This option instructs the compiler to ignore any `OSS.SUPPRESS` (see section 4.4.2.45) directives that are encountered in the ASN.1 module definition. In other words, if the module definition contains an `OSS.SUPPRESS` directive, but the `-ignoreSuppress` option is specified, messages will not be suppressed.

By default, all messages identified by the `OSS.SUPPRESS` directive are suppressed.

3.3.43 -informatoryMessages / -noInformatoryMessages

These options specify whether or not compiler informatory messages should be generated.

By default, the `-noInformatoryMessages` option is implied via the default `-relaxedMode` option.

When either the `-relaxedMode` or `-noWarningMessages` option is specified, the `-noInformatoryMessages` option is implied.

When either the `-noRelaxedMode` or the `-warningMessages` is specified, the `-informatoryMessages` option is implied.

Note, the order in which the explicit `-informatoryMessages` / `-noInformatoryMessages`, `-warningMessages` / `-noWarningMessages` and `-relaxedMode` / `-noRelaxedMode` options appear on the command line is important. The rightmost option is enforced, which overrides any effects of the previous options.

Related options:

`-warningMessages/-noWarningMessages, -relaxedMode/-noRelaxedMode`

3.3.44 -keepNames

This option causes the generation of a `.gen` file containing directives to guard against changes in the `.h` file that can occur if the input changes.

The ASN.1 compiler may generate different names for a given type each time the ASN.1 syntax is compiled (causing you to change your code). To avoid this, use the `-keepNames` option to instruct the ASN.1 compiler to generate a directive in a `.gen` file for each directive found in the input files and for each name that could potentially change if the input changes. When the ASN.1 syntax needs recompiling, the `.gen` file is input to the ASN.1 compiler as the first input file on the command line (along with the files and command-line options previously specified). The compiler will then generate the same names that it previously generated, even if there are explicit identifier changes in the input. For example if the ASN.1 input initially had:

```
Rec  ::= SEQUENCE {
    a SEQUENCE OF SET {name IA5String, age INTEGER},
    b SEQUENCE OF SET {height REAL, weight INTEGER}
  }
```

and this was then changed to:

```
Rec  ::= SEQUENCE {
    b SEQUENCE OF SET {height REAL, weight INTEGER},
    a SEQUENCE OF SET {name IA5String, age INTEGER}
  }
```

the `-keepNames` option will insulate your program from the element-order change.

If the `-keepNames` is specified, but the `-genDirectives` is not specified, the compiler will treat the input as if the `-genDirectives` option without the optional *genDirFilename* parameter had been specified. In this case, the compiler will create the `.gen` file with a name obtained by changing the suffix of the last file on the command line to `.gen` (unless the `-output` option was specified). This `.gen` file will include all directives from the ASN.1 input stream and directives for each name that could change in a subsequent run of the ASN.1 compiler.

For example, for the following ASN.1 input:

```
Module DEFINITIONS ::= BEGIN
    Type ::= INTEGER {one(1), two(2), five(5)}
    S ::= SEQUENCE {
        flagColor  ENUMERATED {red, white, blue}
    }
END
```

the compiler will produce the following lines in `.gen` file when run with the `-keepNames` option:

```
Directives artificially generated for the names to be kept:

--<OSS.TYPENAME Module.Type "Type">--
--<ASN1.Nickname Module.Type.one one>--
--<ASN1.Nickname Module.Type.two two>--
--<ASN1.Nickname Module.Type.five five>--
--<OSS.TYPENAME Module.S "S">--
--<ASN1.Nickname Module.S.flagColor.red red>--
--<ASN1.Nickname Module.S.flagColor.white white>--
--<ASN1.Nickname Module.S.flagColor.blue blue>--
--<OSS.TYPENAME Module.Type "Type">--
```

By default, this option is turned off.

3.3.45 -lean

This option instructs the ASN.1 compiler to generate header files compatible with the Lean encoder/decoder (LED). Like the space-optimized encoder/decoder (SOED), the LED is table-driven; however, it is generally three times faster than the SOED. Additionally, the produced `.c` file is smaller than that of the SOED and much smaller than that of the TOED. Note that the TOED is slightly faster than the LED.

The LED library is not included in the OSS ASN.1 Tools; it is an add-on product that is available for common platforms like Linux, Windows, and Solaris, and may not be available for your embedded system port. If you are interested in LED for your platform, contact OSS Nokalva Sales at info@oss.com.

Specify the `-lean` option along with the `-controlFile` or `-soedFile` options on the ASN.1 compiler command line and link your application code with the LED library file (e.g., `libasn1lean.a`, `asn1lean.lib`, or `asn1lemd.lib`). Then, all calls to the `ossEncode()` and `ossDecode()` functions will result in the use of the optimized Lean encoder/decoder. Alternatively, you can still link the produced files with the space-optimized encoder/decoder library file; however, you must also specify the C-compiler option `-DOSS_SOED_ONLY_USED`.

Note: When `-lean` is specified, the `-noDebug` option is implied by default.

Since the LED is optimized, there are some runtime functions (see the *Restrictions* chapter of the **OSS ASN.1 API Reference Manual**) that are not supported. If you would like to use the LED while still having access to the unsupported functions, you should specify both the LED and SOED library files on your linker's command line; in such a case, you should place the LED library file before the SOED library file and specify the `-DOSS_USE_SOED_AND_LED` C-compiler option.

Using both LED and SOED: when you compile your ASN.1 with the `-lean` option, the generated C structure in the .h file will be incompatible with structures generated when `-lean` is not used. A problem could arise if you have two platforms, one with LED and another with SOED; you would have to write a different application program for each since the generated C structures are so different. The solution is to permit the SOED to also work with the `-lean` generated structures. This is done by compiling the ASN.1 with `-lean` and C-compiling the application with `-DOSS_USE_OSS_FULL_SOED_ONLY`.

Since the LED is optimized, not all C-representations for an ASN.1 type are permitted. Only the representations that ensure efficiency are allowed (see section 6.8).

Once you specify the `-lean` option on the ASN.1 compiler command, along with the `-codeFile` or `-toedFile` options the ASN.1 compiler shall generate an output C file containing the time-optimized encoder/decoder. The generated header file shall contain C-representations supported by the LED. Your application code should be linked with the TOED library in this case. **By default,** the `-lean` option is turned off.

Related options:

`-codeFile`, `-controlFile`, and `-debug` / `-noDebug`, `-soedFile`, `-toedFile`

3.3.46 -listingFile ***<listingFileName>*** / -noListingFile

These options specify whether or not to generate an ASN.1 composite listing of the root module(s), with all macro instances and parameterized items expanded, and all external references (imported items) resolved.

The `-listingFile` option may optionally be followed by an operand to explicitly name the output listing file. If the operand is absent and the `-output` keyword is not used, the output listing filename will default to the last command-line input filename, with a suffix of `.lst` appended.

These options are not available in trial versions of the OSS ASN.1 Tools compiler.

By default, the `-noListingFile` option is implied.

Note: If no error conditions are detected in the input, the listing file contains valid ASN.1, and can be used as input to any ASN.1 compiler.

Related options:

`-modListingFile`

3.3.47 -messageFormat ***<format-scheme>***

The `-messageFormat` option allows you to control the format of the error/warning/informatory messages used by the ASN.1 compiler. This option takes a mandatory parameter which specifies the format scheme to use.

Currently, three format schemes are available:

`oss` the standard OSS Nokalva message format scheme; this is the default message format scheme used by the ASN.1 compiler.

```
filename.asn, line:[(moduleName):]
followed by the error or warning message itself
```

Example:

```
msgfmt.asn, line 9 (Module-1): A0217W: The identifier
'address' is missing from the value and is assumed.
```

`emacs` a format scheme compatible with many text editors; This format scheme looks like:

```
filename.asn[:moduleName:]line:[position:] followed by the
error or warning message itself
```

`msvc` a format scheme compatible with the Microsoft development tools; this format scheme (introduced in version 7.0) looks like:

```
filename.asn(line[, position]) : [moduleName:] {error | warning}
followed by the error or warning message itself
```

Example:

```
msgfmt.asn (9:29) : Mod: warning A0217W: The identifier
'address' is missing from the value and is assumed.
```

You can feed the format schemes to a parser or compatible text editor which can then automatically find the location in the input file for which the error/warning/informatory message was issued.

By default, the `oss` message format schema is used.

3.3.48 -minimize

This option specifies the automatic selection of command-line options which results in the most compact control table. Normally, the following options are automatically selected when `-minimize` is specified: `-noConstraints`, `-noDebug`, and `-compat noValues`. However if the `-xer` / `-cxer` options are also specified, then the `-debug` (and not `-noDebug`) is used along with `-noConstraints` and `-compat noValues`.

By default, this option is turned off.

3.3.49 -modListingFile *<listingFilename>* / -noModListingFile

The `-modListingFile` option tells the compiler to produce a module listing file that contains information from multiple ASN.1 modules.

The `-modListingFile` option may optionally be followed by an operand to explicitly name the output listing file. If the operand is absent and the `-output` keyword is not used, the output listing filename will default to the last command-line input filename, with a suffix of `.lst` appended.

By default, the `-noModListingFile` option is implied.

Notes:
1) The module listing file will have macro definitions and INSTANCE OF uses expanded. However, parameterized items are not expanded.
2) This option is mostly used in diagnosing the compiler.
3) The resulting module listing file may contain invalid ASN.1.

Related options:
`-listingFile`

3.3.50 –noSignalHandler

This option specifies that the OSS Compile-and-Go-Library (CAGL) should not install signal handlers. With signal handling enabled, CAGL will trap signals and return an error code and message to you, rather than just crashing. However, there are disadvantages to trapping signals. In general, performance suffers somewhat. More importantly, signal trapping is not always thread-safe and it could conflict with third party signal handlers.

By default, this option is turned off.

Note: this option is intended for CAGL and not recognized by the OSS ASN.1 Compiler. The OSS Compile-and-Go-Library is not included in the OSS ASN.1 Tools; it is an add-on product that is available for common platforms like Linux, Windows, and Solaris, and may not be available for your embedded system port. If you are interested in CAGL for your platform, contact OSS Nokalva Sales at info@oss.com.

3.3.51 -noStaticValues

This option specifies that the ASN.1 compiler should not statically initialize any variable generated from the ASN.1 value notation. Instead, all such variables are to be initialized in the `ossinit()` function by

means of executable code. For more information on the function `ossinit()` function, see the **OSS ASN.1/C API Manual**. .

Suppose we have to compile the following abstract syntax:

```
A ::= INTEGER
a A ::= 42
```

If the `-noStaticValues` option is specified, an uninitialized external is generated as shown below:

```
A a;
```

and via the function `ossinit()` (which is called once during the initialization phase of your program) the following is executed:

```
a = 42;
```

If the `-noStaticValues` option is not specified, the ASN.1 compiler will generate the following statement into the output `.c` file:

```
A a = 42;
```

This option was added to support certain embedded system software products that do not work properly under some circumstances when extern variables are initialized by means of a C initializer.

By default, this option is turned off.

3.3.52 -output ***<outputFilename | outputDirectoryName>*** / -noOutput

This option specifies whether or not output files should be generated. This option may optionally be followed by a parameter to specify the name prefix of output files or an output directory.

The `outputDirectoryName` optional parameter specifies the directory where output files should be generated. The `outputfileName` optional parameter specifies the name prefix of the output filenames which are to be suffixed with `.c` (`.cpp`), `.h`, `.lst`, `.gen`, *et cetera*, as needed. If no `outputDirectoryName`/`outputfileName` parameter is specified, the prefix of the last `.asn` input file is used for the output filenames.

Examples:

```
asn1 foo.asn -output abc
```

If there is a subdirectory named `abc/` under the present working directory, the ASN.1 compiler will create a subdirectory named `abc/` and place the output files `foo.h` and `foo.c` in this subdirectory; otherwise, the ASN.1 compiler will generate output files with the names `abc.h` and `abc.c` in the present working directory.

```
asn1 foo -output xyz/abc
```

If the `xyz/` subdirectory does not exist, the ASN.1 Compiler will try to create this subdirectory. If the compiler is able to create the named subdirectory successfully, it will generate `abc.h` and `abc.c` files under this subdirectory.

```
asn1 foo -output xyz/abc/
```

Above command line will create the `abc/` directory within the `xyz/` subdirectory and place the output files within the `abc/` directory.

The `-noOutput` option specifies the suppression of all output other than error messages.

By default, the `-output` option with no parameter is implied.

3.3.53 -partialDecodeOnly

The `-partialDecodeOnly` option instructs the compiler to replace standard decoding with partial decoding. In this case, no standard encoding or decoding functions are generated, and constraint checking is disabled (`-noConstraint`). Partial decoding is only supported by the TOED runtime. That is, `-partialDecodeOnly` cannot be used together with the `-soedFile` or `-controlFile` option.

The `-partialDecodeOnly` option is not supported for use with CXER or E-XER.

By default, the `-partialDecodeOnly` option is turned off.

Related options and functions:

`-enablePartialDecode`, `-toedFile`, `ossPartialDecode()` (see the ASN.1/C Runtime API Reference Manual, section 3.5.48)

3.3.54 -pdusForContainingTypes / -noPdusForContainingTypes

The `-pdusForContainingTypes` option instructs the ASN.1 compiler to treat types referenced by contents constraints as PDUs, even if the global or local OSS.NOPDU directive is specified.

The `-noPdusForContainingTypes` option instructs the ASN.1 compiler to disable generation of PDU numbers for types referenced by contents constraints. In this case automatic encoding/decoding of BIT STRING and OCTET STRING types defined with contents constraints will not be performed at runtime.

By default, these two options are turned off.

3.3.55 -pdusForOpenTypes/ -noPdusForOpenTypes

These options instruct the ASN.1 compiler on whether or not to treat open types created by either the ASN1.DeferDecoding or OSS.ENCODABLE directive as PDUs

The `-pdusForOpenTypes` option causes open types created by either the `ASN1.DeferDecoding` or `OSS.ENCODABLE` directive to be treated as PDUs, even if the global or local `OSS.NOPDU` directive is specified. The option also causes open types created by the use of component relation constraints to be treated as PDUs, even if the global or local `OSS.NOPDU` directive is specified.

The `-noPdusForOpenTypes` option instructs the compiler not to generate PDU numbers for any OpenTypes. The option can help to prevent the ASN.1 compiler error:

```
C0923E: Implementation limit exceeded; too many fields in module.
```

for ASN.1 notations with a large number of PDU types.

The `-noPdusForOpenTypes` option can only be used together with the option `-noConstraints`, otherwise it will be ignored.

The `-autoEncDec` option overrides the effect of the `-noPdusForOpenTypes` option.

By default, both the `-pdusForOpenTypes` and `-noPdusForOpenTypes` option are turned off, and only unreferenced types may become PDUs.

3.3.56 -pedantic

This option instructs the compiler to thoroughly check the entire abstract syntax.

The `-pedantic` option specifies that the entire abstract syntax is to be thoroughly syntax checked. Normally the compiler does not consider it an error if a type is referenced but never defined, so long as it is neither directly nor indirectly referenced by the root module(s). The `-pedantic` option implies the `-syntaxOnly` option, so no `.h` or `.c` file is generated. Rather, only a syntax check is performed.

By default, this option is turned off.

3.3.57 -prefix ***<prefix>***

This option specifies one or more characters to prefix to all generated names whose scope is global in your C program file.

The `-prefix` option is useful when an application program supports multiple abstract syntaxes, and the same `typedef struct` tag or manifest constant name is used in the generated files. By adding a prefix to the generated names whose scope is global, the generated files can be C compiled without a multiple declaration error.

By default, names are not prefixed.

3.3.58 -relaxedMode/-norelaxedMode

These options specify the automatic selection of command-line options that results in relaxed or strict compiler behavior. The following options are automatically selected when `-relaxedMode` is specified:

```
-allow BadValues
-allow MixedCaseForSomeBuiltInTypesNames
-allow UnderscoresInAsn1Names
-ignoreError (for all ignorable errors)
-ignoreIncompleteItems
-ignoreRedefinedAssignments
-noWarning
```

Note that some significant warning messages are not suppressed when this option is used. To suppress all warnings, the explicit `-noWarningMessages` option should be specified.

The `-norelaxedMode` option cancels the options implied by `-relaxedMode`. In addition, the `-warningMessages` option is implied.

To enable warnings or informatory messages when `-relaxedMode` is specified, specify `-warningMessages` or `-informatoryMessages` respectively after `-relaxedMode` on the command line.

To disable warnings or informatory messages when `-noRelaxedMode` is specified, specify `-noWarningMessages` or `-noInformatoryMessages` respectively after `-noRelaxedMode` on the command line.

By default, the `-relaxedMode` option is turned on.

The `-relaxedMode` option was introduced in version 7.0, while the `-norelaxedMode` option was introduced in version 8.7.0. Starting with version 9.0, the `-relaxedMode` option is turned on by default.

Related options:

`-allow BadValues, -allow MixedCaseForSomeBuiltInTypesNames, -allowUnderscoresInAsn1Names, -ignoreError, -ignoreIncompleteItems, -ignoreRedefinedAssignments, -noInformatoryMessages, -noWarningMessages`

3.3.59 -relaxPerToedIntegerRangeConstraint

Starting with version 8.4, this option is now available to modify the default behavior of the TOED PER encoder. It applies only to PER encodings.

Standard behavior for the PER encoder is to enforce range constraints even when constraint checking is disabled. This may appear odd, but the reason is that with PER a definition such as:

```
Z ::= INTEGER (50..52)
```

cannot accommodate a value of 49 or 54. It is not simply that 49 and 54 are beyond the limit; it is that 49 and 54 cannot exist because there is no way to physically represent them. In other words, disabling

constraints loses meaning. In its zeal to be compact, PER reserves only enough bits to represent all states in the specified range:

```
1 bit can represent up to 2 states
2 bits can represent up to 4 states
3 bits can represent up to 8 states
...
n bits can represent up to 2^n states
```

Since Z deals with 3 states (50 through 52), we must use 2 bits to encode all possibilities, whereby the bits 00 would represent 50, 01 would represent 51, and 10 would represent 52. What about the bit pattern 11? It is physically possible to have 11 represent 53, but 53 is not allowed by the (50..52) constraint. However, it is impossible to represent anything smaller than 50 or greater than 53. There aren't enough bits in the field.

In summary:

Value	Bits Encoded	Allowed	Physically Possible
<50	??	NO	NO
50	00	YES	YES
51	01	YES	YES
52	10	YES	YES
53	11	NO	YES
>53	??	NO	NO

The `-relaxPerToedIntegerRangeConstraint` compiler option asks the compiler to generate TOED code so as to permit the user to deliberately force the encoding of a disallowed, but possible, value (e.g., 53 in the example above) so that he can test all possible bit patterns for the field in question. In effect, when -relaxPerToedIntegerRangeConstraint is used, if the number of possible INTEGERs is not already a power of 2, the upper bound is extended so that the number of permitted INTEGERs is the next power of 2.

Some additional examples:

```
Y ::= INTEGER (15..19)
-- 5 allowed values, thereby 3 bits are needed.
-- Permits 15 - 19 by default
-- Permits 15 - 22 (a range of 2^3 INTEGERs)
-- when -relaxPerToedIntegerRangeConstraint is in effect.

X ::= INTEGER (0..15)
-- 16 allowed values, thereby 4 bits are needed.
-- Permits 0 - 15 by default
-- Permits 0 - 15 (a range of 2^4 INTEGERs)
-- even when -relaxPerToedIntegerRangeConstraint is in effect.

W ::= INTEGER (-5..5)
-- 11 allowed values, thereby 4 bits are needed.
-- Permits -5 - +5 by default
-- Permits -5 - +10 (a range of 2^4 INTEGERs)
-- when -relaxPerToedIntegerRangeConstraint is in effect.
```

NOTE: If constraint checking is in effect (the `-constraint` compiler option is specified and the `NOCONSTRAIN` runtime flag is not set), the encoder ignores the `-relaxPerToedIntegerRangeConstraint` option.

By default, this option is turned off.

3.3.60 –relaySafe / -noRelaySafe

This option specifies the use of relay-safe decoding and re-encoding of messages with extension addition fields in SEQUENCE, SET, and CHOICE types.

When the `-relaySafe` option is not used, and a firewall supported version 1 of H.323 was used, and it later received a version 2 H.323 message that contained some added fields (not defined in version 1), the decoder would skip over these added fields and return only the version 1 fields to the firewall application. The problem with this is that if the firewall is to decode and re-encode the message with a new IP address, it would re-encode the message as a version 1 message since the version 2 fields were skipped over during decoding.

With relay-safe decoding and encoding, the decoder will retain the value of the fields added in version 2 and, when the message is re-encoded by the firewall application, the encoder will automatically insert the version 2 fields back into the newly encoded message, resulting in a version-2-compliant message with the new IP address - even though the firewall understands only version 1 messages and even though the firewall has no clue as to what is contained in the version 2 fields.

In summary, this relay-safe encoder/decoder behavior occurs only when the ASN.1 compiler flag, `-relaySafe`, is specified. If this flag is specified, then any of your source files that reference the ASN.1-compiler generated header file will have to be re-compiled. No source code modification is required to use this feature, just a re-compile. If `-relaySafe` is not used, the encoder/decoder will will silently ignore unknown extension addition values (i.e. it will behave the way that it did prior to version 5.1).

Notes:
1) When the `-relaySafe` option is specified, a special field to store unknown extensions is generated in the representation of the SEQUENCE, SET, and CHOICE types. You do not need to concern yourself of the details of how this special field works as the encoder/decoder accesses/modifies it automatically.
2) You may not re-encode the unknown extension fields with a set of encoding rules that differ from those that were used to encode them originally.

By default, this option is turned off.

3.3.61 -reservedWords ***<words>***

By default, the compiler applies a disambiguation algorithm to guarantee that all generated names do not conflict with words that are considered as reserved on most popular platforms. For example, such words as "double", "default", "const", and so on are reserved. If a conflict is detected, the compiler resolves the

conflict by changing the generated names to eliminate the conflict. This process of name disambiguation is called "name mangling".

C/C++ compilers on various platforms may have their own lists of reserved words that are not included in the list of reserved words automatically supported by the ASN.1 compiler. Starting from the 8.1.2/8.2 versions, the ASN.1 compiler supports the `-reservedWords` option.

Example:

```
asn1 fname.asn -reservedWords word1,word2,...
```

The `-reservedWords` option allows you to specify a list of words, separated by commas, that should be treated as reserved words by the compiler. As a result, any compiler-generated names matching those words will always be mangled. Depending on where such names appear, they are mangled either with the prefix derived from the ASN.1 module where the corresponding top-level ASN.1 item is defined, or with the name(s) of the outer component(s), if the conflicting name belongs to some component that appears inside those components.

Note: if a user-defined name was assigned using one of the OSS-specific directives `Nickname`, `TYPENAME`, or `FIELDNAME` and it matches the word specified in the `-reservedWords` option, the directive has a precedence. A user-defined name is never changed.

For example, the ASN.1 type `Enum` is defined as follows:

```
Enum ::= ENUMERATED {send, bind}
```

The compiler generates the following without the `-reservedWords` option:

```
typedef enum Enum {
    send = 0,
    bind = 1
} Enum;
```

The Microsoft C/C++ Optimizing Compiler considers such words as "bind" and "send" as reserved and C/C++ compiling of the above structure will fail on the Windows platform.

The `Enum` type generated with the "`-reservedWords send,bind`" option looks like the following. In this case, no C compiling error occurs.

```
typedef enum Enum {
    Enum_send = 0,
    Enum_bind = 1
} Enum;
```

By **default**, this option is turned off.

3.3.62 -restrictedConstraintChecking

This option instructs the compiler not to ignore the `OSS.NoConstrain` directive in the presence of the explicit `-constraints` option.

When the explicit `-constraints` option is specified along with the `-restrictedConstraintChecking` option, the ASN.1 compiler turns on runtime constraint checking for all types except ones that have the `OSS.NoConstrain` directive applied.

This option has no effect unless the `-constraints` option is explicitly specified.

By **default**, this option is turned off.

Related options and directives:
`-constraints / -noConstraints` (see section 3.3.14),
`OSS.NoConstrain` (see section 4.4.2.28)

3.3.63 -root

This option instructs the compiler to treat all input modules as if they were root modules. This means that all modules are thoroughly checked and all unreferenced types (defined types that are not referenced by any other type) in all modules are considered **PDU**s. For more information on root modules refer to the documentation of the OSS.ROOT directive (see section 4.4.2.39).

By default, only the last module specified on the command-line is treated as a root module.

Definition: *PDU - A protocol data unit is a message for transfer contained in an ASN.1-compiler-generated data structure. PDUs can be encoded/decoded as separate complete units. For each PDU, the OSS ASN.1 compiler* `#defines` *an identification tag in the header file. For example, the ASN.1 syntax:*

```
MyBoolean          ::= BOOLEAN
```

yields the following header-file definition:

```
#define            MyBoolean_PDU 1
```

Note that the ASN.1 compiler assigns the tags in sequential order. So, the first PDU encountered gets a tag of '1', the second a tag of '2', et cetera. However, you should always use the `#defined` *name (e.g.,* `MyBoolean_PDU`*) and not the number tag to identify the PDU.*

3.3.64 -sampleCode <**pdus**|**values**> / -noSampleCode

The `-sampleCode` option (introduced in version 9.0) instructs the compiler to generate a sample application that demonstrates how to populate and use the generated C representations of PDU types defined in the input ASN.1 syntax. This option takes an optional parameter to control generation of the sample application:

`values` The compiler creates a sample application by generating test code for each `valuereference` in the ASN.1 input that is defined with a PDU type.

`pdus` The compiler creates a sample application by generating test code using sample values that the compiler automatically creates for each "true" PDU type. That is, for each defined type that is not referenced by any other type, is not imported, and is not used in information object sets or as a type in the ContentsConstraint.

If the operand is absent, the compiler generates a sample application containing test code for each `valuereference` in the ASN.1 input that is defined with a PDU type, and each value is automatically created by the compiler for "true" PDU types.

The test code created for each value demonstrates how to create and initialize the C representation for the corresponding value, call the OSS ASN.1/C runtime API to encode and decode the value, and traverse and print the decoded value.

The sample application is separated into six files. These files are created in the `samplecode/` directory in the same location where the output code-file/control-file is created, as shown below:

```
...
|-- outputname.c
...
`-- samplecode
    |-- makefile
    |-- outputname-createvalue.c
    |-- outputname-createvalue.h
    |-- outputname-printvalue.c
    |-- outputname-printvalue.h
    `-- outputname-testmain.c
```

Where *`outputname`* is the name of a generated control-table file/code-file without a file extension.

The *`outputname`*`-createvalue.c` file contains an implementation of functions that demonstrate how to initialize the C representation of values specified by the `-sampleCode` option.

The *`outputname`*`-createvalue.h` file contains declarations of functions defined in the *`outputname`*`-createvalue.c` for values defined with a PDU type.

The *`outputname`*`-printvalue.c` file contains an implementation of functions that demonstrate how to traverse and print the C representation of PDU types from the ASN.1 input.

The *`outputname`*`-printvalue.h` file contains declarations of functions defined in the *`outputname`*`-printvalue.c`.

The *`outputname`*`-testmain.c` file provides an application entry point (a `main()` routine) containing test code for each PDU value from the *`outputname`*`-createvalue.c` file. The test code calls the functions defined in *`outputname`*`-createvalue.c` to create C representations of the corresponding values, the OSS ASN.1/C runtime API to encode and decode these values, and the functions defined in *`outputname`*`-printvalue.c` to traverse and print the decoded values.

The `makefile` file is a sample makefile that illustrates how to build an executable from all the generated files.

The –sampleCode command-line option can be used along with the OSS.SampleCode directive (see section 4.4.2.40), the latter provides the ability to customize how sample values for some ASN.1 types, such as CHOICE, SEQUENCE OF and SET OF, are created by the compiler.

The -sampleCode option has no effect if no PDUs were found in the input ASN.1 syntax or when one of the following options is present: -noOutput, -syntaxOnly, -noHeader, -noToed (-noCodefile) or -noSoed (-noControlFile). It also has no effect when -sampleCode with the values parameter is specified and the input ASN.1 syntax has no value references for PDU types.

The -noSampleCode option specifies that a sample application should not be generated even if the OSS.SampleCode directive is present in the ASN.1 input.

By default, these two options are turned off.

Related options:

-test (see section 3.3.71)

Related directives:

OSS.SampleCode / OSS.NoSampleCode (see section 4.4.2.40)

3.3.65 -shippable

This option instructs the ASN.1 compiler to generate a special version of .h file suitable for distribution in a toolkit.

Note that that a file named ossship.h is included in shipments to customers who purchased a toolkit builder license. If they have a license, customers are authorized to distribute ossship.h (along with the OSS Nokalva .h file generated with the -shippable ASN.1 compiler option) to their customers. If the -shippable option is specified, no .c file will be generated and the generated .h file will #include the shippable ossship.h file instead of the OSS Nokalva internal header files, such as ossglobl.h, asn1code.h, asn1hdr.h, etc. This will allow their customers to successfully compile their code without unresolved symbol errors. It is important to note that even if customers have a toolkit builder's license, they may **not** ship the produced .c files nor internal OSS Nokalva header files to their customers. Customers should contact info@oss.com if they need the ossship.h header file.

This option is available only if you have a toolkit builder's license and is turned off **by default**.

Note: The .h file produced when the -shippable option is specified is only for distribution to your customers. You should still compile you application with the .h file produced without the -shippable option as normal.

3.3.66 -shortenNames / -noShortenNames

These options specify whether or not long variable names should be shortened to 31 characters by the ASN.1 compiler. This option accommodates maximum name length requirements that exist in some C

compilers. Also see the documentation for the OSS.SHORTENNAMES compiler directive in section 4.4.2.43.

By default, the `-noShortenNames` option is implied on most platforms.

3.3.67 -soedFile <soedCFileName> / -noSoedFile

This option is an alias for the '-controlFile' ASN.1 compiler option. It informs the ASN.1 compiler to generate a control table for use by the space-optimized encoder/decoder or LEAN encoder/decoder (if the '-lean' option is also specified).

The -soedFile option specifies the generation of the encoder/decoder control table in the output C file. It may optionally be followed by an operand to explicitly name the output C file. If the operand is absent and the -output keyword is not used, the name of the output C file will default to the last command-line input filename, with a suffix of .c (or `.cpp` if the `-c++` option is specified) appended.

Note: The `-soedFile (-controlFile)` and `-toedFile (-codeFile)` options are mutually exclusive.

The -noSoedFile option specifies that a control table C file should not be generated. This option is useful when you only want a header file (and no C file) to be generated by the ASN.1 compiler.

By default, both options are turned off.

Related options:

`-toedFile`

3.3.68 -sort / -noSort

These options indicate whether forward references should be eliminated or preserved in the compiler-generated header file.

The `-sort` option should be used when the header file will be input to a C compiler, since C imposes restrictions on forward references.

The `-noSort` option suppresses the sort, so that the order of the generated data types is the same as that of the ASN.1 types in the input file. This proves convenient if you wish to compare the ASN.1 definitions to the generated header file; however, unsorted output results in C compiler errors if there are forward references.

By default, the `-sort` option is implied.

3.3.69 -splitForSharing [<*splitFilename*>]

The `-splitForSharing` command-line option is similar to the `-splitHeaders` option except that each of the multiple header files that are generated may be used autonomously. This is useful if you wish to use only a subset of a large ASN.1 specification in your application.

Additionally under this mode, you will not have to recompile your application if ASN.1 definitions from a module that you do not use change.

When this option is specified, a `.spl` file for the preservation of indices and names across several versions of an ASN.1 specification is also produced. You may specify the filename prefix for this `.spl` using the optional argument *splitFilename*.

By default, this option is turned off.

Related options:

`-splitHeaders`

3.3.70 -splitHeaders [<*splitFilename*>]

The `-splitHeaders` option allows you to have multiple header files generated for a single ASN.1 specification. This is achieved by placing the data structures for each particular ASN.1 module in a separate header file.

Each header file generated when the `-splitHeaders` option is specified contains the data definitions for a particular module. However, the corresponding PDU constants and pointer to the control-table/code-file is generated in a master header file (whose name is derived from the name of the last input ASN.1 file). This master header file `#include`s all of the individual header files produced. Thus, to use the produced data structures, you would simply `#include` the master header file in your application code. If you do that, all of the individual header files will be automatically included.

In addition to the header files generated, the ASN.1 compiler also generates a file with a `.spl` extension. This file contains global compiler directives to preserve the names and indices of the types generated in the multiple header files. If you modify your ASN.1 specification and wish to regenerate multiple files, you should pass this `.spl` file on the command-line before all files that contain your ASN.1 specification. This way, you will not have to change your application program if you make minor changes to the input ASN.1 specification. (You may specify the name prefix for this `.spl` using the optional filename argument.)

Although multiple header files are generated, only one control-table/code-file is still produced for each ASN.1 specification when the `-splitHeaders` command-line option is used. However under the split-headers mode, you can easily use more than one control-table/code-file in a single application with a common set of ASN.1 modules without name conflict problems. (You can even load these control-tables/code-files at runtime if you compiled the control-table/code-file as a DLL (see the related FAQ in the *Linking related questions* section of the FAQs chapter of the **OSS ASN.1 API Reference Manual**).) In order to produce such control tables for simultaneous use, you can follow the following simple procedure:

a) ASN.1 compile your first ASN.1 specification with the `-splitHeaders` option specified and retrieve the produced `.spl` file.
b) Compile your second ASN.1 specification passing the `.spl` file obtained from step (a) as the first file on the command line and retrieve the second produced `.spl` file.
c) Compile your third ASN.1 specification passing the `.spl` file obtained from step (b) as the first file on the command line and retrieve this third produced `.spl` file. You may repeat this process for each of you ASN.1 specifications that you want to use.

After you have produced the appropriate files, you can then `#include` into your application the master header files corresponding to the modules you wish to use and then use `ossinit()`, `ossWinit()`, or `ossUinit()` to load the specific control-table/code-file which you desire.

Notes:
1) You may have to also specify the `-root` option along with the `-splitHeaders` option to have all of the input modules translated and processed.

By default, this option is turned off.

Related options:
`-splitForSharing`

Related directives:
`OSS.HeaderName` (see section 4.4.2.21), `OSS.Preserve` (see section 4.4.2.37)

3.3.71 -suppress ***<messageNumber>***

This option instructs the compiler to suppress a specific warning or informatory message. More than one `-suppress` option may be specified on the command line. The *`messageNumber`* may be either the complete message identifier (e.g., A0210W) or just the numeric portion (e.g., 210).

By default, this option is turned off.

3.3.72 -syntaxOnly

This option instructs the ASN.1 compiler to only do a syntax check on the ASN.1 input. This keyword is useful if you do not wish to generate a header nor a C file when debugging an abstract syntax.

By default, a header and control file are generated (as long as no error is detected and no other option to suppress them is specified).

Related options:
`-pedantic`

3.3.73 -test

This option instructs the ASN.1 compiler to generate a `main()` routine in the output C file which calls the function `ossTest()` for each `valuereference` in the ASN.1 input defined with a type that is a PDU. This is provided for simplifying the generation of sample encodings and for testing the encoder/decoder without the need to write any C code.

If the `-test` option is specified together with the `-enablePartialDecode` option, the `main()` routine calls `ossTestPD()` instead of `ossTest()`.

To view the output of the generated `test` function, you simply compile and link the generated C file and then run the resultant executable.

Note that if your ASN.1 syntax does not contain any value notation, that is, there are no predefined test cases, error `C1261E` will be issued since there's nothing to test.

For more information about this topic, see the description for the function `ossTest()`in the **OSS ASN.1/C Runtime API Reference Manual**.

By default, the `main()` routine described above is not generated.

Related options:
`-sampleCode / -noSampleCode` (see section 3.3.62)

3.3.74 -toedFile***<toedCFileName>*** / -notoedFile

This option is an alias for the '-codeFile' ASN.1 compiler option. It informs the ASN.1 compiler whether or not it should generate an output C file containing the time-optimized encoder/decoder.

The -toedFile keyword can be optionally followed by a name for the output C file. If no name is specified and the -output keyword is not used, the output C filename will default to the last command-line input filename, with a suffix of .c (or `.cpp` if the `-c++` option is specified) appended.

By default the `-toedFile` option is implied.

Related options:
`-soedFile / -controlFile, -codeFile`

Note: The -soedFile (`-controlFile`) and -toedFile (`-codeFile`) options are mutually exclusive.

3.3.75 -uniquePDU / -noUniquePDU

These options specify whether or not Protocol Data Unit (PDU) tags must be unique.

The `-uniquePDU` option specifies that the tags for all PDUs must be unique, while the `-noUniquePDU` specifies that PDUs can have identical tags.

When the `-uniquePDU` option is specified, the OSS ASN.1 compiler issues an error message upon finding a non-unique PDU tag. When the `-noUniquePDU` option is specified, no error or warning messages are issued. When neither option is specified, only warning messages are isssued.

Note: In some ASN.1 specifications such as the Presentation Layer Protocol (ISO/IEC 8823/ITU-T X.226), the PDUs do not have unique tags, whereas in other ASN.1 specifications, such as the Association Control Service Element (ACSE), the PDU tags are unique.

By default, the compiler issues a warning message if it detects non-unique PDU tags when BER or DER is expected to be used at runtime.

3.3.76 –useQualifiedNames ***<maxPrefixLevel>***

The `-useQualifedNames` compiler option was introduced in release 10.1. It instructs the ASN.1 compiler to add a prefix, depending on the type name, to

- Enumerators generated for an ENUMERATED ASN.1 type.
- Constants generated for named numbers of an INTEGER type.
- Constants generated for named bits of a BIT STRING type.

For nested, unnamed types, these qualified names also contain field names that could result in long identifiers. By using the optional `maxPrefixLevel` operand, you can shorten such names by limiting the number of nested fields.

Example:

The following ASN.1 syntax

```
T ::= SEQUENCE {
    outerField SEQUENCE {
        innerField  ENUMERATED {en1, en2, en3}
    }
}
```

compiled without `-useQualifedNames` is represented as the following C enum:

```
typedef enum T_outerField_innerField {
    en1 = 0,
    en2 = 1,
    en3 = 2
} T_outerField_innerField;
```

The same compiled with `-useQualifedNames` is represnted as

```
typedef enum T_outerField_innerField {
    T_outerField_innerField_en1 = 0,
    T_outerField_innerField_en2 = 1,
    T_outerField_innerField_en3 = 2
} T_outerField_innerField;
```

Specifying `-useQualifedNames 1` results in the following enum:

```
typedef enum T_outerField_innerField {
    innerField_en1 = 0,
    innerField_en2 = 1,
    innerField_en3 = 2
} T_outerField_innerField;
```

Note: this option does not change any user-defined names that were assigned using the ASN1.Nickname directive.

By default, the `-useQualifedNames` option is turned off.

3.3.77 -userConstraints / -noUserConstraints

These options specify whether or not user-defined constraints should be generated into the control table and constraint-checking function prototypes into the header file.

The `-userConstraints` option is typically used with the `-noConstraints` option. It causes information for user-defined constraints to be generated into the control table even though all other constraint information is excluded.

The `-noUserConstraints` option is used with the `-constraints` option. It causes user-defined constraint information to be excluded from the control table even though all other constraint information is included.

When `-userConstraints` is in effect, an ASN.1 type definition that uses a user-defined constraint, such as:

```
OddNumber ::= INTEGER (CONSTRAINED BY {-- Must be an odd number--})
```

causes code to be generated into the control table that results in the encoder/decoder calling the user-written function whose prototype is:

```
OddNumber_fn(OssGlobal *world, OddNumber *value, void **handle);
```

to verify the constraint imposed on values of type OddNumber. (Refer to section 4.4.2.20 for more details about specifying the name of this function.)

So, when `-userConstraints` is specified, you are required to write constraint checking functions for all user-defined constraints (unless the OSS.NoConstrain directive is specified to disable constraint checking for a particular type).

With `-noUserConstraints` specified, you do not write functions to check user-defined constraints, for this option instructs the ASN.1 compiler to not generate code to call such functions at run-time.

Note: User-defined constraints are not supported by the TOED.

By default, the `-noUserConstraints` option is used.

3.3.78 -useXMLNames

The `-useXMLNames` compiler option causes generated names in the output header and C file to be the same as the names used for the components of the corresponding E-XER encoding. The OSS ASN.1 Compiler derives the name for such components from the E-XER `NAME` encoding instruction.

In the absence of the `NAME` encoding instruction, this compiler option does not affect the generated output files.

Example:

```
S ::= SEQUENCE {
    r [NAME AS "Red"]   IA5String,
    g [NAME AS "Green"] IA5String,
    yellow [NAME AS CAPITALIZED] IA5String,
    black [NAME AS UPPERCASED] IA5String
}
```

By default, above ASN.1 syntax generates following code in the header file:

```
typedef struct S {
    char            *r;
    char            *g;
    char            *yellow;
    char            *black;
} S;
```

However, if the same ASN.1 syntax is compiled with `-useXMLNames` command line option the name of types and fields are derived from the `NAME` encoding instruction. The generated name `r` is changed to `Red`, `g` to `Green`, `yellow` to `Yellow`, and `black` to `BLACK`, respectively. In this case, the names of the fields of structure `S` will be the same as the names used in the corresponding E-XER encoding:

```
typedef struct S {
    char            *Red;
    char            *Green;
    char            *Yellow;
    char            *BLACK;
} S;
```

E-XER encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<S xmlns:asn1="urn:oid:2.1.5.2.0.1">
  <Red>shirt</Red>
  <Green>trousers</Green>
```

```
  <Yellow>hat</Yellow>
  <BLACK>shoes</BLACK>
</S>
```

3.3.79 -valueRefs / -noValueRefs

These options specify whether or not special information about `valuereferences` should be generated into control table.

The `-valueRef` option instructs the ASN.1 compiler to generate information into the control table that allows the IAAPI (Interpretive ASN.1 Application Program Interface) to dynamically determine what `valuereferences` are present in the compiler input.

The `-noValueRef` option suppresses such information.

By default, the compiler uses the `-noValueRef` alternative.

3.3.80 -verbose / -noVerbose

These options specify whether or not compiler status messages should be issued.

The `-verbose` option instructs the ASN.1 compiler to issue status messages (e.g., indicate which input file is currently being read).

The `-noVerbose` option causes the suppression of status messages.

By default, the compiler uses the `-noVerbose` alternative.

3.3.81 -warningMessages / -noWarningMessages

These options specify whether or not compiler warning and informatory messages should be generated.

`-warningMessages`/`-noWarningMessages` implies `-informatoryMessages`/`-noInformatoryMessages` respectively.

By default, the `-noWarningMessages` option is implied via `-relaxedMode`. Note that some significant warning messages are not suppressed in the default mode. To suppress all warnings, the explicit `-noWarningMessages` option should be specified.

When the `-noRelaxedMode` option is specified, the `-warningMessages` option is implied.

Note, the order in which the explicit `-warningMessages`/`-noWarningMessages` and `-relaxedMode`/`-noRelaxedMode` options appear on the command line is important. The rightmost option is enforced, which overrides any effects of the previous options.

Related options:

`-informatoryMessages/-noInformatoryMessages`,
`-relaxedMode/-noRelaxedMode`

3.3.82 -xsl

The `-xsl` option allows you to generate multiple default stylesheet files one for each PDU. Normally, no separate stylesheet is produced even when the XER encoding rules are requested. (Instead, you may choose to use your own XML stylesheets or customized versions of previously-generated ones.)

The produced stylesheets will have a filename extension of `.xsl` and a filename prefix identical to the name generated for the PDU that they correspond to. You can specify a different filename prefix and extension by using the OSS.Stylesheet directive (refer to section 4.4.2.44).

The `-output` <*output-pathname*> and `-prefix` <*prefix-for-identifiers*> options affect the name of the produced `.xsl` file.

If you want `ossEncode()` to generate a reference for your stylesheet when producing its XML output for a particular PDU, you should call the `ossSetXmlStylesheet()` function prior to invoking the encoder.

By default, no XML stylesheets are generated by the ASN.1 compiler

3.4 Command-line help

You can obtain a synopsis of the command-line syntax by entering `asn1` with no arguments. For example:

```
> asn1

OSS ASN.1 Compiler Version 10.2
Copyright (C) 2015 OSS Nokalva, Inc.  All rights reserved.

asn1 <infile(s)>   [-c    | -c++   ]          [-debug      | -nodebug    ]
  [-ber ]          [-suppress msg# ]          [-short      | -noshort    ]
  [-der ]          [-extern <name> ]          [-unique     | -nounique   ]
  [-per ]          [-sort | -nosort]          [-inform     | -noinform   ]
  [-uper]          [-warn | -nowarn]          [-defines    | -nodefines  ]
  [-xer ]          [-1990 | -2008  ]          [-noout      | -out <name> ]
  [-cxer]          [-charintegers  ]          [-pedantic   | -nopedantic ]
  [-cer ]          [-ignoresuppress]          [-encodeonly | -decodeonly ]
  [-exer]          [-useXmlNames   ]          [-constrain  | -noconstrain]
  [-oer ]          [-messageFormat ]          [-cont <ctrlfile> | -nocont]
  [-coer]          [-nostaticvalues]          [-relaySafe  | -norelaySafe]
  [-lean]          [-extendopentype]          [-errorfile    <errorfile> ]
  [-xsl ]          [-allowbadvalues]          [-h <headerfile>  | -noh   ]
  [-dtd ]          [-gen <gendirfl>]          [-list <listfile> | -nolist]
  [-syntax    ]    [-exportdlldata ]          [-cod  <codefile> | -nocod ]
  [-verbose   ]    [-messageFormat ]          [-cont <ctrlfile> | -nocont]
  [-minimize  ]    [-comments | -noComments ] [-test            | -notest]
  [-compress  ]    [-designerWarnings]        [-commandfile <commandfile>]
  [-keepnames ]    [-ignoreError msg#]        [-mod <modlistingfile>|-nom]
  [-valuerefs ]    [-splitheaders <splfile>   |-splitforsharing <splfile>]
  [-shippable ]    [-pdusforopentypes]        [-nopdusforopentypes       ]
  [-autoencdec ]   [-userconstraints          |-nouserconstraints        ]
  [-dualheader ]   [-cstylecomments]          [-pdusforcontainingtypes   ]
  [-prefix prfx]   [-assignments   ]          [-nopdusforcontainingtypes ]
  [-cont <ctrlfile>|-nocontrolFile ]          [-code <cdfile>|-nocodeFile]
  [-root] [-ansi]  [-compat ver#   ]          [-sampleCode   |-noSampleCode ]
  [-exportDllAPI]  [-helperDefineNames]       [-helperNames  |-noHelperNames]
  [-hdebug     ]   [-helperEnumNames]         [-helperAPI    |-noHelperAPI  ]
  [-relaxedMode    | -norelaxedMode ]         [-helperMacros |-noHelperMacro]
  [-verbose    ]                             [-helperListAPI|-noHelperList ]
  [-reservedWords  <reserved_words> ]         [-restrictedConstraintChecking]
  [-ignoreRedefinedAssignments           ]    [-ignoreIncompleteItems       ]
  [-relaxPerToedIntegerRangeConstraint   ]    [-allow badcase]
  [-enablePartialDecode                  ]    [-partialDecodeOnly           ]
  [-useQualifiedNames   <maxPrefixLevel> ]

enter "asn1 -help" for more information

> asn1 -help

OSS ASN.1 Compiler Version 10.2
Copyright (C) 2015 OSS Nokalva, Inc.  All rights reserved.

Compiler parameters: (.D. indicates default)
  <input> <...input> . one or more ASN.1 input files

  -cont <ctrlfile> ... alias to -soed
  -soed <soedfile> ... generate encoder/decoder control table
  -nocont          ... alias to -nosoed
  -nosoed          ... do not generate encoder/decoder control table
  -cod <codefile>  ... alias to -toed
  -toed <toedfile> .D. generate time-optimized encoder/decoder code
```

```
                    into <codefile>
-nocode          ... alias to -notoed
-notoed          ... do not generate time-optimized encoder/decoder code
-lean            ... generate LED compatible header files
-syntax          ... perform syntax checking only

-h <headerfile>  .D. generate C include file into <headerfile>
-noh             ... do not generate include file
-output name     ... use name for output file names
-nooutput        ... do not generate output files, same as syntaxonly.
-errorfile file  ... write all messages to 'file' instead of stdout/stderr
-list <listfile> ... generate an ASN.1 combined listing into 'listfile'
-nolist          .D. do not generate an ASN.1 combined listing
-mod <listfile>  ... generate single ASN.1 module listings into 'listfile'
-nomod           .D. do not generate single ASN.1 module listings
-gendir <genfile>... generate list of directives into <genfile>

-sort            .D. rearrange generated C types to remove forward references
-nosort          ... generate C types in the same order as the ASN.1 input
-shortennames    ... shorten generated variable names to be 31 bytes max
-noshortennames  .D. do not shorten generated names
-uniquepdu       .D. check that pdu id's are unique
-nouniquepdu     ... do not check that pdu id's are unique
-prefix id       ... prefix variables with unique id
-externalname name . external variable name of encoder/decoder info
-useXmlNames     ... generates C type and field names according to E-XER
                    "NAME" encoding instructions

-constraints     .D. check constraints at runtime
-restrictedConstraintChecking
                 ... do not ignore the OSS.NoConstrain directive
                     when -constraints is specified
-noconstraints   ... do not check constraints at runtime
-userconstraints ... check user constraints (even for -noconstraints)
-nouserconstraints D do not check user constraints (even for -constraint)
-valuerefs       ... generate valuereference info needed for IAAPI use
-novaluerefs     .D. do not generate valuereference info needed for IAAPI use
-encodeonly      ... generate only encoder routines
-decodeonly      ... generate only decoder routines
-nostaticvalues  ... initialize generated values at runtime
-test            ... generate a program to encode/decode ASN.1 values
-notest          .D. do not test encoding/decoding of all ASN.1 values
-debug           .D. issue detailed trace data and error messages
-nodebug         ... do not issue detailed trace data and error messages
                     to slightly improve CPU performance

-c               .D. generate C compatible data types and names
-c++             ... generate C++ compatible data types and names
-dualheader      ... generate headers files compatible with C and C++
-ansi            .D. generate ansi c function prototypes
-defines         .D. generate #define constants
-nodefines       ... generate const-declarations instead of #define constants

-warning         ... allow warning messages
-nowarning       ... suppress warning messages
-suppress msg#   ... suppress a specific message
-ignoresuppress  ... ignore the SUPPRESS directive (display all messages)
-verbose         ... display compilation progress messages
-noverbose       .D. do not display compilation progress messages
-pedantic        ... rigorously check all modules
-nopedantic      .D. rigorously check only the root module(s)
-relaxedMode     .D. automatically select command-line options that
                     result in relaxed compiler behavior, for example,
                     -allowBadValue, -noWarning, etc.
-norelaxedMode   ... automatically select command-line options that
```

Compiler options

```
                    result in strict compiler behavior
-informatory     ... allow informatory messages
-noinformatory   ... suppress informatory messages
-ignoreError msg#... ignore a specific error (issue a warning instead)
-designerWarnings... issue additional warnings for protocol designers
-compat version# ... disable new features for backward compatibility
-allow badcase   ... enable badcases for backward compatibility
-1990            ... all modules conform to 1990 ASN.1 syntax
-2008            ... all modules conform to 2008 ASN.1 syntax

-per             ... generate output file for Aligned PER encoder/decoder
-uper            ... generate output file for Unaligned PER encoder/decoder
-ber             .D. generate output file for BER encoder/decoder
-der             ... generate output file for BER & DER encoder/decoder
-xer             ... generate output file for Basic XER encoder/decoder
-cxer            ... generate output file for Canonical XER encoder/decoder
-cer             ... generate output file for BER & CER encoder/decoder
-exer            ... generate output file for EXTENDED XER encoder/decoder
-oer             ... generate output file for OER encoder/decoder
-coer            ... generate output file for Canonical OER encoder/decoder
-compress        ... use encoding compression (all encoding rules)

-commandfile <file> ... read command line from 'file'
-minimize        ... generate the smallest possible output files
-root            ... treat all modules as root modules
-keepnames       ... generate directives for keeping names
-extendopentype  ... add a userfield to the OpenType structure
-charintegers    ... generate char for INTEGERs constrained to small values
-pdusforopentypes... ignore NOPDU directive for opentypes
-nopdusforopen   ... do not generate PDUs for info objects with -noconstraints
-shippable       ... generate .h file suitable for shipment
-allowbadvalues  ... allow generation of bad values
-sampleCode <pdus|values>
                 ... generate a sample code to initialize, encode, decode
                     and traverse values of PDU types defined in the input
                     ASN.1 syntax
-noSampleCode    ... do not generate a sample code
-relaySafe       ... retain unrecognized extension additions at runtime
-norelaySafe     .D. do not retain unrecognized extension additions at runtime
-autoencdec      ... permit automatic encoding/decoding even if -noconstraint
                     is used
-splitforsharing <splfile> ... generate personal header for each ASN.1 module
-splitheaders    <splfile> ... generate header for each ASN.1 module and
                               a header which includes all module headers
-messageFormat   <format>
                 ... display error messages in <format>:
                     'oss', 'emacs' or 'msvc'
-xsl             ... generate default stylesheet for each PDU
-dtd             ... generate document type definition file for each PDU
-assignments     ... process type and value assignments defined outside
                     ASN.1 modules
-exportdlldata   ... exports all the external variables from a control table
                     or code file DLL for Windows platforms
-comments        .D. generate ASN.1 comments as C style comments without -c++
                     and as C++ comments with -c++ in header file
-noComments      ... do not generate ASN.1 comments in header file
-CStyleComments  ... generate C style comments into header file with -c++
-pdusforcontainingtypes
                 ... ignore NOPDU directive for types referenced by a
                     contents constraint
-nopdusforcontainingtypes
                 ... do not generate PDUs for types referenced by a
                     contents constraint

-noHelperAPI     .D. do not generate helper and list API macros
```

```
-helperAPI       ... generate helper macros and list API macros for
                     all ASN.1 types represented by C structures
-nohelperNames   .D. do not generate context-based names
-helperNames     ... generate context-based names for nested ASN.1
                     built-in types represented by C structures
-helperDefineNames . generate context-based names for #define constants
                     used for bit-masks, named bits and numbers
-helperEnumNames ... generate context-based names for enumerators
-nohelperMacros  .D. do not generate helper macros
-helperMacros    ... generate helper macros for all ASN.1 types
                     represented by C structures
-nohelperListAPI .D. do not generate helper list API macros
-helperListAPI   ... generate helper list API macros for all
                     SET OF and SEQUENCE OF types
-exportDllAPI    ... export all helper list API functions from control table
                     or code file DLL for Windows platforms
-hdebug          ... generate compile-time type checking diagnostic inside
                     helper macros
-ignoreIncompleteItems
                 ... ignore incomplete ASN.1 definitions of types and values
-ignoreRedefinedAssignments
                 ... ignore redefined ASN.1 definitions of types and values
-reservedWords <reserved_words>
                 ... treat words specified in the parameter and separated
                     by commas as reserved words in the generated files
                     that should be always mangled
-relaxPerToedIntegerRangeConstraint
                 ... relax check for upper bound of integer range
                     constraints in TOED PER encoding functions
-useQualifiedNames <maxPrefixLevel>
                 ... generate context-based names for enumerators, named
                     numbers and bits
-enablePartialDecode
                 ... generate time-optimized full and partial decoders
-partialDecodeOnly
                 ... generate time-optimized partial decoder only
-noComments      ... do not generate ASN.1 comments in header file
-CStyleComments  ... generate C style comments into header file with -c++
-pdusforcontainingtypes
                 ... ignore NOPDU directive for types referenced by a
                     contents constraint
-nopdusforcontainingtypes
                 ... do not generate PDUs for types referenced by a
                     contents constraint

-noHelperAPI     .D. do not generate helper and list API macros
-helperAPI       ... generate helper macros and list API macros for
                     all ASN.1 types represented by C structures
-nohelperNames   .D. do not generate context-based names
-helperNames     ... generate context-based names for nested ASN.1
                     built-in types represented by C structures
-helperDefineNames . generate context-based names for #define constants
                     used for bit-masks, named bits and numbers
-helperEnumNames ... generate context-based names for enumerators
-nohelperMacros  .D. do not generate helper macros
-helperMacros    ... generate helper macros for all ASN.1 types
                     represented by C structures
-nohelperListAPI .D. do not generate helper list API macros
-helperListAPI   ... generate helper list API macros for all
                     SET OF and SEQUENCE OF types
-exportDllAPI    ... export all helper list API functions from control table
                     or code file DLL for Windows platforms
-hdebug          ... generate compile-time type checking diagnostic inside
                     helper macros
-ignoreIncompleteItems
```

```
                    ... ignore incomplete ASN.1 definitions of types and values
    -ignoreRedefinedAssignments
                    ... ignore redefined ASN.1 definitions of types and values
    -reservedWords <reserved_words>
                    ... treat words specified in the parameter and separated
                        by commas as reserved words in the generated files
                        that should be always mangled
    -relaxPerToedIntegerRangeConstraint
                    ... relax check for upper bound of integer range
                        constraints in TOED PER encoding functions
    -useQualifiedNames <maxPrefixLevel>
                    ... generate context-based names for enumerators, named
                        numbers and bits
    -enablePartialDecode
                    ... generate time-optimized full and partial decoders
    -partialDecodeOnly
                    ... generate time-optimized partial decoder only
```

Note that the above information may vary slightly depending on the version of the ASN.1 compiler that you are using.

3.5 Chapter appendix: List of available -compat flags

The following `-compat` flags are listed in alphabetical and numerical order.
(Refer to section 3.3.12 for more details about the `-compat` command-line option)

`-compat` flag name	Brief Description	Section
`addBadDirForInstancesOfParamRef`	generates incorrect directives in .gen file with -gen or -keep options for shared instances of parameterized types for which a separate typedef is created	3.5.1
`allowBadDEFAULTValues`	allows the compiler to silently truncate the size-constrained DEFAULT value, and to issue a warning message instead of an error. The DER/CER/COER encoder will not encode the value if it is equal to the truncated version of the DEFAULT value.	3.5.2
`allowLinkedDirectiveForStringTypes`	allows the LINKED compiler directive to be applied to string types for compatibility with very early ASN.1/C releases	3.5.2
`allow namedBitThatExceedsConstraint`	instructs the compiler to issue a warning instead of an error if a named bit exceeds the BIT STRING size	3.5.3
`allowUnnamed`	allows the generation of unnamed nested C structures	3.5.2
`bad1994ExternalWithContentsConstr`	causes 1994 EXTERNAL representation to be incorrectly generated for EXTERNAL types with ContentsConstraint within MultipleTypeConstraint	3.5.3
`badDefineNamesForComponentsOfFields`	restores the behavior where incorrect #define names can be generated for OPTIONAL (or DEFAULTed) fields from the COMPONENTS OF section when the directive FIELDNAME (DefineName, NickName) is applied to the corresponding field of the original type.	3.5.4
`badEnumeratorsConflictingWithDefineNames`	permits the compiler to possibly generate duplicate names for enumeration values and definitions when some combination of names and directives is used in the ASN.1 syntax	3.5.5
`badExternalPrefix`	calls for prefixing '_' to structures in EXTERNAL type	3.5.6
`badLengthDirectives`	lets SHORT, INT, LONG dir. affect `length` and `count`	3.5.7
`badNameConflicts`	causes unneeded name disambiguation	3.5.8
`badPointerTypesWithNestedContConstr`	causes an extra pointer to be generated for ASN.1 types with ContentsConstraint within InnerSubtypes and with compiler-generated helper names	3.5.9
`badRefNames`	causes the incorrect generation of the base type's type reference for tagged types with the OSS.POINTER directive	3.5.10
`badSetOfOidWithPointer`	restores behavior of disregarding the effect of the other global directives when the global OSS.POINTER directive is applied to SET OF, SEQUENCE OF, and OBJECT IDENTIFIER types	3.5.11
`badSharingForTypesWithInlineTypeDir`	ignores OSS.UseThis directives for similar types that include similar fields with OSS.InlineType directives	3.5.12
`badTypedefsForUserNames`	causes extra `typedef`s in .h file for TYPENAME direct	3.5.13
`badTYPENAMEdirectives`	forces the ASN.1 compiler to process invalid OSS.TYPENAME directives applied to referenced fields of structured types	3.5.14
`badUnboundedBitStringsWithNa`	disables the new behavior in 8.6.0 where the compiler	3.5.15

medBits	does not limit an unbounded BIT STRING with a NamedBits representation based on the named bits values.	
badUnderscorePrefix	causes prefixing of internal names with '_'	3.5.16
badValuePrefix	disables prefixing valuereferences for disambiguity	3.5.17
BMPleanUTF8String	reverts UTF8String representation to pre-v7.0 standard when -lean is specified	3.5.18
charUTF8String	causes UTF8String to be represented with character strings made up of one-byte characters	3.5.19
decoderUpdatesInputAddress	lets ossDecode() modify input buf. addr. and length	3.5.20
extensionWithMask	enables bitmasks for pointered extensible elements	3.5.21
extraLinkedSETOFPointer	makes circular SET OF def. have double pointer (**)	3.5.22
extraNameShortening	calls for the unneeded shortening of names	3.5.23
extSizeNotUnbounded	calls for a fixed-length array for certain extensible types	3.5.24
ignore1994ExternalConstr	causes no constraint info to be provided to the runtime about constraints applied to 1994 EXTERNAL types converted to the 1990 EXTERNAL representation.	3.5.26
ignoreInlineTypeDirForSharedTypes	restores the behavior of disregarding the effect of the InlineType directive applied to type nested in a shared one	3.5.27
ignoreNicknamesInConflictResolution	generates disambiguated names for a #define constant or elements of an enum type even if the ASN1.Nickname directive	3.5.28
ignorePointerDirForLean	used to ignore the effect of the POINTER directive for Lean, thus providing compatibility with Lean encoder/decoder runtime libraries of earlier versions.	3.5.29
ignorePrefixForSpecialStructures	disables prefixing C structures generated for ASN.1 types, such as CHARACTER STRING, EMBEDDED PDV and EXTERNAL, when the -helperNames ASN.1 compiler option is specified	3.5.30
implicitTypeInArray	Generates an implicit type definition for the CHOICE, SET OF and SEQUENCE OF type which is an element of the SET OF or SEQUENCE OF type in an ARRAY representation.	3.5.31
intEnums	causes C enums to be initialized with integers and restricts the scope of enum typedefs	3.5.32
multipleUserFunctions	allows multiple constraint functions for a single type	3.5.33
nestUnions	leaves alone nested structures within CHOICE type	3.5.34
noASN.1comments	disable ASN.1 comment transfer to header file	3.5.35
noBTypeValues	causes SEQUENCE/SET in value notation. to be ignored	3.5.36
noConstDeclarations	instructs the ASN.1 compiler to not generate const declarations for values of simple types	3.5.37
noDecoupledNames	specifies that the pre-v5.0.0 rules of name-mangling should be used when generating .c and .h files	3.5.38
noDefaultValues	makes DEFAULT items be treated like OPTIONAL ones	3.5.39
noMacroArgumentPDUs	instructs compiler not to treat macro arguments as PDUs	3.5.40
noObjectSetsFromDummyObjects	calls for the generation of extra _OSET declarations for instances of parameterized information object sets used in table constraints	3.5.41
noOssterm	disables generation of ossterm() with -test option	3.5.42
noParamTypesharing	disables typesharing for parameterized types	3.5.43
noPduForContainedExternal	disables generation of PDU constant in the header file for the EXTERNAL type used in ContentsConstraint	3.5.44
noPDUsForImports	instructs compiler not to treat imported types as PDUs	3.5.45

noSharedTypes	disables the type-sharing optimization used by compiler	3.5.46
noUInt	disallows unsigned int for constrained integers.	3.5.48
noULength	disallows unsigned int for length and count	3.5.49
noUserConstraintPDUs	labels parameters to CONSTRAINED BY as non-PDUs.	3.5.51
noValues	disables C initializations for value notation	3.5.52
oldBooleanType	equates the Boolean and ossBoolean types	3.5.53
oldEncodableNames	generates extra typedefs for referenced/parameter types with DeferDecoding / ENCODABLE directive.	3.5.54
oldExternObjHdl	causes "ObjHandle" to be appended to EXTERNAL	3.5.55
oldInternalDefineNames	allows mangling of #define names in internal structures	3.5.56
oldInternalNamesWithinUseThisSharedTypes	generates names of artificial types within parameterized types that are based on the name of the type that contains instances of param types, when sharing of the type's instances occurs and UseThis directives are present	3.5.57
oldNamesManglingWithPrefix	reproduce rare bug with mangling some typenames when both the -c++ and -prefix options	3.5.58
oldObjectNames	disables prefixing of information object names	3.5.59
oldParamTypesharing	excludes types used as actual parameters in parameterized types from typesharing optimization	3.5.60
oldSharingFieldsWithDirect	prevents typesharing of nested structures which have the OSS.DefineName/OSS.FIELDNAME directive applied to one of their fields	3.5.61
padded	allows PADDED directive on variable length strings	3.5.62
paddedForNamedBits	causes BIT STRING types with named bit lists to be represented with PADDED representation instead of UNBOUNDED	3.5.63
pointeredParamTypesWithContConstrAndUseThis	generates typedefs with an extra pointer for parameterized types with ContentsConstraint and the UseThis directive	3.5.64
terseComments	enables more brief comments in the header file	3.5.65
typedefsForGenNames	causes extra typedefs in .h file for internal compiler-generated names	3.5.66
unbndBit	lets SizeConstraints have precedence over NamedBitLists	3.5.67
unnamedStructForConstrBy	causes the generation of an unnamed struct for types with CONSTRAINED BY nested in SET, SEQUENCE, SET OF, and SEQUENCE OF with OSS.LINKED, OSS.DLINKED, or OSS ARRAY directives applied	3.5.68
useUShortForBitStringsWithNamedBits	disables the new behavior in 8.6.0 where the compiler generates representations for BIT STRINGs with NamedBits that have values exceeding SHRT_MAX and less than USHRT_MAX. Such representations could cause problems at runtime.	3.5.69
v2.0	provides compatibility with version 2.0 of compiler	3.5.70
v3.0	provides compatibility with version 3.0 of compiler	3.5.71
v3.5	provides compatibility with version 3.5 of compiler	3.5.72
v3.6	provides compatibility with version 3.6 of compiler	3.5.73
v4.0	provides compatibility with version 4.0 of compiler	3.5.74
v4.1.0	provides compatibility with version 4.1.0 of compiler	3.5.75
v4.1typesharing	instructs compiler to use version 4.1 type-sharing	3.5.76
v4.1.1 / v4.1.2 / v4.1.3 / v4.1.4 / v4.1.5 / v4.1.6 / v4.1.7 / v4.1.8	provides compatibility with respective version numbers	3.5.77
v4.1.6encodable	causes unneeded typedefs for ENCODABLE types	3.5.78

v4.1.6extraLinkedSETOFPtr	makes circular SET OF def. have double pointer (**)	3.5.79
v4.1.9	provides compatibility with version 4.1.9 of compiler	3.5.80
v4.2.0	provides compatibility with version 4.2.0 of compiler	3.5.81
v4.2.5	provides compatibility with version 4.2.5 of compiler	3.5.82
v4.2.5typesharing	instructs compiler to use version 4.2.5 type-sharing	3.5.83
v4.2.6	provides compatibility with version 4.2.6 of compiler	3.5.84
v4.2badSetOfWithGlobalDir	specifies the incorrect handling of mutually exclusive global directives applied to SET OF and SEQUENCE OF	3.5.85
v4.2badUnderscorePrefix	causes prefixing of internal names with '_'	3.5.86
v4.2defaults	causes use of version 4.2's default representation schema	3.5.87
v4.2namesForOpenTypes	instructs the ASN.1 compiler to generate old-style names which include the original ASN.1 built-in type name for which an open type is being generated	3.5.88
v4.2nicknames	causes Nickname and TYPENAME directives applied to parameterized types to affect the actual parameters	3.5.89
v4.2objHandleCstrPointer	causes pointered structs. for char. str. w/OBJHANDLE	3.5.90
v4.2octetStringDefault	makes OCTET STRINGS (>256 bytes) to be VARYING	3.5.91
v5.0.0	provides compatibility with version 5.0.0 of compiler	3.5.92
v5.0.0badNamesForNamedItems	causes Nickname directive to have a general effect. on parameterized types for nested NamedNumberLists.	3.5.93
v5.0.0namesPrefixes	causes unneeded prefixing of module names to #define constants	3.5.94
v5.0.0nicknames	causes Nickname and TYPENAME directives to have no effect on certain parameterized types	3.5.95
v5.0.1	provides compatibility with version 5.0.1	3.5.96
v5.0.4	provides compatibility with version 5.0.4	3.5.97
v5.0.6	provides compatibility with version 5.0.6	3.5.98
v5.1extraPointer	allows the application of an OSS.POINTER directive to circularly referenced linked SET OF / SEQUENCE OF types	3.5.99
v5.1parameterizedTypes	causes the most general representation to be used for restricted character types and INTEGER types with parameterized size/range constraints	3.5.100
v5.1typesharing	specifies the use of the version 5.1 typesharing algorithm	3.5.101
v5.1unnamedStructForConstrBy	calls for the generation inline types for unnamed structures defined within CONSTRAINED BY clauses when -C++ and -userConstraints are specified	3.5.102
v5.1.0	provides compatibility with version 5.1.0	3.5.103
v5.1.3	provides compatibility with version 5.1.3	3.5.104
v5.1.4	provides compatibility with version 5.1.4	3.5.105
v5.2nestedUsethisTypes	causes OSS.UseThis directive to be ignored if applied to a type referencing another type	3.5.106
v5.2noExtraParamRef	causes no separate typedef to be generated for an ASN.1 type referenced by a parameterized type	3.5.107
v5.2paramNicknames	reproduces rare bug with OSS.TYPENAME and ASN1.Nickname directive applied to some parameterized types	3.5.108
v5.2paramRangeConstraints	generates the long int representation for instances of INTEGER types with dummy parameterized range constraints	3.5.109
v5.2sharing	specifies the use of the version 5.2 typesharing algorithm	3.5.110
v5.2typesharing	reproduce bug in typesharing with a rare type of parameterization	3.5.111
v5.2.0	provides compatibility with version 5.2.0	3.5.112

v5.2.1	provides compatibility with version 5.2.1	3.5.113
v5.3.0	provides compatibility with version 5.3.0	3.5.114
v5.3.1	provides compatibility with version 5.3.1	3.5.115
v5.4integer	provides compatibility with version 5.4integer	3.5.116
v5.4.0	provides compatibility with version 5.4.0	3.5.117
v5.4.2	provides compatibility with version 5.4.2	3.5.118
v5.4.4	provides compatibility with version 5.4.4	3.5.119
v6.0.0	provides compatibility with version 6.0.0	3.5.120
v6.0stringswithMAXsize	provides compatibility with versions prior to 6.1.4 in regards to the generation of C-representations for character strings with SIZE constraints defined with the MAX keyword	3.5.121
v6.1.3varyingbitstring	provides compatibility with versions prior to 6.1.4 in regards to the C-representation of unconstrained BIT STRING types with named bits and the POINTER and VARYING directives applied	3.5.122
v6.1.3	provides compatibility with version 6.1.3	3.5.123
v6.1.4DefineNameSharing	disables the name mangling changes providing compatibility with version numbers 6.1.4 or lower.	3.5.124
v6.1.4extrapointer	provides compatibility with versions 6.1.2, 6.1.4 and 7.0BetaA in regards to applying an extra POINTER directive in certain rare cases to the fields of a SEQUENCE, SET or CHOICE type	3.5.125
v6.1.4ReferencesToLeanTypes	causes use of built-in LED typedefs instead of generated typedefs for references to named simple types with -lean.	3.5.126
v6.1.4	provides compatibility with version 6.1.4	3.5.127
v7.0DefineNames	restores the names generated in some #define	3.5.128
v7.0pdusForBuiltinTypesInObjectSet	provides compatibility with versions lower than 7.0.2	3.5.129
v8.0.0	provides compatibility with version 8.0.0	3.5.131
v8.1.0	provides compatibility with version 8.1.0	3.5.132
v8.1.2SharingTypesWithEXERInstruction	disables the new behavior in 8.2 where the compiler uses typesharing to share identical types with the assigned EXER encoding instructions and provides compatibility with version 8.1.2.	3.5.133
v8.1.2	provides compatibility with version 8.1.2	3.5.134
v8.1.2ParamRefSharingWithDirectives	generates a different typedef for parameterized types with shared instances in the presence of FIELDNAME, DefineName, some TYPENAME and ExtractType directives	3.5.135
v8.2.0	provides compatibility with version 8.2.0 of the ASN.1 compiler	3.5.136
v8.2.0DefineNamesSharingWhenUsingXmlNames	disables the new behavior in 8.3.0 where the compiler uses the XML name of a type to resolve any name conflicts related to the type's components when the -useXmlNames option is specified	3.5.137
v8.3.1	provides compatibility with version 8.3.1 of the ASN.1 compiler	3.5.138
v8.4.0	provides compatibility with version 8.4.0 of the ASN.1 compiler	3.5.139
v8.4DirForInstancesOfParamRef	disables new behavior where a .gen file, generated with the -keepnames option, can be used to restore old names for shared instances of parameterized types	3.5.140
v8.4ExtraTypedefForParamRefWithUseThis	generates extra unused typedefs for one of shared instances of a parameterized type	3.5.141

v8.4InvalidSizeForUnionConstraint	disables the new behavior in 8.5.0 where the compiler properly handles a union of size and single value subtype constraints rather than truncating the type C representation using the upper bound of the size constraint	3.5.142
v8.4PrimitiveTypeSharing	generates extra unneeded typedefs for similar types that appear within other similar types linked by the OSS.UseThis directive	3.5.143
v8.4TypesSharingWithAutomaticTagging	generates duplicate structures in the header file for similar ASN.1 types that are defined in the module with automatic tagging, instead of sharing such structures	3.5.144
v8.5.0	provides compatibility with version 8.5.0 of the ASN.1 compiler	3.5.145
v8.5FalseExtendedConstraintEncoding	restores bad pre-8.6 compiler and runtime behavior in the handling of some extensible constraints by the PER encoder/decoder	3.5.146
v8.5DlinkedSeqOfSetOfWithExtSizeConstraint	disables the new behavior in 8.6.0 where the compiler properly generates a DLINKED/DLINKED-PLUS representation for a SET/SEQUENCE OF type with extensible size constraints and a DLINKED or DLINKED-PLUS directive applied.	3.5.147
v8.5TableConstraintForExtensibleObjectSets	restores pre-8.6 compiler and runtime behavior in handling table/component relation constraint violations for extensible object sets.	3.5.148
v8.6namesForSetOfAndSeqOfWithNamedElements	restores pre-9.0 compiler behavior by generating a fabricated name for the C-structure representing an element of a SET OF / SEQUENCE OF type when the latter type had no ASN.1 name.	3.5.149
8.6UTF8StringRepresentation	restores pre-9.0 compiler behavior by generating (when -lean is specified) different UTF8String C-representations depending on the presence of wide characters in the type values in the input syntax	3.5.150
v8.7.0	provides compatibility with version 8.7.0 of the ASN.1 compiler	3.5.151
v8.7BitStringsWithMAXUpperBound	restores pre-9.0 compiler behavior by generating different C-representations for BIT STRING types having the SIZE constraint with MAX upper bound and the PADDED or the VARYING directive applied	3.5.152
v8.7reservedWords	restores pre-9.0 compiler behavior by generating the names "SID" and "small" without mangling.	3.5.153
v9.0.0extractionForDeeplyNestedTypes	restores pre-9.0.2 compiler behavior by generating code that does not limit the nesting level of structures and unions	3.5.154
v9.0reservedWords	restores pre-10.0 compiler behavior by generating the names "floor" and "send" without mangling.	3.5.155
v10.0valueTruncation	restores pre-10.1 compiler behavior by truncating a value of a SEQUENCE OF/SET OF type with an extensible size constraint.	3.5.156
V10.0reservedWords	restores pre-10.1 compiler behavior by generating the name "interface" without mangling.	3.5.157

3.5.1 -compat addBadDirForInstancesOfParamRef

Prior to version 8.2, the ASN.1 compiler generated incorrect directives in .gen files with the `-gen` and `-keep` options for shared instances of parameterized types for which extra typedefs were created. As the result, such .gen files could not be used to preserve names from the original ASN.1 syntax. Starting with version 8.2, these incorrect directives are no longer generated. The compat flag `"addBadDirForInstancesOfParamRef"` has been added and can be used to reproduce the old behavior.

Example:

ASN.1:

```
M DEFINITIONS AUTOMATIC TAGS ::= BEGIN
     T1 ::= SET { a  P1 {BIT STRING} OPTIONAL}
     T2 ::= SET OF P1 {BIT STRING}
     P1 {PT} ::= SET OF PT
END
```

Generated .h file:

```
typedef struct T1 {
    unsigned char   bit_mask;
#       define      a_present 0x80
    struct P1       *a;  /* optional; set in bit_mask a_present if present
*/
} T1;

typedef struct T2 {
    struct T2       *next;
    struct P1       *value;
} *T2;

typedef struct P1 {
    struct P1       *next;
    struct {
        unsigned int    length;  /* number of significant bits */
        unsigned char   *value;
    } value;
} *P1;
```

Generated .gen without `-compat addBadDirForInstancesOfParamRef`**:**

```
-- Directives artificially generated by the compiler:

--<OSS.TYPENAME M.P1 "P1">--
--<OSS.ExtractType M.P1>--

-- Directives artificially generated for the names to be kept:

--<OSS.TYPENAME M.T1 "T1">--
--<OSS.DefineName M.T1.a "a">--
--<OSS.TYPENAME M.T2 "T2">--
```

Using this .gen file along with .asn results in exactly the same generated structures.

Generated .gen with `-compat addBadDirForInstancesOfParamRef`:

```
-- Directives artificially generated by the compiler:

--<OSS.TYPENAME M.T2.* "P1">--
--<OSS.ExtractType M.T2.*>--
--<OSS.ExtractType M.P1>--
--<OSS.UseThis M.P1 M.T2.*>--

-- Directives artificially generated for the names to be kept:

--<OSS.TYPENAME M.T1 "T1">--
--<OSS.DefineName M.T1.a "a">--
--<OSS.TYPENAME M.T2 "T2">--
```

Using this .gen file along with .asn changes the generated structures:

```
typedef struct _bit1 {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _bit1;

typedef struct M_P1 {
    struct M_P1     *next;
    _bit1           value;
} *M_P1;

typedef struct T1 {
    unsigned char   bit_mask;
#       define      a_present 0x80
    struct M_P1     *a;  /* optional; set in bit_mask a_present if present
*/
} T1;

typedef M_P1 P1;

typedef struct T2 {
    struct T2       *next;
    struct M_P1     *value;
} *T2;
```

3.5.2 -compat allowBadDEFAULTValues

Starting with version 10.2, the ASN.1 compiler reports an error message when the DEFAULT value exceeds the SIZE constraints on the DEFAULT field.

Previously, the compiler silently truncated the size-constrained DEFAULT value that was too long, and issued a warning message. As a result, the DER/CER/COER encoder did not encode the value if it was equal to the truncated version of the DEFAULT value.

The `allowBadDEFAULTValues` compiler compat flag provides compatibility with previous versions.

3.5.3 -compat allowLinkedDirectiveForStringTypes

Starting with version 10.2, the ASN.1 compiler ignores the `LINKED` and the `DLINKED` directives if they are applied to string types (character strings, BIT STRINGs, OCTET STRINGs).

Previously, the `LINKED` compiler directive could be applied to string types for compatibility with very early ASN.1/C releases, although this feature hasn't been documented.

Starting with version 10.2, the compiler ignores the `OSS.LINKED` and `OSS.DLINKED` directives applied to a string type.

The `allowLinkedDirectivesForStringTypes` compiler compat flag provides compatibility with previous versions.

3.5.4 -compat allow namedBitThatExceedsConstraint

Starting with version 10.2, the compiler issues an error if a named bit exceeds the BIT STRING size. Previously, if a named bit exceeded the BIT STRING size, the compiler issued a warning.

The new `-allow namedBitThatExceedsConstraint` flag can be used to restore the previous behavior.

3.5.5 -compat allowUnnamed

Prior to version 5.0, the ASN.1 compiler allowed the generation of unnamed nested C structures even when the `-C++` command-line option was used. Starting with version 5.0, all structures get names when the `-C++` command-line option is specified.

The `-compat allowUnnamed` option disables the new behavior providing compatibility with version numbers lower than 5.0.

Example:

ASN.1:

```
A  ::= SEQUENCE {
         b SEQUENCE OF SEQUENCE {
                  c INTEGER
         }
     }
```

Representation with `-compat allowUnnamed` specified:

```
typedef struct A {
    struct _seqof1 {
        struct _seqof1  *next;
        struct {
            int             c;
```

```
        } value;
    } *b;
} A;
```

Representation without `-compat allowUnnamed` **specified:**

```
typedef struct A {
    struct _seqof1 {
        struct _seqof1  *next;
        struct _seq1 {
            int             c;
        } value;
    } *b;
} A;
```

3.5.6 -compat autoDetectPDUnumber

Prior to version 10.2, the OSS TOED/SOED BER decoder ignored the PDU number passed to determine it from the encoding, even if the number was not zero.

Starting with version 10.2, if the passed number is zero, the OSS ASN.1 compiler and the OSS runtime can determine the actual PDU number from the encoding.

The `-compat autoDetectPDUnumber` option disables the new behavior and provides compatibility with version numbers lower than 10.2.

3.5.7 -compat bad1994ExternalWithContentsConstr

Between versions 7.0.1/8.0B and 8.0.1/8.1, the ASN.1 compiler was changed to generate an incorrect 1994 representation for EXTERNAL types with ContentsConstraint within a MultipleTypeConstraint. Due to such an incorrect representation, the BER decoder could not correctly decode BER encodings produced according to the X.690 specification. The problem was fixed in the 8.0.1/8.1 versions.

The standalone compat flag `-bad1994ExternalWithContentsConstr` restores the incorrectly generated 1994 EXTERNAL representation. This compat flag is not a part of any version compat flags.

Example:

ASN.1:

```
A ::= EXTERNAL ( WITH COMPONENTS {data-value (CONTAINING INTEGER)} )
```

Representation with `-compat bad1994ExternalWithContentsConstr` **specified:**

```
typedef struct A {
    struct _choice1 {
        unsigned short  choice;
#           define      syntaxes_chosen 1
#           define      syntax_chosen 2
#           define      presentation_context_id_chosen 3
#           define      context_negotiation_chosen 4
#           define      transfer_syntax_chosen 5
```

```
#           define      fixed_chosen 6
        union {
            struct _seq1 {
                ObjectID        abstract;
                ObjectID        transfer;
            } syntaxes;  /* to choose, set choice to syntaxes_chosen */
            ObjectID        syntax;  /* to choose, set choice to
                                      * syntax_chosen */
            int             presentation_context_id;  /* to choose, set
                                                       * choice to
                                    * presentation_context_id_chosen */
            struct _seq2 {
                int             presentation_context_id;
                ObjectID        transfer_syntax;
            } context_negotiation;  /* to choose, set choice to
                                     * context_negotiation_chosen */
            ObjectID        transfer_syntax;  /* to choose, set choice
                                               * to transfer_syntax_chosen */
            Nulltype        fixed;  /* to choose, set choice to
                                                     * fixed_chosen */
        } u;
    } identification;
    char            *data_value_descriptor;  /* NULL for not present */
    struct _seq3 {
        /* ContentsConstraint is applied to data_value */
        struct {
            unsigned int    length;
            unsigned char   *value;
        } encoded;
        A_integer       *decoded;
    } data_value;
} A;
```

Representation without `-compat bad1994ExternalWithContentsConstr` **specified:**

```
typedef struct A {
    unsigned char   bit_mask;
#       define      direct_reference_present 0x80
#       define      indirect_reference_present 0x40
    ObjectID        direct_reference;  /* optional; set in bit_mask
                                        * direct_reference_present if
                                        * present */
    int             indirect_reference;  /* optional; set in bit_mask
                                          * indirect_reference_present
                                          * if present */
    char            *data_value_descriptor;  /* NULL for not present */
    struct _choice1 {
        unsigned short  choice;
#           define      single_ASN1_type_chosen 1
#           define      octet_aligned_chosen 2
#           define      arbitrary_chosen 3
        union {
            OpenType        single_ASN1_type;  /* to choose, set choice
                                               * to single_ASN1_type_chosen */
            struct External_octet_aligned {
                /* ContentsConstraint is applied to octet_aligned */
                struct {
                    unsigned int    length;
```

```
                    unsigned char   *value;
                } encoded;
                A_integer       *decoded;
            } octet_aligned;  /* to choose, set choice to
                               * octet_aligned_chosen */
            struct External_arbitrary {
                unsigned int    length;  /* number of significant
                                          * bits */
                unsigned char   *value;
            } arbitrary;     /* to choose, set choice to arbitrary
                              *_chosen */
        } u;
    } encoding;
} A;
```

The `-compat bad1994ExternalWithContentsConstr` flag also causes the 8.0.0 representation of the `single-ASN1-type` field from the EXTERNAL type with an extra typedef to be generated in the helper mode.

For example:

ASN.1:

```
B ::= EXTERNAL
```

Representation with `-helperNames` and `-compat bad1994ExternalWithContentsConstr` specified:

```
typedef struct External {
    unsigned char   bit_mask;
#       define      indirect_reference_present 0x80
    struct _OID     *direct_reference;  /* NULL for not present */
    int             indirect_reference;  /* optional; set in bit_mask
                                          * indirect_reference_present
                                          * if present */
    char            *data_value_descriptor;  /* NULL for not present */
    struct External_encoding *encoding;
} External;

typedef struct External_encoding {
    unsigned short  choice;
#       define      single_ASN1_type_chosen 1
#       define      octet_aligned_chosen 2
#       define      arbitrary_chosen 3
    union {
        OpenType        *single_ASN1_type;  /* to choose, set choice to
                                             * single_ASN1_type_chosen */
        External_octet_aligned *octet_aligned;  /* to choose, set
                                    * choice to octet_aligned_chosen */
        External_arbitrary *arbitrary;  /* to choose, set choice to
                                         * arbitrary_chosen */
    } u;
} External_encoding;
```

Representation with `-helperNames` and without `-compat bad1994ExternalWithContentsConstr` specified:

```
typedef struct External_encoding {
    unsigned short  choice;
#       define      single_ASN1_type_chosen 1
#       define      octet_aligned_chosen 2
#       define      arbitrary_chosen 3
    union {
        OpenType      *single_ASN1_type;  /* to choose, set choice to
                                           * single_ASN1_type_chosen */
        External_octet_aligned *octet_aligned;  /* to choose, set
                                     * choice to octet_aligned_chosen */
        External_arbitrary *arbitrary;  /* to choose, set choice to
                                         * arbitrary_chosen */
    } u;
} External_encoding;
```

3.5.8 -compat badDefineNamesForComponentsOfFields

Usually the fields included into a type that uses a `COMPONENTS OF` statement inherits directives applied to those fields in the original type. Sometimes code is incorrectly generated due to naming directives that affect the generated names and could result in name conflicts, as seen the following example:

```
M DEFINITIONS ::=
BEGIN
 T1 ::= [0] SET {
     f1 INTEGER OPTIONAL
 }
 T2 ::= [1] SET {
     f0 BOOLEAN OPTIONAL,
     COMPONENTS OF T1
 }
END
--<OSS.PDU M.T1>--
--<OSS.FIELDNAME M.T1.f1 "T1_f1">--
```

Since field `T2.f1` inherits the `FIELDNAME` directive applied to `T1.f1`, the generated output contains two definitions of the `T1_f1_present` bitmask bit:

```
typedef struct T1 {
    unsigned char   bit_mask;
#       define      T1_f1_present 0x80
    int             T1_f1;  /* optional; set in bit_mask T1_f1_present if
                             * present */
} T1;

typedef struct T2 {
    unsigned char   bit_mask;
#       define      f0_present 0x80
#       define      T1_f1_present 0x40
    ossBoolean      f0;  /* optional; set in bit_mask f0_present if
present */
    int             T1_f1;  /* optional; set in bit_mask T1_f1_present if
                             * present */
} T2;
```

Starting with version 8.3.0, the ASN.1/C compiler generates separate define names when the define values differ, such as in this example:

```
typedef struct T1 {
    unsigned char   bit_mask;
#       define      T1_f1_present 0x80
    int             T1_f1;  /* optional; set in bit_mask T1_f1_present if
                             * present */
} T1;

typedef struct T2 {
    unsigned char   bit_mask;
#       define      f0_present 0x80
#       define      T2_T1_f1_present 0x40
    ossBoolean      f0;  /* optional; set in bit_mask f0_present if
present */
    int             T1_f1;  /* optional; set in bit_mask T2_T1_f1_present
if
                             * present */
} T2;
```

The compat flag `badDefineNamesForComponentsOfFields` restores the old compiler behavior.

3.5.9 -compat badConflictingEnumeratedNames

When some combination of names and directives is used in the ASN.1 syntax the compiler could generate duplicate names for enumeration values and definitions, as seen in the example that follows.

Example:

ASN.1:

```
TEST DEFINITIONS ::= BEGIN
    Enum ::= ENUMERATED { rrc-uea0 }
    BitStr ::= BIT STRING { rrc-uea0(2) }
    Ch ::= CHOICE {
        alt1    INTEGER,
        notUsed NULL
    }
END
--<ASN1.Nickname TEST.Enum.rrc-uea0 rrc_uea0>--
--<OSS.DefineName TEST.Ch.notUsed "rrc_PenaltyTime_ECN0_notUsed">--
```

The generated code contains the enumerated value `rrc_uea0` and a preprocessor definition with the same name:

```
typedef enum Enum {
    rrc_uea0 = 0
} Enum;
typedef struct BitStr {
    unsigned short  length;  /* number of significant bits */
    unsigned char   *value;
```

```
} BitStr;
#define                     rrc_uea0 0x20
#define                     rrc_uea0_byte 0
```

Starting with version 8.3, the ASN.1/C compiler generates a separate define name for the named bit `BitStr.rrc-uea0`:

```
typedef struct BitStr {
    unsigned short  length;  /* number of significant bits */
    unsigned char   *value;
} BitStr;
#define                     BitStr_rrc_uea0 0x20
#define                     BitStr_rrc_uea0_byte 0
```

The `badConflictingEnumeratedNames` compat flag restores the old compiler behavior.

3.5.10 -compat badExternalPrefix

Prior to version 4.0, the ASN.1 compiler prefixed the names of the `octet_aligned` and `arbitrary` C `structs` located in the EXTERNAL type with an underscore. Starting with version 4.0, the names for these two structures are no longer prefixed with an underscore. Instead, they are renamed `External_octet_aligned` and `External_arbitrary`.

The `-compat badExternalPrefix` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
MyExtern ::= EXTERNAL
```

Representation with `-compat badExternalPrefix` specified:

```
typedef struct External {
. . . .
    struct _octet1 {. . . .} octet_aligned;
    struct _bit1   {. . . .} arbitrary;
. . . .
} External;

typedef External        MyExtern;
```

Representation without `-compat badExternalPrefix` specified :

```
typedef struct External {
. . . .
    struct External_octet_aligned {. . . .} octet_aligned;
    struct External_arbitrary     {. . . .} arbitrary;
. . . .
} External;
```

```
typedef External        MyExtern;
```

3.5.11 -compat badLengthDirectives

Prior to version 4.0, the ASN.1 compiler allowed the global OSS.SHORT, OSS.INT, and OSS.LONG compiler directives to affect the `length` and `count` fields of compiler-generated structures. Starting with version 4.0, these two fields are no longer affected by the above mentioned global compiler directives.

The `-compat badLengthDirectives` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
--<OSS.SHORT>--

Module DEFINITIONS ::= BEGIN
        MyOctString ::= OCTET STRING
END
```

Representation with `-compat badLengthDirectives` specified:

```
typedef struct MyOctString {
    unsigned short  length;
    unsigned char   *value;
} MyOctString;
```

Representation starting without `-compat badLengthDirectives` specified:

```
typedef struct MyOctString {
    unsigned int    length;
    unsigned char   *value;
} MyOctString;
```

3.5.12 -compat badNameConflicts

Prior to version 4.1, the ASN.1 compiler attempted to disambiguate some names even when not required. For example, if multiple identical NamedBitLists were defined for different types, the sets of `#define`s for the bits in the NamedBitList would all have their identifiers prefixed with the type name of the declared BIT STRING. Starting with version 4.1, only enough names are altered to achieve disambiguation. So, the last set of `#define`s does not have its identifiers prefixed with the declared type's name.

The `-compat badNameConflicts` option disables the new behavior providing compatibility with version numbers lower than 4.1.

Example 1:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
 Name17 ::= [17] BIT STRING  { serena(0), melissa(20),
                               ansara(40) }
 name17  Name17 ::= { serena, ansara }

 Name18 ::= [18] BIT STRING  { serena(0), melissa(20),
                               ansara(40) }
 name18  Name18 ::= { serena, ansara }

 Name21 ::= [21] BIT STRING { serena(5), melissa(10),
                               ansara(30) }
 name21  Name21 ::= { serena , ansara }
END
```

Representation with `-compat v4.2.0` and `-compat badNameConflicts` specified:

```
typedef unsigned char   Name17[6];
#define                     Name17_serena 0x80
#define                     Name17_serena_byte 0
#define                     Name17_melissa 0x08
#define                     Name17_melissa_byte 2
#define                     Name17_ansara 0x80
#define                     Name17_ansara_byte 5

typedef unsigned char   Name18[6];
#define                     Name18_serena 0x80
#define                     Name18_serena_byte 0
#define                     Name18_melissa 0x08
#define                     Name18_melissa_byte 2
#define                     Name18_ansara 0x80
#define                     Name18_ansara_byte 5

typedef unsigned int    Name21;
#define                     Name21_serena 0x04000000
#define                     Name21_melissa 0x00200000
#define                     Name21_ansara 0x00000002

extern Name17 name17;

extern Name18 name18;

extern Name21 name21;
```

Representation without `-compat badNameConflicts` and with `-compat v4.2.0` specified:

```
typedef unsigned char   Name17[6];
#define                     Name17_serena 0x80
#define                     Name17_serena_byte 0
#define                     Name17_melissa 0x08
#define                     Name17_melissa_byte 2
#define                     Name17_ansara 0x80
#define                     Name17_ansara_byte 5
```

```
typedef unsigned char   Name18[6];
#define                     Name18_serena 0x80
#define                     Name18_serena_byte 0
#define                     Name18_melissa 0x08
#define                     Name18_melissa_byte 2
#define                     Name18_ansara 0x80
#define                     Name18_ansara_byte 5

typedef unsigned int    Name21;
#define                     serena 0x04000000
#define                     melissa 0x00200000
#define                     ansara 0x00000002

extern Name17 name17;

extern Name18 name18;

extern Name21 name21;
```

Example 2:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN
       Name ::= BIT STRING { serena(0), melissa(50)}
       melissa-byte INTEGER ::= 110
    END
```

Representation with `-compat badNameConflicts` but without `-compat v4.2.0` specified:

```
#define                     serena 0x80
#define                     serena_byte 0
#define                     Name_melissa 0x20
#define                     Name_melissa_byte 6

extern int melissa_byte_1;
```

Representation without `-compat badNameConflicts` and without `-compat v4.2.0` specified:

```
#define                     serena 0x80
#define                     serena_byte 0
#define                     Name_melissa 0x20
#define                     Name_melissa_byte 6

extern int melissa_byte;
```

3.5.13 -compat badPointerTypesWithNestedContConstr

Prior to version 8.1, the ASN.1 compiler generated extra pointers for structures with helper names derived from ASN.1 types with ContentsConstraint within InnerSubtypes. This behavior was not consistent with other generated structures in the helper mode.

The compat flag '`badPointerTypesWithNestedContConstr`', which is a part of the version compat flag v8.1.0, restores the old structures for types with nested ContentsConstraint.

Example:

ASN.1:

```
V ::= SEQUENCE OF OCTET STRING
A ::= SEQUENCE { a1 V (WITH COMPONENT (CONTAINING INTEGER )) }
```

Representation with `-compat bad1994ExternalWithContentsConstr` specified:

```
typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;

typedef int             A_integer;

typedef struct A_a1_seq {
    /* ContentsConstraint is applied to A_a1_seq */
    _OctStr         encoded;
    A_integer       *decoded;
} *A_a1_seq;

typedef struct A_a1 {
    struct A_a1_node *head;
    struct A_a1_node *tail;
    unsigned int    count;
} *A_a1;

typedef struct A_a1_node {
    struct A_a1_node *next;
    struct A_a1_node *prev;
    struct A_a1_seq *value;
} A_a1_node;

typedef struct A {
    struct A_a1     *a1;
} A;
```

Representation without `-compat bad1994ExternalWithContentsConstr` specified:

```
typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;

typedef struct A {
    struct A_a1     *a1;
} A;
```

```
typedef int             A_integer;

typedef struct A_a1_seq {
    /* ContentsConstraint is applied to A_a1_seq */
    _OctStr         encoded;
    A_integer       *decoded;
} *A_a1_seq;

typedef struct A_a1 {
    struct A_a1_node *head;
    struct A_a1_node *tail;
    unsigned int    count;
} A_a1;

typedef struct A_a1_node {
    struct A_a1_node *next;
    struct A_a1_node *prev;
    struct A_a1_seq *value;
} A_a1_node;
```

3.5.14 -compat badRefNames

Prior to version 5.0.8 for tagged types with the OSS.POINTER directive, the ASN.1 compiler incorrectly generated the base type's name instead of the actual tagged type's name when the base type itself was also a tagged type. Starting with version 5.0.8, the actual name for all tagged types is generated in the header file.

The `-compat badRefNames` option disables the new behavior providing compatibility with version numbers lower than 5.0.8.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN
   ProtocolIdentifier  ::= A
   A ::= SET OF INTEGER
   Alerting-UUIE ::= SEQUENCE {
      protocolIdentifier ProtocolIdentifier --<POINTER>--
   }
END
```

Representation with `-compat badRefNames` specified:

```
typedef struct Alerting_UUIE {
   struct A        **protocolIdentifier;
} Alerting_UUIE;
```

Representation starting without `-compat badRefNames` specified:

```
typedef struct Alerting_UUIE {
   struct ProtocolIdentifier *protocolIdentifier;
} Alerting_UUIE;
```

3.5.15 -compat badSetOfOidWithPointer

Prior to version 5.0.4, the ASN.1 compiler incorrectly handled the global OSS.POINTER directive applied to the SET OF, SEQUENCE OF, and OBJECT IDENTIFIER types when it appeared after some other global directives (i.e. OSS.UNBOUNDED, OSS.ARRAY, OSS.LINKED, and DLINKED). The incorrect behavior was to disregard the effect of the latter directives if OSS.POINTER was specified after them. Starting with version 5.0.4, the use of OSS.POINTER on a type does not disable the effect of the OSS.UNBOUNDED, OSS.ARRAY, OSS.LINKED, and DLINKED directives.

The `-compat badSetOfOidWithPointer` option disables the new behavior providing compatibility with version numbers lower than 5.0.4.

Example:

ASN.1:

```
--<OSS.ARRAY OBJECT IDENTIFIER>--
--<OSS.POINTER OBJECT IDENTIFIER>--

Module DEFINITIONS ::= BEGIN
   O ::= OBJECT IDENTIFIER
END
```

Representation with `-compat badSetOfOidWithPointer` specified:

```
typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} *ObjectID;

typedef struct ObjectID *O;
```

Representation starting without `-compat badSetOfOidWithPointer` specified:

```
typedef struct ObjectID {
    unsigned short  count;
    unsigned short  value[1];
} *ObjectID;

typedef struct ObjectID *O;
```

3.5.16 -compat badSharingForTypesWithInlineTypeDir

Prior to version 8.4.0, the ASN.1 compiler correctly shared similar types to which `OSS.InlineType` directives were applied, but incorrectly shared types when only one of the types had an `OSS.InlineType` directive. Prior to version 8.5.0, the problem of ignoring the `OSS.InlineType` directives was fixed, but it also forbade sharing of similar types with `OSS.InlineType` directives.

The `-compat badSharingForTypesWithInlineTypeDir` option is a stand-alone flag that restores the incorrect behavior of the 8.4.0 version.

Example:

ASN.1:

```
--<OSS.InlineType M.T2.list.*>--
--<OSS.InlineType M.T1.list.*>--
--<OSS.UseThis M.T1.list M.T2.list>--

M DEFINITIONS AUTOMATIC TAGS ::= BEGIN
  T1 ::= SEQUENCE { list SEQUENCE OF CHOICE { f INTEGER } }
  T2 ::= SEQUENCE { list SEQUENCE OF CHOICE { f INTEGER } }
END
```

Representation without `-compat badSharingForTypesWithInlineTypeDir` specified:

```
typedef struct _seqof1 {
    struct _seqof1  *next;
    struct _choice1 {
        unsigned short  choice;
#           define      f_chosen 1
        union {
            int             f;  /* to choose, set choice to f_chosen */
        } u;
    } value;
} *_seqof1;

typedef struct T1 {
    struct _seqof1  *list;
} T1;

typedef struct T2 {
    struct _seqof1  *list;
} T2;
```

Representation with `-compat badSharingForTypesWithInlineTypeDir` specified:

The compiler prints the following warnings:

```
"test.asn", line 3 (M): C0567W: The OSS.UseThis directive was not applied to
type 'M.T1.list' because its C representation differs from those of type
'M.T2.list'.

C0492I: There are unused standard directives.  Specify the -gendirectives
command line option, then look at all lines containing "WARNING:" in the
generated .gen file.
```

```
typedef struct T1 {
    struct _seqof1 {
        struct _seqof1  *next;
        struct {
            unsigned short  choice;
#               define      f_chosen 1
            union {
```

```
                int             f; /* to choose, set choice to f_chosen */
            } u;
        } value;
    } *list;
} T1;

typedef struct T2 {
    struct _seqof2 {
        struct _seqof2  *next;
        struct {
            unsigned short  choice;
#               define      f_chosen 1
            union {
                int             f; /* to choose, set choice to f_chosen */
            } u;
        } value;
    } *list;
} T2;
```

3.5.17 -compat badTypedefsForUserNames

Prior to version 5.0, the ASN.1 compiler generated extra `typedef`s in the header file for names specified with the OSS.TYPENAME directive applied to fields defined with tagged types which had the OSS.ExtractType directive applied to them. Starting with version 5.0, these extra `typedef`s are no longer generated.

The `-compat badTypedefsForUserNames` option disables the new behavior providing compatibility with version numbers lower than 5.0.

Example:

ASN.1:

```
--<OSS.ExtractType Mod.Type1.nested>--

Mod DEFINITIONS ::= BEGIN
        Type1 ::= SET {
                nested SetOfInt --<TYPENAME "MyIntSet">--
        }
        SetOfInt ::= SET OF INTEGER
END
```

Representation with `-compat badTypedefsForUserNames` specified:

```
typedef struct SetOfInt *MyIntSet;

typedef struct Type1 {
    struct SetOfInt *nested;
} Type1;

typedef struct SetOfInt {
    struct SetOfInt *next;
    int             value;
```

```
} *SetOfInt;
```

Representation starting without `-compat badTypedefsForUserNames` **specified:**

```
typedef struct Type1 {
    struct SetOfInt *nested;
} Type1;

typedef struct SetOfInt {
    struct SetOfInt *next;
    int             value;
} *SetOfInt;
```

3.5.18 -compat badTYPENAMEdirectives

Prior to version 5.1, the ASN.1 compiler mistakenly allowed (without issuing a warning message) OSS.TYPENAME directives to be applied to referenced components within structured types. Although such TYPENAME directives did not actually affect the name of the specified component, they sometimes caused invalid information to be generated into the header file. Starting with version 5.1, such OSS.TYPENAME directives no longer cause any invalid information to be generated into the header file. Additionally, a warning message is issued when such a directive is detected.

The `-compat badTYPENAMEdirectives` option disables the new behavior providing compatibility with version numbers lower than 5.1.

Example:

ASN.1:

```
--<OSS.TYPENAME Mod.Alerting-UUIE.tokens.* "NewClearToken">--

      Mod DEFINITIONS AUTOMATIC TAGS ::= BEGIN
      ClearToken ::= INTEGER

      Alerting-UUIE ::= SEQUENCE {
            tokens      SEQUENCE OF ClearToken OPTIONAL
      }
      END
```

The header file representation with and without this `-compat` flag is usually the same.

3.5.19 -compat badUnboundedBitStringsWithNamedBits

Prior to version 8.6, the ASN.1 compiler incorrectly generated an unbounded representation with an 'unsigned short' 'length' field for BIT STRING types with NamedBits, if the maximum value of the named bits was less than USHORT_MAX. Starting with version 8.6, the ASN1 compiler generates an 'unsigned int' 'length' field for such types.

The `-compat badUnboundedBitStringsWithNamedBits` option restores the old behavior.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN
        B ::=  BIT STRING {bit(20)}
END
```

Representation without the `-compat badUnboundedBitStringsWithNamedBits` flag for release 8.6.0 or later:

```
typedef struct B {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} B;
#define                     bit 0x08
#define                     bit_byte 2
```

Representation with the `-compat badUnboundedBitStringsWithNamedBits` flag for release 8.6.0 or later:

```
typedef struct B {
    unsigned short    length;  /* number of significant bits */
    unsigned char   *value;
} B;
#define                     bit 0x08
#define                     bit_byte 2
```

3.5.20 -compat badUnderscorePrefix

Prior to version 4.0, the ASN.1 compiler prefixed internal names (such as the `value` field in UNBOUNDED structures) with an underscore when a user-defined name was found to be the same as these internal names. Starting with version 4.0, these internal names are no longer prefixed with an underscore when some user-defined variable has a name that is the same as an internal name (e.g., `count`, `next`, `length`, `prev`, `value`, etc.)

The `-compat badUnderscorePrefix` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
        YourOctStr ::= OCTET STRING

        MyOctStr ::= OCTET STRING
        value MyOctStr ::= 'FF'H
        length INTEGER ::= 10
END
```

Representation with `-compat badUnderscorePrefix` specified:

```
typedef struct YourOctStr {
    unsigned int    _length;
    unsigned char   *_value;
} YourOctStr;

typedef struct MyOctStr {
    unsigned int    _length;
    unsigned char   *_value;
} MyOctStr;

extern MyOctStr value;

extern int length;
```

Representation without `-compat badUnderscorePrefix` specified:

```
typedef struct YourOctStr {
    unsigned int    length;
    unsigned char   *value;
} YourOctStr;

typedef struct MyOctStr {
    unsigned int    length;
    unsigned char   *value;
} MyOctStr;

extern MyOctStr value;

extern int length;
```

3.5.21 -compat badValuePrefix

Prior to version 4.1, the ASN.1 compiler did not prefix ambiguous valuereferences with their containing module's name. Instead, these ambiguous valuereferences had a '_#' suffix added to them. Starting with version 4.1, all such ambiguous valuereferences are prefixed with the name of their containing module.

The `-compat badValuePrefix` option disables the new behavior providing compatibility with version numbers lower than 4.1.

Example:

ASN.1:

```
--<OSS.ROOT>--

Mod1 DEFINITIONS ::= BEGIN
        A ::= INTEGER
        a A ::= 20
END

Mod2 DEFINITIONS ::= BEGIN
```

```
        A ::= INTEGER
        a A ::= 21
END
```

Representation with `-compat badValuePrefix` specified:

```
typedef int             Mod1_A;

typedef int             Mod2_A;

extern Mod1_A a_1;

extern Mod2_A a_2;
```

Representation without `-compat badValuePrefix` specified:

```
typedef int             Mod1_A;

typedef int             Mod2_A;

extern Mod1_A Mod1_a;

extern Mod2_A Mod2_a;
```

3.5.22 -compat BMPleanUTF8String

Prior to version 7.0, the default representation of UTF8String for the LED runtime was `ossBMPString`, which is different from the SOED/TOED representation. Starting from 7.0, this default representation has been changed to the UNBOUNDED representation (`ossCharString`) to ease switching between the SOED/TOED library and the LED library.

The `-compat BMPleanUTF8String` option disables the new behavior providing compatibility with version numbers lower than 7.0.

Example:

ASN.1:

```
X1 DEFINITIONS ::= BEGIN
U ::= UTF8String
END
```

Representation without compat flag for release 7.0 or later (only with `-lean`):

```
typedef ossCharString U;
```

Representation with compat flag for release 7.0 or later (`-lean -compat BMPleanUTF8String`):

```
typedef ossBMPString U;
```

3.5.23 -compat charUTF8String

Prior to version 5.3.0, the ASN.1 compiler generated a character string made up of one-byte characters to represent a UTF8String type with a size constraint. This behavior may have caused problems for those who use multiple byte character strings. Starting with version 5.3.0, the ASN.1 compiler now allocates six-bytes for each character in a UTF8String.

The `-compat charUTF8String` option disables the new behavior providing compatibility with version numbers lower than 5.3.0.

Example:

ASN.1:

```
Mod1 DEFINITIONS ::= BEGIN
  U ::= UTF8String (SIZE(10))
END
```

Representation with `-compat charUTF8String` specified:

```
typedef char            U[11];
```

Representation without `-compat charUTF8String` specified:

```
typedef unsigned char           U[61];
```

3.5.24 -compat decoderUpdatesInputAddress

Prior to version 5.0, the `ossDecode()` function modified the address and length of the input buffer before returning to its calling function. This behavior may have been useful in decoding multiple concatenated PDUs from a single buffer. However, starting with version 5.0, the `ossDecode()` function no longer modifies the address and length of the input buffer.

The `-compat decoderUpdatesInputAddress` option disables the new behavior providing compatibility with version numbers lower than 5.0.

This `-compat` option does not change the representation of the input ASN.1 syntax in the header file.

3.5.25 -compat extensionWithMask

Prior to version 5.0, the ASN.1 compiler generated a bitmask for all elements appearing after an extensibility marker (i.e. '`...`'), even if the element was represented by a pointer. Starting with version 5.0, only non-pointer elements have a bitmask generated for them.

The `-compat extensionWithMask` option disables the new behavior providing compatibility with version numbers lower than 5.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
    A  ::= SEQUENCE {
        field1 INTEGER,
        field2 BOOLEAN,
        ... ,
        field3 REAL --<POINTER>--
    }
END
```

Representation with `-compat extensionWithMask` **specified:**

```
typedef struct A {
    unsigned char   bit_mask;
#       define      field3_present 0x80
    int             field1;
    ossBoolean      field2;
    double          *field3;  /* extension #1; set in bit_mask
                               * field3_present present */
} A;
```

Representation without `-compat extensionWithMask` **specified:**

```
typedef struct A {
    int             field1;
    ossBoolean      field2;
    double          *field3;  /* extension #1; NULL for not
                                 present */
} A;
```

3.5.26 -compat extraLinkedSETOFPointer

Prior to version 4.2, the ASN.1 compiler generated a double pointer (**) to represent a circular reference to a SEQUENCE OF or SET OF type which had a SizeConstraint and used the LINKED representation (due to a global OSS.LINKED directive). Starting with version 4.2, only a single pointer is used for such a type.

The `-compat extraLinkedSETOFPointer` option disables the new behavior providing compatibility with version numbers lower than 4.2.

Example:

ASN.1:

```
--<OSS.LINKED SET OF>--

Circle DEFINITIONS --<PDU>-- ::= BEGIN
   Circ ::= SET (SIZE(10)) OF SEQUENCE {b Circ}
```

```
END
```

Representation with `-compat extraLinkedSETOFPointer` **specified:**

```
typedef struct Circ {
    struct Circ     *next;
    struct {
        struct Circ     **b;
    } value;
} *Circ;
```

Representation without `-compat extraLinkedSETOFPointer` **specified:**

```
typedef struct Circ {
    struct Circ     *next;
    struct {
        struct Circ     *b;
    } value;
} *Circ;
```

3.5.27 -compat extraNameShortening

Prior to version 4.2, the ASN.1 compiler unnecessarily shortened names for certain types. Starting with version 4.2, this unnecessary shortening of names is no longer done.

The `-compat extraNameShortening` option disables the new behavior providing compatibility with version numbers lower than 4.2.

3.5.28 -compat extSizeNotUnbounded

Prior to version 4.2, the ASN.1 compiler generated a fixed-length array for character string, SEQUENCE OF, and SET OF types which had an extensible size constraint. Starting with version 4.2, such types are represented using the UNBOUNDED representation.

The `-compat extSizeNotUnbounded` option disables the new behavior providing compatibility with version numbers lower than 4.2. Note: starting with version 6.0 of the ASN.1/C compiler, this option has no effect unless the `-compat 5.4.0stringrepresentations` option is also specified.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
        String1 ::= PrintableString (SIZE(12, ...))
END
```

Representation with `-compat extSizeNotUnbounded` **specified:**

```
typedef char            String1[13];
```

Representation without `-compat extSizeNotUnbounded` **specified:**

```
typedef struct String1 {
    unsigned short  length;
    char            *value;
} String1;
```

The later representation is the UNBOUNDED one.

3.5.29 -compat generateInlineDeeplyNestedTypes

Prior to version 9.0, the ASN.1 compiler for deeply nested ASN.1 types generated a C-type definition that was uncompilable by the Microsoft Visual C/C++ compiler. This was because the nesting level of inline structures exceeded 15, which is the C compiler implementation limit. Starting with 9.0, the compiler generates an intermediate typedef to decrease the nesting level of generated inline structures.

The `-compat generateInlineDeeplyNestedTypes` option can be used to disable the generation of the intermediate typedef, providing compatibility with version numbers lower than 9.0.

Example:

ASN.1:

```
M DEFINITIONS ::= BEGIN
  S ::= SEQUENCE {
    f1 SEQUENCE {
      f2 SEQUENCE {
        f3 SEQUENCE {
          f4 SEQUENCE {
            f5 SET {
              f6 SEQUENCE {
                f7 SET {
                  f8 SEQUENCE {
                    f9 SET {
                      f10 SEQUENCE {
                        f11 SET {
                          f12 SEQUENCE {
                            f13 SET {
                              f14 SEQUENCE {
                                f15 SET {
                                  i INTEGER
                                }
                              }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
```

```
            }
          }
        }
      }
    }
}
END
```

Representation with `-compat generateInlineDeeplyNestedTypes` **specified:**

```
typedef struct S {
    struct {
        struct {
            struct {
                struct {
                    struct {
                        struct {
                            struct {
                                struct {
                                    struct {
                                        struct {
                                            struct {
                                                struct {
                                                    struct {
                                                        struct {
                                                            int i;
                                                        } f15;
                                                    } f14;
                                                } f13;
                                            } f12;
                                        } f11;
                                    } f10;
                                } f9;
                            } f8;
                        } f7;
                    } f6;
                } f5;
            } f4;
        } f3;
    } f2;
} f1;
} S;
```

Representation without `-compat generateInlineDeeplyNestedTypes` **specified:**

```
typedef struct _seq1 {
    struct {
        struct {
            struct {
                struct {
                    struct {
                        int  i;
                    } f15;
                } f14;
            } f13;
        } f12;
```

```
    } f11;
} _seq1;

typedef struct S {
    struct {
        struct {
            struct {
                struct {
                    struct {
                        struct {
                            struct {
                                struct {
                                    struct {
                                        struct {
                                            _seq1  f10;
                                        } f9;
                                    } f8;
                                } f7;
                            } f6;
                        } f5;
                    } f4;
                } f3;
            } f2;
        } f1;
} S;
```

3.5.30 -compat ignore1994ExternalConstr

Prior to versions 8.0.1/8.1, the ASN.1 compiler ignored constraints applied to the fields within a 1994 EXTERNAL type using `MultipleTypeConstraint`. This behavior has been changed. Now the runtime is able to identify values that violate such constraints.

The `-compat ignore1994ExternalConstr` option disables the new behavior providing compatibility with version numbers lower than 8.0.1/8.1. The ASN.1 compiler gives names to some nested generated structures if ValueConstraints are present. The `-compat ignore1994ExternalConstr` flag instructs the compiler to generate the old unnamed structure for the `encoding` field within the `EXTERNAL` type.

Example:

ASN.1:

```
A ::= EXTERNAL ( WITH COMPONENTS {
        identification (WITH COMPONENTS { presentation-context-id
                                                    PRESENT}),
        data-value-descriptor ABSENT,
        data-value (CONTAINING INTEGER ) } )

a A ::= {identification syntax : {1 2 3},
        data-value CONTAINING 64}
```

Representation with `-compat ignore1994ExternalConstr` specified:

```
typedef struct A {
    unsigned char   bit_mask;
```

```
#       define      direct_reference_present 0x80
#       define      indirect_reference_present 0x40
    ObjectID        direct_reference;  /* optional; set in bit_mask
                                        * direct_reference_present if
                                        * present */
    int             indirect_reference;  /* optional; set in bit_mask
                                         * indirect_reference_present
                                         * if present */
    char            *data_value_descriptor;  /* NULL for not present */
    struct {
        unsigned short  choice;
#           define      single_ASN1_type_chosen 1
#           define      octet_aligned_chosen 2
#           define      arbitrary_chosen 3
        union {
            OpenType        single_ASN1_type;  /* to choose, set choice
                                               * to single_ASN1_type_chosen */
            struct External_octet_aligned {
                /* ContentsConstraint is applied to octet_aligned */
                struct {
                    unsigned int    length;
                    unsigned char   *value;
                } encoded;
                A_integer       *decoded;
            } octet_aligned;  /* to choose, set choice to
                               * octet_aligned_chosen */
            struct External_arbitrary {
                unsigned int    length;  /* number of significant
                                          * bits */
                unsigned char   *value;
            } arbitrary;  /* to choose, set choice to arbitrary
                                                    *_chosen */
        } u;
    } encoding;
} A;
```

The structure generated for the field `encoding` **gets** `_choice1` **name without** `-compat ignore1994ExternalConstr`**:**

```
typedef struct A {
    unsigned char   bit_mask;
#       define      direct_reference_present 0x80
#       define      indirect_reference_present 0x40
    ObjectID        direct_reference;  /* optional; set in bit_mask
                                        * direct_reference_present if
                                        * present */
    int             indirect_reference;  /* optional; set in bit_mask
                                         * indirect_reference_present
                                         * if present */
    char            *data_value_descriptor;  /* NULL for not present */
    struct _choice1 {
        unsigned short  choice;
#           define      single_ASN1_type_chosen 1
#           define      octet_aligned_chosen 2
#           define      arbitrary_chosen 3
        union {
            OpenType        single_ASN1_type;  /* to choose, set choice
                                               * to single_ASN1_type_chosen */
```

```
                struct External_octet_aligned {
                    /* ContentsConstraint is applied to octet_aligned */
                    struct {
                        unsigned int    length;
                        unsigned char   *value;
                    } encoded;
                    A_integer       *decoded;
                } octet_aligned;  /* to choose, set choice to
                                   * octet_aligned_chosen */
                struct External_arbitrary {
                    unsigned int    length;  /* number of significant
                                              * bits */
                    unsigned char   *value;
                } arbitrary;  /* to choose, set choice to arbitrary
                               *_chosen */
            } u;
        } encoding;
    } A;
```

Note that using files generated with the `-compat ignore1994ExternalConstr` option in your application will not result in the following runtime error:

```
E0065S: Absence constraint violated; check field 'direct-reference' (type:
OBJECT IDENTIFIER) of PDU #2 'A'.
```

3.5.31 -compat ignoreInlineTypeDirForSharedTypes

Prior to version 8.4, the ASN.1 compiler ignored the `InlineType` directive applied to a type nested in a shared one. Starting with version 8.4, this behavior has been changed.

The `-compat ignoreInlineTypeDirForSharedTypes` option disables the new behavior providing compatibility with pre-8.4 compilers.

Example:

ASN.1:

```
--<OSS.InlineType M.T2.list.*>--
M DEFINITIONS AUTOMATIC TAGS ::= BEGIN
  T1 ::= SEQUENCE { list SEQUENCE OF CHOICE { f INTEGER } }
  T2 ::= SEQUENCE { list SEQUENCE OF CHOICE { f INTEGER } }
END
```

Representation with `-compat ignoreInlineTypeDirForSharedTypes` specified:

```
typedef struct _choice1 {
    unsigned short  choice;
#       define      f_chosen 1
    union {
        int             f;  /* to choose, set choice to f_chosen */
    } u;
} _choice1;
```

```
typedef struct _seqof1 {
    struct _seqof1  *next;
    _choice1        value;
} *_seqof1;

typedef struct T1 {
    struct _seqof1  *list;
} T1;

typedef struct T2 {
    struct _seqof1  *list;
} T2;
```

Representation without `-compat ignoreInlineTypeDirForSharedTypes` **specified**:

```
typedef struct _choice1 {
    unsigned short  choice;
#       define      f_chosen 1
    union {
        int             f;  /* to choose, set choice to f_chosen */
    } u;
} _choice1;

typedef struct T1 {
    struct _seqof1 {
        struct _seqof1  *next;
        _choice1        value;
    } *list;
} T1;

typedef struct T2 {
    struct _seqof2 {
        struct _seqof2  *next;
        struct {
            unsigned short  choice;
#               define      f_chosen 1
            union {
                int             f;  /* to choose, set choice to
f_chosen */
            } u;
        } value;
    } *list;
} T2;
```

3.5.32 -compat ignoreNicknamesInConflictResolution

Prior to version 7.0, the ASN1 compiler disambiguated names generated for the `#define` constant or elements of an enum type even if the `ASN1.Nickname` directive that contains another (not conflicted) name is applied to one of the disputed names. As a result another name (to which the ASN1.Nickname was not applied) was mangled in spite of the absence of a conflict.

The `-compat ignoreNicknamesInConflictResolution` option can be used to disable the above-mentioned changes providing compatibility with version numbers lower than 7.0

ASN.1 Example:

```
--<ASN1.Nickname M.FirstEnum.one is_one>--
--<ASN1.Nickname M.BtStr.first first_bit>--
M DEFINITIONS ::= BEGIN

FirstEnum ::= ENUMERATED {one}
SecondEnum ::= ENUMERATED {one}

BtStr ::= BIT STRING {first(0)}

Int ::= INTEGER {first(100)}

END
```

Representation with `-compat ignoreNicknamesInConflictResolution` **specified:**

```
typedef enum FirstEnum {
    is_one = 0
} FirstEnum;

typedef enum SecondEnum {
    SecondEnum_one = 0
} SecondEnum;

typedef struct BtStr {
    unsigned short  length;  /* number of significant bits */
    unsigned char   *value;
} BtStr;
#define                     first_bit 0x80
#define                     first_bit_byte 0

typedef int             Int;
#define                     Int_first 100
```

Representation without `-compat ignoreNicknamesInConflictResolution` **specified**:

```
typedef enum FirstEnum {
    is_one = 0
} FirstEnum;

typedef enum SecondEnum {
    one = 0
} SecondEnum;

typedef struct BtStr {
    unsigned short  length;  /* number of significant bits */
    unsigned char   *value;
} BtStr;
#define                     first_bit 0x80
#define                     first_bit_byte 0

typedef int             Int;
#define                     first 100
```

3.5.33 -compat ignorePointerDirForLean

Starting with version 8.5, the ASN.1 compiler supports the `POINTER` directive for the Lean encoder/decoder.

The `-compat ignorePointerDirForLean` option can be used to ignore the effect of the `POINTER` directive for Lean, thus providing compatibility with Lean encoder/decoder runtime libraries of earlier versions.

Example:

ASN.1:

```
B1835 DEFINITIONS ::= BEGIN
   S ::= SEQUENCE { f INTEGER --<POINTER>-- }
END
```

The S type representation with `-compat ignorePointerDirForLean -lean` specified:

```
typedef struct S {
    OSS_INT32       f;
} S;
```

Note that the '`f`' field is generated with a non-pointered type and the ASN.1 compiler will issue a warning message:

```
C0541W: Directive POINTER is not supported and will be ignored.
   S ::= SEQUENCE { f INTEGER --<POINTER>-- }
```

as it has been issued by previous versions of the ASN.1 compiler.

The S type representation with `-lean` specified:

```
typedef struct S {
    OSS_INT32       *f;
} S;
```

Note that the '`f`' field type is a pointer to `OSS_INT32`.

3.5.34 -compat ignorePrefixForSpecialStructures

Prior to version 8.4, the ASN.1 compiler did not prefix the structures generated for ASN.1 types, such as CHARACTER STRING, EMBEDDED PDV and EXTERNAL, when the `-helperNames` and `-prefix` ASN.1 compiler options were specified. Starting with version 8.4, this behavior has been changed.

The `-compat ignorePrefixForSpecialStructures` option disables the new behavior, providing compatibility with pre-8.4 compilers.

Example:

ASN.1:

```
ASN1SPEC1 DEFINITIONS ::= BEGIN
  PDU1 ::= SEQUENCE {
    character-string CHARACTER STRING
  }
END
```

Representation with `-prefix Spec1_-compat ignorePrefixForSpecialStructures` **specified:**

```
typedef struct UnrestrictedChar_negotiation {
    int             presentation_context_id;
    struct Spec1__OID *transfer_syntax;
} UnrestrictedChar_negotiation;
```

Representation with `-prefix Spec1_` **without** `-compat ignorePrefixForSpecialStructures` **specified**:

```
typedef struct Spec1_UnrestrictedChar_negotiation {
    int             presentation_context_id;
    struct Spec1__OID *transfer_syntax;
} Spec1_UnrestrictedChar_negotiation;
```

3.5.35 -compat implicitTypeInArray

Prior to version 6.1.4, the ASN.1 compiler generated an implicit type definition for the CHOICE, SET OF and SEQUENCE OF type which is an element of the SET OF or SEQUENCE OF type in an ARRAY representation.

Starting with version 6.1.4, all such element types have their own typedefs.

The `-compat implicitTypeInArray` option disables the new behavior providing compatibility with version numbers prior to 6.1.4.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN

SeqOfChoice ::= [0] SEQUENCE --<ARRAY>-- OF
        CHOICE {
                        i       INTEGER,
                        s       IA5String
        }

SeqOfSet ::= [1] SEQUENCE --<ARRAY>-- OF
        SET OF INTEGER
END
```

Representation with `-compat implicitTypeInArray` specified:

```
typedef struct SeqOfChoice {
    unsigned int    count;
    struct {
        unsigned short  choice;
#           define      i_chosen 1
#           define      s_chosen 2
        union {
            int             i;  /* to choose, set choice to i_chosen */
            char            *s;  /* to choose, set choice to s_chosen */
        } u;
    } value[1];  /* first element of the array */
} *SeqOfChoice;

typedef struct SeqOfSet {
    unsigned int    count;
    struct _setof1 {
        struct _setof1  *next;
        int             value;
    } *value[1];  /* first element of the array */
} *SeqOfSet;
```

Representation without `-compat implicitTypeInArray` specified:

```
typedef struct _choice1 {
    unsigned short  choice;
#       define      i_chosen 1
#       define      s_chosen 2
    union {
        int             i;  /* to choose, set choice to i_chosen */
        char            *s;  /* to choose, set choice to s_chosen */
    } u;
} _choice1;

typedef struct SeqOfChoice {
    unsigned int    count;
    _choice1        value[1];  /* first element of the array */
} *SeqOfChoice;

typedef struct _setof1 {
    struct _setof1  *next;
    int             value;
} *_setof1;

typedef struct SeqOfSet {
    unsigned int    count;
    struct _setof1  *value[1];  /* first element of the array */
} *SeqOfSet;
```

3.5.36 -compat intEnums

Prior to version 3.6, the ASN.1 compiler initialized C `enum` variables with integers. Starting with version 3.6, such variables are initialized with enumerators. Additionally, you may use this `-compat` option to restrict the scope of generated `typedef`s for enumeration values.

The `-compat intEnums` option disables the new behavior providing compatibility with version numbers lower than 3.6.

Example:

ASN.1:

```
Level ::= ENUMERATED {low(0), medium(1), high(2)}
empty Level ::= low
```

Shared Representation and Declaration:

```
typedef enum Level {
    low = 0,
    medium = 1,
    high = 2
} Level;

extern Level empty;
```

Initialization prior to version 3.6:

```
Level empty = 0;
```

Initialization starting with version 3.6

```
Level empty = low;
```

Alias: `-compat 2`

3.5.37 -compat multipleUserFunctions

Prior to version 4.2, the ASN.1 compiler generated multiple constraint-checking functions for multiple CONSTRAINED BY clauses (even if these multiple CONSTRAINED BY clauses referred to the same type). Starting with version 4.2, only one constraint-checking function is generated for any particular type, even if the type is defined with more than one CONSTRAINED BY clause.

The `-compat multipleUserFunctions` option disables the new behavior providing compatibility with version numbers lower than 4.2.

Example:

ASN.1:

```
  Module DEFINITIONS ::= BEGIN
        SpecialNum ::= INTEGER (CONSTRAINED BY
                                  {-- must be a prime number--})
                                 (CONSTRAINED BY
                                {-- must be in the Fibonacci
--                                  series --})
END
```

Representation with `-compat multipleUserFunctions` specified:

```
typedef int             SpecialNum; /* must be a prime number*/
/* must be in the Fibonacci */
               /* series */

/* SpecialNum_fn is user-defined constraint function for ASN.1 item
 * Module.SpecialNum */
extern int DLL_ENTRY SpecialNum_fn(struct ossGlobal *, SpecialNum *,
                                   void **);
/* SpecialNum_fn1 is user-defined constraint function for ASN.1 item
 * Module.SpecialNum */
extern int DLL_ENTRY SpecialNum_fn1(struct ossGlobal *,SpecialNum *,
                                   void **);
```

Representation without `-compat multipleUserFunctions` specified:

```
typedef int             SpecialNum; /* must be a prime number*/
/* must be in the Fibonacci */
                /* series */
/* SpecialNum_fn is user-defined constraint function for ASN.1 item
 * Module.SpecialNum */
extern int DLL_ENTRY SpecialNum_fn(struct ossGlobal *, SpecialNum *,
                                   void **);
```

Note: Starting with version 5.0, user-defined constraint-checking functions are only generated if the `-userConstraints` compiler option is specified..

3.5.38 -compat nestUnions

Prior to version 4.0, the ASN.1 compiler did not necessarily generate a `typedef` for CHOICE types when the `-C++` option was specified. In addition, any `structs` nested inside a CHOICE type were generated inside the structure where they were nested, instead of being pulled out and `typedef`ined separately. Also, the `_union` tag was not added to the `union` containing the elements of the CHOICE. Starting with version 4.0, all CHOICE types get `typedef`s when the `-C++` option is specified, nested structures are pulled out and made into `typedef`s, and the `_union` tag is added to the `union` containing the elements of the CHOICE.

The `-compat nestUnions` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
        Data ::= SEQUENCE {
```

```
                a INTEGER,
                b CHOICE {
                        c BOOLEAN,
                        d SEQUENCE {
                          e IA5String,
                          f PrintableString
                        }
                }
        }
END
```

Representation with `-compat nestUnions` specified:

```
typedef struct Data {
    int             a;
    struct _choice1 {
        unsigned short  choice;
#           define      c_chosen 1
#           define      d_chosen 2
        union {
            ossBoolean      c;  /* to choose, set choice to
                                 *  c_chosen */
            struct _seq1 {
                char            *e;
                char            *f;
            } d;  /* to choose, set choice to d_chosen */
        } u;
    } b;
} Data;
```

Representation without `-compat nestUnions` specified:

```
typedef struct _choice1 {
    unsigned short  choice;
#       define      c_chosen 1
#       define      d_chosen 2
    union _union {
        ossBoolean      c;  /* to choose, set choice to
                             * c_chosen */
        struct _seq1 {
            char            *e;
            char            *f;
        } d;  /* to choose, set choice to d_chosen */
    } u;
} _choice1;

typedef struct Data {
    int             a;
    _choice1        b;
} Data;
```

Both of the outputs above were generated with the `-C++` option specified.

Alias: `-compat 3`

Related options:

`-compat intEnums` (see section 3.5.28)

3.5.39 -compat noASN.1Comments

Prior to version 5.4.0, the ASN.1 compiler did not generate transfer ASN.1 comment content to the generated header file. Starting with version 5.4.0, all ASN.1 comment content is by default transferred to the generated header file using valid C/C++ comment constructs. Refer to section 7.10 for more details about this topic.

The `-compat noASN.1Comments` option disables the new behavior providing you the ability to make the generated header file smaller in size.

3.5.40 -compat noBTypeValues

Prior to version 4.0, the ASN.1 compiler did not generate representations in the header file for SEQUENCE, SEQUENCE OF, SET or SET OF built-in types used in value assignments. In addition, the corresponding initializations were excluded from the generated `.c` file. Starting with version 4.0, these built-in types have representations generated for them in the header file and initializations made for them in the `.c` file.

The `-compat noBTypeValues` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   AsciiType ::= SEQUENCE OF IA5String
   success AsciiType ::= {"Address  found."}

   failure SEQUENCE OF PrintableString ::= {"Address found."}
END
```

Output with `-compat noBTypeValues` specified:
header file contains:

```
typedef struct AsciiType {
    struct AsciiType *next;
    char            *value;
} *AsciiType;

extern AsciiType success;
```

`.c` file contains:

```
static char _v1[] = "Address found.";
static struct AsciiType _v0[] = {
    {NULL, _v1}
};
AsciiType success = _v0;
```

Output without `-compat noBTypeValues` **specified:**
header file contains:

```
typedef struct AsciiType {
    struct AsciiType *next;
    char             *value;
} *AsciiType;

extern AsciiType success;

typedef struct _seqof1 {
    struct _seqof1  *next;
    char            *value;
} *_seqof1;
extern _seqof1 failure;
```

.c file contains:

```
static char _v1[] = "Address found.";
static struct AsciiType _v0[] = {
    {NULL, _v1}
};
AsciiType success = _v0;

static char _v3[] = "Address not found.";
static struct _seqof1 _v2[] = {
    {NULL, _v3}
};
_seqof1 failure = _v2;
```

3.5.41 -compat noConstDeclarations

The `-compat noConstDeclarations` option instructs the ASN.1 compiler to not generate const declarations for values of simple types. This flag is implied when the `-noDefines`, `-noStaticValues`, or `-test` option is specified.

Example:

ASN.1:

```
ASN.1:
B ::= BOOLEAN
b B ::= TRUE
```

Representation with `-compat noConstDeclarations` **specified:**

```
.h file:
extern B b;
.c file:
B b = TRUE;
```

Representation without `-compat noConstDeclarations` **specified:**

```
.h file:
extern const B b;
.c file:
const B b = TRUE;
```

3.5.42 -compat noDecoupledNames

Starting with version 5.0.0, the following name-mangling rules were introduced:

- Names generated for `#define` bitmask constants are mangled if they are in conflict with another generated name. However, the name of the component referenced by the bitmask constant is not mangled.
- Names of fields generated in a C `struct` are never mangled.
- All names which are in conflict with an OSS or C reserved word are always mangled.
- Names for `extern` type variables are only mangled if they are in conflict with the name of the control table or with another generated `extern` name.
- Both names involved in a conflict are mangled (unless one of the names is never mangled as mentioned above).

The `-compat noDecoupledNames` option can be used to disable the above-mentioned changes providing compatibility with version numbers lower that 5.0.0.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN
      Type1 ::= CHOICE { ambig BOOLEAN }
      Type2 ::= SEQUENCE { ambig INTEGER  OPTIONAL}
      ambig-present Type1 ::= ambig : TRUE
END
```

Representation with `-compat noDecoupledNames` specified:

```
typedef struct Type1 {
    unsigned short  choice;
#       define      ambig_chosen 1
    union {
        ossBoolean      ambig;  /* to choose, set choice to
                                   ambig_chosen */
    } u;
   } Type1;

   typedef struct Type2 {
    unsigned char   bit_mask;
#       define      Type2_ambig_present 0x80
    int             Type2_ambig;  /* optional; set in bit_mask
                                     * Type2_ambig_present if present */
```

```
} Type2;

extern Type1 Mod_ambig_present;
```

Note above how the fieldname `ambig` for `Type1` is not mangled while the one in `Type2` is. Also note how both of the conflicting names are mangled (i.e. `Type2.ambig` and the extern variable `ambig_present`.

Representation without `-compat noDecoupledNames` specified:

```
   typedef struct Type1 {
    unsigned short  choice;
#       define      ambig_chosen 1
    union {
      ossBoolean      ambig;  /* to choose, set choice to ambig_chosen */
    } u;
   } Type1;

   typedef struct Type2 {
    unsigned char   bit_mask;
#       define      Type2_ambig_present 0x80
    int             ambig;  /* optional; set in bit_mask Type2_ambig_present
if
                             * present */
   } Type2;

   extern Type1 ambig_present;
```

Note that neither the `extern` variable name nor the `ambig` field names are mangled in conformity with the introduced rules.

3.5.43 -compat noDefaultValues

Prior to version 4.0, the OSS decoder skipped components of SEQUENCE, and SET types marked as DEFAULT if no value was passed for them in the encoding (i.e. They were treated like OPTIONAL elements). Starting with version 4.0, any DEFAULT components which have no value passed for them are filled with their specified default value by the decoder.

The `-compat noDefaultValues` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Note: When the `-compat noDefaultValues` flag is used, it is possible that a bitmask field may not be generated for a DEFAULT element in a SEQUENCE or SET which is the first field specified in the definition.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
        DataCard ::= SEQUENCE {
                a     [0]  BOOLEAN DEFAULT TRUE ,
                b     [1]  BOOLEAN
        }
END
```

Representation with `-compat noDefaultValues` specified:

```
typedef struct DataCard {
    unsigned char   bit_mask;
#       define      a_present 0x80
    ossBoolean      a;  /* optional; set in bit_mask a_present
                           if present */
    ossBoolean      b;
} DataCard;
```

Representation without `-compat noDefaultValues` specified:

```
typedef struct DataCard {
    unsigned char   bit_mask;
#       define      a_present 0x80
    ossBoolean      a;  /* a_present not set in bit_mask
                         implies value is TRUE */
    ossBoolean      b;
} DataCard;
```

Alias: `-compat 8`

3.5.44 -compat noMacroArgumentPDUs

Prior to version 5.0, the ASN.1 compiler did not automatically treat macro arguments as PDUs. Starting with version 5.0, macro arguments are automatically considered to be PDUs.

The `-compat noMacroArgumentPDUs` option disables the new behavior providing compatibility with version numbers lower than 5.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   ERROR MACRO ::=
   BEGIN
      TYPE NOTATION ::= "PARAMETER" NamedType | empty
      VALUE NOTATION ::= value(VALUE INTEGER)
```

```
      NamedType ::= identifier type | type
   END

   invalidName ERROR PARAMETER
      reason BIT STRING {nameTooLong(1), illegalCharacter(2),
                         unspecified(3)}
   ::= 2
END
```

Representation with `-compat noMacroArgumentPDUs` specified:

No output files are generated and the following messages are printed to the standard output device:

```
"nomcrpdu.asn" (Module): C0220I: No PDUs in abstract syntax.

"nomcrpdu.asn" (Module): C0244I: No types to print, so header file was not
created.

"nomcrpdu.asn" (Module): C0213I: No PDUs to print, so control table was
not created.
```

Representation without `-compat noMacroArgumentPDUs` specified:

```
#define          InvalidName_PARAMETER_PDU 1

typedef struct InvalidName_PARAMETER {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} InvalidName_PARAMETER;
#define                     nameTooLong 0x40
#define                     nameTooLong_byte 0
#define                     illegalCharacter 0x20
#define                     illegalCharacter_byte 0
#define                     unspecified 0x10
#define                     unspecified_byte 0

extern int invalidName;
```

3.5.45 -compat noObjectSetsFromDummyObjects

Prior to version 5.2, the ASN.1 compiler did not generate extra `_OSET` declarations for instances of parameterized information object sets used in table constraints; additionally, these parameterized information object sets were not used to initialize table constraints. This caused some memory violation problems during runtime for certain applications. Starting with version 5.2, extra `_OSET` declaration are generated into the header file and table constraints are appropriately initialized.

The `-compat noObjectSetsFromDummyObjects` option disables the new behavior providing compatibility with version numbers lower than 5.2.

Example:

ASN.1:

```
Errors {OPERATION:Operations} ERROR ::= {Operations.&Errors}

ROS {InvokeId:InvokeIdSet, OPERATION:Invokable, OPERATION:Returnable}
     ::=  CHOICE {
             returnError [3]     ReturnError {{Errors{{Returnable}}}}
          }
ReturnError {ERROR:Errors} ::= SEQUENCE {
     errcode    ERROR.&errorCode ({Errors}),
     parameter  ERROR.&ParameterType
                     ({Errors}{@errcode}) OPTIONAL
}
```

3.5.46 -compat noOssterm

Prior to version 4.2, the ASN.1 compiler did not generate a call to the function `ossterm()` when the `-test` command-line option (see section 3.3.71) was used. Starting with version 4.2, a call to `ossterm()` is generated in the `.c` file when the `-test` command-line option is used.

The `-compat noOssterm` option disables the new behavior providing compatibility with version numbers lower than 4.2.

Note: The `ossterm()` function frees additional memory used by the memory manager when no more encoding/decoding is to be done by an application. For further information on this function, refer to the ***OSS ASN.1/C API Reference Manual***.

3.5.47 -compat noParamTypesharing

Prior to version 5.1, the ASN.1 compiler did not implement typesharing for parametrized types. Starting with version 5.1, an efficient typesharing algorithm is applied to parameterized types which share the same structure (but have different field names). This change does not effect the runtime processing of such types.

The `-compat noParamTypesharing` option disables the new behavior providing compatibility with version numbers lower than 5.1.

Example:

ASN.1:

```
A {Type} ::= SEQUENCE {a Type}
B ::= A {SET {a INTEGER} }
C ::= SET {
```

```
  b A {SEQUENCE {a INTEGER}}
}
```

Representation with `-compat noParamTypesharing` specified:

```
typedef struct _set1 {
    int             a;
} _set1;

typedef struct B {
    _set1           a;
} B;

typedef struct C {
    struct {
        _set1           a;
    } b;
} C;
```

Representation without `-compat noParamTypesharing` specified:

```
typedef struct _set1 {
    int             a;
} _set1;

typedef struct A {
    _set1           a;
} A;

typedef A B;

typedef struct C {
    A               b;
} C;
```

3.5.48 -compat noPduForContainedExternal

Starting with version 8.1.0, and prior to version 8.3.0, the ASN.1 compiler did not generate a PDU constant in the header file for an EXTERNAL type used in ContentsConstraint. Starting with version 8.3.0, the ASN.1 compiler now generates a PDU constant for an EXTERNAL type used in ContentsConstraint. The `-compat noPduForContainedExternal` flag disables the new behavior, which provides compatibility with version numbers 8.1.0 through 8.2.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
Param ::= BIT STRING  (CONTAINING EXTERNAL)
END
```

Output with `-compat noPduForContainedExternal` specified:
the header file contains:

```
#define          Param_PDU 1
```

Output without `-compat noPduForContainedExternal` **specified: the header file contains:**

```
#define          External_PDU 1
#define          Param_PDU 2
```

3.5.49 -compat noPDUsForImports

Prior to version 5.0, the ASN.1 compiler did not treat types imported into root modules as PDUs unless they were actually referenced from within a root module. Starting with version 5.0, types imported to root modules are treated as PDUs, even if they are not referenced from within a root module.

The `-compat noPDUsForImports` option disables the new behavior providing compatibility with version numbers lower than 5.0.

Example:

ASN.1:

mod1.ans (a non-root module):

```
Mod1 DEFINITIONS ::= BEGIN
   Mod1Data ::= SEQUENCE {
           type    ENUMERATED {random, sequential},
           address INTEGER
   }
END
```

mod2.asn (specified as a root module):

```
Mod2 DEFINITIONS ::= BEGIN
   IMPORTS Mod1Data FROM Mod1;

   Mod2Data ::= SEQUENCE {
        cmdCode INTEGER,
        success BOOLEAN
   }
END
```

Representation with `-compat noPDUsForImports` **specified:**

```
#define          Mod2Data_PDU 1

typedef struct Mod2Data {
    int             cmdCode;
    ossBoolean      success;
} Mod2Data;
```

Representation without `-compat noPDUsForImports` **specified:**

```
#define          Mod1Data_PDU 1
#define          Mod2Data_PDU 2
```

```
typedef struct Mod1Data {
    enum {
        random = 0,
        sequential = 1
    } type;
    int             address;
} Mod1Data;

typedef struct Mod2Data {
    int             cmdCode;
    ossBoolean      success;
} Mod2Data;
```

3.5.50 -compat noSharedTypes

Prior to version 3.6, the ASN.1 compiler did not implement a type-sharing optimization in which a structure shared among different data units is separately type-defined and replaced in its various locations with its type definition. Starting with version 3.6, a shared-type optimization is used on such shared structures.

The `-compat noSharedTypes` option disables the new behavior providing compatibility with version numbers lower than 3.6. Note: starting with version 5.2.0 of the ASN.1/C compiler, this option has no effect unless the `-compat v5.1typesharing` option is also specified.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN
        Type1 ::= SET {
           SET OF INTEGER
        }
        Type2 ::= SET {
           SET OF INTEGER
        }
        Type3 ::= SET {
           SET OF INTEGER
        }
END
```

Representation with `-compat noSharedTypes` specified:

```
typedef struct Type1 {
    struct _setof1 {
        struct _setof1  *next;
        int             value;
    } *setOf;
} Type1;

typedef struct Type2 {
    struct _setof2 {
        struct _setof2  *next;
```

```
        int             value;
    } *setOf;
} Type2;

typedef struct Type3 {
    struct _setof3 {
        struct _setof3  *next;
        int             value;
    } *setOf;
} Type3;
```

Representation without `-compat noSharedTypes` specified:

```
typedef struct _setof1 {
    struct _setof1  *next;
    int             value;
} *_setof1;

typedef struct Type1 {
    struct _setof1  *setOf;
} Type1;

typedef struct Type2 {
    struct _setof1  *setOf;
} Type2;

typedef struct Type3 {
    struct _setof1  *setOf;
} Type3;
```

Alias: `-compat 1`

3.5.51 -compat noTypeRefWithUseThisSharing

Prior to version 8.1.2, the ASN.1 compiler did not always share top-level types that had both the `OSS.UseThis` and `OSS.TYPENAME` directives applied. Often this situation occurred when a .gen file was specified on the ASN.1 compiler command line. When the previously separate types become shared, the names of their corresponding C-types and macros in the header file might be changed.

The `-compat noTypeRefWithUseThisSharing` option disables the new behavior, which provides compatibility with versions previous to 8.1.2.

3.5.52 -compat noUInt

Prior to version 4.0, the ASN.1 compiler generated signed integers for non-negative constrained INTEGER types. Starting with version 4.0, unsigned integers are generated for such non-negative constrained INTEGER types.

The `-compat noUInt` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   MilesPerGallon  ::=  INTEGER (1..70)
   Price           ::=  INTEGER (60000..100000)
END
```

Representation with `-compat noUInt` specified:

```
typedef short           MilesPerGallon;
typedef int             Price;
```

Representation without `-compat noUInt` specified:

```
typedef unsigned short  MilesPerGallon;
typedef unsigned int    Price;
```

Alias: `-compat 5`

3.5.53 -compat noULength

Prior to version 4.0, the ASN.1 compiler generated signed integers for `count` and `length` fields in the header file. Starting with version 4.0, unsigned integers are generated for such `count` and `length` fields.

The `-compat noULength` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
    VoiceClip  ::= BIT STRING
END
```

Representation with `-compat noULength` specified:

```
typedef struct VoiceClip {
    int             length;  /* number of significant bits */
    unsigned char   *value;
} VoiceClip;
```

Representation without `-compat noULength` specified:

```
typedef struct VoiceClip {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
```

```
} VoiceClip;
```

Alias: `-compat 4`

3.5.54 -compat noUnionRepresentationForOpenTypes

Prior to version 9.0, the ASN.1 compiler generated a common representation for each open type as the predefined OpenType structure. Starting with version 9.0, the representation for an open type with table or component relation constraints has been changed. Now, instead of the untyped pointer, the decoded open type value is represented as a union of all of the possible type alternatives specified in the object set associated with the open type via table constraints.

The `-compat noUnionRepresentationForOpenTypes` option disables the new behavior providing compatibility with version numbers lower than 9.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
    C ::= CLASS {
        &key INTEGER,
        &Type
    }

    Object C ::= {
        { &key 1, &Type INTEGER } |
        { &key 2, &Type UTF8String }
    }

    S ::= SEQUENCE {
        key C.&key ({Object}),
        value C.&Type ({Object}{@key})
    }
END
```

Representation with `-compat noUnionRepresentationForOpenTypes` specified:

```
typedef struct S {
    int             key;
    OpenType        value;
} S;
```

Representation without `-compat noUnionRepresentationForOpenTypes` specified:

```
enum Object_Type_PDUs {
    PDU_Object_Type_UNKNOWN        = 0,

    PDU_Object_Type_integer        = Object_integer_PDU,
    PDU_Object_Type_UTF8String     = Object_UTF8String_PDU
};

union Object_Type_union {
    Object_integer  *pdu_Object_integer; /* PDU_Object_Type_integer */
```

```
    Object_UTF8String *pdu_Object_UTF8String; /* PDU_Object_Type_UTF8String */
};

typedef struct Object_Type {
    enum Object_Type_PDUs pduNum;
    OssBuf          encoded;
    union Object_Type_union decoded;
} Object_Type;

typedef struct S {
    int             key;
    Object_Type     value;
} S;
```

3.5.55 -compat noUserConstraintPDUs

Prior to version 4.2, the ASN.1 compiler did no treat parameters to CONSTRAINED BY constraints as PDUs. Starting with version 4.2, such parameters are treated as PDUs.

The `-compat noUserConstraintPDUs` option disables the new behavior providing compatibility with version numbers lower than 4.2.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
 SpecialNumber ::= INTEGER (CONSTRAINED BY
   {INTEGER --Integer parameter determines allowed values.--})
END
```

Representation with `-compat noUserConstraintPDUs` specified:

```
#define          SpecialNumber_PDU 1

typedef int             SpecialNumber;

/* SpecialNumber_fn is user-defined constraint function for
 * ASN.1 item Module.SpecialNumber */
extern int DLL_ENTRY SpecialNumber_fn(struct ossGlobal *,
                                      SpecialNumber *, void **);
```

Representation without `-compat noUserConstraintPDUs` specified:

```
#define          SpecialNumber_PDU 1
#define          SpecialNumber_integer_PDU 2

typedef int             SpecialNumber;

typedef int             SpecialNumber_integer;

/* SpecialNumber_fn is user-defined constraint function for
 * ASN.1 item Module.SpecialNumber */
```

```
extern int DLL_ENTRY SpecialNumber_fn(struct ossGlobal *,
                                      SpecialNumber *, void **);
```

The above example was compiled with the `-userConstraints` option specified.

3.5.56 -compat noValues

Prior to version 4.0, the ASN.1 compiler did not generate initializations for valuereferences in the ASN.1 input. Starting with version 4.0, initializations are generated in the `.c` file for each valuereference in the input.

The `-compat noValues` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   Magnitude ::= INTEGER (1..10)
   powerLevel Magnitude ::= 8
END
```

Representation with `-compat noValues` specified:

`.h` file contains:

```
typedef unsigned short  Magnitude;
```

`.c` file contains no initializations.

Representation without `-compat noValues` specified:

`.h` file contains:

```
typedef unsigned short  Magnitude;
extern Magnitude powerLevel;
```

`.c` file contains:

```
Magnitude powerLevel = 8;
```

Alias: `-compat 9`

3.5.57 -compat oldBooleanType

Prior to version 4.1, the ASN.1 compiler generated a type called `Boolean` to represent the ASN.1 BOOLEAN type. Starting with version 4.1, ASN.1 BOOLEAN types are represented using a type called `ossBoolean`.

The `-compat oldBooleanType` option instructs the compiler to generate a type definition that equates the `Boolean` and `ossBoolean` types to provide compatibility with version numbers lower than 4.1.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   Married  ::= BOOLEAN
END
```

Representation with `-compat oldBooleanType` **specified:**

```
typedef ossBoolean      Boolean;

typedef ossBoolean      Married;
```

Representation without `-compat oldBooleanType` **specified:**

```
typedef ossBoolean      Married;
```

3.5.58 -compat oldEncodableNames

Prior to version 5.1.0, the ASN.1 compiler generated an extra `typedef` for some types that had the ASN1.DeferDecoding or OSS.ENCODABLE directive applied to them. This behavior occurred when the type was a parameter of a parameterized type or was referenced from another part of the specification. Starting with version 5.1.0, the extra typedefs noted above are no longer generated.

The `-compat oldEncodableNames` option disables the new behavior providing compatibility with version numbers lower than 5.1.0.

Example:

ASN.1:

```
--<ASN1.DeferDecoding Module.Parm.*>--

Module DEFINITIONS ::= BEGIN
   Parm {Type} ::= SET OF Type
   N ::= SET OF INTEGER
   B ::= SEQUENCE { n Parm {N} }
END
```

Representation with `-compat oldEncodableNames` **specified:**

```
typedef struct Parm {
    struct Parm     *next;
    OpenType        value;  /* Parm_N_encodable type */
} *Parm;

typedef struct N {
   struct N       *next;
   int             value;
} *N;
```

```
typedef struct B {
    struct Parm     *n;
} B;

typedef struct N       *Parm_N_encodable;
```

Representation without `-compat oldEncodableNames` **specified:**

```
typedef struct Parm {
    struct Parm     *next;
    OpenType        value;  /* N type */
} *Parm;

typedef struct N {
   struct N       *next;
   int             value;
} *N;

typedef struct B {
   struct Parm     *n;
} B;
```

3.5.59 -compat oldExternObjHdl

Prior to version 5.0, the ASN.1 compiler appended the string `ObjHandle` to the type names representing the EXTERNAL, CHARACTER STRING, and EMBEDDED PDV types when the OBJHANDLE directive was specified. Starting with version 5.0, the names for these types no longer have the string `ObjHandle` appended to them when the OBJHANDLE directive is specified.

The `-compat oldExternObjHdl` option disables the new behavior providing compatibility with version numbers lower than 5.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
  MyExtern ::= EXTERNAL --<OBJHANDLE>--
END
```

Representation with `-compat oldExternObjHdl` **specified:**

```
typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} ObjectID;

typedef struct External ObjHandle {
    . . . .
} External ObjHandle;

typedef External ObjHandle MyExtern;
```

Representation without `-compat oldExternObjHdl` **specified:**

```
typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} ObjectID;

typedef struct External {
   . . . .
} External;

typedef External        MyExtern;
```

3.5.60 -compat oldInternalDefineNames

Prior to version 5.0.1, the ASN.1 compiler mangled `#define` names in internal structures such as `External`, `EmbeddedPDV`, and `UnrestrictedChar` when a name conflict was detected. Starting with version 5.0.1, such name mangling is no longer done to `#define` in the names in such C structures.

The `-compat oldInternalDefineNames` option disables the new behavior providing compatibility with version numbers lower than 5.0.1.

Example:

ASN.1:

```
SNI-Extensions DEFINITIONS ::= BEGIN
 E ::= EXTERNAL
 MySeq   ::=     SEQUENCE        {
    presentation-data-values        CHOICE  {
        octet-aligned       OCTET STRING ,
        arbitrary           BIT STRING
    }
 }
END
```

Representation with `-compat oldInternalDefineNames` specified:

```
typedef struct ObjectID {  . . . . } ObjectID;

typedef struct External {
 . . . .
#           define      encoding_octet_aligned_chosen 2
#           define      encoding_arbitrary_chosen 3
 . . . .
} External;

typedef External        E;

typedef struct MySeq {
 . . . .
} MySeq;
```

Representation without `-compat oldInternalDefineNames` specified:

```
typedef struct ObjectID { . . . . } ObjectID;

typedef struct External {
 . . . .
   #           define      octet_aligned_chosen 2
   #           define      arbitrary_chosen 3
 . . . .
} External;

typedef External        E;

typedef struct MySeq {
 . . . .
} MySeq;
```

3.5.61 -compat oldInternalNamesWithinUseThisSharedTypes

Prior to version 8.2, the ASN.1 compiler generated names of artificial types created due to the presence of the ASN1.DeferDecoding or OSS.ENCODABLE directives, as well as for types with the CONTAINING subtype and open types within information objects that appear within parameterized types whose instances are shared by the compiler and UseThis directives are also applied. These generated names were based on the names of types that contain one of the instances of such parameterized types instead of names of the parameterized types themselves.

Starting with version 8.2, the names of such artificial types are derived from the name of the parameterized type whose instances are shared in the presence of the UseThis directives.

Example:

```
--<ASN1.DeferDecoding M.P.*>--
--<OSS.UseThis M.T.a M.P>--
--<OSS.UseThis M.T-T.* M.P-T>--

M DEFINITIONS AUTOMATIC TAGS ::= BEGIN
     P {PT} ::= SET OF PT
     T ::= SET { a  P {INTEGER}}
     P-T {P-PT} ::= BIT STRING (CONTAINING P-PT )
     T-T ::= SET OF P-T {INTEGER}
END
```

The generated .h file contains:

```
#define          T_PDU 1
#define          T_T_PDU 2
#define          P_integer_encodable_PDU 3
#define          P_T_integer_PDU 4

typedef struct P {
    struct P        *next;
    OpenType        value;  /* P_integer_encodable type */
} *P;

typedef struct T {
```

```
    struct P        *a;
} T;

typedef int             P_T_integer;

typedef struct P_T {
    /* ContentsConstraint is applied to P_T */
    struct {
        unsigned int    length;  /* number of significant bits */
        unsigned char   *value;
    } encoded;
    P_T_integer     *decoded;
} P_T;

typedef struct T_T {
    struct T_T      *next;
    P_T             value;
} *T_T;
typedef int             P_integer_encodable;
```

The .h file generated with `-compat oldInternalNamesWithinUseThisSharedTypes` contains the artificially generated types `T_integer_encodable` and `T_T_integer` instead of `P_integer_encodable` and `P_T_integer`:

```
#define          T_PDU 1
#define          T_T_PDU 2
#define          T_integer_encodable_PDU 3
#define          T_T_integer_PDU 4

typedef struct P {
    struct P        *next;
    OpenType        value;  /* T_integer_encodable type */
} *P;

typedef struct T {
    struct P        *a;
} T;

typedef int             T_T_integer;

typedef struct P_T {
    /* ContentsConstraint is applied to P_T */
    struct {
        unsigned int    length;  /* number of significant bits */
        unsigned char   *value;
    } encoded;
    T_T_integer     *decoded;
} P_T;

typedef struct T_T {
    struct T_T      *next;
    P_T             value;
} *T_T;

typedef int             T_integer_encodable;
```

3.5.62 -compat oldNamesManglingWithPrefix

Prior to version 5.3.0, the ASN.1 compiler incorrectly mangled some typenames when both the `-c++` and `-prefix` options were specified. Starting with version 5.3.0, such names are no longer incorrectly mangled.

The `-compat oldNamesManglingWithPrefix` option disables the new behavior providing compatibility with version numbers lower than 5.0.

3.5.63 -compat oldObjectNames

Prior to version 4.2.6, the ASN.1 compiler did not prefix ambiguous names within separate information objects with their containing module's name. Instead a '_#' suffix was added for disambiguation. Starting with version 4.2.6, ambiguous names within information objects are prefixed with their containing module's name.

The `-compat oldObjectNames` option disables the new behavior providing compatibility with versions earlier than 4.2.6. To have an effect, either the `-genDirectives` or `-keepNames` option must be specified in the ASN.1 compiler command line.

Example:

ASN.1:

```
--<OSS.ROOT>--
Mod1 DEFINITIONS ::= BEGIN
   C1 ::= CLASS {&Type DEFAULT INTEGER --<LONG>--,
                 &ValueSet &Type DEFAULT { INTEGER } }
   myClass C1 ::=
   {
      &Type   IA5String --<UNBOUNDED>--
   }
END

Mod2 DEFINITIONS ::= BEGIN
   C1 ::= CLASS {&Type DEFAULT INTEGER,
                 &ValueSet &Type DEFAULT { INTEGER } }
   myClass C1 ::=
   {
      &Type   BOOLEAN
   }
END
```

Representation with `-compat oldObjectNames` specified:

```
#define          C1_integer_1_PDU 1
#define          MyClass_IA5String_1_PDU 2
#define          C1_integer_2_PDU 3
#define          MyClass_IA5String_2_PDU 4

typedef struct Mod1_C1 {
```

```
    unsigned char   bit_mask;
#       define      Mod1_C1_ValueSet_present 0x80
#       define      Mod1_C1_ValueSet_present 0x40
      unsigned short  Type;  /* optional; set in bit_mask
                              * Mod1_C1_Type_present if present */

    unsigned short  ValueSet;  /* optional; set in bit_mask
                                * Mod1_C1_ValueSet_present if present */
    long            _oss_unique_index;
} Mod1_C1;

typedef struct Mod2_C1 {
    unsigned char   bit_mask;
#       define      Mod2_C1_ValueSet_present 0x80
#       define      Mod2_C1_ValueSet_present 0x40
      unsigned short  Type;  /* optional; set in bit_mask
                              * Mod1_C1_Type_present if present */
    unsigned short  ValueSet;  /* optional; set in bit_mask
                                * Mod2_C1_ValueSet_present if present */
    long            _oss_unique_index;
} Mod2_C1;

typedef long            C1_integer_1;

typedef struct MyClass_IA5String_1 {
    unsigned int    length;
    char            *value;
} MyClass_IA5String_1;

typedef int             C1_integer_2;

typedef char            *MyClass_IA5String_2;

extern Mod1_C1 Mod1_myClass;

extern Mod2_C1 Mod2_myClass;
```

Representation without `-compat oldObjectNames` **specified:**

```
#define          C1_integer_1_PDU 1
#define          Mod1_MyClass_IA5String_PDU 2
#define          C1_integer_2_PDU 3
#define          Mod2_MyClass_IA5String_PDU 4

typedef struct Mod1_C1 {
    unsigned char   bit_mask;
#       define      Mod1_C1_ValueSet_present 0x80
#       define      Mod1_C1_ValueSet_present 0x40
        unsigned short  Type;  /* optional; set in bit_mask
                                * Mod1_C1_Type_present if present */
    unsigned short  ValueSet;  /* optional; set in bit_mask
                                * Mod1_C1_ValueSet_present if present */
    long            _oss_unique_index;
} Mod1_C1;

typedef struct Mod2_C1 {
    unsigned char   bit_mask;
#       define      Mod2_C1_ValueSet_present 0x80
```

```
#       define      Mod1_C1_ValueSet_present 0x40

    unsigned short  Type;
    unsigned short  ValueSet;  /* optional; set in bit_mask
                                * Mod2_C1_ValueSet_present if present */
    long            _oss_unique_index;
} Mod2_C1;

typedef long            C1_integer_1;

typedef struct Mod1_MyClass_IA5String {
    unsigned int    length;
    char            *value;
} Mod1_MyClass_IA5String;

typedef int             C1_integer_2;

typedef char            *Mod2_MyClass_IA5String;
extern Mod1_C1 Mod1_myClass;

extern Mod2_C1 Mod2_myClass;
```

Note:
Use of the `-compat oldObjectNames` option with the `-genDirectives` command-line option could cause some names for information object types to be omitted from the `.gen` file.

3.5.64 -compat oldParamTypesharing

Prior to version 5.0, the ASN.1 compiler did not include types used as actual parameters in a parameterized type in the typesharing optimization. Starting with version 5.0, parameters to parameterized types are included in the typesharing optimization.

The `-compat oldParamTypesharing` option disables the new behavior providing compatibility with version numbers lower than 5.0.

Note: Note that use of the `-compat oldParamTypesharing` option may result in duplicate structures with the same name being generated in the `.h` file

3.5.65 -compat oldSharingFieldsWithDirectives

Prior to version 5.2, the ASN.1 compiler did not allow typesharing of nested structures which had the OSS.DefineName or OSS.FIELDNAME directive applied to one of their fields. Starting with version 5.2, a shared-type optimization is used on such common structures.

The `-compat oldSharingFieldsWithDirectives` option disables the new behavior providing compatibility with version numbers lower than 5.2. Note: for versions 5.2.1 up to 7.0.0 of the ASN.1/C compiler, this option has no effect unless the `-compat v5.1typesharing` option is also specified. Starting with version 7.0.0, pre-5.2 sharing is restored so the `-compat oldSharingFieldsWithDirectives` option is not needed.

Example:

ASN.1:

```
--<OSS.DefineName Mod.IN-EntryInformation.information.attribute
-- "attribute">--
--<OSS.DefineName Mod.EntryInformation.information.attribute
-- "attribute">--

Mod DEFINITIONS ::= BEGIN

        Attribute ::= INTEGER

        IN-EntryInformation ::= SEQUENCE {
                information     CHOICE {
                        attribute        Attribute
                } OPTIONAL
        }

        EntryInformation        ::=     SEQUENCE {
                information     CHOICE

                        attribute       Attribute
                } OPTIONAL
        }
END
```

Representation with `-compat oldSharingFieldsWithDirectives` specified:

```
    typedef int             Attribute;

    typedef struct IN_EntryInformation {
        unsigned char   bit_mask;
    #       define      IN_EntryInformation_information_present 0x80
        struct {
            unsigned short  choice;
    #           define      attribute_chosen 1
            union {
                Attribute       attribute;  /* to choose, set choice to
                                             * attribute_chosen */
            } u;
        } information;  /* optional; set in bit_mask
                         * IN_EntryInformation_information_present if present
    */
    } IN_EntryInformation;

    typedef struct EntryInformation {
        unsigned char   bit_mask;
    #       define      EntryInformation_information_present 0x80
```

```
        struct {
            unsigned short  choice;
#           define      attribute_chosen 1
            union {
                Attribute       attribute;  /* to choose, set choice to
                                             * attribute_chosen */
            } u;
        } information;  /* optional; set in bit_mask
                         * EntryInformation_information_present if present */
    } EntryInformation;
```

Representation without `-compat oldSharingFieldsWithDirectives` **specified:**

```
    typedef int             Attribute;

    typedef struct _choice1 {
        unsigned short  choice;
#       define      attribute_chosen 1
        union {
            Attribute       attribute;  /* to choose, set choice to
                                         * attribute_chosen */
        } u;
    } _choice1;

    typedef struct IN_EntryInformation {
        unsigned char   bit_mask;
#       define      information_present 0x80
        _choice1        information;  /* optional; set in bit_mask
                                       * information_present if present */
    } IN_EntryInformation;

    typedef struct EntryInformation {
        unsigned char   bit_mask;
#       define      information_present 0x80
        _choice1        information;  /* optional; set in bit_mask
                                       * information_present if present */
    } EntryInformation;
```

3.5.66 -compat padded

Prior to version 4.0, the ASN.1 compiler allowed the use of the PADDED directive on variable length strings. Starting with version 4.0, the PADDED directive is only allowed on fixed length strings.

The `-compat padded` option disables the new behavior providing compatibility with version numbers lower than 4.0.

Example:

ASN.1:

```
    --<OSS.PADDED IA5String>--

    Module DEFINITIONS ::= BEGIN
       VariableString       ::= [1] IA5String (SIZE (2..10))
```

```
   NullTerm             ::= [2] IA5String (SIZE(10))
END
```

Representation with `-compat padded` specified:

```
typedef char            VariableString[10];

typedef char            NullTerm[10];
```

Representation without `-compat padded` specified:

Compiler will refuse to compile the above syntax issuing the following message:

```
C0455E: PADDED directive for VariableString requires a single size
constraint.
```

Alias: `-compat 6`

3.5.67 -compat paddedForNamedBits

Prior to version 6.0, the ASN.1 compiler represented BIT STRING with a named bit list in the PADDED representation by default. Starting from version 6.0, the default representation was changed to UNBOUNDED.

The `-compat paddedForNamedBits` disables the new behavior providing compatibility with version numbers lower than 6.0.

Example:

```
B DEFINITIONS ::= BEGIN
   BSN ::= BIT STRING { a(0) }
END
```

Representation with `-compat paddedForNamedBits` specified:

```
typedef unsigned char   BSN;
#define                     a 0x80
```

Representation without `-compat paddedForNamedBits` specified:

```
typedef struct BSN {
     unsigned short  length;  /* number of significant bits */
     unsigned char   *value;
} BSN;
#define                     a 0x80
#define                     a_byte 0
```

3.5.68 -compat pointeredParamTypesWithContConstrAndUseThis

Prior to version 8.2, the ASN.1 compiler generated typedefs with an extra pointer for parameterized types with ContentsConstraint and the UseThis directive. Starting with version 8.2, no extra pointer is generated for such typedefs.

Example:

```
--<OSS.UseThis M.T-T.* M.P-T>--

M DEFINITIONS AUTOMATIC TAGS ::= BEGIN
     P-T {P-PT} ::= BIT STRING (CONTAINING SEQUENCE { a P-PT } )
     T-T ::= SET OF P-T {INTEGER}
END
```

The generated .h file contains the following typedefs:

```
typedef struct P_T_seq {
    int             a;
} P_T_seq;

typedef struct P_T {
    /* ContentsConstraint is applied to P_T */
    _BitStr         encoded;
    P_T_seq         *decoded;
} P_T;

typedef P_T T_T_seq;
```

The .h file generated with `-compat pointeredParamTypesWithContConstrAndUseThis` contains a typedef for the parameterized type "P_T" with an extra pointer:

```
typedef struct P_T_seq {
    int             a;
} P_T_seq;

typedef struct P_T {
    /* ContentsConstraint is applied to P_T */
    _BitStr         encoded;
    P_T_seq         *decoded;
} *P_T;
```

3.5.69 -compat terseComments

Prior to version 5.0, the ASN.1 compiler printed brief comments in the header file. Starting with version 5.0, more verbose comments are produced for many types.

The `-compat terseComments` option disables the new behavior producing the old-style terse comments.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
        A ::= SEQUENCE {
                a INTEGER OPTIONAL,
                b BOOLEAN DEFAULT TRUE
        }
END
```

Representation with `-compat terseComments` specified:

```
typedef struct A {
    unsigned char   bit_mask;
#       define      a_present 0x80
#       define      b_present 0x40
    int             a;  /* optional */
    ossBoolean      b;  /* default assumed if omitted */
} A;
```

Representation without `-compat terseComments` specified:

```
typedef struct A {
    unsigned char   bit_mask;
#       define      a_present 0x80
#       define      b_present 0x40
    int             a;  /* optional; set in bit_mask a_present
                         * if present */
    ossBoolean      b;  /* b_present not set in bit_mask
                         * implies value is TRUE */
} A;
```

Note that this option can also be used to suppress descriptive comments for user-defined constraint-checking functions.

3.5.70 -compat typedefsForGenNames

Between versions 4.2 (inclusive) and 4.2.6 (exclusive), the ASN.1 compiler generated extra `typedef`s into the header file for user-defined CHOICE, SEQUENCE, and SET types. These extra `typedef`s were generated when the following conditions were met:

- The OSS.FIELDNAME directive was used to rename a user-defined field in a CHOICE, SEQUENCE, or SET type
- Multiple CHOICE, SEQUENCE, or SET types contained such a field.
- This user-defined field referenced a tagged CHOICE, SEQUENCE, or SET type.
- The name given by the OSS.FIELDNAME for such a field in two or more different CHOICE, SEQUENCE, or SET types was the same.

Starting with version 4.2.6, these extra `typedef`s are no longer generated.

The `-compat typedefsForGenNames` option disables the new behavior providing compatibility with version numbers between 4.2 (inclusive) and 4.2.6 (exclusive).

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   C ::= SEQUENCE {
      a  AlgorithmIdentifier --<FIELDNAME
                                "signatureAlgorithm">--
   }
   C1 ::= SEQUENCE {
      b  AlgorithmIdentifier --<FIELDNAME
                                "signatureAlgorithm">--
   }
   AlgorithmIdentifier ::= SEQUENCE {
      idStamp           INTEGER,
      returnType        INTEGER
   }
END
```

Representation with `-compat typedefsForGenNames` specified:

```
typedef struct AlgorithmIdentifier {
    int             idStamp;
    int             returnType;
} AlgorithmIdentifier;

typedef AlgorithmIdentifier _seq1;

typedef struct C {
    _seq1           signatureAlgorithm;
} C;

typedef struct C1 {
    _seq1           signatureAlgorithm;
} C1;
```

Representation without `-compat typedefsForGenNames` specified:

```
typedef struct AlgorithmIdentifier {
    int             idStamp;
    int             returnType;
} AlgorithmIdentifier;

typedef struct C {
    AlgorithmIdentifier signatureAlgorithm;
} C;

typedef struct C1 {
    AlgorithmIdentifier signatureAlgorithm;
} C1;
```

3.5.71 -compat unbndBit

In version 3.6, the ASN.1 compiler gave SizeConstraints precedence over NamedBitLists when determining the representation of the BIT STRING type. Additionally, an UNBOUNDED structure was generated for the BIT STRING by default. In versions prior to 3.6 and after 3.6, the NamedBitList takes precedence over the SizeConstraint if both are specified with the BIT STRING type. Also, an UNBOUNDED structure is not always generated.

The `-compat undbndBit` option provides compatibility with version 3.6. Note: starting with version 6.0 of the ASN.1/C compiler, BIT STRING types are represented the same as in version 3.6, so this option became pointless.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
 AccessLevel1 ::= BIT STRING {guest(1), user(2), superuser(3)}
                             (SIZE (10))
END
```

Representation `-compat undbndBit` specified (SizeConstraint used):

```
typedef struct AccessLevel1 {
    unsigned short  length;  /* number of significant bits */
    unsigned char   *value;
} AccessLevel1;
#define                     guest 0x40
#define                     guest_byte 0
#define                     user 0x20
#define                     user_byte 0
#define                     superuser 0x10
#define                     superuser_byte 0
```

Representation without `-compat undbndBit` specified (SizeConstraint disregarded):

```
typedef unsigned short  AccessLevel1;
#define                     guest 0x4000
#define                     user 0x2000
#define                     superuser 0x1000
```

Alias: `-compat 7`

Note: The Draft of the International Standard (DIS) of **ASN.1:1994** stated that SizeConstraints were to have precedence over NamedBitLists when applied to the BIT STRING type. However, the actual International Standard (IS) of **ASN.1:1994** reverted back to the **ASN.1:1990** decision of letting the NamedBitList have precedence over the SizeConstraint, in such a case.

3.5.72 -compat unnamedStructForConstrBy

Prior to version 5.0, the ASN.1 compiler generated unnamed structures for SEQUENCE and SET types defined with CONSTRAINED BY nested in SEQUENCE OF, and SET OF types with the OSS.LINKED, OSS.DLINKED, or OSS.ARRAY directive. (This scenario results in the generation of the type `unnamed_type` in the constraint function prototype when `-userconstraints` is specified.) Starting with version 5.0, such unnamed structures are no longer generated.

The `-compat unnamedStructForConstrBy` disables the new behavior providing compatibility with version numbers lower than 5.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   DataPacket ::= SEQUENCE {
      oddNumber SET --<ARRAY>-- (SIZE(10)) OF SET {
         evenOrOdd  INTEGER
      } (CONSTRAINED BY {--Must be odd--})
   }
END
```

Representation with `-compat unnamedStructForConstrBy` specified:

```
typedef struct DataPacket {
    struct {
        unsigned short  count;
        struct {
            int             evenOrOdd;
        } value[10];
    } oddNumber;
} DataPacket;

/* _cnstr_by1_fn is user-defined constraint function for ASN.1
 *  item Module.DataPacket.oddNumber.* */
extern int DLL_ENTRY _cnstr_by1_fn(struct ossGlobal *,
                                   unknown_type *, void **);
```

Representation without `-compat unnamedStructForConstrBy` specified:

```
typedef struct DataPacket {
    struct {
        unsigned short  count;
        struct _set1 {
            int             evenOrOdd;
        } value[10];
    } oddNumber;
} DataPacket;

/* _set1_fn is user-defined constraint function for ASN.1 item
```

```
 * Module.DataPacket.oddNumber.* */
extern int DLL_ENTRY _set1_fn(struct ossGlobal *,
                struct _set1 *, void **);
```

3.5.73 -compat useUShortForBitStringsWithNamedBits

Prior to version 8.6, the ASN.1 compiler sometime generated representations for BIT STRINGs with NamedBits that have values that exceed SHRT_MAX and are less than USHRT_MAX, which could cause problems with runtimes that access the values of those named bits.

The `-compat useUShortForBitStringsWithNamedBits` option restores the old behavior.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN
    B ::=  BIT STRING {bit(65534)} (SIZE(1..65535))
END
```

Representation without the `-compat useUShortForBitStringsWithNamedBits` flag for release 8.6.0 or later:

```
typedef struct B {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} B;
#define                     bit 0x02
#define                     bit_byte 8191
```

Representation with the `-compat useUShortForBitStringsWithNamedBits` flag for release 8.6.0 or later:

```
typedef struct B {
    unsigned short    length;  /* number of significant bits */
    unsigned char   *value;
} B;
#define                     bit 0x02
#define                     bit_byte 8191
```

3.5.74 -compat v2.0

Provides compatibility with version 2.0 of the ASN.1 compiler.

The `v2.0` flag is equivalent to specifying the following `-compat` options: `noSharedTypes`, `intEnums`, `nestUnions`, `noULength`, `noUInt`, `padded`, `noDefaultValues`, `noBTypeValues`, `badExternalPrefix`, `badLengthDirectives`, and `v4.0`.

3.5.75 -compat v3.0

Provides compatibility with version 3.0 of the ASN.1 compiler.

The `v3.0` flag is equivalent to specifying the `-compat v2.0` option.

3.5.76 -compat v3.5

Provides compatibility with version 3.5 of the ASN.1 compiler.

The `v3.5` flag is equivalent to specifying the following `-compat` options: `noSharedTypes`, `intEnums`, `nestUnions`, `noULength`, `noUInt`, `padded`, `badExternalPrefix`, `badUnderscorePrefix`, `badLengthDirectives, and v4.0`.

3.5.77 -compat v3.6

Provides compatibility with version 3.6 of the ASN.1 compiler.

The `v3.6` flag is equivalent to specifying the following `-compat` options: `nestUnions`, `noULength`, `noUInt`, `padded`, `unbndBit`, `badUnderscorePrefix`, `badLengthDirectives`, and `v4.0`.

3.5.78 -compat v4.0

Provides compatibility with version 4.0 of the ASN.1 compiler.

The `v4.0` flag is equivalent to specifying the following `-compat` options: `badValuePrefix`, `oldBooleanType`, `badNameConflicts`, and , `v4.1.0.`

3.5.79 -compat v4.1.0

Provides compatibility with version 4.1.0 of the ASN.1 compiler.

The `v4.1.0` flag is equivalent to specifying the `-compat options: extraLink and v4.1.6`.

3.5.80 -compat v4.1typesharing

During version 4.1, the ASN.1 compiler used a less aggressive typesharing optimization. Starting with version 4.2, a more aggressive typesharing optimization is used.

The `-compat v4.1typesharing` option disables the new behavior providing compatibility with version number 4.2.

3.5.81 -compat v4.1.1 / -compat v4.1.2 / -compat v4.1.3 / -compat v4.1.4 / -compat v4.1.5 / -compat v4.1.6 / -compat v4.1.7 / -compat v4.1.8

These options provide compatibility with their respective version numbers.

All of these options are equivalent to specifying the `-compat` options: `extSizeNotUnbounded` and `v4.1.9`.

3.5.82 -compat v4.1.6encodable

Prior to version 4.2, the ASN.1 compiler generated unneeded `typedef`s for complex user-defined types marked with the OSS.ENCODABLE directive. Starting with version 4.2, these unneeded `typedef`s are no longer generated.

The `-compat v4.1.6encodable` option disables the new behavior providing compatibility with version numbers lower than 4.2.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
         Certificate ::= [0] SEQUENCE {
             signedData     CertificateInfo --<ENCODABLE>--,
             signature      BIT STRING
         }
         CertificateInfo ::= [1] SEQUENCE {
             version            IA5String,
             serialNumber       INTEGER
         }
END
```

Representation with `-compat v4.1.6encodable` specified:

```
#define          Certificate_PDU 1
#define          Certificate_signedData_encodable_PDU 2

typedef struct Certificate {
   OpenType   signedData;  /* Certificate_signedData_encodable
                            * type */
```

```
        struct {
         unsigned int    length;  /* number of significant bits */
         unsigned char   *value;
        } signature;
    } Certificate;

    typedef struct CertificateInfo {
        char            *version;
        int             serialNumber;
    } CertificateInfo;

    typedef CertificateInfo Certificate_signedData_encodable;
```

Representation without `-compat v4.1.6encodable` **specified:**

```
    #define          Certificate_PDU 1
    #define          CertificateInfo_PDU 2

    typedef struct Certificate {
        OpenType        signedData;  /* CertificateInfo type */
        struct {
         unsigned int    length;  /* number of significant bits */
         unsigned char   *value;
        } signature;
    } Certificate;

    typedef struct CertificateInfo {
        char            *version;
        int             serialNumber;
    } CertificateInfo;
```

3.5.83 -compat v4.1.6extraLinkedSETOFPtr

Prior to version 4.2, the ASN.1 compiler generated a double pointer (`**`) to represent a circular reference to a SEQUENCE OF or SET OF type which had a SizeConstraint and used the LINKED representation (due to a global OSS.LINKED directive). Starting with version 4.2, only a single pointer is used for such a type.

The `-compat v4.1.6extraLinkedSETOFPtr` option disables the new behavior providing compatibility with version numbers lower than 4.2 (such as version 4.1.6).

Example:

ASN.1:

```
    --<OSS.LINKED SET OF>--

    Circle DEFINITIONS --<PDU>-- ::= BEGIN
       Circ ::= SET (SIZE(10)) OF SEQUENCE {b Circ}
    END
```

Representation with `-compat v4.1.6extraLinkedSETOFPtr` **specified:**

```
typedef struct Circ {
    struct Circ     *next;
    struct {
        struct Circ     **b;
    } value;
} *Circ;
```

Representation without `-compat v4.1.6extraLinkedSETOFPtr` **specified:**

```
typedef struct Circ {
    struct Circ     *next;
    struct {
        struct Circ     *b;
    } value;
} *Circ;
```

3.5.84 -compat v4.1.9

Provides compatibility with version 4.1.9 of the ASN.1 compiler.

The `v4.1.9` flag is equivalent to specifying the `-compat` options: `multipleUserFunctions`, `noUserConstraintPDUs`, `v4.1typesharing`, `extraNameShortening`, `v4.1.6extralink`, `v4.1.6encodable`, `noOssterm`, `v4.2.5typesharing`, and `v4.2.6`.

3.5.85 -compat v4.2.0

Provides compatibility with version 4.2.0 of the ASN.1 compiler.

The `v4.2.0` flag is equivalent to specifying the `-compat v4.2.5` option.

3.5.86 -compat v4.2.5

Provides compatibility with version 4.2.5 of the ASN.1 compiler.

The `v4.2.5` flag is equivalent to specifying the `-compat` options: `typedefsForGenNames`, `v4.2.5typesharing`, and `v4.2.6`.

3.5.87 -compat v4.2.5typesharing

During version 4.2.5, the ASN.1 compiler used a typesharing optimization different from later versions. Starting with version 4.2.6, a different typesharing optimization was introduced.

The `-compat v4.2.5typesharing` option disables the new behavior providing compatibility with version number 4.2.5.

3.5.88 -compat v4.2.6

Provides compatibility with version 4.2.6 of the ASN.1 compiler.

The `v4.2.6` flag is equivalent to specifying the `-compat` options: `noDecoupledNames`, `terseComments`, `extensionWithMask`, `oldExternObjHdl`, `oldObjectNames`, `v4.2defaults`, `unnamedStructForConstrBy`, `noMacroArgumentPDUs`, `noPDUsForImports`, `v4.2objhandleCstrPointer`, `v4.2namesforopentypes`, `decoderUpdatesInputAddress`, `oldParamTypesharing`, and `v5.0.0`.

3.5.89 -compat v4.2badSetOfWithGlobalDir

During version 4.2, the ASN.1 compiler incorrectly gave precedence to the earlier of two or more mutually exclusive global directives applied to the SET OF or SEQUENCE OF types. Starting with version 5.0.4, the last mutually exclusive global directive specified is given precedence in conformity with how the rest of the mutually exclusive OSS global directives are handled.

The `-compat v4.2badSetOfWithGlobalDir` option disables the new behavior providing compatibility with version 4.2.

Note that this flag is not included in any `v4.2.`*x* or `v5.0.`*x* compat flags.

Example:

ASN.1:

```
--<OSS.UNBOUNDED SEQUENCE OF>--
--<OSS.ARRAY SEQUENCE OF>--

Module DEFINITIONS ::= BEGIN
   Seq ::= SEQUENCE {
      a SEQUENCE OF INTEGER OPTIONAL
   }
END
```

Representation with `-compat v4.2badSetOfWithGlobalDir` specified:

```
typedef struct Seq {
    struct _seqof1 {
        unsigned int    count;
        int             *value;
    } *a;  /* NULL for not present */
} Seq;
```

Representation without `-compat v4.2badSetOfWithGlobalDir` **specified:**

```
typedef struct Seq {
    struct _seqof1 {
        unsigned int    count;
        int             value[1];  /* first element of the array */
    } *a;  /* NULL for not present */
} Seq;
```

3.5.90 -compat v4.2badUnderscorePrefix

During version 4.2, the ASN.1 compiler prefixed internal names (such as the `value` field in UNBOUNDED structures) with an underscore when a user-defined name was found to be the same as these internal names. Starting with version 5.0.1, these internal names are no longer prefixed with an underscore when some user-defined variable has a name that is the same as an internal name (e.g., `count`, `next`, `length`, `prev`, `value`, etc.).

The `-compat v4.2badUnderscorePrefix` option disables the new behavior providing compatibility with version 4.2.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
        YourOctStr ::= OCTET STRING

        MyOctStr ::= OCTET STRING
        value MyOctStr ::= 'FF'H
        length INTEGER ::= 10
END
```

Representation with `-compat v4.2badUnderscorePrefix` **specified:**

```
typedef struct YourOctStr {
    unsigned int    _length;
    unsigned char   *_value;
} YourOctStr;

typedef struct MyOctStr {
    unsigned int    _length;
    unsigned char   *_value;
} MyOctStr;

extern MyOctStr value;

extern int length;
```

Representation without `-compat v4.2badUnderscorePrefix` **specified:**

```
typedef struct YourOctStr {
    unsigned int    length;
```

```
    unsigned char   *value;
} YourOctStr;

typedef struct MyOctStr {
    unsigned int    length;
    unsigned char   *value;
} MyOctStr;

extern MyOctStr value;

extern int length;
```

3.5.91 -compat v4.2defaults

Instructs the ASN.1 compiler to use the default representation schema present during version 4.2 of the ASN.1 compiler. For example in version 4.2, the default representation of SEQUENCE OF and SET OF types with SizeConstraints was ARRAY not LINKED (as is in version 5.0).
The `-compat v4.2defaults` option provides compatibility with version 4.2.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN
   MySet        ::= SET (SIZE(5)) OF INTEGER
   YourSeq      ::= SEQUENCE (SIZE (5)) OF INTEGER
END
```

Representation with `-compat v4.2defaults` specified:

```
typedef struct MySet {
    unsigned short  count;
    int             value[5];
} MySet;

typedef struct YourSeq {
    unsigned short  count;
    int             value[5];
} YourSeq;
```

Representation without `-compat v4.2defaults` specified:

```
typedef struct MySet {
    struct MySet    *next;
    int             value;
} *MySet;

typedef struct YourSeq {
    struct YourSeq  *next;
    int             value;
} *YourSeq;
```

3.5.92 -compat v4.2namesForOpenTypes

During version 4.2.x when the ASN1.DeferDecoding directive, the OSS.ENCODABLE directive, or a component relation constraint was specified for a type, the affected type had its name replaced with a name of the form: **`Type`**_*`builtInTypeName`*_**`encodable`** in its generated `typedef`. Where *`builtInTypeName`* was a name of an ASN.1 built-in type in lower-case letters. Starting with version 5.0, such a type no longer loses its name. Rather only a suffix of `_encodable` is added.

Example:

ASN.1:

```
--<ASN1.DeferDecoding Module.Type.setComponent>--

Module DEFINITIONS ::= BEGIN
        Type ::= SET {
                setComponent INTEGER,
                b            BOOLEAN
        }
END
```

Representation with `-compat v4.2namesForOpenTypes` specified:

```
#define          Type_PDU 1
#define          Type_integer_encodable_PDU 2

typedef struct Type {
    OpenType        setComponent;  /* Type_integer_encodable
                                    * type */
    ossBoolean      b;
} Type;

typedef int             Type_integer_encodable;
```

Representation without `-compat v4.2namesForOpenTypes` specified:

```
#define          Type_PDU 1
#define          Type_setComponent_encodable_PDU 2

typedef struct Type {
    OpenType        setComponent;
    /* Type_setComponent_encodable type */
    ossBoolean      b;
} Type;

typedef int             Type_setComponent_encodable;
```

3.5.93 -compat v4.2nicknames

Prior to version 5.0.5, the ASN.1 compiler incorrectly applied the ASN1.Nickname and OSS.TYPENAME directives to parameters of parameterized types and other types referenced by

parmaterized types. Starting with version 5.0.5, such types are no longer incorrectly affected by the ASN1.Nickname and OSS.TYPENAME directives.

The `-compat v4.2nicknames` option disables the new behavior providing compatibility with version numbers lower than 5.0.5. Note: starting with version 5.2.1 of the ASN.1/C compiler, this option has no effect unless the `-compat v5.2noextraparamref` option is also specified.

Example:

ASN.1:

```
--<ASN1.Nickname  Mod.CA "MyCa">--

Mod DEFINITIONS ::= BEGIN
  CA ::= SET OF INTEGER
  Param {Type} ::= CA
  P1 ::= Param {INTEGER}
  P2 ::= Param {BOOLEAN}
END
```

Representation with `-compat v4.2nicknames` and `-compat v4.2.0` specified:

```
#define          MyCa_PDU 1
#define          MyCa_PDU 2

typedef struct MyCa {
    struct MyCa     *next;
    int             value;
} *MyCa;

typedef struct MyCa     *MyCa;
typedef struct MyCa     *MyCa;
```

Representation without `-compat v4.2nicknames` and `-compat v4.2.0` specified:

```
#define          P1_PDU 1
#define          P2_PDU 2

typedef struct MyCa {
    struct MyCa     *next;
    int             value;
} *MyCa;

typedef struct MyCa     *Param;
typedef struct MyCa     *P1;
typedef struct MyCa     *P2;
```

3.5.94 -compat v4.2objHandleCstrPointer

Prior to version 5.0, the ASN.1 compiler incorrectly generated pointered structures for character string types specified with the OBJHANDLE/NOCOPY directive. Starting with version 5.0 such character string structures no longer are pointered.

The `-compat v4.2objHandleCstrPointer` option disables the new behavior providing compatibility with version numbers lower than 5.0.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
        StringA ::= IA5String --<OBJHANDLE>--
END
```

Representation with `-compat v4.2objHandleCstrPointer` specified:

```
typedef struct StringA {
    unsigned int    length;
    char            *value;
} *StringA;
```

Representation without `-compat v4.2objHandleCstrPointer` specified:

```
typedef struct StringA {
    unsigned int    length;
    char            *value;
} StringA;
```

3.5.95 -compat v4.2octetStringDefault

During version 4.2, the ASN.1 compiler used the VARYING representation for OCTET STRING types with a SizeConstraint larger than 256. Starting with version 5.0.1, such OCTET STRING types take on the UNBOUNDED representation.

The `-compat v4.2octetStringDefault` option disables the new behavior providing compatibility with version 4.2.

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   DefaultOctetString   ::= OCTET STRING (SIZE(300))
END
```

Representation with `-compat v4.2octetStringDefault` specified:

```
typedef struct DefaultOctetString {
    unsigned short  length;
    unsigned char   value[300];
} DefaultOctetString;
```

Representation without `-compat v4.2octetStringDefault` specified:

```
typedef struct DefaultOctetString {
    unsigned short  length;
    unsigned char   *value;
} DefaultOctetString;
```

3.5.96 -compat v5.0.0

Provides compatibility with version 5.0.0 of the ASN.1 compiler.

The `v5.0.0` flag is equivalent to specifying the compat options: `oldInternalDefineNames, v4.2badUnderScorePrefix, v5.0.0namesPrefixes, v5.0.0badNamesForNamedItem,` and, `v5.0.1`.

3.5.97 -compat v5.0.0badNamesForNamedItems

During version 5.0.0, the ASN.1 compiler interpreted the ASN1.Nickname directive as having a general effect when used on a named number (for the INTEGER type), named bit (for the BIT STRING type), or named enumerator (for the ENUMERATED type) nested in a component of a parameterized type. In other words, applying the Nickname directive to an instance of a parameterized type in order to change the name of some named item in a nested INTEGER, BIT STRING, or ENUMERATED type caused the changing of this name in all other instance of this parameterized type. Starting with version 5.0.1, the ASN1.Nickname directive only affects the parameterized-type instance that it is applied to.

The `-compat v5.0.0badNamesForNamedItems` disables the new behavior providing compatibility with version 5.0.0.

Example:

ASN.1:

```
--<ASN1.Nickname Mod.SearchArgument.signed.subset.baseObjectInt
-- nicknamed_baseObjectInt>--

Mod DEFINITIONS ::= BEGIN
   SearchArgument  ::=     Param {SET {
           subset          [1]     INTEGER {
                   baseObjectInt(0), oneLevelInt(1)}
                   DEFAULT baseObjectInt }} --<PDU>--
   Param {Type} ::= SET {
    signed [1] Type,
    unsigned [2] Type,
    a [3] Type
   }
END
```

Representation with `-compat v5.0.0badNamesForNamedItems` specified:

```
typedef struct _set1 {
```

```
    unsigned char   bit_mask;
#       define      subset_present 0x80
    int             subset;  /* subset_present not set in bit_mask implies
value
                             * is baseObjectInt */
#       define      nicknamed_baseObjectInt 0
#       define      oneLevelInt 1
#       define      nicknamed_baseObjectInt 0
#       define      Mod_Param_signed_subset_oneLevelInt_1 1
#       define      nicknamed_baseObjectInt 0
#       define      Mod_Param_signed_subset_oneLevelInt_2 1
} _set1;

typedef struct Param {
    _set1           Param_signed;
    _set1           Param_unsigned;
    _set1           a;
} Param;

typedef Param SearchArgument;
```

Representation without `-compat v5.0.0badNamesForNamedItems` **specified:**

```
typedef struct _set1 {
    unsigned char   bit_mask;
#       define      subset_present 0x80
    int             subset;  /* subset_present not set in bit_mask implies
value
                             * is baseObjectInt */
#       define      nicknamed_baseObjectInt 0
#       define      oneLevelInt 1
#       define      Mod_Param_signed_subset_baseObjectInt_1 0
#       define      Mod_Param_signed_subset_oneLevelInt_1 1
#       define      Mod_Param_signed_subset_baseObjectInt_2 0
#       define      Mod_Param_signed_subset_oneLevelInt_2 1
} _set1;

typedef struct Param {
    _set1           Param_signed;
    _set1           Param_unsigned;
    _set1           a;
} Param;

typedef Param SearchArgument;
```

3.5.98 -compat v5.0.0namesPrefixes

During version 5.0.0, the ASN.1 compiler unnecessarily prefixed module names to `#define` statements with suffixes of `_chosen` and `_present` when the OSS.TYPENAME directive was used to rename a CHOICE alternative or an OPTIONAL field. Starting with version 5.0.1, such unnecessary prefixing is no longer done.

The `-compat v5.0.0namesPrefixes` disables the new behavior providing compatibility with version 5.0.0.

Example:

ASN.1:

```
--<OSS.TYPENAME  SNI-Extensions.AA.unsigned "AA_unsigned">--

SNI-Extensions DEFINITIONS ::= BEGIN
   AA ::= CHOICE { unsigned        SET OF INTEGER }
END
```

Representation `-compat v5.0.0namesPrefixes` **specified:**

```
typedef struct AA {
    unsigned short  choice;
#       define      SNI_Extensions_AA_unsigned_chosen 1
    union {
        struct AA_unsigned {
            struct AA_unsigned *next;
            int             value;
        } *AA_unsigned; /* to choose, set choice to
                         * SNI_Extensions_AA_unsigned_chosen */
    } u;
} AA;
```

Representation without `-compat v5.0.0namesPrefixes` **specified:**

```
typedef struct AA {
    unsigned short  choice;
#       define      AA_unsigned_chosen 1
    union {
        struct AA_unsigned {
            struct AA_unsigned *next;
            int             value;
        } *AA_unsigned;  /* to choose, set choice to AA_unsigned_chosen */
    } u;
} AA;
```

3.5.99 -compat v5.0.0nicknames

Prior to version 5.0.5, the application of the ASN1.Nickname and OSS.TYPENAME directives to certain parameterized types had no effect. Starting with version 5.0.5, such types are affected by the ASN1.Nickname and OSS.TYPENAME directives.

The `-compat v5.0.0nicknames` option disables the new behavior providing compatibility with version numbers lower than 5.0.5. Note: starting with version 5.2.1 of the ASN.1/C compiler, this option has no effect unless the `-compat v5.2noextraparamref` option is also specified.

Example:

ASN.1:

```
--<ASN1.Nickname  Mod.CA "MyCa">--

Mod DEFINITIONS ::= BEGIN
  CA ::= SET OF INTEGER
  Param {Type} ::= CA
  P1 ::= Param {INTEGER}
  P2 ::= Param {BOOLEAN}
END
```

Representation with `-compat v5.0.0nicknames` and `-compat v5.0.4` specified:

```
#define          P1_PDU 1
#define          P2_PDU 2

typedef struct CA {
    struct CA       *next;
    int             value;
} *CA;

typedef struct CA       *P1;
typedef struct CA       *P2;
```

Representation without `-compat v5.0.0nicknames` or `-compat v5.0.4` specified:

```
#define          P1_PDU 1
#define          P2_PDU 2

typedef struct MyCa {
    struct MyCa     *next;
    int             value;
} *MyCa;

typedef struct MyCa     *Param;
typedef struct MyCa     *P1;
typedef struct MyCa     *P2;
```

3.5.100 -compat v5.0.1

Provides compatibility with version 5.0.1 of the ASN.1 compiler.
The `v5.0.1` flag is equivalent to specifying the compat option: `v4.2octetStringDefault, v5.0.4`

3.5.101 -compat v5.0.4

Provides compatibility with version 5.0.4 of the ASN.1 compiler. The `v5.0.4` flag is equivalent to specifying the compat options: `badSetOfOidWithPointer, v5.0.0nicknames, v5.0.6`

3.5.102 -compat v5.0.6

Provides compatibility with version 5.0.6 of the ASN.1 compiler. The `v5.0.6` flag is equivalent to specifying the compat options: `oldEncodableNames, v5.0.8`

3.5.103 -compat v5.1extraPointer

During version 5.1, the ASN.1 compiler allowed the application of an OSS.POINTER directive to linked SET OF / SEQUENCE OF types even if they circularly referenced each other. Starting from version 5.2, the ASN.1 compiler does not allow the application of the OSS.POINTER directive to SET OF / SEQUENCE OF types in such cases.

The `-compat v5.1extraPointer` option disables the new behavior providing compatibility with version number 5.1.

3.5.104 -compat v5.1parameterizedTypes

In version 5.1, the ASN.1 compiler produced the most general representation for restricted character types, BIT STRING, OCTET STRING, and INTEGER types with parameterized size or range subtype constraints. Subsequent versions of the ASN.1 compiler generate an UNBOUNDED representation for such restricted character strings, BIT STRING, and OCTET STRING types, and an appropriate `int` representation for INTEGER types.

The `-compat v5.1parameterizedTypes` option disables the new behavior providing compatibility with version number 5.1.

3.5.105 -compat v5.1typesharing

During version 5.1, the ASN.1 compiler used a different algorithm to perform typesharing for parameterized types. Subsequent versions of the ASN.1 compiler use a slightly different algorithm in certain rare cases.

The `-compat v5.1typesharing` option disables the new behavior providing compatibility with version number 5.1.

3.5.106 -compat v5.1unnamedStructForConstrBy

Prior to version 5.2, the ASN.1 compiler generated inline types for unnamed structures within CONSTRAINED BY clauses when the `-C++` and `-userConstraints` options were both specified.

Starting with version 5.2, such unnamed structures are extracted and defined outside of their containing structure. The new behavior results in a better and simpler naming scheme for parameterized types in the generated user-defined-constraint-checking functions.

The `-compat v5.1unnamedStructForConstrBy` option disables the new behavior providing compatibility with version numbers lower than 5.2.

Related options:

`-compat unnamedStructForContrBy`

3.5.107 -compat v5.1.0

Provides compatibility with version 5.1.0 of the ASN.1 compiler. The `v5.1.0` flag is equivalent to specifying the compat options: `noParamTypesharing`, `v5.1extraPointer`, `v5.1typesharing`, `v5.1.0parameterizedTypes`, `v5.1.3`

3.5.108 -compat v5.1.3

Provides compatibility with version 5.1.3 of the ASN.1 compiler. The `v5.1.3` flag is equivalent to specifying the compat options: `oldSharingFieldsWithDirectives`, `v5.1unnamedStructForConstrBy`, `v5.1.4`

3.5.109 -compat v5.1.4

Provides compatibility with version 5.1.4 of the ASN.1 compiler. The `v5.1.4` flag is equivalent to specifying the compat options: `badRefNames`, `noParamTypeSharing`, `v5.1parameterizedtypes`, `v5.1typesharing`, `v5.1unnamedStructForConstrBy`, `v5.2.0`.

3.5.110 -compat v5.2nestedUsethisTypes

During version 5.2, the ASN.1 compiler incorrectly handled types with the OSS.UseThis directive applied to them if they were defined as a reference to another type. Starting with version 5.3.0, such nested applications of the OSS.UseThis directive are now correctly handled.

The `-compat v5.2nestedUsethisTypes` option disables the new behavior providing compatibility with version number 5.2.

Example:

ASN.1:

```
--<OSS.UseThis Mod.B Mod.A>--
```

```
Mod DEFINITIONS ::= BEGIN
     A ::= SEQUENCE OF INTEGER
     B ::= BB
     BB ::= SET (SIZE(1..20)) OF INTEGER
     C ::= CHOICE { a B}
END
```

Representation with `-compat v5.2nestedUsethisTypes` specified:

```
typedef struct A {
    struct A        *next;
    int             value;
} *A;

typedef struct BB {
    struct BB       *next;
    int             value;
} *BB;

typedef struct C {
    unsigned short  choice;
#       define      a_chosen 1
    union {
        struct BB       *a;  /* to choose, set choice to a_chosen */
    } u;
} C;
```

Representation without `-compat v5.2nestedUsethisTypes` specified:

```
typedef struct A {
    struct A        *next;
    int             value;
} *A;

typedef struct BB {
    struct BB       *next;
    int             value;
} *BB;

typedef struct C {
    unsigned short  choice;
#       define      a_chosen 1
    union {
        struct A        *a;  /* to choose, set choice to a_chosen */
    } u;
} C;
```

3.5.111 -compat v5.2noExtraParamRef

During version 5.2, the ASN.1 compiler did not generate a separate `typedef` for an ASN.1 type referenced by a parameterized type. Starting with version 5.3.0, a separate `typedef` is produced in such a case.

The `-compat v5.2noExtraParamRef` option disables the new behavior providing compatibility with version number 5.2.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN
   MyType ::= IA5String
   P {Type} ::= MyType (CONSTRAINED BY {Type})
   P1 ::= SET OF P {INTEGER (1..2)}
   P2 ::= CHOICE { a P {INTEGER (20..22)}}
END
```

Representation with `-compat v5.2noExtraParamRef` specified:

```
typedef char            *MyType;

typedef struct P1 {
    struct P1       *next;
    MyType          value;
} *P1;

typedef struct P2 {
    unsigned short  choice;
#       define      a_chosen 1
    union {
        MyType          a;  /* to choose, set choice to a_chosen */
    } u;
} P2;

typedef unsigned short  P1_integer;

typedef unsigned short  P2_integer;
```

Representation without `-compat v5.2noExtraParamRef` specified:

```
typedef char            *MyType;

typedef MyType          P;

typedef struct P1 {
    struct P1       *next;
    P               value;
} *P1;

typedef struct P2 {
    unsigned short  choice;
#       define      a_chosen 1
    union {
        P               a;  /* to choose, set choice to a_chosen */
    } u;
} P2;
```

```
typedef unsigned short  P1_integer;

typedef unsigned short  P2_integer;
```

3.5.112 -compat v5.2paramNickname

During version 5.2, the ASN.1 compiler sometimes incorrectly handled the application of the OSS.TYPENAME and ASN1.Nickname directive on parameterized types; previously, some of the instances of such types were not affected by the directive application while they should have been (e.g., those instances with the OSS.ExtractType directive applied to them). Additionally, some parameterized SET OF and SEQUENCE OF structures may have gotten an incorrect C representation (ARRAY instead of LINKED) if the OSS.TYPENAME directive was applied to them. Starting with version 5.3.0, these errors were fixed.

The `-compat v5.2paramNickname` option disables the new behavior providing compatibility with version number 5.2.

3.5.113 -compat v5.2paramRangeConstraints

During version 5.2, the ASN.1 compiler produced the `long int` representation for instances of INTEGER types with dummy parameterized range constraints, even if the instances were of a value that surpassed the capacity of signed `long int`. Starting with version 5.3.0, the ASN.1 compiler now generates either a signed `long int` or the `LONG_LONG` representation for all instances of INTEGER types with dummy parameter range constraints depending upon the size of the largest instance.

The `-compat v5.2paramRangeConstraints` option disables the new behavior providing compatibility with version number 5.2.

3.5.114 -compat v5.2sharing

During version 5.2, the ASN.1 compiler used a special typesharing algorithm which was replaced by an improved one in version 5.3.0. Specifically, two things have been changed:

i) Now, one common C `typedef` is produced for several ASN.1 types only if no TYPENAME/Nickname/FIELDNAME/DefineName directives are applied to these types or if the same directives with identical operators are applied to each of these types.

ii) The ASN.1 compiler now will not produce a common `typedef` for ARRAY or UNBOUNDED SET OF/SEQUENCE OF with different representations for their `count` fields but will still produce such for PADDED BIT STRINGs or LINKED OBJECT IDENTIFIERs with identical C representations but different representations for their `length` field .

The `-compat v5.2sharing` option disables the new behavior providing compatibility with version number 5.2.

3.5.115 -compat v5.2typesharing

During version 5.2, the ASN.1 compiler experienced problems in typesharing with a rare type of parameterization. Starting with version 5.3.0, the ASN.1 compiler now correctly handles such syntax.

The `-compat v5.2typesharing` option disables the new behavior providing compatibility with version number 5.2.

3.5.116 -compat v5.2.0

Provides compatibility with version 5.2.0 of the ASN.1 compiler. The `v5.2.0` flag is equivalent to specifying the compat options: `charUTF8String`, `v5.2paramRangeConstraints`, `v5.2paramNicknames`, `v5.2sharing`, `v5.2noExtraParamRef`, `v5.2nestedUsethisTypes`, `v5.2.1`.

3.5.117 -compat v5.2.1

Provides compatibility with version 5.2.1 of the ASN.1 compiler. The `v5.2.1` flag is equivalent to specifying the compat options: `oldnamesmanglingwithprefix`, `v5.2typesharing`, `v5.3.0`.

3.5.118 -compat v5.3.0

Provides compatibility with version 5.3.0 of the ASN.1 compiler.

3.5.119 -compat v5.3.1

Provides compatibility with version 5.3.1. This includes forcing the ASN.1 compiler to print numbers instead of LLONG_MIN, LLONG_MAX, ULLONG_MAX definitions into the .c output file., allowing one to proceed with old c-header files that do not contain these definitions.

3.5.120 -compat v5.4integer

During version 5.4, LONG_LONG was the default representation for INTEGER types with an extensible size constraint. Without using the -compat flag, it is still possible to get LONG_LONG representation either by applying the LONGLONG directive or by specifying constraint boundaries that do not fit within the 4-byte int range.

3.5.121 -compat v5.4.0

Provides compatibility with version 5.4.0 of the ASN.1 compiler.

3.5.122 -compat v5.4.2

Provides compatibility with version 5.4.2 of the ASN.1 compiler.

3.5.123 -compat v5.4.4

Provides compatibility with version 5.4.4 As of 5.4.4, the ASN.1 compiler handles cases where an INTEGER with the HUGE directive is present as a field of a structured type and the -code option is specified. Previously COMPILER ERROR #22 had been issued. To restore the previous behavior use the -compat noHugeIntegerStructNames or -compat v5.4.4 options. Note that these -compat options can cause a compiler error.

As of 5.4.4, the ASN.1 compiler correctly handles CHOICE types acting as fields to a CLASS definition when the -code option is specified. The union representing the CHOICE type is defined outside the C struct of the containing CLASS definition. Previously, the C-representation of the field was generated inline within the CLASS type. This may have resulted in duplications of the manifest constants (used to identify which alternative is contained within the union) if there was a field name conflict. The 'v5.4typesharing' or 'v5.4.4' -compat flag can be used to restore the previous behavior of the ASN.1 compiler.

3.5.124 -compat v6.0.0

Provides compatibility with version 6.0.0 of the ASN.1 compiler.

3.5.125 -compat v6.0stringswithMAXsize

The ASN.1 compiler was changed in such a way as to generate the NULLTERM C-representation for character strings with SIZE constraints defined with the MAX keyword on 64-bit platforms. With version 6.0 the ASN.1 compiler generated different C-representations for such strings on 64-bit and on 32-bit platforms.

For backward compatibility, a new compiler option, "-compat v6.0stringswithMAXsize", is available to restore the previous behavior of the ASN.1 compiler on 64-bit platforms.

3.5.126 -compat v6.1.3varyingbitstring

Prior to version 6.1.4, the ASN.1 compiler generated the same representation for unconstrained BIT STRING type with present named bits and the POINTER and VARYING directives applied as for a constrained BIT STRING with an upper bound of its SIZE constraint equal to the highest named bit plus one.

Starting with version 6.1.4, the representation for such BIT STRING types has been made identical to the representation for the same unconstrained BIT STRING without named bits.

The -compat v6.1.3varyingbitstring option disables the new behavior providing compatibility with version numbers lower than 6.1.4.

Example:

ASN.1:

```
Mod DEFINITIONS ::=
BEGIN
    PVaryNamedBits ::= BIT STRING {b1(1), b12(12)} --<POINTER>-- --<VARYING>-
-
END
```

Representation with -compat v6.1.3varyingbitstring **specified:**

```
typedef struct PVaryNamedBits {
    unsigned short  length;  /* number of significant bits */
    unsigned char   value[2];
} *PVaryNamedBits;
#define                     b1 0x40
#define                     b1_byte 0
#define                     b12 0x08
#define                     b12_byte 1
```

Representation without -compat v6.1.3varyingbitstring **specified:**

```
typedef struct PVaryNamedBits {
    unsigned short  length;  /* number of significant bits */
    unsigned char   value[1];  /* first element of the array */
} *PVaryNamedBits;
#define                     b1 0x40
#define                     b1_byte 0
#define                     b12 0x08
#define                     b12_byte 1
```

3.5.127 -compat v6.1.3

Provides compatibility with version 6.1.3 of the ASN.1 compiler.

3.5.128 -compat v6.1.4DefineNameSharing

Starting with version 7.0, the following changes in name-mangling rules were introduced:

- names generated for `#define bitmask` constants and enumeration values used by some shared types include names of fields which satisfy all the types using this shared type, the rest of the mangled name is some number to make the name unique;
- names generated for `#define bitmask` constants and enumeration values used by one type only contain the name of this type or its fields for mangling. Previously, the mangled names might include names of other types or fields

The `-compat v6.1.4DefineNameSharing` option can be used to disable the above-mentioned changes providing compatibility with version numbers 6.1.4 or lower.

Example:

ASN.1:

```
Z DEFINITIONS AUTOMATIC TAGS ::= BEGIN
  A ::= SET { f1 BOOLEAN OPTIONAL }
  B ::= SET { f1 BOOLEAN OPTIONAL }
  C ::= SET OF SET { f1 BOOLEAN OPTIONAL }
  D ::= SET OF SET { f1 BOOLEAN OPTIONAL }
  E ::= SET { f1 BOOLEAN OPTIONAL, f2 INTEGER }
END
```

Representation without `-compat v6.1.4DefineNameSharing` specified:

```
typedef struct A {
     unsigned char   bit_mask;
 #       define      A_f1_present 0x80
     ossBoolean      f1;  /* optional; set in bit_mask A_f1_present if
                             present */
} A;

typedef struct B {
     unsigned char   bit_mask;
 #       define      B_f1_present 0x80
     ossBoolean      f1;  /* optional; set in bit_mask B_f1_present if
                            present */
} B;

typedef struct _set1 {
    unsigned char   bit_mask;
#       define      f1_1_present 0x80
    ossBoolean      f1;  /* optional; set in bit_mask f1_1_present if
                            present */
} _set1;

typedef struct C {
    struct C        *next;
    _set1           value;
} *C;

typedef struct D {
    struct D        *next;
    _set1           value;
```

```
} *D;

typedef struct E {
     unsigned char   bit_mask;
 #       define      E_f1_present 0x80
     ossBoolean      f1;  /* optional; set in bit_mask E_f1_present if
                             present */
    int             f2;
} E;
```

Note that the `B` type uses the `B_f1_present` define name and the `shared _seq1` type used by both the `C` and `D` types includes the `f1_1_present` define name.

Representation with -compat v6.1.4DefineNameSharing **specified:**

```
typedef struct A {
     unsigned char   bit_mask;
#       define      A_f1_present 0x80
    ossBoolean      f1;  /* optional; set in bit_mask A_f1_present if
                            present */
} A;

typedef struct B {
    unsigned char   bit_mask;
#       define      A_f1_present 0x80
    ossBoolean      f1;  /* optional; set in bit_mask A_f1_present if
                            present */
} B;

typedef struct _set1 {
    unsigned char   bit_mask;
#       define      A_f1_present 0x80
    ossBoolean      f1;  /* optional; set in bit_mask A_f1_present if
                            present */
} _set1;

typedef struct C {
    struct C        *next;
    _set1           value;
} *C;

typedef struct D {
    struct D        *next;
    _set1           value;
} *D;

typedef struct E {
    unsigned char   bit_mask;
#       define      E_f1_present 0x80
    ossBoolean      f1;  /* optional; set in bit_mask E_f1_present if
                            present */
    int             f2;
} E;
```

Note that the `A_f1_present` define name used by the `A, B, C, D` types.

3.5.129 -compat v6.1.4extrapointer

The ASN.1 compiler no longer applies an extra POINTER directive in rare cases to the fields of a SEQUENCE, SET, or CHOICE type when the field types are recursive and circularly reference themselves via a certain SEQUENCE OF or SET OF type and one of the following conditions occurs:

a. the SEQUENCE OF or SET OF type has a LINKED/DLINKED C-representation and the SIZE constraint is applied, and the recursive field is not the first component of the SEQUENCE, SET, or CHOICE type;

b. the SEQUENCE OF or SET OF type has an UNBOUNDED C-representation and the SIZE constraint is applied, and the UNBOUNDED representation is assigned using a global directive:

--<OSS.UNBOUNDED SEQUENCE OF, SET OF>--

Previously in cases a) and b) the ASN.1 compiler internally applied an extra POINTER directive to such recursive components of SEQUENCE, SET, or CHOICE types, causing in case a) these components to be double-pointered. Now such recursive components do not get an extra pointer.

This incorrect behavior is present in versions 6.1.2 to 6.1.4 and 7.0BetaA of the ASN.1/C compiler. The new compat flag "-compat v6.1.4extrapointer", restores the old incorrect behavior of the ASN.1 Compiler.

3.5.130 -compat v6.1.4ReferencesToLeanTypes

When the `-lean` option was introduced in 5.1, the compiler produced references to named simple types incorrectly, using the LED built-in type name (such as ossOctetString) instead of a user-defined type name.

Additionally, if the referenced type is a BIT STRING with a named bit list, defines for the named bits are no longer duplicated in the output header file. For instance, given

```
B ::= BIT STRING {b0(0)}

Name ::= SEQUENCE { bit B }
```

The compiler previously produced the following C typedefs:

```
typedef ossBitString B;
#define                     b0 0x80
#define                     b0_byte 0


typedef struct Name {
    ossBitString    bit;
#       define      b0 0x80
#       define      b0_byte 0
} Name;
```

Starting with version 7.0, it produces:

```
typedef ossBitString B;
#define                     b0 0x80
```

```
#define                     b0_byte 0

typedef struct Name {
   B                bit;
} Name;
```

The `-compat v6.1.4ReferencesToLeanTypes` option disables the new behavior providing compatibility with version 6.1.4.

3.5.131 -compat v6.1.4

Provides compatibility with version 6.1.4 of the ASN.1 compiler.

3.5.132 -compat v7.0DefineNames

Starting with version 8.2.0, the new compat flag is added to the list of all 7.0 version compat flags that restore the names generated in some #define. In prior 8 versions of the compiler, such names were qualified with top-level types names.

3.5.133 -compat v7.0pdusForBuiltinTypesInObjectSet

Prior to version 7.0.2 the ASN.1 compiler generated a separate PDU for every built-in type used in an information object set. Starting with version 7.0.2 the ASN.1 compiler does not generate a separate PDU for these built-in types.

The `-compat v7.0pdusForBuiltinTypesInObjectSet` option disables the new behavior providing compatibility with version numbers lower than 7.0.2

Example:

ASN.1:

```
KEY-IDENTIFIER ::= CLASS {
    &id  INTEGER UNIQUE,
    &Value
} WITH SYNTAX {
    SYNTAX &Value IDENTIFIED BY &id
}

KeyIdentifiers KEY-IDENTIFIER ::= {
    {SYNTAX OCTET STRING IDENTIFIED BY 2} |
    {SYNTAX OCTET STRING IDENTIFIED BY 3}
}

TestType ::= SEQUENCE {
    identifier SEQUENCE {
        idType  KEY-IDENTIFIER.&id ({KeyIdentifiers}),
```

```
            idValue KEY-IDENTIFIER.&Value ({KeyIdentifiers}{@.idType})
        }
    }
```

Representation with `-compat v7.0pdusForBuiltinTypesinObjectSet` specified:

```
    #define          TestType_PDU 1
    #define          M_KeyIdentifiers_SYNTAX_1_PDU 2
    #define          M_KeyIdentifiers_SYNTAX_2_PDU 3
    #define          KeyIdentifiers_OSET 1       /* Class is KEY-IDENTIFIER */

    typedef struct KEY_IDENTIFIER {
        int             id;
        unsigned short  Value;
    } KEY_IDENTIFIER;

    typedef struct M_KeyIdentifiers_SYNTAX_1 {
        unsigned int    length;
        unsigned char   *value;
    } M_KeyIdentifiers_SYNTAX_1;

    typedef M_KeyIdentifiers_SYNTAX_1 M_KeyIdentifiers_SYNTAX_2;

    enum KeyIdentifiers_Value_PDUs {
        PDU_KeyIdentifiers_Value_UNKNOWN = 0,

        PDU_KeyIdentifiers_Value_SYNTAX_1 = M_KeyIdentifiers_SYNTAX_1_PDU,
        PDU_KeyIdentifiers_Value_SYNTAX_2 = M_KeyIdentifiers_SYNTAX_2_PDU
    };

    union KeyIdentifiers_Value_union {
        M_KeyIdentifiers_SYNTAX_1 *pdu_KeyIdentifiers_SYNTAX_1;
                                            /* PDU_KeyIdentifiers_Value_SYNTAX_1
    */
        M_KeyIdentifiers_SYNTAX_1 *pdu_KeyIdentifiers_SYNTAX_2;
                                            /* PDU_KeyIdentifiers_Value_SYNTAX_2
    */
    };
    typedef struct KeyIdentifiers_Value {
        enum KeyIdentifiers_Value_PDUs pduNum;
        OssBuf          encoded;
        union KeyIdentifiers_Value_union decoded;
    } KeyIdentifiers_Value;

    typedef struct TestType {
        struct {
            int             idType;
            KeyIdentifiers_Value idValue;
        } identifier;
    } TestType;
```

Representation without `-compat v7.0.1pdusForBuiltinTypesinObjectSet` specified:

```
    #define          TestType_PDU 1
    #define          KeyIdentifiers_SYNTAX_PDU 2
    #define          KeyIdentifiers_OSET 1       /* Class is KEY-IDENTIFIER */

    typedef struct KEY_IDENTIFIER {
```

```
    int             id;
    unsigned short  Value;
} KEY_IDENTIFIER;

typedef struct KeyIdentifiers_SYNTAX {
    unsigned int    length;
    unsigned char   *value;
} KeyIdentifiers_SYNTAX;

enum KeyIdentifiers_Value_PDUs {
    PDU_KeyIdentifiers_Value_UNKNOWN = 0,

    PDU_KeyIdentifiers_Value_SYNTAX = KeyIdentifiers_SYNTAX_PDU
};

union KeyIdentifiers_Value_union {
    KeyIdentifiers_SYNTAX *pdu_KeyIdentifiers_SYNTAX;
                                          /* PDU_KeyIdentifiers_Value_SYNTAX
*/
};

typedef struct KeyIdentifiers_Value {
    enum KeyIdentifiers_Value_PDUs pduNum;
    OssBuf          encoded;
    union KeyIdentifiers_Value_union decoded;
} KeyIdentifiers_Value;

typedef struct TestType {
    struct {
        int             idType;
        KeyIdentifiers_Value idValue;
    } identifier;
} TestType;
```

3.5.134 -compat v7.0typeSharing

Starting with version 5.1, when sharing of generated types was introduced, up to version 7.0.0, the compiler did not share structures generated for `OCTET STRING` types with structures generated for `BIT STRING` types, even if both had the same representation. However, structures which included fields of such types could share the same C structure even if one structure contained a field of an `OCTET STRING` type and the other structure contained a field of a `BIT STRING` type. Starting with version 7.0.2 the compiler shares structures generated for these fields as well. That is, a field of an `OCTET STRING` type shares the structure generated for a field of a `BIT STRING` type if the outer structures containing these fields share the same generated type. However, when the `-lean` command line option is specified no structures are generated for `OCTET STRING`, `BIT STRING` and character string types. The structures `ossOctetString`, `ossBitString` and `ossCharString`, predefined in the ossasn1.h header file, are used instead. Therefore, starting with version 7.0.2, when `-lean` is specified the compiler does not share a generated structure between two types with the same binary representation if one of the types contains a field of an `OCTET STRING` type (generated as `ossOctetString`) and another type contains a respective field of a `BIT STRING` (or a character string) type.

In the following example, `S1.a` and `S2.a` have the same C representation only if the `-lean` option is not specified.

```
S1 ::= SEQUENCE {
    a SEQUENCE OF
            OCTET STRING
}
S2 ::= SET {
    a SEQUENCE OF
            BIT STRING,
    b BIT STRING
}
```

1. When the ASN.1 Compiler generates structures for the SOED or TOED, the fields `S1.a` and `S2.a` share the same generated structure, `_seqof1.` The generated type `_octet1` is shared between `S1.a.*`, `S2.a.*` and `S2.b`:

```
/* no compiler options specified */
typedef struct _octet1 {
    unsigned int    length;
    unsigned char   *value;
} _octet1;

typedef struct _seqof1 {
    struct _seqof1  *next;
    _octet1         value;
} *_seqof1;

typedef struct S1 {
    struct _seqof1  *a;
} S1;

typedef struct S2 {
    struct _seqof1  *a;
    _octet1         b;
} S2;
```

With the `-compat v7.0typeSharing` option the ASN.1 compiler does not share structures generated for `S1.a.*`, `S2.a.*` and `S2.b`. However, the compiler shares structures generated for `S1.a` and `S2.a`, and for the below example no `_octet1` type definition will be generated:

```
/* options specified: -compat v7.0typeSharing */
typedef struct _seqof1 {
    struct _seqof1  *next;
    struct {
        unsigned int    length;
        unsigned char   *value;
    } value;
} *_seqof1;

typedef struct S1 {
    struct _seqof1  *a;
} S1;

typedef struct _bit1 {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _bit1;

typedef struct S2 {
```

```
    struct _seqof1  *a;
    _bit1           b;
} S2;
```

This compatibility parameter may be useful only to revert to the previous enumeration of artificially generated names like "`_octet1`" and "`_bit1`". Note: starting with version 8.5.0 of the ASN.1/C compiler, this option has no effect unless the `-compat v8.4PrimitiveTypesSharing` option is also specified.

2. When the ASN.1 Compiler generates structures for the LED, the structures generated for `S1.a` and `S2.a` do not share the same type:

```
/* options specified: -lean */
typedef struct S1 {
    struct _seqof1 {
        struct _seqof1  *next;
        ossOctetString  value;
    } *a;
} S1;

typedef struct S2 {
    struct _seqof2 {
        struct _seqof2  *next;
        ossBitString    value;
    } *a;
    ossBitString    b;
} S2;
```

The `-compat v7.0typeSharing` option allows sharing of types generated for `S1.a` and `S2.a`:

```
/* options specified: -lean -compat v7.0typeSharing */
typedef struct _seqof1 {
    struct _seqof1  *next;
    ossOctetString  value;
} *_seqof1;

typedef struct S1 {
    struct _seqof1  *a;
} S1;

typedef struct S2 {
    struct _seqof1  *a;
    ossBitString    b;
} S2;
```

When the `-lean` command line option is specified, this compatibility parameter may be useful if the application code refers to `BIT STRING` field `S2.a.*` using an `ossOctetString` type as in the following fragment of code:

```
S2 s2;
ossOctetString *field = &(s2.a->value);
```

3.5.135 -compat v8.0.0

Provides compatibility with version 8.0.0 of the ASN.1 compiler. The `v8.0.0` flag is equivalent to specifying the compat option `ignore1994ExternalConstr`.

3.5.136 -compat v8.1.0

Provides compatibility with version 8.1.0 of the ASN.1 compiler. The `v8.1.0` flag is equivalent to specifying the compat options `bad1994ExternalWithContentsConstr` and `noTypeRefWithUseThisSharing`.

3.5.137 -compat v8.1.2SharingTypesWithEXERInstruction

Prior to version 8.2, the ASN.1 compiler did not always share identical types with the assigned encoding EXER instructions. Starting with version 8.2, the ASN.1 compiler uses typesharing in such a case.

The `-compat v8.1.2SharingTypesWithEXERInstruction` option disables the new behavior and provides compatibility with version 8.1.2.

3.5.138 -compat v8.1.2

Provides compatibility with version 8.1.2 of the ASN.1 compiler.

3.5.139 -compat v8.1.2ParamRefSharingWithDirectives

Prior to version 8.2, the ASN.1 compiler changed typedefs generated for parameterized types with shared instances, if certain directives were applied.

Starting with version 8.2, the same typedef is generated for a parameterized type if all instances of this type can be shared and the `OSS.TYPENAME`, `OSS.ExtractType`, `OSS.FIELDNAME`, or `OSS.DefineName` directives are applied to some of the instances or to the parameterized type itself. The new compat flag, `v8.1.2ParamRefSharingWithDirectives`, can be used to restore the old behavior.

Example:

ASN.1:

```
M DEFINITIONS AUTOMATIC TAGS ::= BEGIN
     T1 ::= SET { a  P1 {BIT STRING} OPTIONAL}
     T2 ::= SET OF P1 {BIT STRING}
```

```
     P1 {PT} ::= SET OF PT
END
```

Generated .h file:

```
typedef struct T1 {
    unsigned char   bit_mask;
#       define      a_present 0x80
    struct P1       *a;  /* optional; set in bit_mask a_present if present
*/
} T1;

typedef struct T2 {
    struct T2       *next;
    struct P1       *value;
} *T2;

typedef struct P1 {
    struct P1       *next;
    struct {
        unsigned int    length;  /* number of significant bits */
        unsigned char   *value;
    } value;
} *P1;
```

The following directives are added:

```
--<OSS.TYPENAME M.P1 "P1">--
--<OSS.FIELDNAME M.T1.a "a">--
```

The compiler now generates exactly the same structure for the parameterized type in the header file as if the compat option was not specified. The old .h file is generated with -compat v8.1.2ParamRefSharingWithDirectives:

```
typedef struct _bit1 {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _bit1;

typedef struct P1 {
    struct P1       *next;
    _bit1           value;
} *P1;

typedef struct T1 {
    unsigned char   bit_mask;
#       define      a_present 0x80
    struct P1       *a;  /* optional; set in bit_mask a_present if present
*/
} T1;

typedef struct T2 {
    struct T2       *next;
    struct P1       *value;
} *T2;
```

3.5.140 -compat v8.2.0

Provides compatibility with version 8.2.0 of the ASN.1 compiler.

3.5.141 -compat v8.2.0DefineNamesSharingWhenUsingXmlNames

Starting with version 8.3.0, the ASN.1 compiler always uses the XML name of a type, instead of its ASN.1 name, to resolve any name conflicts related to the type's components when the `-useXmlNames` option is specified.

The `-compat v8.2.0DefineNamesSharingWhenUsingXmlNames` option can be used to disable the above-mentioned change, thus providing compatibility with version numbers 8.2.0 and lower, when the compiler sometimes used the ASN.1 names of types to resolve name conflicts. Note: starting with version 8.5.0 of the ASN.1/C compiler, this option has no effect unless the `-compat v8.4TypesSharingWithAutomaticTagging` option is also specified.

Note that this bug did not exist in the compiler in helper mode, that is, when the `-helperNames` option is explicitly specified or is implied.

Example:

ASN.1:

```
B1357-2 DEFINITIONS AUTOMATIC TAGS ::= BEGIN
   A ::= SEQUENCE { s SET { f1 INTEGER OPTIONAL }}
   B ::= [XER:NAME AS "b-type"] SEQUENCE {
     s SET { f1 INTEGER OPTIONAL }
   }
   C ::= SEQUENCE { s SET { f1 INTEGER OPTIONAL, f2 BOOLEAN OPTIONAL }}
END
```

The B type representation with both `-compat v8.2.0DefineNamesSharingWhenUsingXmlNames` and `-useXmlNames` specified:

```
typedef struct b_type {
     struct {
         unsigned char   bit_mask;
#           define      B_s_f1_present 0x80
         int             f1;  /* optional; set in bit_mask B_s_f1_present
if
                                * present */
     } s;
} b_type;
```

Note that the ASN.1 name of the `B` type is used in the `B_s_f1_present` define name.

The B type representation without `-compat v8.2.0DefineNamesSharingWhenUsingXmlNames` and with `-useXmlNames` specified:

```
typedef struct b_type {
     struct {
```

```
          unsigned char   bit_mask;
#            define      b_type_s_f1_present 0x80
          int             f1;  /* optional; set in bit_mask
b_type_s_f1_present if
                                * present */
     } s;
} b_type;
```

Note that the xml name of the `B` type is used in the `b_type_s_f1_present` define name here.

3.5.142 -compat v8.3.1

Provides compatibility with version 8.3.1 of the ASN.1 compiler. The `-compat v8.3.1` flag is equivalent to specifying the `-compat ignoreInlineTypeDirForSharedTypes` option.

3.5.143 -compat v8.4.0

Provides compatibility with version 8.4.0 of the ASN.1 compiler.
This flag does not replace the `-compat badSharingForTypesWithInlineTypeDir`, the `-compat v8.4PrimitiveTypeSharing` or the `-compat v8.4ExtraTypedefForParamRefWithUseThis` flags.

3.5.144 -compat v8.4DirForInstancesOfParamRef

Prior to version 8.5.0, the .gen file that was generated with the `-keepnames` option, could not be used to restore old names for shared instances of parameterized types.

The `-compat v8.4DirForInstancesOfParamRef` option, when it is used along with the `-compat v8.4ExtraTypedefForParamRefWithUseThis` option, restores the incorrect old behavior of the compiler.

Example:
The following syntax was compiled with `-keepnames` and was then compiled with the generated .gen file.

```
M DEFINITIONS AUTOMATIC TAGS ::= BEGIN

    PDU-P1 {INTEGER: nVal} ::= SEQUENCE {
        a  INTEGER (nVal)
    }
    PDU-P2 {INTEGER: setSz} ::= CHOICE {
            b SET OF PDU-P1 {setSz}
    }
    PDU-1 ::= SET {
            c PDU-P2 {2}
    }
    PDU-2 ::= PDU-P2 {2}
END
```

The following representation is generated:

```
typedef struct PDU_P1 {
    unsigned short  a;
} PDU_P1;

typedef struct _setof1 {
    struct _setof1  *next;
    PDU_P1          value;
} *_setof1;

typedef struct PDU_P2 {
    unsigned short  choice;
#       define      b_chosen 1
    union {
        struct _setof1  *b;  /* to choose, set choice to b_chosen */
    } u;
} PDU_P2;

typedef struct PDU_1 {
    PDU_P2          c;
} PDU_1;

typedef PDU_P2 PDU_2;
```

Representation with the following additional directives:

```
--<OSS.TYPENAME M.PDU-P1 "PDU_P1">--
--<OSS.ExtractType M.PDU-P1>--
--<OSS.TYPENAME M.PDU-1 "PDU_1">--
--<OSS.ExtractType M.PDU-1.c>--
--<OSS.UseThis M.PDU-1.c M.PDU-P2>--
--<OSS.TYPENAME M.PDU-1.c.b "_setof1">--
--<OSS.ExtractType M.PDU-1.c.b>--
--<OSS.DefineName M.PDU-1.c.b "b">--
--<OSS.UseThis M.PDU-2 M.PDU-1.c>--
--<OSS.TYPENAME M.PDU-2 "PDU_2">--
--<OSS.ExtractType M.PDU-2.b>--
--<OSS.UseThis M.PDU-2.b M.PDU-1.c.b>--
--<OSS.DefineName M.PDU-2.b "b">--
--<OSS.TYPENAME M.PDU-P1 "PDU_P1">--
--<OSS.TYPENAME M.PDU-1.c "PDU_P2">--
```

and compat flags for release 8.5.0 or later:

```
-compat v8.4DirForInstancesOfParamRef
-compat v8.4ExtraTypedefForParamRefWithUseThis
```

```
typedef struct PDU_P1 {
    unsigned short  a;
} PDU_P1;

typedef struct _setof1 {
    struct _setof1  *next;
    PDU_P1          value;
} *_setof1;

typedef struct M_PDU_P2 {
```

```
    unsigned short  choice;
#       define      b_chosen 1
    union {
        struct _setof1  *b;  /* to choose, set choice to b_chosen */
    } u;
} M_PDU_P2;

typedef M_PDU_P2 PDU_P2;

typedef struct PDU_1 {
    M_PDU_P2        c;
} PDU_1;

typedef M_PDU_P2 PDU_2;
```

3.5.145 -compat v8.4ExtraTypedefForParamRefWithUseThis

Prior to version 8.5.0, the ASN.1 compiler could generate extra unused typedefs for one of the shared instances of a parameterized type. This is a stand-alone compat flag.

The `-compat 8.4ExtraTypedefForParamRefWithUseThis` option restores the incorrect old behavior.

Example:

ASN.1:

```
--<OSS.UseThis M.PDU-2.b M.PDU-1.c.b>--
--<OSS.UseThis M.PDU-1.c M.PDU-P2>--
--<OSS.UseThis M.PDU-2 M.PDU-1.c>--

M DEFINITIONS AUTOMATIC TAGS ::= BEGIN

    PDU-P1 {INTEGER: nVal} ::= SEQUENCE {
        a  INTEGER (nVal)
    }
    PDU-P2 {INTEGER: setSz} ::= CHOICE {
            b SET OF PDU-P1 {setSz}
    }
    PDU-1 ::= SET {
            c PDU-P2 {2}
    }
    PDU-2 ::= PDU-P2 {2}
END
```

Representation without the `-compat 8.4ExtraTypedefForParamRefWithUseThis` flag for release 8.5.0 or later:

```
typedef struct PDU_P1 {
    unsigned short  a;
} PDU_P1;

typedef struct _setof1 {
    struct _setof1  *next;
```

```
    PDU_P1          value;
} *_setof1;

typedef struct PDU_P2 {
    unsigned short  choice;
#       define      b_chosen 1
    union {
        struct _setof1  *b;  /* to choose, set choice to b_chosen */
    } u;
} PDU_P2;

typedef struct PDU_1 {
    PDU_P2          c;
} PDU_1;

typedef PDU_P2 PDU_2;
```

Representation with the `-compat 8.4ExtraTypedefForParamRefWithUseThis` flag for release 8.5.0 or later:

An extra typedef is added:

```
typedef PDU_P2 _choice2;
```

3.5.146 -compat v8.4InvalidSizeForUnionConstraint

Prior to version 8.5, the ASN.1 compiler incorrectly processed a Union of SizeConstraint and SingleValue subtype constraints. The compiler truncated the type C representation using the upper bound of the size constraint. Starting with version 8.5, the C representation is limited by the size of the single value, if it is greater than the upper bound of the size constraint.

The `-compat v8.4InvalidSizeForUnionConstraint` option disables the new behavior, providing compatibility with pre-8.5 versions.

Example:

ASN.1:

```
X1 DEFINITIONS ::= BEGIN
OS ::= OCTET STRING ('010203040506'H | SIZE(4))
END
```

Representation without the `-compat v8.4InvalidSizeForUnionConstraint` flag for release 8.5 or later:

```
typedef struct OS {
     unsigned short  length;
     unsigned char   value[6];
} OS;
```

Representation with the `-compat v8.4InvalidSizeForUnionConstraint` flag for release 8.5 or later:

```
typedef struct OS {
     unsigned short  length;
     unsigned char   value[4];
} OS;
```

3.5.147 -compat v8.4PrimitiveTypeSharing

Prior to version 8.5.0, the ASN.1 compiler could generate extra unneeded typedefs for similar types that appear within other similar types linked by OSS.UseThis directives.

The `-compat v8.4PrimitiveTypeSharing` option is a stand-alone flag that restores the incorrect behavior of the 8.4.0 version.

Example:

ASN.1:

```
--<OSS.TYPENAME M1.T1.a "_seq1">--
--<OSS.ExtractType M1.T1.a>--
--<OSS.TYPENAME M1.T2.a "_seq2">--
--<OSS.ExtractType M1.T2.a>--
--<OSS.ExtractType M1.T2.b>--

--<OSS.UseThis M1.T2.a.f1 M1.T1.a>--
--<OSS.UseThis M1.T2.a.f2 M1.T1.a>--
--<OSS.UseThis M1.T2.b M1.T2.a>--

M1 DEFINITIONS AUTOMATIC TAGS ::= BEGIN

T1 ::= CHOICE {a SEQUENCE {} }
T2 ::= CHOICE {
        a SEQUENCE {
          f1 SEQUENCE {},
          f2 SEQUENCE {}
        },
   b  SEQUENCE {
          f1 SEQUENCE {},
          f2 SEQUENCE {}
        }
}
END
```

Representation without the `-compat v8.4PrimitiveTypeSharingt` flag for release 8.5.0 or later:

```
typedef struct _seq1 {
    char            placeholder;
} _seq1;

typedef struct T1 {
    unsigned short  choice;
#       define      T1_a_chosen 1
    union {
        _seq1           a;  /* to choose, set choice to T1_a_chosen */
```

```
    } u;
} T1;

typedef struct _seq2 {
    _seq1           f1;
    _seq1           f2;
} _seq2;

typedef struct T2 {
    unsigned short  choice;
#       define      T2_a_chosen 1
#       define      b_chosen 2
    union {
        _seq2           a;  /* to choose, set choice to T2_a_chosen */
        _seq2           b;  /* to choose, set choice to b_chosen */
    } u;
} T2;
```

Representation with the `-compat v8.4PrimitiveTypeSharing` flag for release 8.5.0 or later, an extra "`_seq1_2`" typedef is generated:

```
typedef struct _seq1 {
    char            placeholder;
} _seq1;

typedef struct T1 {
    unsigned short  choice;
#       define      T1_a_chosen 1
    union {
        _seq1           a;  /* to choose, set choice to T1_a_chosen */
    } u;
} T1;

typedef struct _seq2 {
    _seq1           f1;
    _seq1           f2;
} _seq2;

typedef struct _seq1_2 {
    char            placeholder;
} _seq1_2;

typedef struct T2 {
    unsigned short  choice;
#       define      T2_a_chosen 1
#       define      b_chosen 2
    union {
        _seq2           a;  /* to choose, set choice to T2_a_chosen */
        _seq2           b;  /* to choose, set choice to b_chosen */
    } u;
} T2;
```

3.5.148 -compat v8.4TypesSharingWithAutomaticTagging

Prior to version 8.5.0, the ASN.1 compiler could generate duplicate structures in the header file for similar ASN.1 types that are defined in the module with automatic tagging, instead of sharing such structures.

The `-compat v8.4TypesSharingWithAutomaticTagging` option restores the incorrect old behavior.

Example:

ASN.1:

```
T DEFINITIONS AUTOMATIC TAGS ::= BEGIN
Seq1 ::= [1]SEQUENCE {
      f1 BOOLEAN,
      f2 CHOICE {a INTEGER, b SET {bf IA5String}}}
Seq2 ::= [2]SEQUENCE {
      f1 BOOLEAN,
      f2 CHOICE {a INTEGER, b SET {bf IA5String}}}
END
```

Representation without the `-compat v8.4TypesSharingWithAutomaticTagging` flag for release 8.5.0 or later:

```
typedef struct _set1 {
    char            *bf;
} _set1;

typedef struct _choice1 {
    unsigned short  choice;
#       define      a_chosen 1
#       define      b_chosen 2
    union {
        int             a;  /* to choose, set choice to a_chosen */
        _set1           b;  /* to choose, set choice to b_chosen */
    } u;
} _choice1;

typedef struct Seq1 {
    ossBoolean      f1;
    _choice1        f2;
} Seq1;

typedef struct Seq2 {
    ossBoolean      f1;
    _choice1        f2;
} Seq2;
```

Representation with the `-compat v8.4TypesSharingWithAutomaticTagging` flag for release 8.5 or later:

```
typedef struct Seq1 {
    ossBoolean      f1;
    struct {
```

```
        unsigned short  choice;
#           define      a_chosen 1
#           define      b_chosen 2
        union {
            int             a;  /* to choose, set choice to a_chosen */
            struct _set1 {
                char            *bf;
            } b;  /* to choose, set choice to b_chosen */
        } u;
    } f2;
} Seq1;

typedef struct Seq2 {
    ossBoolean      f1;
    struct {
     unsigned short  choice;
#           define      a_chosen 1
#           define      b_chosen 2
        union {
            int             a;  /* to choose, set choice to a_chosen */
            struct _set1    b;  /* to choose, set choice to b_chosen */
        } u;
    } f2;
} Seq2;
```

3.5.149 -compat v8.5.0

Provides compatibility with version 8.5.0 of the ASN.1 compiler.
This flag does not replace the `-compat v8.5FalseExtendedConstraintEncoding` or the `-compat v8.5TableConstraintForExtensibleObjectSets` flags.

3.5.150 -compat v8.5FalseExtendedConstraintEncoding

Prior to version 8.6, the OSS PER runtime mishandled extensible SIZE constraints with no upper bound and extensible single value constraints on SEQUENCE/SET OF and OCTET/BIT STRING, for example:

```
SEQUENCE ({0}, ..., {1,2,3}) OF INTEGER
```

As a result, the PER encoder sometimes produced invalid encodings and the PER decoder failed to decode those encodings or incorrectly decoded them. Starting with version 8.6, the OSS ASN.1 compiler and the OSS PER runtime correctly handle such specifications.

The `-compat v8.5FalseExtendedConstraintEncoding` option disables the new behavior providing compatibility with versions earlier than 8.6.

Example:

ASN.1:

```
Exts DEFINITIONS::= BEGIN
```

```
    B ::= BIT STRING (SIZE(6..MAX, ...))
    value B ::= '11111111'B
END
```

PER encoding with `-compat v8.5FalseExtendedConstraintEncoding` specified:

```
08FF
```

PER decoding with `-compat v8.5FalseExtendedConstraintEncoding` specified:

```
D0081S: End of input reached before message was fully decoded;
```

PER encoding without `-compat v8.5FalseExtendedConstraintEncoding` specified:

```
0008FF
```

PER decoding with `-compat v8.5FalseExtendedConstraintEncoding` specified:

```
value B ::= '11111111'B
```

3.5.151 -compat v8.5DLinkedSeqOfSetOfWithExtSizeConstraint

Prior to version 8.6, the ASN.1 compiler incorrectly generated a LINKED representation for SET OF and SEQUENCE OF types with extensible size constraints and the DLINKED/DLINKED-PLUS directive applied. Starting with version 8.6, the ASN1 compiler generates the DLINKED/DLINKED-PLUS representation for such types.

The `-compat v8.5DLinkedSeqOfSetOfWithExtSizeConstraint` option restores the incorrect old behavior.

Example:

ASN.1:

```
M DEFINITIONS::= BEGIN
    SDEB ::= [2]SEQUENCE (SIZE(6..200, ...)) --<DLINKED-PLUS>-- OF
INTEGER
END
```

Representation without the `-compat v8.5DLinkedSeqOfSetOfWithExtSizeConstraint` flag for release 8.6.0 or later:

```
typedef struct SDEB {
    struct SDEB_node *head;
    struct SDEB_node *tail;
    unsigned int    count;
} SDEB;

typedef struct SDEB_node {
    struct SDEB_node *next;
    struct SDEB_node *prev;
```

```
    int             value;
} SDEB_node;
```

Representation with the `-compat v8.5DLinkedSeqOfSetOfWithExtSizeConstraint` flag for release 8.6.0 or later:

```
typedef struct SDEB {
    struct SDEB     *next;
    int             value;
} *SDEB;
```

3.5.152 -compat v8.5TableConstraintForExtensibleObjectSets

Prior to version 8.6, the OSS Constraint Checker mishandled table/component relation constraint violations for extensible object sets. Starting with version 8.6, the OSS ASN.1 compiler and the OSS runtime support X.681 Annex E.2. That is, a constraint violation error is reported if table/component relation constraints are not satisfied for an extensible object set and any of the following conditions is true:

1. The value of a UNIQUE field is found in the object set.
2. The `STRICT_CONSTRAINT_CHECKING` runtime flag is specified.
3. The `OSS_AUTO_ENCODE_WITHOUT_CHECKING_CONSTRAINT` encoder flag is not specified.

The `-compat v8.5TableConstraintForExtensibleObjectSets` option disables the new behavior providing compatibility with versions prior to 8.6.

3.5.153 -compat v8.6namesForSetOfAndSeqOfWithNamedElements

Prior to version 9.0, the ASN.1 compiler generated a fabricated name for the C-structure representing an element of a SET OF / SEQUENCE OF type when the latter type had no ASN.1 name. Starting with 9.0 the compiler uses the name of the element type for the structure (if available).

The `-compat v8.6namesForSetOfAndSeqOfWithNamedElements` option can be used to disable the above-mentioned changes, providing compatibility with version numbers lower than 9.0.

Example:

ASN.1:

```
M DEFINITIONS ::= BEGIN
   Fox ::= SEQUENCE {
     foo SEQUENCE OF bar INTEGER
   }
END
```

Representation with `-compat v8.6namesForSetOfAndSeqOfWithNamedElements` specified:

```
typedef struct Fox {
    struct _seqof1 {
```

```
        struct _seqof1  *next;
        int             value;
    } *foo;
} Fox;
```

Representation without the `-compat v8.6namesForSetOfAndSeqOfWithNamedElements` compat flag specified:

```
typedef struct Fox {
    struct bar {
        struct bar      *next;
        int             value;
    } *foo;
} Fox;
```

3.5.154 -compat 8.6UTF8StringRepresentation

Prior to version 9.0, the OSS ASN1/C Compiler generated (when `-lean` was specified) different UTF8String C-representations when wide characters were present in the type values in the input syntax. Starting with version 9.0, the compiler always generates C-representations independently of the values.

The `-compat 8.6UTF8StringRepresentation` option restores the old incorrect behavior.

Example:

ASN.1:

```
MOD DEFINITIONS ::= BEGIN
    U1b ::= UTF8String
    u1 U1b ::= {{0,0,3,4}}
    U4b ::= UTF8String
    u4 U4b ::= {{0,1,3,4}}
END
```

Representation with `-compat 8.6UTF8StringRepresentation` specified:

```
typedef ossCharString U1b;
typedef ossUniversalString U4b;
```

Representation without `-compat 8.6UTF8StringRepresentation` specified:

```
typedef ossCharString U1b;
typedef ossCharString U4b;
```

3.5.155 -compat v8.7.0

Provides compatibility with version 8.7.0 of the ASN.1 compiler.

3.5.156 -compat v8.7BitStringsWithMAXUpperBound

Prior to version 9.0, the ASN.1 compiler generated different C representations for BIT STRING types having the SIZE constraint with a MAX upper bound and the PADDED or the VARYING directive applied on 64-bit and 32-bit platforms. Starting with version 9.0, the 64-bit version of the ASN.1 compiler generates the same C representations for those types as on 32-bit platforms.

The `-compat v8.7BitStringsWithMAXUpperBound` option can be used to restore the incorrect old behavior.

Example:

ASN.1:

```
M DEFINITIONS ::= BEGIN
  BS1 ::= BIT STRING (SIZE(1..MAX)) --<PADDED>--
  BS2 ::= BIT STRING (SIZE(1..MAX)) --<VARYING>--
END
```

Representations with `-compat v8.7BitStringsWithMAXUpperBound` specified on 64-bit platform:

```
typedef unsigned char   BS1[268435456];

typedef struct BS2 {
    unsigned int    length;  /* number of significant bits */
    unsigned char   value[268435456];
} *BS2;
```

Representations without `-compat v8.7BitStringsWithMAXUpperBound` specified:

```
typedef struct BS1 {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} BS1;

typedef struct BS2 {
    unsigned short  length;  /* number of significant bits */
    unsigned char   value[1];  /* first element of the array */
} *BS2;
```

3.5.157 -compat v8.7reservedWords

Prior to version 9.0, the ASN.1 compiler generated the "SID" and "small" names, if present in the ASN.1 specification, without mangling. This led to name conflicts with identifiers in system headers on MS Windows platform. Starting with 9.0 the compiler mangles these names in generated code.

The `-compat v8.7reservedWords` option can be used to disable the above-mentioned changes, providing compatibility with version numbers lower than 9.0.

Example:

ASN.1:

```
M DEFINITIONS ::= BEGIN
SID ::= SEQUENCE {
    small INTEGER
}
END
```

Representation with `-compat v8.7reservedWords` specified:

```
typedef struct SID {
    int             small;
} SID;
```

Representation without the `-compat v8.7reservedWords` compat flag specified:

```
typedef struct M_SID {
    int             SID_small;
} M_SID;
```

3.5.158 -compat v9.0.0extractionForDeeplyNestedTypes

Prior to version 9.0.2, the ASN.1 compiler could generate code code for deeply nested ASN.1 CHOICE types that was uncompilable by the Microsoft Visual C/C++ compiler. This occurred because the nesting level of structures and unions exceeded the MS VC implementation limit of 15. Starting with version 9.0.2, the compiler correctly limits the nesting level of generated inline structures and unions.

The `-compat v9.0.0extractionForDeeplyNestedTypes` option can be used to disable the above-mentioned changes, providing compatiblity with version 9.0.0.

Example:

ASN.1:

```
M DEFINITIONS ::= BEGIN
C ::= CHOICE {
    f1 CHOICE {
        f2 CHOICE {
            f3 CHOICE {
                f4 CHOICE {
                    f5 CHOICE {
                        f6 CHOICE {
                            f7 SET {
                                f8 SET {
                                    f9 SET {
                    i INTEGER
                                    }
                                }
                            }
                        }
                    }
                }
```

```
            }
        }
    }
}
END
```

Representation with `-compat v9.0.0extractionForDeeplyNestedTypes` specified:

```
typedef struct C {
    unsigned short  choice;
#       define      f1_chosen 1
    union {
        struct _choice6 {
            unsigned short  choice;
#               define      f2_chosen 1
            union {
                struct _choice5 {
                    unsigned short  choice;
#                       define      f3_chosen 1
                    union {
                        struct _choice4 {
                            unsigned short  choice;
#                               define      f4_chosen 1
                            union {
                                struct _choice3 {
                                    unsigned short  choice;
#                                       define      f5_chosen 1
                                    union {
                                        struct _choice2 {
                                            unsigned short  choice;
#                                               define      f6_chosen 1
                                            union {
                                                struct _choice1 {
                                                    unsigned short
                                                              choice;
#                                                   define
                                                        f7_chosen 1
                                                union {
                                                    struct _set1 {
                                                        struct {
                                                            struct {
                                                                int
                                                                   i;
                                                            } f9;
                                                        } f8;
                                                    } f7;  /* to
                                             * choose, set choice to f7_chosen */
                                                } u;
                                            } f6;  /* to choose, set
                                             * choice to f6_chosen */
                                        } u;
                                    } f5;  /* to choose, set choice to
                                             * f5_chosen */
                                } u;
                            } f4;  /* to choose, set choice to
                                             * f4_chosen */
                        } u;
                    } f3;  /* to choose, set choice to f3_chosen */
```

```
                    } u;
                } f2;  /* to choose, set choice to f2_chosen */
            } u;
        } f1;  /* to choose, set choice to f1_chosen */
    } u;
} C;
```

Representation without `-compat v9.0.0extractionForDeeplyNestedTypes` specified:

```
typedef struct _choice2 {
    unsigned short  choice;
#       define      f6_chosen 1
    union {
        struct _choice1 {
            unsigned short  choice;
#               define      f7_chosen 1
            union {
                struct _set1 {
                    struct {
                        struct {
                            int             i;
                        } f9;
                    } f8;
                } f7;  /* to choose, set choice to f7_chosen */
            } u;
        } f6;  /* to choose, set choice to f6_chosen */
    } u;
} _choice2;

typedef struct C {
    unsigned short  choice;
#       define      f1_chosen 1
    union {
        struct _choice6 {
            unsigned short  choice;
#               define      f2_chosen 1
            union {
                struct _choice5 {
                    unsigned short  choice;
#                       define      f3_chosen 1
                    union {
                        struct _choice4 {
                            unsigned short  choice;
#                               define      f4_chosen 1
                            union {
                                struct _choice3 {
                                    unsigned short  choice;
#                                       define      f5_chosen 1
                                    union {
                                        _choice2        f5;  /* to choose,
                                                * set choice to f5_chosen */
                                    } u;
                                } f4;  /* to choose, set choice to
                                        * f4_chosen */
                            } u;
                        } f3;  /* to choose, set choice to f3_chosen */
                    } u;
                } f2;  /* to choose, set choice to f2_chosen */
```

```
            } u;
        } f1;  /* to choose, set choice to f1_chosen */
    } u;
} C;
```

3.5.159 -compat v9.0reservedWords

Prior to version 10.0, the ASN.1 compiler generated the names "floor" and "send", if present in the ASN.1 specification, without mangling. This led to name conflicts with identifiers in system headers on Microsoft Windows platforms. Starting with 10.0, the compiler mangles these names in generated code.

The `-compat v9.0reservedWords` option can be used to disable the above-mentioned changes, providing compatibility with version numbers lower than 10.0.

Example:

ASN.1:

```
M DEFINITIONS ::= BEGIN
    TEN ::= ENUMERATED {send(1), floor(2), small(3)}
END
```

Representation with `-compat v9.0reservedWords` specified:

```
typedef enum TEN {
    send = 1,
    floor = 2
} TEN;
```

Representation without the compat flag specified:

```
typedef enum TEN {
    TEN_send = 1,
    TEN_floor = 2
} TEN;
```

3.5.160 -compat v10.0valueTruncation

Prior to version 10.1, the ASN.1 compiler generated the value of a SEQUENCE OF/SET OF type with an extensible size constraint without truncation. Starting with 10.1, the compiler truncates such values to satisfy the size constraints applied to the type, unless the `-allow BadValues` command line option is in effect.

The `-compat v10.0valueTruncation` option can be used to disable the above-mentioned changes, providing compatibility with version numbers lower than 10.1.

Example:

ASN.1:

```
M DEFINITIONS AUTOMATIC TAGS ::= BEGIN
    A ::= SEQUENCE (SIZE (1..2, ...)) OF INTEGER
    val A ::= {1, 2, 3}
END
```

Generation with `-compat v10.0valueTruncation` specified:

```
static struct A _v0[] = {
    {&_v0[1], 1},
    {&_v0[2], 2},
    {NULL, 3}
};
A val = _v0;
```

Generation without the compat flag specified:

```
static struct A _v0[] = {
    {&_v0[1], 1},
    {NULL, 2}
};
A val = _v0;
```

3.5.161 -compat v10.0reservedWords

Prior to version 10.1, the ASN.1 compiler generated the name "interface", if present in the ASN.1 specification, without mangling. This led to name conflicts with C runtime functions on Microsoft Windows platforms. Starting with 10.1, the compiler mangles this name in the generated code.

The `-compat v10.0reservedWords` option can be used to disable this change, providing compatibility with version numbers lower than 10.1.

Example:

ASN.1:

```
M DEFINITIONS ::= BEGIN
    S ::= SEQUENCE {
        interface  INTEGER,
        connection INTEGER
    }
END
```

Representation with `-compat v10.0reservedWords` specified:

```
typedef struct S {
    int             interface;
    int             connection;
} S;
```

Representation without the compat flag specified:

```
typedef struct S {
    int             S_interface;
    int             connection;
```

```
        } S;
```

3.5.162 -compat v10.1.0

The `-compat v10.1.0` flag provides compatibility with version 10.1.0 of the ASN.1 compiler.

4 Compiler directives

4.1 Introduction

The OSS ASN.1 compiler directives give you control over how data is encoded (see OSS.DEFINITE / OSS.INDEFINITE, section 4.4.2.9), decoded (see OSS.EXTENSIBLE / OSS.INEXTENSIBLE, section 4.4.2.16), or represented locally. The compiler does not require the use of directives. If no directives are used in an ASN.1 module, the compiler utilizes a default set of assumptions to generate the header and control/code files.

For example, the default representation of a 10 character ASN.1 `VisibleString` is (in C) an 11 character null-terminated string; however, the OSS.PADDED directive could be used to alter the representation to a 10 character blank-padded string. As another example, the default representation of a size-constrained `SEQUENCE OF` type is a linked-list of values; however, the OSS.ARRAY directive can be used to alter the representation to that of a length field immediately followed by an array. To learn more about the C representations used for ASN.1 notation by the OSS ASN.1/C compiler, refer to chapter 7.

4.2 Specifying directives

The directive specification (DirectiveSpec) is designed to appear as a comment to other compilers that do not support it. The start of a DirectiveSpec is denoted by two consecutive hyphens followed by a less than symbol (`--<`) and the end is denoted by a greater than symbol followed by two consecutive hyphens (`>--`). There should not be any spaces between the hyphens and the less than or greater than symbol. A DirectiveSpec consists of one or more directives with the general format:

```
--<[prefix.]Directive  [Operand(s)]>--
```

Multiple DirectiveSpecs may be specified. A DirectiveSpec can be continued on the next line by leaving out the end-of-DirectiveSpec marker (`>--`) and by inserting two consecutive hyphens (`--`) at the beginning of the next line (the first two non-space characters). All directives *are* case-sensitive and must be specified as shown in the *Directives reference* (section 4.4)

Examples of valid equivalent directive specifications:

```
--<OSS.NULLTERM>-- --<OSS.ROOT Module1>--
```

or

```
--<OSS.NULLTERM>--
--<OSS.ROOT Module1>--
```

or

```
--<OSS.NULLTERM>--
--<OSS.ROOT
--Module1>--
```

4.2.1 Handling contradictory directives

It is possible that two mutually exclusive directives (e.g., `OSS.DEFINITE` and `OSS.INDEFINITE)` be specified by a user. If the OSS ASN.1 compiler comes across such contradictory directives in a DirectiveSpec, the last directive specified is used and the previous directives are ignored.

4.2.2 Handling directives that take operands

If a directive accepts one or more operands, they must be listed after the directive's name. Multiple operands should be separated by commas (as shown below):

```
--<OSS.ROOT Module1, Module2, Module3>--
```

4.3 Types of directives accepted

It is valuable to note that the OSS ASN.1 compiler recognizes three types of directives (1) standard directives, (2) OSS-specific new-style directives, and (3) OSS-specific old-style directives. The definitions of these types of directives follow:

4.3.1 Standard directives

All standard directives are of the form `--<ASN1.Directive ..... >--`, where `Directive` is replaced with one of the standardized directives defined in section 4.4.1. A standard directive can be placed anywhere in the ASN.1 syntax so long as it appears before the item that it references. Note that standard directives usually have an `ASN1` prefix. The following is a typical example of how a standard directive looks like:

```
--<ASN1.WorkingSet Module1, Module2, Module3>--
```

For further information on standard directives, refer to section 4.4.1.

4.3.2 OSS-specific new-style directives

OSS-specific new-style directives are directives that are unique to the OSS ASN.1 Tools, but which follow the syntax prescribed by the **X/Open ASN.1/C++ API**. All such directives are of the form:

```
--<[OSS].directiveName ..... >--
```

where *`directiveName`* is replaced with one of the directives that are unique to the OSS ASN.1 Tools (see section 4.4.2). Note that it is strongly recommended that the `OSS` prefix be present when the directive is being specified globally; however, its presence is not mandatory. Similarly, when an OSS-specific directive is used at the module-level, it need not have the `OSS` prefix. Finally at the local-level, no directives should be ever prefixed with `OSS`.

4.3.3 OSS-specific old-style directives

The OSS-specific old-style directives are have the same functions as OSS-specific new-style ones except that they never take on the `OSS` prefix. In addition, unlike new-style directives, most of the old-style directives have no capability for specifying globally that only particular instances of a type should be affected.

These types of directives can have local, module, and global scope depending on their placement.

When specified locally, multiple old-style directives can be specified within a single DirectiveSpec separated by vertical bars (i.e. '|'). For example the following directive specifications are equivalent:

```
A ::= [1] REAL  --<DOUBLE>--  --<POINTER>--
```

and:

```
A ::= [1] REAL  --<DOUBLE | POINTER>--
```

Tip: If you want to replace all the OSS old-style directives with the new-style ones and write them into a separate file, you should first run the ASN.1 compiler on your abstract syntax with the `-genDirectives` option (see section 3.3.28). You may then remove all the directives from your ASN.1 syntax and invoke the ASN.1 compiler with the generated `.gen` file as the first parameter on the command line.

4.3.4 Precedence rules for the different types of directives accepted

New-style OSS-specific directives take precedence over old-style ones. In the following example the old-style directive `--<OBJECTID 13>--` will be ignored.

```
--<OSS.OBJECTID Sample.Name 27>--

Sample DEFINITIONS EXPLICIT TAGS ::= BEGIN
      Name ::= [1] OBJECT IDENTIFIER --<OBJECTID 13>--
END
```

Standard directives take precedence over old-style OSS-specific directives. For example, the ASN.1 syntax:

```
--<ASN1.Nickname Module.Int ASN1_Nicknamed_Int>--

Module DEFINITIONS ::= BEGIN
    Int ::= INTEGER --<TYPENAME "OSS_Specific_New_Name">--
END
```

results in the TYPENAME renaming mechanism to be ignored and the following is generated in the header file:

```
typedef int  ASN1_Nicknamed_Int;
```

Local OSS-specific new-style directives take precedence over global OSS-specific new-style directives:

```
--<OSS.FLOAT>--

Module DEFINITIONS ::= BEGIN
   A ::= [1] REAL --<DOUBLE>--
   B ::= [2] REAL
   C ::= [3] REAL
END
```

In the above notation, `A` will be of type `double` while `B` and `C` will be of type `float`.

4.3.5 Treatment of other implementation-specific directives

Other implementation-specific directives (like those starting with : `--<DSET.`, `--<HP.`, `--<TCSI.`, `--<IBM.`, `--<CMIS.`, `--<GDMO.`, `--<XMP.`, `--<XOM.`, etc.) have warning messages printed for them but are otherwise ignored.

4.4 Directives reference

This section contains an alphabetical list of the directives that the OSS ASN.1 compiler recognizes. Each directive entry contains the following information:

- the directive's purpose,
- defaults associated with the directive,
- details about the directive's specification and its operands,
- at least one example of the directive's use.

Quick look-up table for compiler directives

Directive name	Brief description	Section
ASN1.ConstraintFunction	assigns a name for the function used to check a user-defined constraint	4.4.1.1
ASN1.DeferDecoding	defers the decoding of a particular component within a SEQUENCE, SET, or CHOICE structure	4.4.1.2
ASN1.HugeInteger	selects a special representation for integers of a size not supported by the operating system in use	4.4.1.3
ASN1.Nickname	gives an alias to a name from an ASN.1 module for use in the application program	4.4.1.4
ASN1.PDU	specifies that one or more ASN.1 types should be treated as protocol data units	4.4.1.5
ASN1.RealRepresentation	provides a means to select the representation (binary, decimal, or mixed) to be used for ASN.1 values of type REAL	4.4.1.6
ASN1.Remove	identifies those types or components that should be ignored from input ASN.1 modules when generating output files	4.4.1.7
ASN1.Version	specifies the year of the ASN.1 standard in use in a particular ASN.1 module	4.4.1.8
ASN1.WorkingSet	identifies those parts of the ASN.1 input that are central to the application	4.4.1.9
OSS.ARRAY	specifies that a SET OF, SEQUENCE OF, or OBJECT IDENTIFIER type should be represented as an array of values along with a count-field	4.4.2.1
OSS.ASN1VER	specifies which version of the ASN.1 standard is in use in the ASN.1 input	4.4.2.2
OSS.BMPSTRING	causes the ASN.1 UTF8String type to be represented as an unbounded array of two-byte characters (Unicode)	4.4.2.3
OSS.COMMENT	instructs the ASN.1 compiler to place a user-defined comment near the beginning of the `.c` and `.h` files	4.4.2.4
OSS.CommentPreservation	gives you more control over the transfer of ASN.1 comments to the generated header file	4.4.2.5
OSS.DataCallback	associates a field of a SEQUENCE or SET type, an alternative of a CHOICE type, or an element of a SEQUENCE OF or SET OF type with your callback function for partial decoding	4.4.2.6

OSS.DECIMAL	causes the ASN.1 REAL type to be represented with a null-terminated character string	4.4.2.6
OSS.DECODEONLY	instructs the compiler to only generate the time-optimized decoder and not the encoder	4.4.2.7
OSS.DefineName	allows users to give names to generated `#defined` components within CHOICE, SEQUENCE, or SET structures	4.4.2.8
OSS.DEFINITE	specifies that the definite length form should be used when encoding protocol data units (PDUs)	4.4.2.9
OSS.DLINKED	specifies that SET OF, SEQUENCE OF, OCTET STRING, BIT STRING, or restricted character string types should be represented as a doubly-linked list of values	4.4.2.10
OSS.DLINKED-PLUS	specifies that SET OF or SEQUENCE OF should be represented as a doubly-linked-plus list of nodes	4.4.2.11
OSS.DTD	gives you greater control over the XML data type definitions which are produced by the ASN.1 compiler	4.4.2.12
OSS.DOUBLE	causes the ASN.1 REAL type to be represented with a C `double`	4.4.2.19
OSS.ENCODABLE	provides support to DER (Distinguished Encoding Rules) applications that encode encrypted signatures	4.4.2.13
OSS.ENCODED	causes the OBJECT IDENTIFIER type to be represented as a structure which contains a length in octets and the address of an array of characters containing BER-encoded contents	4.4.2.14
OSS.ENCODEONLY	instructs the compiler to only generate the time-optimized encoder and not the decoder	4.4.2.7
OSS.ExerNumericBoolean	Instructs the E-XER encoder to produce numeric Boolean values (0 and 1) instead of textual values (true and false or their replacements)	4.4.2.15
OSS.EXTENSIBLE	enables decoder support for the rules of extensibility	4.4.2.15
OSS.ExtensibleUseType	instructs the E-XER decoder to treat the content of extensible CHOICE types with the USE-TYPE XER encoding instruction as unknown extensions rather than values of the first CHOICE alternative	4.4.2.17
OSS.ExtractType	specifies that a nested structure should be coded using a `typedef` (defined outside of the structure)	4.4.2.18
OSS.FIELDNAME	instructs the ASN.1 compiler to generate specific names for fields in a structure	4.4.2.53
OSS.FLOAT	causes the ASN.1 REAL type to be represented with a C `float`	4.4.2.19
OSS.FUNCNAME	specifies a name for a function that checks a user-defined constraint	4.4.2.20
OSS.HelperListAPI	instructs the ASN.1 compiler to generate a set of list manipulation API functions for the specified SET OF / SEQUENCE OF type or for all such types when it is specified globally	4.4.2.22
OSS.HelperMacro	instructs the ASN.1 compiler to generate helper macros for the specified type or for all supported types when it is used globally	4.4.2.23
OSS.HeaderName	allows you to explicitly specify a name for the header file generated for a particular ASN.1 module in split-headers mode	4.4.2.21
OSS.HUGE	selects a special representation for integers of a size not supported by the operating system in use	4.4.2.24

OSS.INCLUDES	specifies the name of one or more configuration files to be inserted at the point at which this directive is specified	4.4.2.25
OSS.INDEFINITE	specifies that the indefinite length form should be used when encoding protocol data units (PDUs)	4.4.2.9
OSS.INEXTENSIBLE	disables decoder support for the rules of extensibility	4.4.2.15
OSS.InfoCallback	associates a field of a SEQUENCE or SET type, an alternative of a CHOICE type, or an element of a SEQUENCE OF or SET OF type with your callback function for partial decoding	4.4.2.16
OSS.InlineType	specifies that a nested structure should be coded inline within the containing structure	4.4.2.18
OSS.INT	causes the ASN.1 INTEGER type to be represented with a C `int`	4.4.2.41
OSS.LENGTHSIZE	changes the default representation for the `count` and `length` fields of structures generated by the ASN.1 compiler	4.4.2.26
OSS.LINKED	specifies that SET OF, SEQUENCE OF, OCTET STRING, BIT STRING, OBJECT IDENTIFIER, or restricted character string types should be represented as a singly-linked list of values	4.4.2.27
OSS.LONG	causes the ASN.1 INTEGER type to be represented with a C `long`	4.4.2.41
OSS.LONGLONG	causes the ASN.1 INTEGER type to be represented with a `LONG_LONG` on supported platforms	4.4.2.41
OSS.MIXED	allows the ASN.1 REAL type to be represented with either a null-terminated character string or a C `double`	4.4.2.6
OSS.NoConstrain	identifies those types or components for which constraint checking should be disabled	4.4.2.28
OSS.NOCOPY	a synonym for the OSS.OBJHANDLE directive	4.4.2.32
OSS.NODEFAULTVALUE	informs the ASN.1 compiler to treat DEFAULT components as if they were OPTIONAL components instead	4.4.2.29
OSS.NoHelperListAPI	instructs the ASN.1 compiler to not generate a set of list manipulation API functions for the specified SET OF / SEQUENCE OF type or for all such types when it is specified globally	4.4.2.22
OSS.NoHelperMacro	instructs the ASN.1 compiler to not generate helper macros for the specified type or for all supported types when it is used globally	4.4.2.23
OSS.NOPDU	instructs the encoder/decoder not to treat an ASN.1 type as a protocol data unit	4.4.2.35
OSS.NOPOINTER	causes memory to be allocated inline for an ASN.1 type	4.4.2.36
OSS.NOVALUE	specifies that for each ASN.1 value reference a corresponding C initialized variable should not be generated	4.4.2.54
OSS.NULLTERM	indicates that a ASN.1 character string type should be represented in the target language as a null-terminated variable length string	4.4.2.31
OSS.OBJECTID	similar to the OSS.ENCODED directive, but also allows the specification of the size of the `value` field	4.4.2.14
OSS.OBJHANDLE	specifies that certain types or components should receive special handling at run-time	4.4.2.32
OSS.ONECHAR	specifies that the representation of a character string should be of a single character	4.4.2.33

OSS.PADDED	indicates that the character string, BIT STRING, and OCTET STRING types should be represented as fixed-length padded strings	4.4.2.34
OSS.PDU	instructs the encoder/decoder to treat an ASN.1 type as a protocol data unit	4.4.2.35
OSS.POINTER	causes a pointer to the specified type to be generated without inline allocation	4.4.2.36
OSS.Preserve	instructs the ASN.1 compiler to generate an ASN.1 type into the header file which normally would be ignored	4.4.2.37
OSS.PrintFunctionName	allows users to specify their own data formatter for printing values of primitive types	4.4.2.38
OSS.ROOT	identifies the ASN.1 module(s) that are to be used as the root module(s) in which all unreferenced types are considered PDUs	4.4.2.39
OSS.SampleCode	instructs the ASN.1 compiler to generate sample code for the identified PDU types or valuereferences	4.4.2.40
OSS.SelfCompleteWildcard	instructs E-XER decoder to literally preserve wildcard contents in the decoded C value; allows checking the digital signature of a wildcard after decoding	4.4.2.41
OSS.SHORT	causes the ASN.1 INTEGER type to be represented with a C `short`	4.4.2.42
OSS.SHORTENNAMES	instructs the compiler to omit certain letters from long names to make them less than 31 characters in length in the header file	4.4.2.43
OSS.Stylesheet	gives you greater control over the default XML stylesheets which are produced by the ASN.1 compiler	4.4.2.44
OSS.SUPPRESS	instructs the compiler not to print specific informational and/or warning messages	4.4.2.45
OSS.TIMESTRUCT	indicates that an ASN.1 GeneralizedTime or UTCTime type should be represented in the target language as a structure of integers	4.4.2.46
OSS.Truncate	speeds up decoding by skipping extra trailing elements in large SET OF or SEQUENCE OF types	4.4.2.47
OSS.TypeIndex	an internal directive that is generated into the `.spl` file when the `-splitHeaders` or `-splitForSharing` option is in use	4.4.2.48
OSS.TYPENAME	instructs the ASN.1 compiler to generate specific names for user-defined types	4.4.2.53
OSS.UNBOUNDED	specifies that the type is to be represented as a length/pointer pair or as a count/pointer pair	4.4.2.49
OSS.UNIVERSALSTRING	selects the representation of the ASN.1 UTF8String type in C as an unbounded array of four-byte characters (UCS-4)	4.4.2.50
OSS.USERFIELD	allows the generation of a locally defined user field that is of significance only to a particular application	4.4.2.51
OSS.UseThis	permits the reduction of the generated header file's size by eliminating semi-duplicate C `typedef`s	4.4.2.52
OSS.VALNAME	instructs the ASN.1 compiler to generate specific names for the `value` variable in compiler-generated structures	4.4.2.53
OSS.VALUE	specifies that for each ASN.1 value reference a corresponding C initialized variable should be generated	4.4.2.54
OSS.VARYING	specifies that a type should be represented as length-prefixed variable-length strings	4.4.2.55

OSS.XEREncodeFunction	allows you to specify your own data formatter to encode ASN.1 types using XML Encoding Rules (XER Basic or Canonical)	4.4.2.56

4.4.1 Standard Directives

The format for a standard directive is:

```
--<ASN1.nameOfDirective [absoluteReference] [directiveOperand(s)]>--
```

where the *nameOfDirective* is one of the following:

1) ConstraintFunction,
2) DeferDecoding,
3) HugeInteger,
4) Nickname,
5) PDU,
6) RealRepresentation,
7) Remove,
8) Version,
9) WorkingSet

The *absoluteReference* is a fully specified name of an ASN.1 item (see **clause 15 Rec. X.680 ITU-T (1997E)**). Note, however, that the @ symbol in the formal definition of absolute references should be omitted from OSS DirectiveSpecs . The purpose of the absolute reference is to specify which item or items from the ASN.1 syntax the directive should affect. For more information about absolute references refer to the definition box in this section.

As for *directiveOperand(s)*, it consists of one or more tokens that can be used to configure the named directive as outlined on the following pages. Note that the presence of *directiveOperand(s)* is mandatory for some directives and optional for others.

Note that standard directives may also be written as OSS new-style directives with the OSS prefix instead of ASN1. For example:

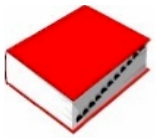
```
--<ASN1.Version 2008 Module>--
```

is equivalent to:

```
--<OSS.Version 2008 Module>--
```

Definition: ***Absolute Reference*** - a single or multiple part dot-notation used for uniquely specifying one or more components in an ASN.1 syntax. This notation is similar to the syntax used in many programming languages to access components within named structures and records. The outermost structure is listed first using its identifier followed by a dot '.'. Then, a component of the outermost structure (which may itself be a structure) can be listed next. If the desired component rests within a nested structure, a dot is added after the name of the containing structure and then the desired component is listed.

Four methods are provided for specifying particular ASN.1 components: (1) use of the component's identifier, (2) use of an asterisk '*' for specifying all rows in the SEQUENCE OF and SET OF arrays, (3) use of an integer index for specifying which element is intended in a SEQUENCE, SET, or CHOICE, and (4) use of a dollar sign '$' followed by a number index to reference a particular CONSTRAINED BY clause (see next page for more information about this last type of absolute reference).



Absolute Reference

Example:

Given the ASN.1 definition:

```
MyMod DEFINITIONS ::= BEGIN
    Comp1 ::= SET OF SEQUENCE {
            a       INTEGER,
            b       OCTET STRING OPTIONAL,
            c       CHOICE {
                        nest1   BOOLEAN,
                        nest2   BIT STRING
                    }
    }
    Comp2 ::= IA5String
END
```

The absolute reference for the entire module is:	`MyMod`
The absolute reference for the SET OF is :	`MyMod.Comp1`
The absolute reference of the SEQUENCE series is:	`MyMod.Comp1.*`
The absolute reference for the INTEGER is:	`MyMod.Comp1.*.a` or `MyMod.Comp1.*.1`
The absolute reference for the OCTET STRING is:	`MyMod.Comp1.*.b` or `MyMod.Comp1.*.2`
The absolute reference for the CHOICE is:	`MyMod.Comp1.*.c` or `MyMod.Comp1.*.3`
The absolute reference for the BOOLEAN is	`MyMod.Comp1.*.c.nest1` or `MyMod.Comp1.*.c.1`
The absolute reference for the BIT STRING is:	`MyMod.Comp1.*.c.nest2` or `MyMod.Comp1.*.c.2`
Finally, the absolute reference for the IA5String is:	`MyMod.Comp2`

(**Important:** Also see the next sections for more about absolute references.)

Using absolute references for CONSTRAINED BY clauses

A special type of absolute reference notation (added in v5.0) is used to access ASN.1 types located within CONSTRAINED BY clauses. This special notation consists of a dollar sign '$' followed by a number index indicating the particular CONSTRAINED BY intended. Further, this number index can be optionally followed by a colon ':' and then another number index (or a component identifier) indicating the particular component within the CONSTRAINED BY braces that is intended. If this latter colon and following index (or identifier) is left out, then the first component within the CONSTRAINED BY is targeted by default. The rest of this section contains some demonstrative examples to help you understand what has been mentioned in the previous sentences:

Example 1: Simple CONSTRAINED BY clauses

ASN.1:

```
--<OSS.FIELDNAME Mod.A.$2.c "my_c">--

Mod DEFINITIONS ::= BEGIN
   A ::= BOOLEAN (CONSTRAINED BY {--Just a comment--})
                 (CONSTRAINED BY {
                    SET {c NULL, d REAL}})
END
```

In the absolute reference above, `$2.c` refers to field `c` in the SET type in the 2nd CONSTRAINED BY clause. Notice how we did not have to write `$2:1.c`, since `:1` is already the implied by default.

Example 2 - CONSTRAINED BY types in instances of parameterized types

ASN.1:

```
--<OSS.TYPENAME Mod.B.$1 "my_bitstr">--
--<OSS.FIELDNAME Mod.B.$3.a "my_a">--

Mod DEFINITIONS ::= BEGIN
   B ::= Param {INTEGER} (CONSTRAINED BY {BIT STRING})

   Param {T} ::= T (CONSTRAINED BY {--Just a comment--})
                   (CONSTRAINED BY {
                       SEQUENCE {a BOOLEAN}})
END
```

In the first absolute reference above, `$1` refers to the BIT STRING type in the 1st CONSTRAINED BY clause applied to `B`.

In the second absolute reference above, `$3.a` refers to field `a` in the SEQUENCE type in the 3rd CONSTRAINED BY clause applied to the parameterized type `B`. Notice how we have to count all of the CONSTRAINED BY clauses that are applied to the type `B` and not just the ones in the parameterized definition `Param`. Thus, determining the proper index to use is different for non-parameterized types with CONSTRAINED BY clauses and different for parameterized types with such.

Example 3 - Nested CONSTRAINED BY clauses

ASN.1:

```
--<ASN1.Nickname Mod.C.$2:3.f.$1.e "my_e">--

Mod DEFINITIONS ::= BEGIN
  C ::= BOOLEAN (CONSTRAINED BY {})
                  (CONSTRAINED BY {
                      INTEGER, -- "1st" parameter
                      IA5String,
                      CHOICE {
                         f REAL (CONSTRAINED BY
                                 {SET {e NULL}})
                                } -- "3rd" parameter
                  })
END
```

In the absolute reference above, `$2:3` refers to the 3rd parameter (i.e. the CHOICE type) in the 2nd CONSTRAINED BY clause applied to the type `C`; furthermore, `$1:e` refers to the component `e` in the SET type which is the 1st (and the only) parameter in the 1st CONSTRAINED BY applied to the component `f` of the CHOICE type.

Directives That Have No Effect on Generated Files

Some standard directives and their operands have no effect on the compiler-generated output files. If a directive "has no effect", it can still be specified in the ASN.1 input file, but a warning message will be issued.

Each time the OSS ASN.1 compiler parses a directive which has no effect, the Compiler prints one of two informatory messages:

(a) if the `-genDirectives` command line option was not specified:

```
C0492I: There are unused standard directives.  Specify the
       -genDirectives command line option, then look at the
       generated .gen file.
```

(b) if `-genDirectives` was specified:

```
C0064I:  There are unused standard or OSS-specific directives.
         See the .gen file.
```

In the second case, the Compiler prints out a commented informatory warning in the `.gen` file:

```
--<ASN1.ConstraintFunction Module>-- -- WARNING: could not use the
-- directive
```

The following subsections give detailed descriptions of the standard directives accepted by the OSS ASN.1 compiler. Note again that these directives are listed in alphabetical order.

4.4.1.1 ASN1.ConstraintFunction

The ASN1.ConstraintFunction directive is used to specify the name of a function in your application which is used to check a user-defined constraint. This directive can only be used with types that have a CONSTRAINED BY clause. **By default,** the constraining function's name is *typereference*_fn.

This directive has the following format:

```
--<ASN1.ConstraintFunction absoluteReference ["]functionName["]>--
```

The absolute reference can refer to a component within a module that has a user-defined constraint attached to it. The *functionName* argument is the name that of the constraint checking function that the user wishes to use. This function name can be a quoted or non-quoted string. If the *functionName* argument is omitted, the ASN.1 compiler will issue an error message.

Example:

```
--<ASN1.ConstraintFunction  Module.OddInteger isConstrOdd>--

Module DEFINITIONS ::= BEGIN
    OddInteger ::= INTEGER (CONSTRAINED BY {-- odd --} )
END
```

In the above example, the application function `isContrOdd()` will be used to check if the type `OddInteger` is indeed odd. If no constraining function name were provided, by default the constraining function's name would be `OddInteger_fn.`

If both the ASN1.ConstraintFunction and OSS.FUNCNAME directive are specified for the same type, the OSS-specific directive takes precedence, as shown in the following example:

```
--<ASN1.ConstraintFunction Module isConstrOdd>--

Module DEFINITIONS ::= BEGIN
    OddInteger ::= INTEGER
    (CONSTRAINED BY {--<FUNCNAME "isLocalOdd">--} )
END
```

The function with the name `isLocalOdd()` will be used to check constraints in the above example.

Notes:
1) The ASN1.ConstraintFunction directive must appear before the user-defined type it refers to.
2) Starting with version 5.0, user-defined constraint-checking functions are only generated if the `-userConstraints` compiler option is specified.
3) User-defined constraints are not supported by the TOED.

Related directive:

OSS.FUNCNAME (section 4.4.2.20)

4.4.1.2 ASN1.DeferDecoding

The ASN1.DeferDecoding directive may be used to indicate that a particular component within a CHOICE, SEQUENCE, or SET structure should not be automatically decoded when its containing type is marked for decoding. **By default**, all components of a CHOICE, SET, or SEQUENCE are decoded with their containing type.

This directive has the following format:

```
--<ASN1.DeferDecoding absoluteReference>--
```

The type referred to by the absolute reference will be replaced by an open type which gets encoded and decoded separately by the user application.

Example:

```
--<ASN1.DeferDecoding Module.Type.setComponent>--

Module DEFINITIONS ::= BEGIN
    Type ::= SET {
        setComponent INTEGER,
        b BOOLEAN
    }
END
```

The following types are produced in the `.h` file for the above example:

```
#define          Type_PDU 1
#define          Type_setComponent_encodable_PDU 2

typedef struct Type {
    OpenType        setComponent;   /* Type_setComponent_encodable type */
    ossBoolean      b;
} Type;

typedef int             Type_setComponent_encodable;
```

The defined type, `Type_setComponent_encodable`, can be later used in decoding `setComponent`. Notice that this type has a separate PDU tag generated for it.

Related directives:

OSS.ENCODABLE (4.4.2.13), OSS.OBJHANDLE / OSS.NOCOPY (section 4.4.2.32)

Also refer to the following note box:

Note:
1) The ASN1.DeferDecoding directive is applicable only if BER or DER is in use. While this directive affects the behavior of the encoder/decoder, it has no effect on the actual encoding that is produced.
2) You may not use the ASN1.DeferDecoding directive if you use PER, XER, or OER.
3) You may not apply the ASN1.DeferDecoding directive to an ANY type or open type component.
4) Note that the time-optimized encoder/decoder did not support the ASN1.DeferDecoding directive prior to version 5.2.0.

4.4.1.3 ASN1.HugeInteger

The ASN1.HugeInteger directive is used to select a special representation for integers of a size not supported by the operating system in use. **By default,** INTEGER types are represented as machine dependent 16-bit, 32-bit or 64-bit numbers.

This directive has the following format:

```
--<ASN1.HugeInteger absoluteReference>--
```

The absolute reference should refer to an ASN.1 INTEGER type that is to be represented using the HugeInteger representation. The HugeInteger representation consists of a structure containing a pointer to a character string and a length field specifying the size in octets of the stored integer:

```
struct {
    unsigned short  length;
    unsigned char   *value;
} integerType;
```

The integer is stored in binary format in the series of octets referenced by the `value` field above.

The example below illustrates how this directive can be used:

Example:

For the following ASN.1 syntax:

```
--<ASN1.HugeInteger LengthModule.LengthData.lengthOfSolarSystem>--

LengthModule DEFINITIONS ::= BEGIN
    LengthData ::= SEQUENCE {
        lengthOfSolarSystem  INTEGER,
        units  ENUMERATED
               {mm(0), cm(1), m(2), km(3), lightyears(4)} DEFAULT mm
    }
END
```

The OSS ASN.1 compiler generates the following code in the header file:

```
typedef struct LengthData {
    unsigned char   bit_mask;
#       define      units_present 0x80
    struct {
        unsigned short  length;
        unsigned char   *value;
    } lengthOfSolarSystem;
    enum _enum1 {
        mm = 0,
        cm = 1,
        m = 2,
        km = 3,
        lightyears = 4
    } units;/* units_present not set in bit_mask implies value is mm */
} LengthData;
```

Related directive:
This directive has the same effect as the OSS-specific directive OSS.HUGE (see section 4.4.2.24).

4.4.1.4 ASN1.Nickname

The Nickname directive can be used to give an alias to a name from an ASN.1 module for use in the application program. **By default,** the name used in the generated data structures is derived from the input ASN.1 syntax.

This directive has the following format:

```
--<ASN1.Nickname absoluteReference ["]nickname["]>--
```

absoluteReference specifies a name for one of the following: (1) a module, (2) type reference, (3) value reference, (4) macro reference, (5) object class reference, (6) object reference, (7) object set reference, (8) identifier or field name, (9) element of a SET or SEQUENCE, (10) anonymous component of a SET or SEQUENCE, or (11) an identifier in a NamedNumberList. The *nickname* parameter should follow the naming rules of the target programming language.

Example:

```
--<ASN1.Nickname MyModule.MySequence.MyTypeName MTN>--
```

The above notation specifies that a nickname of `MTN` should be used for the type, `MyTypeName`, which is an element of the SEQUENCE, `MySequence`, contained in the module, `MyModule`.

As noted above, the ASN1.Nickname directive can be assigned to:

(a) module names

In this case, the ASN.1 compiler substitutes the original ASN.1 module name with the given *nickname* in its output (i.e. in the generated `.h` and `.c` files).

(b) type reference, value reference, macro reference, object class reference, object reference, and object set reference names.

Note, however, that this directive does not currently support `macroreference` names. Also, note that if you would like to rename a type defined with macro TYPE NOTATION, you should place this directive after the module where such a type is defined.

The following ASN.1 definition:

```
--<ASN1.Nickname Module.UserData Monkey>--
--<ASN1.Nickname Module.UserData.* Gorilla>--
--<ASN1.Nickname Module.UserData.*.1 Gibbon>--
--<ASN1.Nickname Module.UserData.*.fvalue Mandrill>--

Module DEFINITIONS ::= BEGIN
    UserData ::= SET OF SEQUENCE {
        key     INTEGER,
        fvalue  OCTET STRING OPTIONAL
    }
END
```

is equivalent to:

```
  Module DEFINITIONS ::= BEGIN
       UserData ::= SET --<TYPENAME "Monkey">-- OF
            SEQUENCE {
                 key    INTEGER --<FIELDNAME "Gibbon">--,
                 fvalue  OCTET STRING --<FIELDNAME "Mandrill">--
                                                    OPTIONAL
            } --<TYPENAME "Gorilla">--
  END
```

and results in:

```
typedef struct Monkey {
    struct Monkey   *next;
    struct Gorilla {
        unsigned char   bit_mask;
#       define      Mandrill_present 0x80
        int             Gibbon;
        struct {
            unsigned int    length;
            unsigned char   *value;
        } Mandrill;  /* optional; set in bit_mask Mandrill_present
                        if present */
    } value;
} *Monkey;
```

Note that the "`*`" in the absolute reference for the nickname "`Gibbon`" refers to the SEQUENCE type while the following "`1`" refers to the first element in the SEQUENCE (i.e. `key`). (For more information on absolute references, refer to the definition box in section 4.4.1.

(c) identifier or field names

The following is a possible use of the Nickname directive in renaming ASN.1 identifiers:

```
--<ASN1.Nickname Module.PasswordChallengeRequestResponse.
-- challengeRequestResponse.challengeResponse pcrrResponse>--

Module DEFINITIONS ::= BEGIN
    PasswordChallengeRequestResponse ::= CHOICE {
        passwordInTheClear               INTEGER,
        challengeRequestResponse         SEQUENCE {
            challengeRequest   BOOLEAN  OPTIONAL,
            challengeResponse  INTEGER  --FIELDNAME "pcrrResponse">--
                                  OPTIONAL
        }
    }
END
```

and it is equivalent to the commented OSS-specific OSS.FIELDNAME directive. (see section 4.4.2.53 for more information on the OSS.FIELDNAME directive.)

(d) element names of a SET OF or SEQUENCE OF

You can specify:

```
--<ASN1.Nickname Module.UserData.* UserDataSeq>--

Module DEFINITIONS ::= BEGIN
    UserData ::= SET OF SEQUENCE {
        key    INTEGER,
        value  OCTET STRING OPTIONAL
    }
END
```

The '*' in the absolute reference of the Nickname directive refers to the name of the intermediate type created for the SEQUENCE. The ASN.1 compiler output would look like:

```
typedef struct UserData {
    struct UserData *next;
    struct UserDataSeq {
        unsigned char   bit_mask;
#       define      value_present 0x80
        int             key;
        struct {
            unsigned int    length;
            unsigned char   *value;
        } value;  /* optional ; set in bit_mask value_present if present */
    } value;
} *UserData;
```

(e) anonymous component of a SET or SEQUENCE

The following is a valid syntax:

```
--<ASN1.Nickname Module.ConnectInformation.*.1
--               ConnectInfoDirection>--

Module DEFINITIONS ::= BEGIN
    ConnectInformation ::= SEQUENCE OF SEQUENCE {
        CHOICE {
            unidirectional  [0]  BOOLEAN,
            bidirectional   [1]  BOOLEAN,
            addleg          [2]  INTEGER
        },
        administrativeState  INTEGER OPTIONAL
    }
END
```

In the absolute reference `Module.ConnectInformation.*.1`, `ConnectInformation` refers to the SEQUENCE OF, while '`*`' refers to the following SEQUENCE, and "`1`" refers to the first element in the SEQUENCE (i.e. the CHOICE construct). The number notation makes it possible to give a name to every component of a SEQUENCE, SET ,or CHOICE, for in ASN.1:1990 it is possible to omit identifiers for elements of these constructs. Note that identifiers are mandatory in ASN.1:2008).

In the above example, the first component of SEQUENCE (i.e. the CHOICE) gets renamed as follows:

```
typedef struct ConnectInformation {
    struct ConnectInformation *next;
    struct {
        unsigned char   bit_mask;
#           define      administrativeState_present 0x80
        struct {
            unsigned short  choice;
#               define      unidirectional_chosen 1
#               define      bidirectional_chosen 2
#               define      addleg_chosen 3
            union {
                ossBoolean      unidirectional; /* to choose, set choice to
                                                 * unidirectional_chosen */
                ossBoolean      bidirectional;  /* to choose, set choice to
                                                 * bidirectional_chosen */
                int             addleg;  /* to choose, set choice to
                                          * addleg_chosen */
            } u;
        } ConnectInfoDirection;
        int             administrativeState; /* optional; set in bit_mask
                                              * administrativeState_present
                                              * if present */
    } value;
} *ConnectInformation;
```

Note that using both the anonymous component syntax and the fieldname syntax for the same component for deeply nested types can have an adverse effect on the performance of the ASN.1 compiler.

(f) identifiers used in a NamedNumberList

If you specify:

```
--<ASN1.Nickname Module.E.enum1 "new_enum1">--
--<ASN1.Nickname Module.B.bit1 "new_bit1">--
--<ASN1.Nickname Module.I.int1 "new_int1">--

Module DEFINITIONS ::= BEGIN
        E ::= ENUMERATED {enum1(1), enum2(2)}
    B ::= BIT STRING {bit1(4), bit2(8)}
        I ::= INTEGER {int1(10), int2(8)}
END
```

the ASN.1 compiler will generate the followin in the header file:

```
typedef enum E {
    new_enum1 = 1,
    enum2 = 2
} E;

typedef unsigned short  B;
#define                     new_bit1 0x0800
#define                     bit2 0x0080

typedef int             I;
#define                     new_int1 10
#define                     int2 8
```

Related directives:

OSS.FIELDNAME, OSS.VALNAME, OSS.TYPENAME (see section 4.4.2.53)

4.4.1.5 ASN1.PDU

The ASN1.PDU directive specifies that one or more ASN.1 types should be treated as protocol data units. **By default**, all unreferenced types (defined types that are not referenced by any other type) are considered to be PDUs by the compiler, and as such can be encoded/decoded as complete units.

This directive has the following format:

```
--<ASN1.PDU type [, type ] ...>--
```

where the one or more `type`s following the directive are operands specifying which ASN.1 types should be treated as PDUs. Note that the desired types should be specified using the absolute reference notation.

Example:

```
--<ASN1.PDU Module.Type>--

Module DEFINITIONS ::= BEGIN
```

```
    TypeRef ::= Type
    Type ::= SET { b BOOLEAN }
END
```

In the above example, the SET, `Type`, will be treated as a PDU, even though it is referenced in another type definition.

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
#define          TypeRef_PDU 1
#define          Type_PDU 2

typedef struct Type {
    ossBoolean      b;
} Type;

typedef Type            TypeRef;
```

Notice the #`defined` PDU number for `Type`.

Related directive:

This directive has the same effect as the OSS-specific directive OSS.PDU (see section 4.4.2.35).

Note: The PDU directive can have three syntaxes:

1) OSS Old-Style Syntax - the old-style OSS-specific directive, `--<PDU>--` , cannot have any operands. Rather, its placement determines its scope (i.e. local or global);

2) OSS New-Style Syntax - the new-style OSS-specific directive, `--<OSS.PDU [type]>--`, must have no more than one operand referring to an ASN.1 type. If no operands are given, this directive has the same effect as the old-style global PDU directive. If an operand is present, this directive has the same effect as the old-style local PDU directive. This directive may be placed anywhere in the ASN.1 input when specified with an operand.

3) Standard Directive Syntax - the standard directive, `--<ASN1.PDU` *`operand(s)`*`>--`, must have one or more operands referring to ASN.1 type(s) which should be treated as PDU('s).

4.4.1.6 ASN1.RealRepresentation

The ASN1.RealRepresentation directive provides a means to select the representation (binary, decimal, or mixed) to be used for ASN.1 values of type REAL. This directive can be applied to particular REAL types, or can specify a default for all REAL types. **By default,** the REAL type is represented as C `double` (i.e. binary).

This directive has the following format:

```
--<ASN1.RealRepresentation realKind [type1 [, type2, ...]]>--
```

The *realKind* operand is one of the words: `binary`, `decimal`, or `mixed`, specifying that the REAL type should be represented in the generated `.c` and `.h` files as a `double`, a character string of the form: `"[-]nnnnn.nnnnn[E[-]nnnnn]"`, or a `struct` with both possible representations, respectively. After the *realKind* operand, a list of affected types can be specified using the absolute reference notation. If no type references are specified, the ASN1.RealRepresentation directive will set the default representation for REAL types locally, on the module-level, or globally, according to its placement.

Note: In order for the ASN1.RealRepresentation directive to have a global effect, it must be used without type references and placed before first module in the ASN.1 input.

Example:

The following ASN.1 definition:

```
--<ASN1.RealRepresentation mixed Module.Type.a1.b2>--

Module DEFINITIONS ::= BEGIN

    Type ::= SEQUENCE {
        a1 SET { b1 INTEGER, b2 REAL},
        a2 REAL
    }
END
```

is equivalent to:

```
Module DEFINITIONS ::= BEGIN

    Type ::= SEQUENCE {
        a1 SET { b1 INTEGER, b2 REAL --<MIXED>--},
        a2 REAL
    }
END
```

and results in:

```
typedef struct Type {
    struct {
        int             b1;
        MixedReal       b2;
    } a1;
    double          a2;
} Type;
```

Note that the OSS type `MixedReal` used above is defined as follows in `asn1hdr.h`:

```
enum MixedReal_kind {OSS_BINARY, OSS_DECIMAL};

typedef struct {
    enum MixedReal_kind kind;
    union {
        double base2;
```

```
        char  *base10;
    } u;
} MixedReal;
```

Related directives:
This directive has the same effect as the OSS-specific directives OSS.DECIMAL, OSS.DOUBLE, and OSS.MIXED (see sections 4.4.2.6 & 4.4.2.19).

4.4.1.7 ASN1.Remove

The ASN1.Remove directive identifies those types or components that should be ignored from input ASN.1 modules when generating files for application use. This is useful when the application designer plans to not use some part of an ASN.1 specification but does not want to (or cannot) modify the original ASN.1 source. **By default,** all parts of the ASN.1 input are included in the compiler-generated files.

The ASN1.Remove directive informs the ASN.1 compiler that an ASN.1 item should be treated as if it were simply not included in the ASN.1 source text (i.e. no associated code is generated either in the `.h` or `.c` file about this type for use by the application). Even though components are omitted from the input ASN.1, the encodings that result are compatible with corresponding applications that do not use the ASN1.Remove directive. This is due to the following practices:

- Automatic tags are assigned as if the removed components were present. In theory, automatic tags are assigned before removal.
- Some encoding rules, such as PER, are sensitive to the number of possible components. By counting any removed components when encoding, an encoded value will be the same whether or not components have been removed.

This directive has the following format:

```
--<ASN1.Remove absoluteReference [, absoluteReference ] ...>--
```

where *absoluteReference* can refer to any of the following:

1) any defined type;
2) any defined value;
3) any component of a CHOICE type (except it is not valid to remove all alternatives leaving an empty CHOICE type);
4) any component that is marked OPTIONAL in a SEQUENCE or SET type;
5) any component that has a DEFAULT value in a SEQUENCE or SET type;
6) any information object class;
7) any information object that is not included in an information object set;
8) any information object set.

Example:

```
--<ASN1.Remove Mod.Int>--
--<ASN1.Remove Mod.S.a>--
--<ASN1.Remove Mod.i>--
```

```
Mod DEFINITIONS ::= BEGIN
    S ::= SET {
        a SET OF SET OF SEQUENCE {
            b Int
        }OPTIONAL
    }
    Int ::= INTEGER
    i Int ::= 10
END
```

After compiling the above syntax, only an empty S structure with an arbitrary placeholder is generated in the header file as shown below:

```
#define          S_PDU 1

typedef struct S {
    char            placeholder;
} S;
```

Notes:
1) When an ASN.1 item is marked for removal, any references to the item must also be removed through the use of the ASN1.Remove directive. If errors related to missing references occur and the `-genDirectives` or the `-keepNames` command line option is specified, the OSS ASN.1 compiler automatically generates the needed ASN1.Remove directives in a file with a `.rmv` extension. (This file will have the same prefix as the `.gen` file.)
2) The generated `.rmv` file can be passed on the command-line (before any compiler options) to let the compiler automatically remove the types that are causing trouble.
3) Note that in some special circumstances, ASN1.Remove directives cannot possibly be generated into the `.rmv` file. In such a case, the user should successively call the ASN.1 compiler several times (correcting his/her syntax) until his/her notation ASN.1-compiles cleanly.

A more complicated example:

```
--<ASN1.Remove Mod.bit, Mod.Int>--
--<ASN1.Remove Mod.Set.a.b.c.d.*.e>--

Mod DEFINITIONS ::= BEGIN
    Int ::= INTEGER
    A ::= SEQUENCE {
        a        Int OPTIONAL,
        b        BIT STRING
    }
    a A ::= { a 1, b bit}
    bit BIT STRING ::= '0'B

    C ::= CHOICE {
```

```
        a        Int
    }

    Set ::= SET {
        a SET {
            b SET {
                c SET {
                    d SET OF SET {
                        e INTEGER OPTIONAL
                    }
                }
            }
        }
    }
    s Set ::= { a { b { c { d { {e 3}, {e 4}, {e 5} } } } } }
END
```

The `.rmv` file will contain the following Remove directives for types, values, components which reference items marked for removal:

`--<ASN1.Remove Mod.s>--` - `s` contains values for the component `e` marked for removal;

`--<ASN1.Remove Mod.C>--` - `C` references the type `Int` marked for removal;

`--<ASN1.Remove Mod.a>--` - `a` contains the reference to the defined value `bit` marked for removal;

`--<ASN1.Remove Mod.A.a>--` - the component `A.a` is of the defined type `Int` marked for removal.

Important notes:

The listing file generated with the `-listingFile` command line option omits removed items entirely. If you need to get a full listing with removed items included, use the `-modListingFile` command line option.

Standard directives applied to or found inside ASN.1 items marked for removal have no effect.

The ASN1.Remove directive applied to the fields of a parameterized type results in removing those fields from each instance of the parameterized type. We do not recommend application of the ASN1.Remove directive to parameterized types.

Removing fields from a type that is referenced in the COMPONENTS OF type notation results in removing those fields from each type specified with the COMPONENTS OF type notation.

Types that are referenced only by items marked for removal may become PDUs after those items are removed.

4.4.1.8 ASN1.Version

The ASN1.Version directive allows the user to specify the year of the ASN.1 standard in use in a particular ASN.1 module. This directive is useful for applications that have several versions of ASN.1 source in use. **By default,** the ASN.1 compiler runs in a neutral version mode until version-specific

syntax (e.g., information objects and macro notation) is encountered; after which, the Compiler latches on to the version to which such notation is peculiar.

The Version directive has the following format:

```
--<ASN1.Version version moduleName [{ oid }]>--
```

where *version* is one of the years that an ASN.1 standard was published by ITU-T (e.g., 1990, 1994, 1997, 2002, and 2008). Secondly, *moduleName* is a mandatory operand that specifies which module is of the version specified. Following the *moduleName* argument an OBJECT IDENTIFIER enclosed in curly braces may be optionally given for unique module identification.

Example:

```
--<ASN1.Version 1990 Module>--

Module DEFINITIONS --<ASN1VER 1994>-- ::= BEGIN
a A ::= INTEGER {one(1), two(2)} : one
A ::= ANY
Type ::= SET {
typeComponent INTEGER,
b BOOLEAN
}
END
```

Note that the ASN1.Version directive takes precedence over the OSS-specific old-style ASN1VER directive in the above example.

If a compiler version option (e.g., -1990 and -2008 (see section 3.3.1)) and the ASN1.Version directive conflict, the higher version year is used.

Related directives and command-line options:

OSS.ASN1VER (see section 4.4.2.2), -1990 / -2008 (see section 3.3.1)

4.4.1.9 ASN1.WorkingSet

The ASN1.WorkingSet directive is used to identify those parts of the ASN.1 input that are central to the application. All of the names (types, values, information objects, etc.) defined in the working-set are directly available for use by the application. Parts of the ASN.1 specification not included in the working-set are only available to the application if they are referenced from those parts in a working-set. **By default,** the ASN.1 compiler chooses a working set by examining the type of compiler directives present. Refer to section 4.4.1.9.1 for more information about this default behavior.

The ASN1.WorkingSet directive has the format:

```
--<ASN1.WorkingSet WSName absoluteReference [, absoluteReference] … >--
```

WSName is a working-set name and is required to be present for consistency between the ASN.1/C product and the ASN.1/C++ and ASN.1/Java products; although required, it is otherwise ignored. *absoluteReference* can refer to a module name, type reference, value reference, object class reference, object reference, or object set reference. If *absoluteReference* is a module name, this

directive has the same effect as the OSS.ROOT directive which causes all unreferenced types within this module to be considered PDUs.

Including a type reference into a working-set does not necessary result in making this type a PDU:

```
--<ASN1.WorkingSet MyWorkingSet A.Bar>--

   A DEFINITIONS ::= BEGIN
       Foo ::= SEQUENCE OF Bar
       Bar ::= SEQUENCE {a INTEGER, b Foo OPTIONAL}
       Z ::= BOOLEAN
   END
```

Even though `A.Foo` was not explicitly placed in the working-set it still goes in there since it is referenced by `A.Bar`. This leads to two types in the working-set referencing each other, so neither is a PDU (unless an implementation chooses to make either or both a PDU).

Example:

```
--<ASN1.WorkingSet WSName Module2, Module3.Type4>--

Module2 DEFINITIONS ::= BEGIN
    Type1 ::= SET {
        typeComponent INTEGER,
        b BOOLEAN
    }

    Type2 ::= INTEGER
END

Module3 DEFINITIONS ::= BEGIN
    Type3 ::= SET {
        aComponent INTEGER,
        b BOOLEAN
    }

    Type4 ::= IA5String
END
```

This example makes `Type1`, `Type2` and `Type4` directly available to the application. In addition, these three types are treated as PDUs.

Note: A component of a SET, SEQUENCE, CHOICE, SET OF, or SEQUENCE OF cannot be added to a working-set by itself.

Tip: If you want extend an existing working-set, you may specify a separate instance of the ASN1.WorkingSet directive and give the name of the working-set that you want to extend. For illustration:

```
--<ASN1.WorkingSet commonSet module1>--
. . . .
--<ASN1.WorkingSet commonSet module14>--
```

The working-set, `commonSet` , in the above example will contain both `module1` and `module14`.

4.4.1.9.1 ASN.1 compiler's default determination of the working set

By default, the ASN.1 compiler decides on the scope of the working set by studying the types of directives present in the input. In regard to working sets, it considers directives of four types: (1) ASN1.WorkingSet directives, (2) non-WorkingSet ASN.1 standard directives, (3) OSS.ROOT directives, and (4) non-ROOT OSS-specific directives. Based on the presence and absence of these directives in the input, the ASN.1 compiler assumes one of the following four modes of operation:

1) *default working set* mode (abbreviation: **dws**) means that **all** input module names are included in the working set. Additionally when the `-genDirectives` command-line option is specified, the compiler generates an ASN1.WorkingSet directive for each module name in `.gen` file.

2) *root default* mode (abbreviation: **rd**) means that only the last module will be included in the working set. In this mode, when the `-genDirectives` command-line option is specified, no ASN1.WorkingSet or OSS.ROOT directives are included in the `.gen` file.

3) *working set* mode (abbreviation: **ws**) means that the ASN.1 compiler will only include modules explicitly specified by the ASN1.WorkingSet and OSS.ROOT directives in the working set. In addition, when the `-genDirectives` command-line option is specified, all modules from the working set will have an ASN1.WorkingSet directive generated for them in the `.gen` file, even if they were included via the OSS.ROOT directive. However in some special circumstances (particularly when the `-root` option or global OSS.ROOT directive is specified) , both ASN1.WorkingSet and OSS.ROOT directives are generated in the `.gen` file.

4) *root* mode (abbreviation: **r**) means that **no** module will be included into the working set unless it is specified in an OSS.ROOT directive. Additionally when the `-genDirectives` command-line option is specified, the compiler generates an OSS.ROOT directive with one or more module name arguments (one argument for each module in the working set) in the `.gen` file.

The following table illustrates when the above four modes are used by the ASN.1 compiler:

Directive Type	Types of Directives Present in Input (**X** = present, blank = absent)															
OSS.ROOT					X	X	X	X	X	X	X					X
ASN1.WorkingSet							X	X	X	X		X	X	X	X	
ASN.1 standard directive (non-WorkingSet)	X			X				X	X		X			X	X	X
OSS-specific directives			X	X		X			X	X			X		X	X

(non-ROOT)																
Resultant Mode	dws	rd	rd	rd	r	r	ws	ws	ws	ws	ws	ws	ws	ws	ws	ws

4.4.2 OSS-Specific Directives

The format for OSS-specific directives is shown below:

```
--<OSS.directiveName [absoluteReference] [directiveOperands]>--
```

where *directiveName* is the name of one of the OSS-specific directives which is currently supported by the ASN.1 compiler. An OSS-specific directive may also include an absolute reference (*absoluteReference*) of the item to which the directive is supposed to be applied. (For detailed information on absolute references, refer to the definition box in section 4.4.1). No absolute reference needs to be specified if an OSS-specific directive is being used for a global effect. In such a case, *absoluteReference* is just left out. The following OSS-specific directives can be used without absolute references and have the same effect as the corresponding old-style directives specified globally:

```
OSS.DECIMAL / OSS.MIXED
OSS.DEFAULTVALUE / -OSS.NODEFAULTVALUE
OSS.DEFINITE / -OSS.INDEFINITE
OSS.EXTENSIBLE / -OSS.INEXTENSIBLE
OSS.FLOAT / OSS.DOUBLE
OSS.OBJECTID
OSS.PDU / -OSS.NOPDU
OSS.SHORT / OSS.INT / OSS.LONG / OSS.LONGLONG / OSS.HUGE
OSS.VALUE / -OSS.NOVALUE
```

Placement of OSS-Specific Directives

The OSS-specific directives can be used in the ASN.1 input wherever spaces or tabs are allowed (i.e. locally, at the module-level, and globally). However, when used globally, they should be specified prior to the first module definition. In addition, the directives can be captured in a `.gen` file by the use of the `-genDirectives` command line option (see section 3.3.28). Then, all of the directives in the ASN.1 syntax may be removed and the `.gen` file can be passed as the first parameter on the command-line when invoking the ASN.1 compiler.

The following subsections give detailed descriptions of the OSS-specific directives accepted by the OSS ASN.1 compiler. Note again that these directives are listed in alphabetical order.

4.4.2.1 OSS.ARRAY

The OSS.ARRAY directive specifies that a SET OF, SEQUENCE OF, or OBJECT IDENTIFIER type should be represented as an array of values along with a count field. **By default,** the SET OF and SEQUENCE OF types are represented as singly-linked lists and the OBJECT IDENTIFIER is represented in a structure with a character string pointer and a length field (giving the size of the character string in bytes).

Note that the OSS.ARRAY directive is not supported when the -helperNames option is specified.

This directive has the following format:

```
--<OSS.ARRAY [[absoluteReference] or [asn1Type [, asn1type]…]]>--
```

When used globally, the OSS.ARRAY directive requires operands specifying which ASN.1 types are to be affected. If the user desires to only affect a particular instance of a type, he/she should use an absolute reference (*absoluteReference*) to the desired type. On the other hand if the user wants to set the representation for all instances of a particular type, he/she should specify the name of the desired ASN.1 built-in type (`asn1Type` can take the values of: SET OF, SEQUENCE OF, or OBJECT IDENTIFIER.) Note that more than one `asn1type` operand may be present, separated by commas.

Although the OSS.ARRAY directive must have either an absolute reference or an ASN.1 built-in type name specified, both an absolute reference and an ASN.1 built-in type name may not be present in a single OSS.ARRAY directive.

When used locally, the OSS.ARRAY directive accepts no operands and affects only the type it is placed next to.

Example:

The following notation shows both the local and global use of the OSS.ARRAY directive:

```
--<OSS.ARRAY SET OF>--
--<OSS.ARRAY ArraySample.RealArray>--

ArraySample DEFINITIONS ::= BEGIN
    IntArray        ::= SEQUENCE SIZE (5) --<ARRAY>-- OF INTEGER
    RealArray       ::= SEQUENCE SIZE (5) OF REAL
    RealLinkedList  ::= SEQUENCE SIZE (5) OF REAL
    BooleanArray    ::= SET SIZE (5) OF BOOLEAN
END
```

In the above example `BooleanArray` is set to the array representation using a global directive that specifies that all SET OF structures should be represented as arrays. Similarly, `RealArray` is set to be represented with an array by a global directive that has `RealArray`'s absolute reference as an operand. `IntArray` is set to the array representation by the use of a local directive which takes no operands. (As mentioned in section 4.3.2, local directives do not take on the `OSS` prefix.) Finally, `RealLinkedList` will default to the linked-list representation since no OSS.ARRAY directives are applied to it.

After passing the above notation through the Compiler, the following is generated in the header file:

```
typedef struct IntArray {
    unsigned short  count;
    int             value[5];
} IntArray;

typedef struct RealArray {
    unsigned short  count;
    double          value[5];
} RealArray;
```

```
typedef struct RealLinkedList {
    struct RealLinkedList *next;
    double          value;
} *RealLinkedList;

typedef struct BooleanArray {
    unsigned short  count;
    ossBoolean      value[5];
} BooleanArray;
```

Related directives:
OSS.DLINKED (see section 4.4.2.10), OSS.LINKED (see section 4.4.2.27), OSS.UNBOUNDED (see section 4.4.2.49)

4.4.2.2 OSS.ASN1VER

The OSS.ASN1VER directive is used to specify which version of the ASN.1 standard is in use in the ASN.1 input. **By default,** the ASN.1 compiler runs in a neutral version mode until version-specific syntax (e.g., information objects and macro notation) is encountered; after which, the Compiler latches on to the version to which such notation is peculiar.

This directive has the following format:

```
--<OSS.ASN1VER absoluteReference version>--
```

When specified globally, this directive takes an absolute reference which can be a name of an ASN.1 module or some component within a module that should be treated as if it is in the ASN.1:*version* syntax. *version* can take on the values: `1990`, `1994`, `1997`, `2002`, and `2008`. If specified globally without an absolute reference, the OSS.ASN1VER directive sets the default version of the entire ASN.1 input.

When specified locally, this directive does not take an absolute reference and only affects the type it is placed next to.

Example:

```
--<OSS.ASN1VER Module.NoNames 1990>--

Module DEFINITIONS ::= BEGIN
    NoNames ::= SEQUENCE {
        BOOLEAN,
        BOOLEAN,
        INTEGER
    }
END
```

The OSS.ASN1VER directive in the above syntax tells the Compiler to allow the unnamed elements in the `NoNames` SEQUENCE. (Starting from **ASN.1:1994**, all elements of CHOICE, SEQUENCE, and SET are required to have an identifier.)

Related directives and command-line options:
ASN1.Version (see section 4.4.1.8), `-1990`, `-2008` (see section 3.3.1)

4.4.2.3 OSS.BMPSTRING

The OSS.BMPSTRING directive is used to represent the ASN.1 UTF8String type in C as an unbounded array of two-byte characters (Unicode). This is the default representation for the ASN.1 BMPString type. For more information on the UTF8String type refer to **ITU-T Rec. X.680 (2008), Annex H**. **By default,** the ASN.1 UTF8String type is represented as a NULL terminating character string.

This directive has the following format:

```
--<OSS.BMPSTRING [absoluteReference]>--
```

absoluteReference refers to an item in the ASN.1 syntax that is declared to be a UTF8String. If used without arguments, this directive can only function as a local directive.

When specified globally, this directive should take an absolute reference specifying a UTF8String type which should be represented representation.

When specified locally, this directive does not take any arguments and only affects the type it is placed next to.

Example:

```
--<OSS.BMPSTRING Module.TwoByteGlobal>--

Module DEFINITIONS ::= BEGIN

        Nullterm         ::= UTF8String
        TwoByteLocal     ::= UTF8String --<BMPSTRING>--
        TwoByteGlobal    ::= UTF8String

END
```

The above syntax shows the use of the OSS.BMPSTRING directive with and without operands. `TwoByteGlobal` is set to the BMPString representation by a global OSS-prefixed directive specified with an absolute reference. `TwoByteLocal` is set to the BMPString representation through the use if a local directive. Finally `Nullterm` has no OSS.BMPSTRING directives applied to it, so it defaults to the Null-terminated string representation. Part of the generated header file for the above notation is shown below:

```
typedef unsigned char   *Nullterm;

typedef struct TwoByteLocal {
    unsigned int    length;
    unsigned short  *value;
} TwoByteLocal;

typedef struct TwoByteGlobal {
    unsigned int    length;
    unsigned short  *value;
} TwoByteGlobal;
```

Note that the `length` fields in `TwoByteLocal` and `TwoByteGlobal` specify the number of characters in the string and not the number of bytes.

Notes:
1) The BMPSTRING and UNIVERSALSTRING representations of UTF8String are incompatible with the OSS.OBJHANDLE (or OSS.NOCOPY) directive.
2) With the BMPSTRING and UNIVERSALSTRING representations, character data will be automatically converted to the UTF-8 format before encoding and converted from UTF-8 format after decoding.
3) No attempt is made at run-time to ensure that the generated C variable contains a valid UTF-8 converted character string. The string octets are passed to the encoder unmodified as the user places them. Also, unmodified data octets are used to fill the C variable when decoding. It is the user's responsibility to provide valid UTF-8 data before calling the encoder and to interpret the UTF-8 data received from the decoder.

Related directives:
OSS.NULLTERM (see section 4.4.2.31), OSS.UNBOUNDED (see section 4.4.2.49), OSS.UNIVERSALSTRING (see section 4.4.2.50), OSS.VARYING (see section 4.4.2.55)

4.4.2.4 OSS.COMMENT

The OSS.COMMENT directive instructs the ASN.1 compiler to place a user-defined comment near the beginning of the `.c` and `.h` files generated by the Compiler. **By default,** no user-defined comment is generated.

This directive has the following format:

```
--<OSS.COMMENT "commentText">--
```

The OSS.COMMENT directive can only be specified globally and should only occur once in the ASN.1 input. If two or more OSS.COMMENT statements are found, the last one encountered is used and the others are disregarded. `commentText` is the actual comment (enclosed in double parentheses) that the user wishes to appear in the beginning of the generated files.

Example:

The following ASN.1 notation alerts the application developer by the use of a comment of a special transfer requirement:

```
--<OSS.COMMENT "!ALERT!  PDU should be encrypted before sending.">--

AccessData DEFINITIONS ::= BEGIN
        TopSecret ::= SEQUENCE {
                username        IA5String,
                password        IA5String,
```

```
                    accessLevel     INTEGER
            }
    END
```

The compiler generates the following header file with desired comment under the copyright information comment:

```
/*********************************************************************/
/* Copyright (C) 2002 OSS Nokalva.  All rights reserved.             */
/*********************************************************************/
/************************************************/
/* !ALERT!  PDU should be encrypted before sending.*/
/************************************************/
/* Generated for: OSS Nokalva */
/* Abstract syntax: comment */
/* Created: Wed Jan 24 18:08:15 2002 */
/* ASN.1 compiler version: 5.4.0 */
/* Target operating system: Windows NT/Windows 9x */
/* Target machine type: Intel x86 */
/* C compiler options required: -Zp8 (Microsoft) */
/* ASN.1 compiler options and file names specified:
 * userdata
 */

#ifndef OSS_userdata
#define OSS_userdata

#include "asn1hdr.h"
#include "asn1code.h"


#define          TopSecret_PDU 1

typedef struct TopSecret {
    char            *username;
    char            *password;
    int             accessLevel;
} TopSecret;


extern void *userdata;    /* encoder-decoder control table */
#endif /* OSS_userdata */
```

Notice that the rest of the file is generated as usual. The `.c` file will also have the same comment near the top.

Note: Starting from version 5.4.0, ASN.1 single line and multi-line comments are transferred by default to the generated header file. Thus, you may also enter a normal ASN.1 comment in the input ASN.1 source and a valid C/C++ comment will be generated in the output file. Refer to section 7.10 for more information about the transfer of ASN.1 comments into the ASN.1 compiler-generated header file.

4.4.2.5 OSS.CommentPreservation

The OSS.CommentPreservation directive gives you more control over the transfer of ASN.1 comments to the generated header file. This directive allows you to turn comment transfer on for certain sections of your ASN.1 source and to turn it off for other sections. **By default,** all ASN.1 comments encountered are transferred to the generated header file.

This directive has the following format:

```
--<OSS.CommentPreservation [ON | OFF]>--
```

Either the literal `ON` or the literal `OFF` must appear as an argument to this directive.

This directive affects all lines of ASN.1 source that follow it until another OSS.CommentPreservation directive is encountered.

Example:

```
--<OSS.CommentPreservation OFF>--

I1 ::= INTEGER   -- I1 type

--<OSS.CommentPreservation ON>--

i I1 ::= 1       -- i variable
```

The above syntax causes the following comments to be generated in the header file:

```
typedef int             I1;

extern I1 i;       /* i variable */
```

Note: the `--<OSS.CommentPreservation OFF>--` directive is listed as the first directive of any: (1) `.gen` files you produce with the `-genDirectives` option, (2) any `.spl` file you produce with the `-splitForSharing` or the `-splitHeader` option, and (3) any OSS-supplied *asn1dflt* files passed to the compiler used by the compiler.

4.4.2.6 OSS.DataCallback

The OSS.DataCallback compiler directive associates a field of a SEQUENCE or SET type, an alternative of a CHOICE type, or an element of a SEQUENCE OF or a SET OF type with your callback function. This directive is useful when using partial decoding.

When specified globally (either in an ASN.1 file before the module definition or in a separate directives file), the OSS.DataCallback directive takes an absolute reference to a field:

```
--<OSS.DataCallback [absoluteReference] ["]FunctionName["]>--
```

`FunctionName` is the user-defined name for the callback function.

When specified locally, the OSS.DataCallback directive must be placed next to the field definition. For example:

```
age INTEGER(1..100) --<DataCallback "FunctionName">--,
```

It's possible for a named type to appear more than once in a PDU. When an OSS.DataCallback directive is applied to such a field, the decoder will call the same callback function for each occurrence. The callback, however, cannot differentiate a call for one field from a call for the other since they are both associated with the same function. To discern the difference, consider the OSS.InfoCallback directive described in section 4.4.2.27.

Related directives, options, and functions:
`OSS.InfoCallback` (section 4.4.2.27), `-enablePartialDecode` (section 3.3.22), `-partialDecodeOnly` (section 3.3.53), `ossPartialDecode()` (See the OSS ASN.1 Runtime API Manual, section 3.5.48)

4.4.2.7 OSS.DECIMAL / OSS.MIXED

The directives OSS.DECIMAL and OSS.MIXED specify representations for the ASN.1 REAL type.

Note that the OSS.MIXED directive is not supported when the -helperNames option is specified.

When the OSS.DECIMAL directive is used, the representation is a null-terminated character string of the form:

`"[-]nnnnn.nnnnn[E[-]nnnnn]"`.

Note that the character string pointer can also be set to point to the special variables `OSS_PLUS_INFINITY`, `OSS_MINUS_INFINITY` and `ossNaN`.

When the OSS.MIXED directive is used, the following representation is used, permitting either a decimal or binary representation:

```
enum MixedReal_kind {BINARY, DECIMAL};

typedef struct {
    enum MixedReal_kind kind;
    union {
        double base2;
        char  *base10;
    } u;
} MixedReal;
```

By default, the REAL type is represented as a `double`.

These directives have the following format:

```
--<OSS.DECIMAL [absoluteReference]>--
--<OSS.MIXED [absoluteReference]>--
```

When specified globally, these directives can take an absolute reference specifying which component of the ASN.1 syntax is intended for the desired representation. If specified globally without an absolute reference, the OSS.DECIMAL and OSS.MIXED directives determine the default representation for all REAL types within their scope.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to.

Note: Constraining the base of the REAL type to 2 implies the binary representation which is represented as a `double` by default. Constraining the base to 10 implies the decimal representation which is represented as null-terminated `char*` by default.

Examples:

The following ASN.1 notation shows the various ways of determining the representation of the REAL type:

```
Module DEFINITIONS ::= BEGIN
   Decimal ::= REAL (WITH COMPONENTS {..., base (10) } )
   Binary  ::= REAL (WITH COMPONENTS {..., base (2) } )
   BinaryAlso ::= REAL
   Mixed   ::= REAL --<MIXED>--
END
```

After passing the above syntax through the compiler, the following is generated:

```
typedef char            *Decimal;

typedef double          Binary;

typedef double          BinaryAlso;

typedef MixedReal       Mixed;
```

Notice how `BinaryAlso` defaults to the `double` representation, since no directives are applied to it.

The following example shows the use of global and local uses of the OSS.DECIMAL and OSS.MIXED directive:

```
--<OSS.DECIMAL>--

Module DEFINITIONS ::= BEGIN
        Decimal         ::= REAL (WITH COMPONENTS {..., base (10) } )
        DecimalAlso     ::= REAL (WITH COMPONENTS {..., base (2) } )
        DecimalToo      ::= REAL
        Mixed           ::= REAL --<MIXED>--
END
```

After compiling the above syntax, the following is generated in the header file:

```
typedef char            *Decimal;

typedef char            *DecimalAlso;

typedef char            *DecimalToo;
```

```
typedef MixedReal       Mixed;
```

Notice how the global directive `--<OSS.DECIMAL>--` overrides the default representation of the base 2 constraint. However, it does not take precedence over the local `--<MIXED>--` directive.

Warning: `double` is provided as the default representation because it is OSS's assessment that this is what most users want. However, this is inconsistent with the ASN.1 standard which implies that values encoded in base 10 be represented in a format suitable for base 10 and values encoded in base 2 be represented in a format suitable for base 2 in order not to lose precision in conversions to and from their floating point representation.

If you want to ensure that no precision is ever lost, either constrain the type REAL to base 2 or base 10, or used the MIXED directive to ensure that the C representation is capable of encoding/decoding without any conversion.

Related directives:

ASN1.RealRepresentation (see section 4.4.1.6), OSS.DOUBLE / OSS.FLOAT (see section 4.4.2.19)

4.4.2.8 OSS.DECODEONLY / OSS.ENCODEONLY

The OSS.DECODEONLY and OSS.ENCODEONLY directives instruct the compiler that when it is generating the time-optimized encoder/decoder, it should generate only encoder routines (OSS.ENCODEONLY) or only decoder routines (OSS.DECODEONLY). This option has no effect when the space-optimized encoder/decoder is being used, since the information generated in the control table is used for both encoding and decoding. **By default**, both encoder and decoder routines are generated when the `-codeFile` option is specified.

These directives have the following format:

```
--<OSS.DECODEONLY [absoluteReference]>--
--<OSS.ENCODEONLY [absoluteReference]>--
```

When specified globally, these directives can take an absolute reference which specifies which part of the ASN.1 input is affected. If specified globally without an absolute reference, the OSS.ENCODEONLY and OSS.DECODEONLY directives determine the default contents of the time-optimized `.c` file.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to.

Example:

```
--<OSS.DECODEONLY Module.InBox>--
```

```
Module DEFINITIONS ::= BEGIN
    InBox ::= SEQUENCE {
        senderAddress     IA5String,
        messageType       ENUMERATED {binary(0), text(1)},
        messageContents   UTF8String
    }
    OutBox ::= SEQUENCE {
        receiverAddress   IA5String,
        messageType       ENUMERATED {binary(0), text(1)},
        messageContents   UTF8String
    } --<ENCODEONLY>--
END
```

The above notation will instruct the Compiler to generate only decoding routines for `InBox` and only encoding routines for `OutBox` in the time-optimized `.c` file.

4.4.2.9 OSS.DefineName

The OSS.DefineName directive allows users to give names to generated `#defined` components within CHOICE, SEQUENCE, or SET structures. **By default,** the Compiler generates its own names for such `#defined` components.

This directive has following format:

```
--<OSS.DefineName absoluteReference ["]userDefinedName["]>--
```

`absoluteReference` refers to a particular field in some CHOICE, SEQUENCE, or SET; this operand must be present. `userDefinedName` is the name that the user wishes to be assigned to the specified `#defined` field; this operand must also be present and can be optionally contained in a pair of double quotes.

Although the OSS.DefineName directive can appear anywhere white-space is allowed, the `OSS` prefix, a full absolute reference, and a user defined name must all be present (i.e. It is used like a global directive).

Example:

```
--<OSS.DefineName Module.Type.field1 aNewName>--

Module DEFINITIONS ::= BEGIN
    Type ::= CHOICE {
        field1   BOOLEAN,
        field2  REAL
    }
END
```

After running the ASN.1 compiler, the following will be generated in the header file:

```
typedef struct Type {
    unsigned short  choice;
    #       define      aNewName_chosen 1
    #       define      field2_chosen 2
    union {
```

```
            ossBoolean      field1;
            double          field2;
        } u;
    } Type;
```

Notice that Compiler named the first `#define:  aNewName_chosen` instead of `field1_chosen`.

4.4.2.10 OSS.DEFINITE / OSS.INDEFINITE

The OSS.DEFINITE and OSS.INDEFINITE directives specify that the definite or indefinite length form should be used when encoding protocol data units (PDUs). **By default,** the definite length form is used.

These directives have following format:

```
--<OSS.DEFINITE [absoluteReference]>--
--<OSS.INDEFINITE [absoluteReference]>--
```

When used globally, these directives can take an absolute reference to a particular ASN.1 component (which should be a PDU). Application of these directives to non-PDU ASN.1 types has no effect. If specified globally without an absolute reference, the OSS.DEFINITE and OSS.INDEFINITE directives determine the default type of length encoding for all PDUs.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to.

Note: Note that the time-optimized BER encoder and the Lean BER encoder do not currently support the indefinite length form of encoding. However, all of the provided decoders (i.e. space-optimized/Lean/time-optimized) *do* support the indefinite length form of encoding.

Example:

The following example illustrates both the local and global use of these directives:

```
--<OSS.INDEFINITE>--

Module DEFINITIONS ::= BEGIN
        ByteStream ::= OCTET STRING
        DataPacket ::= SEQUENCE {
                username        IA5String,
                userID          INTEGER
        } --<DEFINITE>--
END
```

In the above example, the `--<OSS.INDEFINITE>-- directive` sets a global default for indefinite length encoding. Consequently, `ByteStream` will be encoded using the indefinite length form, since no other directives are applied to it. However, `DataPacket` will be encoded using the definite length form due to the local `--<DEFINITE>--` directive placed next to it.

These two directives do not affect the header file generated.

4.4.2.11 OSS.DLINKED

The OSS.DLINKED directive specifies that SET OF, and SEQUENCE OF should be represented as a doubly-linked list of values. **By default,** the SET OF and SEQUENCE OF types are represented as a singly-linked list of values. Refer to section 7.6 for more details about the default representation of these ASN.1 types.

Note that the OSS.DLINKED directive is not supported when the -helperNames option is specified.

This directive has following format:

```
--<OSS.DLINKED [[absoluteReference] or [asn1Type [, asn1Type] …]]>--
```

When used globally, this directive can take an absolute reference to a SEQUENCE OF, or SET OF, type which the user desires to represent as a doubly-linked list of values. Conversely, an ASN.1 built-in type name (`asn1Type`) can be specified. `asn1Type` can be the name of any of the above-mentioned built-in types; it instructs the ASN.1 compiler to represent all instances of the specified type as doubly-linked lists. Although more than one `asn1Type` may be present, both an absolute reference and a built-in type name may not be specified.

When specified locally, this directive does not take any arguments and only affects the type that it placed next to.

Example:

The following ASN.1 notation shows the global use of the OSS.DLINKED directive:

```
--<OSS.DLINKED  SET OF>--

Sample1 DEFINITIONS EXPLICIT TAGS ::= BEGIN
    IntDL  ::= [1] SET SIZE (5) OF INTEGER
    IntAR  ::= [2] SET SIZE (15) --<ARRAY>-- OF INTEGER
    StrDL  ::= [3] SET OF IA5String
END
```

The global directive `--<OSS.DLINKED  SET OF>--` makes the default representation for all instances of SET OF to be doubly-linked lists. However, note that the local directive `--<ARRAY>--` overrides the default representation for `IntAR` above.

After passing the above notation through the ASN.1 compiler, the following is generated in the header file

```
typedef struct IntDL {
    struct IntDL    *next;
    struct IntDL    *prev;
    int             value;
} *IntDL;

typedef struct IntAR {
    unsigned short  count;
    int             value[15];
```

```
} IntAR;

typedef struct StrDL {
    struct StrDL    *next;
    struct StrDL    *prev;
    char            *value;
} *StrDL;
```

Notice that the double-linked list representation has two pointers: `next*` and `prev*`. These pointers point to the next and previous elements in the SEQUENCE OF or SET OF. The last element in the list will have its `next` pointer set to `NULL` while the first element in the list will have its `prev` pointer set to `NULL`.

Related directives:

OSS.ARRAY and OSS.LINKED

4.4.2.12 OSS.DLINKED-PLUS

The OSS.DLINKED-PLUS directive was introduced in the 8.0 version. It specifies that SET OF, and SEQUENCE OF types should be represented as doubly-linked-plus lists of nodes. A doubly-linked-plus representation is similar to the doubly-linked representation, and in addition an extra structure is generated for each such list that keeps pointers to the head and tail nodes and keeps a total of the number of nodes in the list. **By default** if the `-helperNames` option is specified or implied, the SET OF and SEQUENCE OF types with elements whose types are structured or pointered are represented as a doubly-linked-plus list of values. SET OF and SEQUENCE OF types with elements whose types are not structured and not pointered (ex. INTEGER without HUGE, REAL with DOUBLE, BOOLEAN, NULL and ENUMERATED) are represented as an unbounded structures. Refer to section 7.2.2 for more details about the default representation of these ASN.1 types in the helper mode.

This directive has the following format:

```
--<OSS.DLINKED-PLUS [[absoluteReference] or [asn1Type [, asn1Type]
…]]>--
```

When used globally, this directive can take an absolute reference to a SEQUENCE OF, or SET OF type which the user desires to represent as a doubly-linked-plus list of nodes. Alternatively, an ASN.1 built-in type name (`asn1Type`) can be specified. `asn1Type` can be the name of any of the above-mentioned built-in types; it instructs the ASN.1 compiler to represent all instances of the specified type as doubly-linked lists. Although more than one `asn1Type` may be present, both an absolute reference and a built-in type name may not be specified.

When specified locally, this directive does not take any arguments and only affects the type to which it's applied i.e. the type next to which the directive is placed.

Note that if a type-specific OSS.HelperListAPI is applied to a SET OF or SEQUENCE OF type, it overrides any other global or local directives and the DLINKED-PLUS representation is used.

Example:

The following ASN.1 notation shows the global use of the OSS.DLINKED-PLUS directive:

```
--<OSS.DLINKED-PLUS  SET OF>--

M DEFINITIONS AUTOMATIC TAGS::= BEGIN
S1      ::= [1] SET OF INTEGER
S2      ::= [2] SET --<UNBOUNDED>-- OF INTEGER
S3      ::= [3] SET OF IA5String
END
```

The global directive `--<OSS.DLINKED-PLUS  SET OF>--` makes the default representation for all instances of SET OF to be doubly-linked-plus lists. However, note that the local directive `--<UNBOUNDED>--` overrides the default representation for `S2` above.

After passing the above notation through the ASN.1 compiler with -helperNames, the following is generated in the header file:

```
typedef struct S1 {
    struct S1_node  *head;
    struct S1_node  *tail;
    unsigned int    count;
} S1;

typedef struct S1_node {
    struct S1_node  *next;
    struct S1_node  *prev;
    int             value;
} S1_node;

typedef struct S2 {
    unsigned int    count;
    int             *value;
} S2;

typedef struct S3 {
    struct S3_node  *head;
    struct S3_node  *tail;
    unsigned int    count;
} S3;

typedef struct S3_node {
    struct S3_node  *next;
    struct S3_node  *prev;
    char            *value;
} S3_node;
```

Notice that the doubly-linked-plus list representation has two structures being generated. The structure with the `_node` suffix has two pointers: `next*` and `prev*`. These pointers point to the next and previous elements in the SEQUENCE OF or SET OF. The last element in the list will have its `next` pointer set to `NULL`, while the first element in the list will have its `prev` pointer set to `NULL`. The third field in the structure is the actual element for which the node was created. The second structure (for example `S1` or `S3`) also has two pointers: `head*` and `tail*`. These pointers point to the first and last nodes in the doubly-linked list. The third field keeps the total number of nodes in the list.

Using the helper list API functions generated when the `-helperListAPI` option or the `OSS.HelperListAPI` directive is present in order to create values for DLINKED-PLUS data types in C code is recommended.

Related directives:

OSS.DLINKED
OSS.HelperListAPI/OSS.NoHelperListAPI

4.4.2.13 OSS.DTD

The OSS.DTD directive was introduced in version 6.0 and gives you greater control over the XML data type definition (DTD) files which are produced by the ASN.1 compiler. Using this directive, you can instruct the ASN.1 compiler to generate a separate DTD with your own desired filename for any particular PDU. **By default**, no DTDs are produced by the ASN.1 compiler unless the `-dtd` option is specified. When the `-dtd` option is specified, a data type definition for each PDU in the input is generated into a separate file whose name is derived from that of the PDU. (Note that you can use data type definitions to make the XER output produced by `ossEncode()`editable in a non-ASN.1-specific XML tool.)

Note: You can use the `ossSetXmlDTD()` API function to have the `ossEncode()` function generate a reference to your desired DTD in the ensuing XER encoding produced. Refer to the **OSS ASN.1 API/C Reference Manual** for more details.

This directive has the following format:

--<OSS.DTD [*<absoluteReference> "<OptionalNewName>.dtd"*]**>--**

absoluteReference refers to any PDU in the input ASN.1 syntax that you wish to have a separate XML DTD generated for. *OptionalNewName* is an optional/non-mandatory argument for this directive that allows you to specify a new full pathname for the DTD which will be generated.

When an `absoluteReference` is not specified, it is assumed that a DTD is to be produced for each and every PDU in the input syntax. This form of the OSS.DTD directive (i.e. a global directive application) is identical in behavior to the `-dtd` ASN.1 compiler option.

Although the OSS.DTD directive can appear anywhere white-space is allowed, the OSS prefix must be present.

Example:

```
--<OSS.DTD XModule.StringType>--
--<OSS.DTD XModule.IntType "AnotherNameForInt.dtd">--

XModule DEFINITIONS ::= BEGIN
   StringType ::= UTF8String
   IntType  ::= INTEGER
```

```
   studentName ::= <StringType>John H. Doe</StringType>
   studentAge ::= <IntType>23</IntType>
END
```

For the above syntax, two separate DTD files (one named `StringType.dtd` and the other named `AnotherNameForInt.dtd`) will be produced for their respective PDUs.

Related ASN.1 compiler options:
`-dtd` (see section 3.3.21)

4.4.2.14 OSS.ENCODABLE

The OSS.ENCODABLE directive is used to support DER (distinguished encoding rules) applications that encode encrypted signatures. When applied to a component of a CHOICE, SEQUENCE, or SET, it causes that component to be represented as an open type. In addition, the type that would normally have been generated as a field in the `struct` is generated as a `typedef` which can be encoded independently of the CHOICE, SEQUENCE, or SET in which it was embedded. **By default,** all components within a CHOICE, SEQUENCE, or SET are represented using their normal forms and no `typedef` is generated for them.

This directive has following format:

```
--<OSS.ENCODABLE [absoluteReference]>--
```

When used globally, this directive can take an absolute reference to a component within a CHOICE, SEQUENCE, or SET which the user desires to represent as an open type.

When specified locally, this directive does not take any arguments and only affect the type it placed next to.

Note:
1) The OSS.ENCODABLE directive should not be applied to a tagged type. To achieve the same effect for tagged types, you can use the ASN1.DeferDecoding directive (see section 4.4.1.2). Generally speaking, we recommend you use the ASN1.DeferDecoding directive instead of the OSS.ENCODABLE directive since the former is designed for greater applicability.
2) This directive does NOT work when PER or XER is in use.

Example:

The following ASN.1 syntax shows the local use of the OSS.ENCODABLE directive:

```
Module DEFINITIONS ::= BEGIN
        A ::= SEQUENCE {
               id    [0] INTEGER,
               data  [1] OCTET STRING --<ENCODABLE>--
        }
```

```
END
```

After passing the above notation through the compiler, the following is generated in the header file:

```
#define          A_PDU 1
#define          A_data_encodable_PDU 2

typedef struct A {
    int             id;
    OpenType        data;  /* A_data_encodable type */
} A;

typedef struct A_data_encodable {
    unsigned int    length;
    unsigned char   *value;
} A_data_encodable;
```

Note that a separate PDU tag was generated for the type marked with the OSS.ENCODABLE directive. Also, notice how `data` is represented as an open type and the associated OCTET STRING has a `typedef` created for it.

`OpenType` is defined as follows in `asn1hdr.h`:

```
typedef struct {
  int            pduNum;
  long           length;       /* length of encoded */
  void          *encoded;
  void          *decoded;
#ifdef OSS_OPENTYPE_HAS_USERFIELD
  void          *userField;
#endif
} OpenType;
```

Note that `userField` can carry any information that the user desires. This field is only generated if the `-extendOpenType` compiler option is specified and is always ignored by the encoder/decoder.

4.4.2.15 OSS.ENCODED / OSS.OBJECTID

The OSS.ENCODED and OSS.OBJECTID directives allow the representation of the OBJECT IDENTIFIER type to be a structure which contains a length in octets and an address of an array of characters containing the BER-encoded contents of the OBJECT IDENTIFIER (e.g., `{1 2 3 4 5}` would be stored as: `{0x2A, 0x03, 0x04, 0x05}`). This permits the representation of OBJECT IDENTIFIER types with node values exceeding the maximum integer representation on a given machine. The encoded value is derived according to the **ITU-T Rec.X.690 (2008)** document (clause 8.19). **By default,** OBJECT IDENTIFIERs are represented as a structure with a character string pointer and a length field giving the size of the character string in bytes (i.e. the ENCODED representation).

Note that the OSS.OBJECTID directive is not supported when the -helperNames option is specified.

These directives have the following format:

```
--<OSS.OBJECTID [[ENCODED] or [length]]>--
--<ENCODED>--
```

The OSS.OBJECTID directive can be used globally or locally to set the default representation for all OBJECT IDENTIFIER types within scope. This directive must take either a *length* as an operand (indicating the size [2 to 32,767] in octets of the character string to contain the BER-encoded contents of the OBJECT IDENTIFIER) or the literal **ENCODED** as an operand (indicating that the string to contain the OBJECT IDENTIFIER value is of arbitrary length).

The OSS.ENCODED directive can only be used locally and is equivalent to OSS.OBJECTID ENCODED.

Note: To optimize runtime performance the encoder does not verify that OBJECT IDENTIFIER values in the ENCODED representation are actually of a valid format. It is left up to the application developer to instate such checks.

Example:

The following ASN.1 notation demonstrates the local and global use of the OSS.OBJECTID and OSS.ENCODED directives if -helperNames is not specified:

```
--<OSS.OBJECTID ENCODED>--

Module DEFINITIONS ::= BEGIN
ObjectIDRep     ::= [1] OBJECT IDENTIFIER --<OBJECTID 82>--
EncodedRep      ::= [2] OBJECT IDENTIFIER --<ENCODED>--
UnboundedRep    ::= [3] OBJECT IDENTIFIER --<UNBOUNDED>--
DefaultRep      ::= [4] OBJECT IDENTIFIER

END
```

After passing the above notation through the ASN.1 compiler, the following is generated:

```
typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} ObjectID;

typedef struct ObjectIDRep {
    unsigned short  length;
    unsigned char   value[82];
} ObjectIDRep;

typedef ObjectID        EncodedRep;

typedef struct UnboundedRep {
    unsigned short  count;
    unsigned short  *value;
} UnboundedRep;

typedef ObjectID        DefaultRep;
```

Note again that the default representation is the same as the ENCODED representation.

Example:

The following ASN.1 notation demonstrates the use of the OSS.ENCODED directives if -helperNames is specified:

```
Module DEFINITIONS ::= BEGIN
EncodedRep      ::= [1] OBJECT IDENTIFIER --<ENCODED>--
DefaultRep      ::= [3] OBJECT IDENTIFIER

END
```

After passing the above notation through the ASN.1 compiler, the following is generated:

```
typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

typedef _OID            EncodedRep;

typedef _OID            DefaultRep;
```

4.4.2.16 OSS.ExerNumericBoolean

The `OSS.ExerNumericBoolean` directive may be applied to a BOOLEAN type in the presence of a GLOBAL-DEFAULTS MODIFIED-ENCODINGS encoding instruction. It may also be applied to an ASN.1 module (in this case all such types that are textually defined within this module are affected) and it may be applied globally to affect all such types in all modules in the input syntax.

The directive affects the way values of such BOOLEAN types are encoded. By default, the encoder produces the textual BOOLEAN values "true" and "false", or their replacements provided with the help of the TEXT encoding instruction. In the presence of the `OSS.ExerNumericBoolean` directive, the encoder produces numeric BOOLEAN values (1 and 0) instead.

The directive is useful for reducing the size of the encoded XML documents and for exchanging information with decoders that understand only numeric BOOLEAN values.

Example:

Note: this example is valid beginning with version 8.2 of the OSS ASN.1 Tools for C.

```
Test DEFINITIONS XER INSTRUCTIONS AUTOMATIC TAGS ::= BEGIN

Presence ::= BOOLEAN -- <ExerNumericBoolean>--
p0 Presence ::= FALSE

ENCODING-CONTROL XER
    GLOBAL-DEFAULTS MODIFIED-ENCODINGS
    TEXT Presence:false AS "absent"
    TEXT Presence:true AS  "present"
```

```
END
```

The value `p0` is encoded as

```
<Presence>absent</Presence>
```

by default and as

```
<Presence>0</Presence>
```

in the presence of the `OSS.ExerNumericBoolean` directive.

4.4.2.17 OSS.EXTENSIBLE / OSS.INEXTENSIBLE

The OSS.EXTENSIBLE and OSS.INEXTENSIBLE directives are used to enable or disable decoder support for the rules of extensibility. Their only effect at compile time is the setting of an internal flag which is associated with SEQUENCE, SET and BIT STRING types. The rules of extensibility state that when decoding a PDU:

- Ignore all elements of a SET or SEQUENCE whose tags are not defined in the abstract syntax definition that was input to the compiler.

- Where named bits are used, treat any bit as insignificant if no name is assigned to it in the original ASN.1 source file (e.g., if a BIT STRING defines named bits, and bits other than the named bits are encountered in the input stream, the decoder ignores them.)

By default, the OSS.INEXTENSIBLE directive is implied.

These directives have the following format:

```
--<OSS.EXTENSIBLE [absoluteReference]>--
--<OSS.INEXTENSIBLE [absoluteReference]>--
```

When specified globally, these directives can take an absolute reference which refers to the BIT STRING, SEQUENCE, or SET type that is to be affected. If specified globally without an absolute reference, the OSS.EXTENSIBLE and OSS.INEXTENSIBLE directives determine the default handling of extensible elements by the decoder.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to.

Note: The EXTENSIBLE directive provides the same function as the ASN.1 formal extension marker (…). One may use either the directive or the extension marker but not both.

Example:

The following example shows the global and local use of the OSS.EXTENSIBLE and OSS.INEXTENSIBLE:

```
--<OSS.EXTENSIBLE>--

Sample1 DEFINITIONS EXPLICIT TAGS ::= BEGIN

    Name1 ::= SEQUENCE {
        a INTEGER,
        b VisibleString
    }

    Name2 ::= SET {
        c INTEGER,
        d VisibleString
    } --<INEXTENSIBLE>--
END
```

The local OSS.INEXTENSIBLE directive overrides the global OSS.EXTENSIBLE default. Note that the OSS.EXTENSIBLE and OSS.INEXTENSIBLE directives do not affect the C representation in the header file.

4.4.2.18 OSS.ExtensibleUseType

The `OSS.ExtensibleUseType` directive may be applied to an extensible CHOICE type with a final XER USE-TYPE encoding instruction. It may also be applied to an ASN.1 module (in this case all extensible CHOICE types with a final USE-TYPE instruction textually defined within this module are affected) and it may be applied globally to affect all such types in all modules in the input syntax.

The directive affects the way unknown values of such CHOICE types are decoded. Unknown values are the ones that contain the `xsi:type` attribute with an unrecognized value. By default, such data is decoded as data matching the first alternative and all unknown elements/attributes are ignored. In the presence of the `OSS.ExtensibleUseType` directive, the decoder treats such data as an unknown extension rather than as a value of the first CHOICE alternative. If the input ASN.1 specification was compiled with the `-relaySafe` option, the entire XML sub-element that corresponds to such an unknown extension is preserved in the special ‘unknown alternative’ field by the decoder and may be safely relayed to other parties. In the absence of the `-relaySafe` option, the entire XML sub-element is skipped and the decoded CHOICE type indicates the use of an unknown alternative.

The directive is useful for dealing with unknown elements and attributes from the USE-TYPE content model on a low level. The decoder skips them by default, so if such data needs to be analyzed, the `-relaySafe` option should be used and the `OSS.ExtensibleUseType` directive should be applied to the CHOICE type so as to access the encoded XML data.

Example:

Note: this example is valid beginning with version 8.2 Beta A of the OSS ASN.1 Tools for C.

```
Test DEFINITIONS XER INSTRUCTIONS AUTOMATIC TAGS ::= BEGIN

T ::= [USE-TYPE] CHOICE {
    base SEQUENCE {i INTEGER},
    e1   SEQUENCE {i INTEGER, b BOOLEAN},
    ...
```

```
} --<ExtensibleUseType>--

ENCODING-CONTROL XER
    GLOBAL-DEFAULTS MODIFIED-ENCODINGS
    GLOBAL-DEFAULTS CONTROL-NAMESPACE
    "http://www.w3.org/2001/XMLSchema-instance" PREFIX "xsi"
END
```

For the following XML document:

```
<T xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="e4">
  <i>1</i>
  <j>2</j>
</T>
```

the decoder produces the following results:

a) In the absence of the `ExtensibleUseType` directive (unknown content is skipped):

```
value T ::= base :
  {
    i 1
  }
```

b) In the presence of the `ExtensibleUseType` directive without the `-relaySafe` option (the entire XML subtree is skipped):

```
value T ::= <unknown>
```

and the decoded CHOICE value contains the following data:

```
choice == 0
```

c) In the presence of the `ExtensibleUseType` directive with the `-relaySafe` option (the XML subtree is relayed):

```
value T ::= <unknown>
```

and the decoded CHOICE value contains the following data:

```
choice == ossUnknownAlt_chosen
u.ossUnknownAlt contains
  <T xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="e4">
    <i>1</i>
    <j>2</j>
  </T>
```

Related directives:
The following directive also affects E-XER decoder behavior:
`OSS.SelfCompleteWildcard` (see section 4.4.2.41)

4.4.2.19 OSS.ExtractType / OSS.InlineType

The OSS.ExtractType and OSS.InlineType directives are used for specifying whether a nested structure should be coded using a `typedef` (defined outside of the structure) or inline within the structure. **By default,** when there are identical nested structures in the ASN.1 input, a shared independent `typedef` is created for the identical types. On the other hand if there is a unique nested structure, **by default** it is coded inline in the header file.

These directives have the following format:

```
--<OSS.ExtractType absoluteReference>--
--<OSS.InlineType absoluteReference>--
```

absoluteReference should refer a CHOICE, SEQUENCE, SEQUENCE OF, SET, or SET OF type that is to be affected by these directives; this operand must be present.

Although the OSS.ExtractType and OSS.InlineType directives can appear anywhere white-space is allowed, the OSS prefix and a full absolute reference must both be present (i.e. These directives are used like global directives).

Examples:

The following example shows the effect of the OSS.ExtractType directive on the nested SET OF:

```
--<OSS.ExtractType Mod.Type1.nested>--

Mod DEFINITIONS ::= BEGIN
    Type1 ::= SET {
        nested SET OF INTEGER
    }
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct _setof1 {
    struct _setof1  *next;
    int             value;
} *_setof1;

typedef struct Type1 {
    struct _setof1  *nested;
} Type1;
```

The following is a sample use of the OSS.InlineType directive:

```
--<OSS.InlineType Mod.Type1.1>--

Mod DEFINITIONS ::= BEGIN
   Type1 ::= SET {SET OF INTEGER}
   Type2 ::= SET {SET OF INTEGER}
   Type3 ::= SET {SET OF INTEGER}
END
```

with the C representation:

```
typedef struct Type1 {
    struct _setof1 {
        struct _setof1  *next;
        int             value;
    } *setOf;
} Type1;

typedef struct _setof2 {
    struct _setof2  *next;
    int             value;
} *_setof2;

typedef struct Type2 {
    struct _setof2  *setOf;
} Type2;

typedef struct Type3 {
    struct _setof2  *setOf;
} Type3;
```

Notice how `Type1` has an inline nested structure while `Type2` and `Type3` have `typedef`ined ones.

4.4.2.20 OSS.FLOAT / OSS.DOUBLE

The OSS.FLOAT and OSS.DOUBLE directives specify the C data type that should be used to represent the ASN.1 REAL type. **By default,** the ASN.1 REAL type is represented using the C `double` data type.
Note that the OSS.FLOAT directive is not supported when the -helperNames option is specified.

These directives have the following format:

```
--<OSS.FLOAT [absoluteReference]>--
--<OSS.DOUBLE [absoluteReference]>--
```

When used globally, these directives can take an absolute reference to an ASN.1 REAL type. If specified globally without an absolute reference, the OSS.FLOAT and OSS.DOUBLE directives determine the default representation for all REAL types.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to.

Example:
The following ASN.1 syntax illustrates the local and global use of the OSS.FLOAT and OSS.DOUBLE directives:

```
--<OSS.FLOAT>--
--<OSS.DOUBLE Module.DataFolder.b>--

Module DEFINITIONS ::= BEGIN
```

```
        DataFolder ::= SEQUENCE {
                a       REAL --<DOUBLE>--,
                b       REAL,
                c       REAL
        }
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct DataFolder {
    double          a;
    double          b;
    float           c;
} DataFolder;
```

Notice how the local OSS.DOUBLE directive overrides the global default set by the OSS.FLOAT directive.

Related directives:
ASN1.RealRepresentation (see section 4.4.1.6), OSS.DECIMAL / OSS.MIXED (see section 4.4.2.6)

4.4.2.21 OSS.FUNCNAME

The OSS.FUNCNAME directive is used to specify a name for a function that checks some user-defined constraint. **By default,** the name of the C function that the encoder/decoder will call before encoding and after decoding a user-constrained value is derived from the name of the type reference or identifier with which the constraint is associated. For example, given:

```
Y ::= INTEGER (CONSTRAINED BY { -- Must be a power of 2 -- })
```

the default constraint-checking function name will be `Y_fn`.

This directive has the following format:

```
--<OSS.FUNCNAME [absoluteReference] ["]FunctionName["]>--
```

When specified globally, the OSS.FUNCNAME directive takes an absolute reference to a type that has a user-defined constraint. *`FunctionName`* is the name that the application developer wishes to use for the constraint checking function. Note that when using OSS.FUNCNAME globally, it must appear before the type to which it applies.

When specified locally, the OSS.FUNCNAME directive does not take an absolute reference and only affects the type in whose user-defined constraint it is placed.

Example:

For the following ASN.1 notation:

```
--<OSS.FUNCNAME Module.DataCard.c "checkForPowerOf2">--
```

```
Module DEFINITIONS ::= BEGIN
        DataCard ::= SEQUENCE {
             a BOOLEAN,
             b BOOLEAN,
             c INTEGER (CONSTRAINED BY { -- Must be a power of 2 --})
        }
END
```

results in the following code in the header file:

```
typedef struct DataCard {
    ossBoolean      a;
    ossBoolean      b;
    int             c; /* Must be a power of 2 */
} DataCard;

/* checkForPowerOf2 is user-defined constraint function for ASN.1 item
 * Module.DataCard.c */
extern int DLL_ENTRY checkForPowerOf2(struct ossGlobal *, int *, void **);
```

Notes:

1) Starting with version 5.0, user-defined constraint-checking functions are only generated if the `-userConstraints` compiler option is specified.
2) The ASN1.ConstraintFunction directive has the same function as the OSS.FUNCNAME directive. The only difference is that you may also specify OSS.FUNCNAME locally instead of only globally.
3) User-defined constraints are not supported by the TOED.

Useful Addendum:

The syntax of the user-specified constraint checking functions is:

```
int funcName(struct ossGlobal *g, int *value, void **handle);
```

The first argument, `g`, is precisely the same one that is passed as the first parameter to the encoder/decoder. Within struct `ossGlobal`, there is a field, `userVar`, that you may use in any way that you see fit. The only field that the encoder/decoder will ever modify is `encDecVar`. Similarly, the only field that the memory manager will ever modify is `memMgrVar`.

The second argument is the address of the value whose constraint is to be checked. You are allowed to modify the value. If the value that you provide violates the constraint, it may not be detected by the encoder, and it will never be detected by the decoder.

The third argument is the memory handle of the `value` argument. This argument is useful only on systems where the address of a field may be different from the handle of that field. This argument is of limited use on platforms where mature memory management is provided by the operating system, such as Windows and Unix. However with more complex memory managers, this argument will be needed even on such operating systems.

If your constraint checking function detects a constraint violation it must return a non-zero integer value; otherwise, it must return a value of zero. The encoder/decoder will issue an error message if your function returns a non-zero value.

Related directive:

ASN1.ConstraintFunction (see section 4.4.1.1)

4.4.2.22 OSS.HeaderName

The OSS.HeaderName directive was added in version 5.4.0 and is for use with the split-headers mode of the ASN.1 compiler. Under this mode, each ASN.1 module in the input ASN.1 specification has a separate header file generated for it. Normally, the names of these header files are simply taken from their module references. The OSS.HeaderName directive allows you to explicitly specify a name for the header file generated for a particular ASN.1 module.

This directive has the following format:

```
--<OSS.HeaderName moduleReference "headerFilename">--
```

`moduleReference` refers to the ASN.1 module whose header file is to be renamed. `headerFileName` is the name prefix to use for the chosen module header file.

Although the OSS.HeaderName directive can appear anywhere white-space is allowed, the `OSS` prefix, a full module reference, and a user defined filename must all be present.

Example::

If the following ASN.1 notation:

```
--<OSS.HeaderName MyModule "myInteger.h">--

MyModule DEFINITIONS ::= BEGIN
    A ::= INTEGER
END
```

is passed to the ASN.1 compiler while the `-splitHeader` or `-splitForSharing` option is specified, a header file named `myInteger.h` containing the C representation of the `MyModule` module will be produced.

4.4.2.23 OSS.HelperListAPI / OSS.NoHelperListAPI

The `OSS.HelperListAPI`/`OSS.NoHelperListAPI` directives were introduced in the 8.0 version to control the generation of the helper list APIs. These directives may be applied globally as well as to particular ASN.1 types (note that the `-helperListAPI`/`-noHelperListAPI` options always override these directives). The global `OSS.HelperListAPI` directive affects all types having the `DLINKED-PLUS` representation and not overridden by a local `OSS.NoHelperListAPI` directive. If a

type-specific `OSS.HelperListAPI` is applied to the `SET OF`/`SEQUENCE OF` type, it must have the `DLINKED-PLUS` representation (see 4.4.2.11). An error is issued if the representation is `UNBOUNDED`. You must specify the `-helperNames` option (see section 3.3.37) if at least one such directive is present in the input syntax. As the result, all structures are generated in the header file according to the conditions specified in section 3.3.37.

These directives have the following format:

```
--<OSS.HelperListAPI [[absoluteReference] or [asn1Type [, asn1Type]
…]]>--

--<OSS.NoHelperListAPI [[absoluteReference] or [asn1Type [,
asn1Type] …]]>--
```

When used globally, these directives can take an absolute reference to a `SEQUENCE OF` or `SET OF` type which is represented as a doubly-linked-plus list of nodes; a set of list manipulation functions is produced for this type. Alternatively, an ASN.1 built-in type name (`asn1Type`) can be specified. `asn1Type` can be `SET OF` or `SEQUENCE OF` built-in types; it instructs the ASN.1 compiler to generate list manipulation functions for all instances of the specified ASN.1 built-in type. Although more than one `asn1Type` may be present, both an absolute reference and a built-in type name may not be specified at once.

Generated list API functions are described in section 3.3.35.

Example:

```
--<OSS.DLINKED-PLUS M.Quest.answers>--
--<OSS.HelperListAPI M.Quest.answers>--
M DEFINITIONS AUTOMATIC TAGS ::= BEGIN

Quest ::= SEQUENCE {
    questions   SEQUENCE OF IA5String,
    answers SEQUENCE OF BOOLEAN
}

END
```

When compiled with `-helperNames`, the above syntax results in the following header file (helper list API functions are produced only for ‘Quest_answers’):

```
typedef struct Quest {
    struct Quest_questions *questions;
    struct Quest_answers *answers;
} Quest;

…

typedef struct Quest_answers {
    struct Quest_answers_node *head;
    struct Quest_answers_node *tail;
    unsigned int    count;
} Quest_answers;

typedef struct Quest_answers_node {
    struct Quest_answers_node *next;
    struct Quest_answers_node *prev;
    ossBoolean      value;
```

```
} Quest_answers_node;

/* ********************************************** */
/* Helper List API functions for 'Quest_answers' list */
/* ********************************************** */

/* creates an empty list */
struct Quest_answers * DLL_ENTRY oss_Quest_answers(OssGlobal *_world);

…

/* unlinks and frees node of list */
ossBoolean DLL_ENTRY oss_Quest_answers_unlink(OssGlobal *_world, struct Quest_answers *
_list, Quest_answers_node * _node);
```

Related directives:

OSS.DLINKED-PLUS
OSS.HelperNames

4.4.2.24 OSS.HelperMacro / OSS.NoHelperMacro

The `OSS.HelperMacro`/`OSS.NoHelperMacro` directives were introduced in the 8.0 version. These directives instruct the ASN.1 compiler to generate or not generate helper macros. These directives may be applied globally as well as to particular ASN.1 types (note that the `-helperMacros`/`-noHelperMacros` options always override these directives). If a type-specific OSS.HelperMacro is specified, it overrides any contradicting global helper macro directive.

These directives have the following format:

```
--<OSS.HelperMacro [absoluteReference]>--

--<OSS.NoHelperMacro [absoluteReference]>--
```

When used globally, these directives can take an absolute reference to an ASN.1 typefor which the user wishes to generate helper macros. `OSS.HelperMacro` directives can be used only with context-based compiler-fabricated names for generated structures derived from built-in ASN.1 types. You must specify the `-helperNames` option (see section 3.3.37) if at least one such directive is present in the input syntax. As a result, all structures are generated in the header file according to the conditions specified in section 3.3.37.

Generated helper macros are described in detail in section 3.3.36. When a type is a non-typereference component of a SET, SEQUENCE or CHOICE, its parent type should have its own macros produced in order to produce macros for the component.

Example:

```
--<OSS.HelperMacro>--
--<OSS.NoHelperMacro M.Item.size>--
M DEFINITIONS AUTOMATIC TAGS ::= BEGIN

Size ::= SEQUENCE {
    length INTEGER,
    width  INTEGER,
```

```
    height INTEGER
}

Item ::= SEQUENCE {
    color      IA5String,
    size       Size
}

END
```

Generated helper macros:

```
typedef struct Size {
    int             length;
    int             width;
    int             height;
} Size;

…

typedef struct Item {
    char            *color;
    struct Size     *size;
} Item;

/* allocates memory for Item_PDU */
#define oss_Item_new_pdu(world) \
    (Item *)ossGetInitializedMemory(world, sizeof(Item))

/* gets "color" field value */
#define oss_Item_color_get(inp) \
    (inp)->color

/* sets "color" field value */
#define oss_Item_color_set(outp, color_) \
    (outp)->color = (color_)

/* allocates memory for a string of given length */
#define oss_Item_color_new(world, length_) \
    oss__CharStr_new(world, length_)

/* allocates memory and returns a copy of an input string */
#define oss_Item_color_copy(world, value_) \
    oss__CharStr_copy(world, value_)
```

Related directives:

OSS.HelperNames

4.4.2.25 OSS.HUGE

The OSS.HUGE directive is used to select a special representation for integers of a size not supported by the operating system in use. **By default,** INTEGER types are represented as a machine-dependent integer.

This directive has the following format:

```
--<OSS.HUGE [absoluteReference]>--
```

When specified globally, the *absoluteReference* operand should refer to a ASN.1 INTEGER type that is to be represented using the HugeInteger representation.

When specified locally, the OSS.HUGE directive takes no operands and affects the type that it is placed next to.

The HugeInteger representation without -helperNames consists of a structure containing a pointer to a character string and a length field specifying the size in octets of the stored integer:

```
    struct {
        unsigned short  length;
        unsigned char   *value;
} integerType;
```

The HugeInteger representation with -helperNames consists of a structure containing a pointer to a character string and a length field specifying the size in octets of the stored integer has “unsigned int” C type:

```
    struct _HugeInt {
        unsigned int    length;
        unsigned char   *value;
    } _HugeInt;
```

Notes:
1) If an INTEGER type with the HUGE directive is constrained to be non-negative, then a type value will be treated as an unsigned number rather than as a two's complement binary integer. In the BER/DER encoder, this ensures that the unsigned HUGE INTEGER values are encoded correctly by prepending an extra 0 octet if the high order bit of the value is 1.

2) Value constraints on an INTEGER type with the HUGE directive should not contain an integer constant which can not be represented as a 64bit integer. Such a value will cause the ASN.1 compiler to issue an error message. OSS believes this behavior is sufficient for most applications which require the use of HUGE INTEGERs. Nevertheless, OSS will provide full support for the constrained HUGE INTEGERs, if this is required by customers.

Example::

The following ASN.1 notation:

```
HugeInt DEFINITIONS ::=
BEGIN
    A ::= INTEGER --<HUGE>--
END
```

results in the following code in the header file:

```
The structure generated for A is the following:

   typedef struct A {
       unsigned short  length;
       unsigned char   *value;
   } A;
```

Related directive:
ASN1.HugeInteger (see section 4.4.1.3)

4.4.2.26 OSS.INCLUDES

The OSS.INCLUDES directive is used to specify the name of one or more configuration files (which contain global compiler directives) to be inserted at the point at which this directive is specified. This makes it possible for your organization to define different sets of compiler directives that can be included or excluded depending on the application. **By default,** no configuration files are inserted into the ASN.1 input.

This directive has the following format:

```
--<OSS.INCLUDES ["configFilename" ["configFilename"] …]>--
```

This directive can only be specified globally and must have at least one configuration filename as an operand. In addition, note that the configuration file specified may only contain global directives.

Note: If a directive in a configuration file conflicts with a directive which is either defined in another configuration file or is defined within the ASN.1 module, the last directive (i.e., scanning left to right, top to bottom) takes precedence. All directives that are defined within an ASN.1 source file or INCLUDES file take precedence over directives that are defined in the **asn1dflt** file (see section 5.1).

Example:

The following ASN.1 notation illustrates the use of the OSS.INCLUDES directive:

```
--<OSS.INCLUDES  "file1.cfg", "file2.cfg" >--

Sample1 DEFINITIONS EXPLICIT TAGS ::= BEGIN
    Name1 ::= INTEGER
    Name2 ::= BOOLEAN
END
```

The header file will not mention the OSS.INCLUDES directive, but it will be generated as if the directives in the configuration files were declared at the instance of the OSS.INCLUDES directive.

4.4.2.27 OSS.InfoCallback

The format is

```
--<OSS.InfoCallback [absoluteReference] ["]FunctionName["]>--
```

This directive is provided for those cases where an OSS.DataCallback directive is applied to a type which occurs in multiple locations in a message.

Example

For this definition

```
M DEFINITIONS AUTOMATIC TAGS ::= BEGIN

PackageInfo ::= SEQUENCE {
    from         Address,
    to           Address,
    posted       DATE
}

Address ::= SEQUENCE {
    zipcode      INTEGER( 0..99999 ),
    addressline  VisibleString( SIZE (1..64) )
}
END
```

a directive like this

```
--<OSS.DataCallback M.Address.zipcode "myZipcode">--
```

would cause `myZipcode()` to be called for both `PackageInfo.from.zipcode` and `PackageInfo.to.zipcode`, but `myZipcode()` wouldn't be able to differentiate the two fields. `OSS.InfoCallback` offers a way. By employing

```
--<OSS.InfoCallback M.PackageInfo.from "fromAddress">--
```

you instruct the compiler to generate a prototype for the `fromAddress()` callback function. This function, which you provide, is called by the decoder twice, once before starting the partial decoding of the `from` field and again after finishing its decoding. For details of the information provided to the infocallback function, see the OSS ASN.1/C API Runtime Reference Manual.

Related directives, options, and functions:

`OSS.DataCallback`, `-enablePartialDecode`, `-partialDecodeOnly`, `ossPartialDecode()` (See the OSS ASN.1 Runtime API Manual, section 3.5.48)

4.4.2.28 OSS.LENGTHSIZE

Compiler directives

The OSS.LENGTHSIZE directive is used for changing the default representation for the `count` and `length` fields of structures generated by the ASN.1 compiler. **By default** when no size constraints are specified, the `count` and `length` fields are either `short` (for the OBJECT IDENTIFIER type) or `int` (for all other types). When a size constraint is present these fields take on the smallest representation that can accommodate all of the required values.

Note that the OSS.LENGTHSIZE directive is not supported when the -helperNames option is specified. The `count` and `length` fields are always `unsigned int` for all types).

This directive has the following format:

```
--<OSS.LENGTHSIZE size>--
```

The OSS.LENGTHSIZE directive does not take an absolute reference but has a global effect. The operand, `size`, is mandatory. *`size`* should be substituted for one of the following literals: (1) `SHORT`, (2) `INT`, or (3) `LONG`. Note that the `OSS` prefix must be present when specifying this directive. If more than one OSS.LENGTHSIZE directive is found, the last one encountered is used.

Refer to the example below to see the effect of the OSS.LENGTHSIZE directive and its operands.

Examples:

The following ASN.1 notation will cause the Compiler to generate the default representation of the OCTET STRING:

```
Module DEFINITIONS ::= BEGIN
        A ::= OCTET STRING
END
```

The default representation is:

```
typedef struct A {
    unsigned int    length;
    unsigned char   *value;
} A;
```

However if the OSS.LENGTHSIZE directive is specified as follows:

```
--<OSS.LENGTHSIZE SHORT>--

Module DEFINITIONS ::= BEGIN
        A ::= OCTET STRING
END
```

The resultant header file contains:

```
typedef struct A {
    unsigned short  length;
    unsigned char   *value;
} A;
```

Notice the `short` data type in the second example instead of the `int` data type for the `length` field.

Notes:
1) The use of a size constraint overrides the LENGTHSIZE directive.
2) In some cases, the operand `LONG` will have the same effect as the operand `INT`.

4.4.2.29 OSS.LINKED

Specifies that a SET OF, SEQUENCE OF, or OBJECT IDENTIFIER should be represented as a singly-linked list of values. **By default,** the SET OF and SEQUENCE OF types are represented as a singly-linked list of values; and OBJECT IDENTIFIER types are represented as a structure with a character string pointer and a length field giving the size of the character string in bytes (i.e. the ENCODED representation). Refer to section 7.6 for more details about the default representation of these ASN.1 types.

Note that the OSS.LINKED directive is not supported when the -helperNames option is specified.

This directive has the following format:

```
--<OSS.LINKED [[absoluteReference] or [asn1Type [, asn1Type]|]]>--
```

When specified globally, the *absoluteReference* operand may be present and should refer to an ASN.1 SEQUENCE OF, SET OF, or OBJECT IDENTIFIER that is to be represented using the singly-linked list representation. The asn1Type argument (if present) sets the global the default representation for the specified ASN.1 built-in type (i.e. one of the above-mentioned allowed types). Both an absolute reference and an ASN.1 built-in type name may not be specified together in a single directive.

When specified locally, the OSS.LINKED directive takes no operands and only affects the type that it is place next to.

Example:

```
--<OSS.LINKED Module.DataCards>--
--<OSS.LINKED SET OF>--

Module DEFINITIONS ::= BEGIN
        DataCards ::= SEQUENCE OF INTEGER
        InfoCards ::= SET --<ARRAY>--  SIZE (18) OF IA5String
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct DataCards {
    struct DataCards *next;
    int             value;
} *DataCards;

typedef struct InfoCards {
    unsigned short  count;
    char            *value[18];
} InfoCards;
```

Note that the next field either points to the subsequent element in the list or is NULL.

Related directives:

OSS.ARRAY, OSS.DLINKED

4.4.2.30 OSS.NoConstrain

The OSS.NoConstrain directive identifies those types or components for which constraint checking should be disabled. The directive turns off all constraint checking for the type, including checking of the default permitted alphabet for character strings. For ENUMERATED types, it disables the subtype constraints but does not disable checking for valid enumerators. **By default,** constraint checking is turned on.

This directive has the following format:

```
--<OSS.NoConstrain [absoluteReference]>--
```

When specified globally, the *absoluteReference* operand should refer to an ASN.1 type that has a defined constraint.

The OSS.NoConstrain directive may not be specified locally.

Note:

1. The OSS.NoConstrain directive has no effect if the ASN.1 compiler command-line option `-constraint` is explicitly specified, except in the following cases:
   - the directive is applied to a CHARACTER STRING (unrestricted) or an EMBEDDED PDV type.
   - the directive is applied to a BIT STRING or OCTET STRING type constrained by contents constraint. In this case it disables automatic encoding/decoding of the contained type.
2. If the `-userConstraints` command-line option and the `OSS.NoConstrain` directive are both in use, all constraints except user-constraints are disabled for the specified type or field.

Example:

```
--<OSS.NoConstrain Module.S>--

Module DEFINITIONS ::= BEGIN
   S ::= INTEGER (1..10 | 300) (CONSTRAINED BY { })
END
```

For the above example if the `-constraint` option is not specified on the command line, all constraints for the type `S` are disabled. If the `-userConstraints` command-line option is specified, constraint checking of user-constraints will not be disabled.

Another Example:

```
--<OSS.NoConstrain Mod.ABConly>--
--<OSS.NoConstrain Mod.ShortString>--
--<OSS.PDU>--
```

```
Mod DEFINITIONS ::= BEGIN
   ABConly   ::= PrintableString (FROM ("A" | "B" | "C"))
   abc ABConly ::= "The wrong string"

   ShortString ::= PrintableString (SIZE(1 | 3 ))
   shStr ShortString ::= "The long string"

   ShortABCs ::= PrintableString (ABConly INTERSECTION ShortString)
   shABC ShortABCs ::= "ABCDD"
END
```

Note that the PrintableString `abc` will be passed at run-time without issuing a constraint violation error. Also note that although the OSS.NoConstrain directive is not explicitly applied to the `ShortABCs` type, constraint checking for this type will still be disabled because the type is composed of types marked with OSS.NoConstrain.

Example for the contents constraint:

```
--<OSS.NoConstrain ContConstr.O>--
ContConstr DEFINITIONS::= BEGIN
    O ::= OCTET STRING (CONTAINING INTEGER)
END
```

For the above example, the automatic encoding/decoding will not be performed at runtime even if the `-constraint` option is specified on the command line.

Related options and directives:

`-constraints / -noConstraints` (see section 3.3.14),
`-restrictedConstraintChecking` (see section 3.3.60),

4.4.2.31 OSS.NODEFAULTVALUE

The OSS.NODEFAULTVALUE directive informs the ASN.1 compiler to treat items marked with the ASN.1 DEFAULT tag to be treated as if they were marked with the OPTIONAL tag instead. **By default,** types marked with the DEFAULT tag take on their specified default value when absent from the encoding.

This directive has the following format:

```
--<OSS.NODEFAULTVALUE [absoluteReference]>--
```

When specified globally, the *absoluteReference* operand may be present and refers to an ASN.1 type marked with a DEFAULT tag. When specified globally without any operand, the OSS.NODEFAULTVALUE directive affects all types marked with the DEFAULT keyword.

When specified locally, the OSS.NODEFAULTVALUE directive takes no operands and only affects the type that it is placed next to.

Example:

The following ASN.1 notation shows a usage of the OSS.NODEFAULTVALUE directive:

```
--<OSS.NODEFAULTVALUE Module.DataCard.a>--

Module DEFINITIONS ::= BEGIN
        DataCard ::= SEQUENCE {
                a     [0]  BOOLEAN DEFAULT TRUE ,
                b     [1]  BOOLEAN DEFAULT TRUE,
                c     [2]  BOOLEAN
        }
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct DataCard {
    unsigned char   bit_mask;
#       define      a_present 0x80
#       define      b_present 0x40
    ossBoolean      a;  /* optional; set in bit_mask a_present if present
*/
    ossBoolean      b;  /* b_present not set in bit_mask implies value is
                         * TRUE */
    ossBoolean      c;
} DataCard;
```

Note how 'a' is represented as an OPTIONAL component because the OSS.NODEFAULTVALUE directive is applied to it. On the other hand, 'b' is still treated as a component with a default value.

4.4.2.32 OSS.NonUnionOpenType

The OSS.NonUnionOpenType directive forces generation of the predefined structure OpenType as the C representation for the ASN.1 open type, where the decoded value is represented as the untyped pointer.

By default, in the C representation for an ASN.1 open type the decoded value is represented as a union of PDU type alternatives (see 8.6.16).

This directive has the following format:

```
--<OSS.NonUnionOpenType absoluteReference>--
```

Example:

```
--<OSS.NonUnionOpenType Module.S.value>--

Module DEFINITIONS ::= BEGIN
    C ::= CLASS {
        &code INTEGER,
        &Type
    }

    Object C ::= {
        { &code 1, &Type INTEGER } |
        { &code 2, &Type UTF8String }
```

```
    }

    S ::= SEQUENCE {
        key C.&code ({Object}),
        value C.&Type ({Object}{@key})
    }
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct S {
    int             key;
    OpenType        value;
} S;
```

4.4.2.33 OSS.NULLTERM

The OSS.NULLTERM directive is used to indicate that an ASN.1 character string type, a GeneralizedTime type, a UTCTime type, or (starting with versions 8.1.3 and 8.2) a TIME type should be represented in the target language as a null-terminated variable length string.

By default, most string types (i.e. GeneralString, GraphicString, IA5String, NumericString, PrintableString, TeletexString, UTF8String, VideotexString, and VisibleString) and a TIME type (see 7.6.26) are already represented as null-terminated strings. The NULLTERM directive would have no effect on those types.

Other string types, multi-octet strings like BMPString and UniversalString, have a default representation of UNBOUNDED. Such strings typically contain many embedded NULLs (0x00), so NULLTERM makes little sense. For these, the NULLTERM directive is ignored.

The time types, GeneralizedTime and UTCTime, default to being represented as a structure of integers if the -helperNames option is not specified, and have a default representation of NULLTERM if the -helperNames is specified (see section 7.6.9 and 7.6.27). Using NULLTERM changes the representation to a string. Similarly, using TIMESTRUCT changes the representation to a structure of integers. See the examples below for details.

This directive has three forms. When specified globally, as below, the *absoluteReference* operand should refer to a specific ASN.1 character string or time type that you wish to represent as a null-terminated string.

```
--<OSS.NULLTERM [absoluteReference] >--
```

Alternatively, you may use the asn1Type argument to provide the name of an ASN.1 built-in character string type (other than CHARACTER STRING) whose default representation should be set to a null-terminated character string. This cannot be used for time types.

```
--<OSS.NULLTERM asn1Type [,asn1Type]…>--
```

When specified locally, the NULLTERM directive takes no operands and only affects the type that it is placed next to.

```
--<NULLTERM>--
```

.

Note: When NULLTERM is specified and a SizeConstraint subtype is not present, the POINTER directive is implicitly used.

Warning: When using the null-terminated representation, the user should ensure that the character string does not contain any embedded null characters. The presence of an embedded null character would cause the encoder to mis-encode the string. Likewise, the decoder would prematurely stop decoding such a string, since it would assume the embedded null character specifies the end of the string.

Examples:

The lines below show how to generate a null-terminated representation for all PrintableString types, not just for NameDefault. Note that the argument for OSS.NULLTERM is a built-in type, not a user-defined type.

```
Sample1 DEFINITIONS::=BEGIN
--<OSS.NULLTERM PrintableString>--
NameDefault ::= [1] PrintableString
```

The generated .h file will show

```
typedef char            *NameDefault;
```

The lines below show how to generate a null-terminated representation for a specific type, TimeNULL. TimeDefault remains at its default representation. Note that the argument for OSS.NULLTERM is a user-defined type.

```
--<OSS.NULLTERM Sample1.TimeNULL>--
TimeDefault ::= [3] GeneralizedTime
TimeNULL    ::= [4] GeneralizedTime
```

The generated .h file (without the -helperNames option) will show the following typedefs. Note how TimeDefault and TimeNULL are represented differently.

```
typedef GeneralizedTime TimeDefault;
typedef char            *TimeNULL;
```

The lines below show the first BMPString, called BMPDefault, taking the default. The second, BMPNull, also takes the default since the NULLTERM directive is ignored for BMPString and UniversalString types.

This example also shows how a NULLTERM directive can be used locally by placing it next to the type to be modified.

```
BMPDefault  ::= [5] BMPString
BMPNull     ::= [6] BMPString       --<NULLTERM>--
```

Note how the statements generated for BMPDefault and BMPNull are similar.

```
typedef struct BMPDefault {
    unsigned int    length;
    unsigned short  *value;
} BMPDefault;

typedef struct BMPNull {
    unsigned int    length;
    unsigned short  *value;
} BMPNull;
```

Related directives:
OSS.DLINKED (see section 4.4.2.10), OSS.LINKED (see section 4.4.2.27), OSS.PADDED (see section 4.4.2.34), OSS.UNBOUNDED (see section 4.4.2.49), OSS.VARYING (see section 4.4.2.55)

4.4.2.34 OSS.OBJHANDLE / OSS.NOCOPY

The OSS.OBJHANDLE directive (for which OSS.NOCOPY is a synonym) can be used on an OCTET STRING, BIT STRING, restricted character string, or a type with an ASN1.DeferDecoding directive already applied to specify special handling at run-time. This permits portions of a PDU to be encoded from a file or decoded to a file when the file memory manager is in use. Likewise when the default memory manager is in use, the OSS.OBJHANDLE (OSS.NOCOPY) directive indicates that the decoder should use a pointer to the encoded data during decoding instead of making its own copy of the encoded data. Note that the OSS.OBJHANDLE directive is permitted only on restricted character strings types with a single byte per character, BIT STRING, OCTET STRING, ANY, ANY DEFINED BY, and the open type. **By default,** there is no special handling of components at run-time.

These directives have the following format:

```
--<OSS.OBJHANDLE absoluteReference>--
--<OSS.NOCOPY absoluteReference>--
```

When specified globally, these directives can take an absolute reference referring to an allowed ASN.1 type. If specified globally without an absolute reference, these directives have no effect.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to.

Note:
1) When PER is in use during runtime, the `ossDecode()`, `ossFreePDU()`, and `ossCpyValue()` functions ignore applications of the OSS.OBJHANDLE directive. However when other encoding rules are in use (e.g., BER and DER), the OSS.OBJHANDLE directive is fully implemented; thus, `ossDecode()` and `ossCpyValue()` do not allocate regular memory for fields of a PDU marked with the OSS.OBJHANDLE directive; similarly, `ossFreePDU()` does not de-allocate memory for such fields.
2) The OSS.OBJHANDLE directive is not supported when OER is in use.

Example:

The following ASN.1 notation shows an example use of the OSS.OBJHANDLE directive:

```
--<OSS.OBJHANDLE Module.SpecialHandling>--

Module DEFINITIONS ::= BEGIN
        NoSpecialHandling ::= [1] IA5String
        SpecialHandling   ::= [2] IA5String
END
```

After compiling the above notation, the following is generated in the header file:

```
typedef char            *NoSpecialHandling;

typedef struct SpecialHandling {
    unsigned int    length;
    char            *value;
} *SpecialHandling;
```

Notice above how the IA5String `SpecialHandling` is represented with the UNBOUNDED representation (see section 4.4.2.49).

Note that the `.c` file will contain code to inform the encoder/decoder about the special handling of the type labeled `SpecialHandling`.

Related directives:
ASN1.DeferDecoding (see section 4.4.1.2)

Cross-reference: Refer to the *Using the default memory manager*, *Using the file memory manager*, and *Using the socket memory manager* sections in the *OSS memory managers* chapter of the **OSS ASN.1 API Reference Manual** for more information on the OSS.OBJHANDLE or OSS.NOCOPY directives.

4.4.2.35 OSS.ONECHAR

The OSS.ONECHAR directive specifies that the representation of a restricted character string should consist of a single character. **By default,** character strings can be an indefinite length, or a length specified by a subtype constraint.

Note that the OSS.ONECHAR directive is not supported when the `-helperNames` option is specified.

This directive has the following format:

```
--<OSS.ONECHAR [[absoluteReference] or [asn1Type [, asn1Type]…]]>--
```

Use this directive globally to make an absolute reference to any character string type (except CHARACTER STRING), when only one character in the string is wanted. Similarly, an ASN.1 built-in type name (`asn1Type`) can be specified, and `asn1Type` can name any allowed character string. Although more

than one `asn1type` may be present, both an absolute reference and a built-in type name cannot be specified.

When specified locally, this directive takes no arguments and affects only the type placed next to it.

Note: The character used when the ONECHAR directive is specified can be one byte, two bytes, or four bytes long.

This directive is useful mainly in generating initialized C values for special characters, such as those in the ASN1-CHARACTER-MODULE as defined in ISO/IEC 8824-1:1994.

Example:

The following ASN.1 notation shows the local and global use of the OSS.ONECHAR directive on various string types:

```
--<OSS.ONECHAR UniversalString>--
--<OSS.ONECHAR Module.SingleCharBMP>--

Module DEFINITIONS ::= BEGIN
        SingleCharUTF   ::= UTF8String          --<ONECHAR>--
        SingleCharNum   ::= NumericString       --<ONECHAR>--
        SingleCharPri   ::= PrintableString     --<ONECHAR>--
        SingleCharTel   ::= TeletexString       --<ONECHAR>--
        SingleCharVid   ::= VideotexString      --<ONECHAR>--
        SingleCharIA5   ::= IA5String           --<ONECHAR>--
        SingleCharGra   ::= GraphicString       --<ONECHAR>--
        SingleChraVis   ::= VisibleString       --<ONECHAR>--
        SingleCharGen   ::= GeneralString       --<ONECHAR>--
        SingleCharUni   ::= UniversalString
        SingleCharBMP   ::= BMPString
        RegularBMP      ::= BMPString
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef unsigned char   SingleCharUTF;

typedef unsigned char   SingleCharNum;

typedef unsigned char   SingleCharPri;

typedef unsigned char   SingleCharTel;

typedef unsigned char   SingleCharVid;

typedef unsigned char   SingleCharIA5;

typedef unsigned char   SingleCharGra;

typedef unsigned char   SingleChraVis;

typedef unsigned char   SingleCharGen;
```

```
typedef int             SingleCharUni;

typedef unsigned short  SingleCharBMP;

typedef struct RegularBMP {
    unsigned int    length;
    unsigned short  *value;
} RegularBMP;
```

4.4.2.36 OSS.PADDED

The OSS.PADDED directive indicates that character string, BIT STRING, and OCTET STRING types should be represented in the target language as fixed-length padded strings when the SizeConstraint subtype, and/or the NamedBitList for the BIT STRING type, is used. The character string types are blank-padded and BIT STRING types are zero-padded. In other words, all strings specified with the OSS.PADDED directive are the same size; short character strings and BIT STRINGs could have trailing spaces or zeros, respectively.

By default, character string types are represented as NULLTERM strings. BIT STRING and OCTET STRING types are represented as fixed length PADDED strings when the SizeConstraint subtype or NamedBitList is present or as an UNBOUNDED string when neither is present.

Note that the OSS.PADDED directive is not supported when the `-helperNames` option is specified.

This directive has the following format:

```
--<OSS.PADDED [[absoluteReference] or [asn1Type [, asn1Type]…]]>--
```

When used globally, this directive can take an absolute reference to any restricted character string type to which the PADDED representation is to be applied. Similarly, an ASN.1 built-in type name (`asn1Type`) can be specified. `asn1Type` can be the name of any allowed character string or be the literal "`BIT STRING`" or "`OCTET STRING`". Although more than one `asn1Type` may be present, both an absolute reference and a built-in type name cannot be specified.

When specified locally, this directive does not take any arguments and only affect the type it placed next to.

Notes:

1) When PADDED is specified, the SizeConstraint subtype (and/or NamedBitList for BIT STRING types) must also be present; otherwise, the compiler issues an error message. If the SizeConstraint is not of a single fixed size, the `-compat padded` command-line option must be specified (see section 3.5.62).
2) If the directive is used globally as --<OSS.PADDED OCTET STRING>--, then it will affect all OCTET STRING types constrained to a fixed size.

Example:

The following ASN.1 example shows an example use of the padded directive:

```
--<OSS.PADDED Module.PaddedStr>--

Module DEFINITIONS ::= BEGIN
        PaddedStr   ::= IA5String (SIZE (10))
        UnPaddedStr ::= IA5String (SIZE (10))
END
```

After passing the above syntax through the Compiler, the following is generated in the header file:

```
typedef char            PaddedStr[10];

typedef char            UnPaddedStr[11];
```

Notice how `UnPaddedStr` is one character larger than `PaddedStr`. This extra space is for the null terminated-character.

Related directive:
OSS.DLINKED (see section 4.4.2.10), OSS.LINKED (see section 4.4.2.27), OSS.NULLTERM (see section 4.4.2.31), OSS.UNBOUNDED (see section 4.4.2.49), OSS.VARYING (see section 4.4.2.55)

4.4.2.37 OSS.PDU / OSS.NOPDU

The OSS.PDU and OSS.NOPDU directives indicate whether or not an ASN.1 type can be sent or received as a protocol data unit. **By default**, all unreferenced types (defined types that are not referenced by any other type) are considered to be PDUs by the compiler and, as such, can be encoded/decoded as complete units.

These directives have the following format:

```
--<OSS.PDU [absoluteReference]>--
--<OSS.NOPDU [absoluteReference]>--
```

When specified globally, these directives can take an absolute reference specifying which component of the ASN.1 syntax is intended. If specified globally without an absolute reference, the OSS.PDU and OSS.NOPDU directives set a default for all types within their scope.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to.

Note: The OSS.PDU and ASN1.PDU directives only have effect on types contained in root modules. To treat all input modules as root modules, you have to either specify the `-root` compiler option (when compiling your ASN.1 syntax) or include the global OSS.ROOT directive in your ASN.1 specification.

Example:

The following ASN.1 notation demonstrates the local and global use of the OSS.PDU directive.

```
--<OSS.PDU Module.NameString>--

Module DEFINITIONS ::= BEGIN
        MiddleAgeClubCard ::= SEQUENCE {
                name            NameString,
                phoneNumber     PhoneString,
                age             AgeInt
        }
        NameString ::=  IA5String (SIZE (40))
        PhoneString ::= NumericString (SIZE (15)) --<PDU>--
        AgeInt ::= INTEGER (45..54)

END
```

After passing the above notation through the ASN.1 compiler, the following is generated:

```
#define          MiddleAgeClubCard_PDU 1
#define          NameString_PDU 2
#define          PhoneString_PDU 3

typedef char            NameString[41];

typedef char            PhoneString[16];

typedef unsigned short  AgeInt;

typedef struct MiddleAgeClubCard {
    NameString      name;
    PhoneString     phoneNumber;
    AgeInt          age;
} MiddleAgeClubCard;
```

The `#define`s shown above mark which types are PDUs. Notice how `AgeInt` is not marked to be a PDU, since it is referenced in the SEQUENCE `MiddleAgeClubCard` and it doesn't have a PDU directive applied to it.

Related directive:
ASN1.PDU (see section 4.4.1.5)

4.4.2.38 OSS.POINTER / OSS.NOPOINTER

The OSS.POINTER and OSS.NOPOINTER directives indicate whether memory should be allocated inline for an ASN.1 type (OSS.NOPOINTER) or only a pointer to the type should be created (OSS.POINTER) without inline allocation for the type. **By default,** all C `character string` data types without a size constraint use the pointer representation while other types use inline allocation.

Note that the OSS.NOPOINTER directive is not supported when the -helperNames option is specified. The OSS.POINTER directive is supported only for the following simple types: BOOLEAN, INTEGER without HUGE directive, NULL, and REAL.

These directives have the following format:

```
--<OSS.POINTER [absoluteReference]>--
--<OSS.NOPOINTER [absoluteReference]>--
```

When specified globally, these directives can take an absolute reference specifying which component of the ASN.1 syntax is intended. If specified globally without an absolute reference, the OSS.POINTER and OSS.NOPOINTER directives set a default for all types within their scope.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to.

Example:

The following ASN.1 notation shows the local and global use of these directives:

```
--<OSS.POINTER>--
--<OSS.NOPOINTER Module.InlineIntGlobal>--

Module DEFINITIONS ::= BEGIN

        InlineIntLocal      ::= [1] INTEGER --<NOPOINTER>--
        InlineIntGlobal     ::= [2] INTEGER
        PointeredIntGlobal  ::= [3] INTEGER
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef int             InlineIntLocal;

typedef int             InlineIntGlobal;

typedef int             *PointeredIntGlobal;
```

Notice how the local OSS.NOPOINTER directive overrides the global OSS.POINTER directive. Also, note the use of the absolute reference in the global OSS.NOPOINTER directive. The resulting types are either allocated inline or have a pointer created for them for later allocation.

4.4.2.39 OSS.Preserve

The OSS.Preserve directive was added in version 5.4.0 and is mainly for use with the split-headers mode of the ASN.1 compiler. This directive instructs the ASN.1 compiler to generate the representation of an ASN.1 type into the header file which normally would be ignored. Additionally, all types that are referenced by the type selected for preservation also have data structures generated for them. However, these types which are forced into the header file are still not considered PDUs.

This directive has the following format:

```
--<OSS.Preserve absoluteReference>--
```

When specified globally, the *absoluteReference* operand should refer to the ASN.1 type whose definition you want to appear in the produced header.

Note: This directive is useful in the split-headers mode (see the `-splitHeaders` and `-splitForSharing` command-line options) as it allows you to accumulate selected type definitions in a particular module header file.

The OSS.Preserve directive may not be specified locally.

Example:

```
--<OSS.Preserve Mod1.ShortString>--

Mod1 DEFINITIONS ::= BEGIN
   CompanyName   ::= ShortString
   ShortString ::= PrintableString
END

Mod2 DEFINITIONS ::= BEGIN
   Name ::= IA5String
   Age  ::= INTEGER
END
```

Normally if the above were compiled without the `-root` option, the definitions in the non-root module (i.e. `Mod1`) would not be generated into the header file. However since the OSS.Preserve directive is applied, the following is written to the compiler-generated header file:

```
#define          Name_PDU 1
#define          Age_PDU 2

typedef char            *ShortString;

typedef char            *Name;

typedef int             Age;
```

4.4.2.40 OSS.PrintFunctionName

The OSS.PrintFunctionName directive was introduced in version 5.1.0 and allows users to specify their own data formatter for printing values of primitive types. The function that users specify is called each time an OSS API function (e.g., `ossPrintPDU()` and `ossPrintPER()`) needs to output data for display. **By default,** a generic internal data format and display utility is used to print values of primitive types.

This directive has the following format:

```
--<OSS.PrintFunctionName absoluteReference "yourPrintFunctionName">--
```

This directive may only be specified globally and takes an absolute reference and a print-function name. *absoluteReference* specifies which primitive type of the ASN.1 syntax is to printed with the specified function. The specified function name *yourPrintFunctionName* should be enclosed in double quotes and have no parentheses (i.e. "()") attached at the end.

The print function specified must have the following prototype:

```
void DLL_ENTRY yourPrintFunctionName(struct ossGlobal *, char *);
```

If this directive is specified, the corresponding value will not be displayed by ossPrintPDU()'s internal printer, but will be converted to the *display format* and handed to *yourPrintFunctionName*() via its second parameter. *yourPrintFunctionName*() is then free to display this value in any manner desired.

The *display format* is a character string whose format varies according to the type whose value is to be printed. The following table describes the possible formats for the passed character string:

ASN.1 Primitive Type	Chacracter String Format
INTEGER or ENUMERATED	decimal number
OCTET STRING	'*xx...xx*'H, where *x* is a hexadecimal symbol
BIT STRING	'*xx...xx*'B, where *x* is a binary symbol
REAL	{<*mantissa*>,<*base*>,<*exponent*>}
restricted character string	quoted string
UTC/GeneralizedTime	quoted string
OID-IRI/RELATIVE-OID-IRI	quoted string

Note that the OSS ASN.1 Tools includes two commonly used replacement print functions: (1) ossPrintOctetAsIPAddress() and (2) ossPrintOctetAsASCII(). The definition for these functions is given below:

```
#include <ctype.h>
#include "asn1code.h"

#define ZERO '0'
#define NINE '9'
#define hex_value(x) (((x)>NINE)?((x)-7-ZERO):((x)-ZERO))

void DLL_ENTRY ossPrintOctetAsIPAddress(struct ossGlobal *world,
                                        char* display)
{
    unsigned    int  k, l;
    unsigned    char n;
    /* keep in mind 'xx...xx'H format, where x - hexadecimal sign */
    display++;
    l = strlen(display) - 2;
    for ( k = 0; k < (l >> 1); k++){
        n = (hex_value(display[k<<1]) << 4) +
            hex_value(display[(k<<1) + 1]);
        ossPrint(world, "%s%d", (k == 0)?"":".", n);
    }
}


void DLL_ENTRY ossPrintOctetAsASCII(struct ossGlobal *world, char* display)
```

```
{
    unsigned    int  k, l;
    unsigned    char n;
    /* keep in mind 'xx...xx'H format, where x - hexadecimal sign */
    display++;
    l = strlen(display) - 2;
    ossPrint(world, "\"");
    for ( k = 0; k < (l >> 1); k++){
        n = (hex_value(display[k<<1]) << 4)
            + hex_value(display[(k<<1) + 1]);
        if (isprint(n)) {
            ossPrint(world, "%c", n);
        } else {
            break;
        }
    }
    ossPrint(world, "\"");
}
```

Note: The substitute print function should not free the memory passed to it via the second parameter.

The example below illustrates the use of the OSS.PrintFunctionName directive and also the use these two functions:

Example:

Assume that we would like certain OCTET STRINGs to be displayed as plain ASCII and others as IP addresses, instead of being displayed in the default hexadecimal format. We can add two directives into ASN.1 specification as follows:

```
--<OSS.PrintFunctionName Module.Seq.name  "ossPrintOctetAsASCII">--
--<OSS.PrintFunctionName Module.Seq.ip    "ossPrintOctetAsIPAddress">--

Module DEFINITIONS ::= BEGIN
   Seq ::= SEQUENCE {
      name   OCTET STRING,
      ip     OCTET STRING
   }
END
```

The above directive applications will cause values of `Seq.name` to be displayed/printed as plain ASCII text while values of `Seq.ip` will be displayed/printed as IP addresses.

4.4.2.41 OSS.ROOT

The OSS.ROOT directive identifies the ASN.1 module(s) that are to be used as the root module(s). In a root module, all unreferenced types are considered to be PDUs. In addition, all non-root modules exist only to supply external references to the root modules. **By default,** the last file specified on the command-line is taken to contain the root module.

This directive has the following format:

```
--<OSS.ROOT [moduleName[{objectIdentifier}]
            [moduleName[{objectIdentifier}]]…]>--
```

The OSS.ROOT directive may only be used globally and can take one or more operands to identify the ASN.1 modules that are to be used as the root module(s). The operand may be a module reference (i.e., `moduleName`), or a module reference followed by a built-in `objectIdentifier` value. If no operands are provided, all input modules are considered root modules.

Example:

```
--<OSS.ROOT DirectorySystemProtocol{joint-iso-ccitt ds(5) modules(1)
--                                  dsp(12)}>--
```

Related ASN.1 compiler options:

`-root` (see section 3.3.61)

4.4.2.42 OSS.SampleCode / OSS.NoSampleCode

The `OSS.SampleCode` directive instructs the compiler to generate a sample application containing sample code for PDU types and valuereferences identified by this directive, providing more control over the sample code being created than the `–sampleCode` command-line option (see section 3.3.62). If there are several `OSS.SampleCode` directives in the ASN.1 input, the compiler creates an application containing sample code for all PDU types and valuereferences identified by these directives.

The directive accepts two types of syntax, the first allows you to select those PDU types and valuereferences that should be included in the sample code when generating the sample application, and the second allows you to control how the compiler automatically creates sample values for some ASN.1 types.

In the first case the directive has the following format:

```
--<OSS.SampleCode [pdus|values] [absoluteReference]>--
```

If the absolute reference is present, it should refer to a PDU type, an ASN.1 module or a valuereference for a PDU type; otherwise, the directive is ignored.

The `pdus` and `values` parameters are accepted only when the directive is specified globally and is applied to an ASN.1 module or to all ASN.1 input, i.e., the absolute reference is absent.

When the `OSS.SampleCode` directive is applied to all ASN.1 input, it has the same effect as the `–sampleCode` command-line option with the same parameter.

In the second case the directive has the following format:

```
--<OSS.SampleCode parameter [absoluteReference]>--
```

where the mandatory *parameter* can be either **count:***number* (the *number* is a non-negative integer number) or **selection**.

If *parameter* is set to **count:***number*, the directive should refer to the entire ASN.1 syntax, to an ASN.1 module or to a SEQUENCE OF or SET OF type. The directive specifies the default number of items to be used by the compiler when it automatically creates values for SEQUENCE OF/SET OF types for use in the sample code. If the OSS.SampleCode directive with the **count** parameter is not specified for a particular SEQUENCE OF/SET OF type, the compiler uses 1 as the default number of SEQUENCE OF/SET OF items.

If *parameter* is set to **selection**, the directive should refer to a CHOICE alternative. The directive specifies that the compiler selects the identified CHOICE alternative when it creates values for the corresponding CHOICE type for use in the sample code. If the OSS.SampleCode directive with the **selection** parameter is not specified for a particular CHOICE type, the compiler selects the first CHOICE alternative by default. If there are several such directives assigned to different alternatives of the same CHOICE type, the compiler creates as many sample values as needed to cover all the selected alternatives.

Note, the –noSampleCode command-line option overrides all OSS.SampleCode directives.

The OSS.NoSampleCode directive identifies those PDU types and valuereferences that will be excluded from testing in the sample application.

This directive has the following format:

```
--<OSS.NoSampleCode [absoluteReference]>--
```

If the absolute reference is present, it should refer to a PDU type, an ASN.1 module or a valuereference for a PDU.

Note, the –sampleCode command-line option overrides all OSS.NoSampleCode directives.

Examples:

The following example will cause the compiler to generate sample code for all valuereferences present in the ASN.1 syntax, except those defined in the Module2 module:

```
 --<OSS.ROOT>--
 --<OSS.SampleCode values>--
 --<OSS.NoSampleCode Module2>--
Module1 DEFINITIONS ::= BEGIN
        A ::= OCTET STRING
        value1 A ::= '00'H
END
Module2 DEFINITIONS ::= BEGIN
        B ::= INTEGER
        value2 B ::= 1
END
```

The following ASN.1 notation demonstrates using `OSS.SampleCode` to customize the generation of sample values:

```
--<OSS.SampleCode pdus Module>--
--<OSS.SampleCode selection Module.S.b>--
Module DEFINITIONS ::= BEGIN
    S ::= CHOICE {
       a IA5String,
       b SEQUENCE --<SampleCode count:3>-- OF INTEGER
    }
END
```

The compiler will create the sample application containing test code for the following value:

```
samplevalue1-S S ::= b : {
    0,
    0,
    0
}
```

Related options:

`-sampleCode/-noSampleCode` (see section 3.3.62)

4.4.2.43 OSS.SelfCompleteWildcard

The OSS.SelfCompleteWildcard directive may be applied to a type with a final XER ANY-ELEMENT encoding instruction. It may also be applied to an ASN.1 module (in this case all types with a final ANY-ELEMENT instruction textually defined within this module are affected) and it may be applied globally to affect all such types in all modules in the input syntax.

The directive instructs the compiler to mark the types it affects by a special flag in the control table. When this flag is found at runtime, the E-XER decoder attempts to preserve the wildcard body in the decoded value exactly as it was present in the original XML document. When decoding from plain memory with the SOED, and in all cases of LED usage, the wildcard contents are directly copied to the output C value. When decoding from files and/or sockets with the SOED, the decoder may reconstruct the outermost tag of a wildcard, performing whitespace normalization within it and declaring all namespaces before all attributes. In either case, the following will hold:

- namespaces declared in outer XML elements won't be redeclared in the decoded wildcard value
- the original order of namespaces and attributes will be preserved if the namespaces were declared before the attributes

In particular, if the wildcard contents were normalized according to the W3C Canonical XML specification, the decoded wildcard value will literally match the original XML value. This makes it possible to validate the digital signature of a wildcard after decoding. The directive is also useful to speed up E-XER decoding of wildcards when it is known in advance that they are self-complete and do not use namespace or entity declarations from the outer XML document.

Example:

```
--<OSS.SelfCompleteWildcard M.Root.w>--
M DEFINITIONS XER INSTRUCTIONS ::= BEGIN

Root ::= SEQUENCE {
    w [ANY-ELEMENT] UTF8String (CONSTRAINED BY {}) }

ENCODING-CONTROL XER
    GLOBAL-DEFAULTS MODIFIED-ENCODINGS
    NAMESPACE ALL AS "urn:Root" PREFIX "r"

END
```

For the following XML document:

```
<Root xmlns="urn:Root">
  <n1:wildcard xmlns:n1="urn:n1" xmlns:n2="urn:n2" n1:at1="1" n2:at2="2"/>
</Root>
```

where 'n1:wildcard' is an element wildcard, the decoder produces:

```
<n1:wildcard xmlns:n2="urn:n2" xmlns:n1="urn:n1" xmlns="urn:Root"
             n2:at2="2" n1:at1="1"/>
```

which is the wildcard value by default and

```
<n1:wildcard xmlns:n1="urn:n1" xmlns:n2="urn:n2" n1:at1="1" n2:at2="2"/>
```

which is literally the original, with the OSS.SelfCompleteWildcard directive.

Related directives:
The following directive also affects E-XER decoder behavior:
`OSS.ExtensibleUseType` (see section 4.4.2.17)

4.4.2.44 OSS.SHORT / OSS.INT / OSS.LONG / OSS.LONGLONG

The OSS.SHORT, OSS.INT, OSS.LONG, and OSS.LONGLONG directives determine the representation of the ASN.1 INTEGER type. On a typical system, OSS.SHORT calls for a 16-bit integer, OSS.INT for a 32-bit one (or a 16-bit one on systems such as MS DOS and MS Windows 3.1), OSS.LONG for a 32-bit one, and OSS.LONGLONG for a 64-bit one (Note: OSS.LONGLONG has the same meaning as OSS.LONG on platforms that do not support 8-byte integers). **By default** if no subtype constraints are present, the OSS.INT directive is implied. If a subtype constraint is present, the Compiler chooses the smallest representation able to carry all of the required values.

These directives have the following format:

```
--<OSS.SHORT [absoluteReference]>--
--<OSS.INT  [absoluteReference]>--
--<OSS.LONG [absoluteReference]>--
--<OSS.LONGLONG [absoluteReference]>--
```

When specified globally, these directives can take an absolute reference referring to an ASN.1 INTEGER type in the input. If specified globally without an absolute reference, the OSS.SHORT, OSS.INT, OSS.LONG, and OSS.LONGLONG directives set a default for all INTEGER types within their scope. Note that only INTEGER types which do not have subtype constraints are affected by the global use of these directives.

When specified locally, these directives do not take any arguments and only affect the type they are placed next to. Note that when used locally, these directives override any choices that the ASN.1 compiler would have made based upon a subtype constraint.

Note: Unlike outdated versions of the OSS ASN.1 compiler, the global use of the OSS.SHORT, OSS.INT and OSS.LONG directives no longer affects the representation of the `length` and `count` fields in structures generated by the Compiler. One may now use the OSS.LENGTHSIZE directive to modify the representation of these fields.

Example:

The following ASN.1 notation illustrates the global and local use of the OSS.SHORT, OSS.INT, OSS.LONG, and OSS.LONGLONG directives:

```
--<OSS.SHORT>--
--<OSS.INT Module.GlobalINTInt>--

Module DEFINITIONS ::= BEGIN
        GlobalDefaultInt ::= [1] INTEGER
        GlobalINTInt     ::= [2] INTEGER
        LONGInt          ::= [3] INTEGER --<LONG>--
        LONGLONGInt      ::= [4] INTEGER --<LONGLONG>--
END
```

In the above syntax, `--<OSS.SHORT>--` sets the global default representation for INTEGER types. `--<OSS.INT Module.GlobalINTInt>--` affects only the INTEGER with a tag of `[2]` in the module above. The local `--<LONG>--` and `--<LONGLONG>--` directives override the default representation for the types that they are placed next to.

After passing the above notation through the Compiler, the following is generated in the header file:

```
typedef short           GlobalDefaultInt;

typedef int             GlobalINTInt;

typedef long            LONGInt;

typedef LONG_LONG       LONGLONGInt;
```

Note that `LONG_LONG` is defined in the file `asn1hdr.h` as the C compiler-specific representation of a 64-bit integer. Note again that if your platform does not support 64-bit integers, the OSS.LONG and OSS.LONGLONG directives have the same effect.

4.4.2.45 OSS.SHORTENNAMES

The OSS.SHORTENNAMES directive instructs the compiler to omit certain letters from long names to make them less than 31 characters in length in the header file. This directive accommodates maximum name length requirements that exist on some C compilers. Note that the OSS ASN.1 compiler insures that all shortened names are unique (adding _# suffixes if necessary). **By default,** name-shortening is turned off.

This directive has the following format:

```
--<OSS.SHORTENNAMES [shortenToLength]>--
```

The OSS.SHORTENNAMES directive may only be used globally and takes one optional operand (*shortenToLength*) which specifies the exact integer length (16 or greater) to which long variable names should be shortened (e.g., `--<OSS.SHORTENNAMES 27>---`).

If no operands are specified, all names in the ASN.1 input that are longer than 31 characters in length are shortened to 31 characters.

Example:

The following ASN.1 notation shows an example use of the OSS.SHORTENNAMES directive:

```
--<OSS.SHORTENNAMES>--

Mod DEFINITIONS ::= BEGIN
   WeWereTryingToThinkOfaNameButWeCouldNotFindaSuitableOne  ::= INTEGER
   NormalName                                               ::= BOOLEAN
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef int             WWTTTONBWCNFSOn;

typedef ossBoolean      NormalName;
```

Notice how the name that is longer than 31 characters is renamed in the header file while the name that is shorter than 31 characters is left as is.

Related ASN.1 compiler options:

`-shortenNames` (see section 3.3.64)

4.4.2.46 OSS.Stylesheet

The OSS.Stylesheet directive was introduced in version 6.0 and gives you greater control over the default XML stylesheets which are produced by the ASN.1 compiler. Using this directive, you can instruct the ASN.1 compiler to generate a separate XML stylesheet with your own desired filename for any particular PDU. **By default**, no stylesheets are produced by the ASN.1 compiler unless the `-xsl` option is specified. When the `-xsl` option is specified, a stylesheet for each PDU in the input is generated into a

separate file whose name is derived from that of the PDU. (Note that stylesheets can be used to make the XER output of the `ossEncode()` function more visually appealing when viewed in a web browser.)

This directive has the following format:

`--<OSS.Stylesheet` [*`<absoluteReference>`* *`"<OptionalNewName>.xsl"`*]**`>--`**

`absoluteReference` refers to any PDU in the input ASN.1 syntax that you wish to have a separate XML stylesheet generated for. *`OptionalNewName`* is an optional/non-mandatory argument for this directive that allows you to specify a new full pathname for the stylesheet which will be generated.

When an `absoluteReference` is not specified, it is assumed that a stylesheet is to be produced for each and every PDU in the input syntax. This form of the OSS.Stylesheet directive (i.e. a global directive application) is identical in behavior to the `-xsl` ASN.1 compiler option.

Although the OSS.Stylesheet directive can appear anywhere white-space is allowed, the OSS prefix must be present.

Note: You can use the `ossSetXmlStylesheet()` API function to have the `ossEncode()` function generate a reference to your desired stylesheet in the ensuing XER encoding produced. Refer to the **OSS ASN.1 API/C Reference Manual** for more details.

Example:

```
--<OSS.Stylesheet XModule.StringType>--
--<OSS.Stylesheet XModule.IntType "AnotherNameForInt.xsl">--

XModule DEFINITIONS ::= BEGIN
   StringType ::= UTF8String
   IntType  ::= INTEGER

   studentName ::= <StringType>John H. Doe</StringType>
   studentAge ::= <IntType>23</IntType>
END
```

For the above syntax, two separate stylesheet files (one named `StringType.xsl` and the other named `AnotherNameForInt.xsl`) will be produced containing browser default style information for their respective PDUs.

Related ASN.1 compiler options:

`-xsl` (see section 3.3.80)

4.4.2.47 OSS.SUPPRESS

The OSS.SUPPRESS directive instructs the compiler not to print specific informatory and/or warning messages. OSS.SUPPRESS proves especially useful when compiling abstract syntaxes that are known to

contain specifications which are not fully compliant with the ASN.1 standard. **By default,** informatory and warning messages are not suppressed.

This directive has the following format:

```
--<OSS.SUPPRESS messageID[, messageID]…>--
```

The OSS.SUPPRESS directive may only be used globally and takes one or more operands (`messageID`) specifying which message needs to be suppressed. This `messageID` can be either the full message identifier (e.g., **A0178W**) or just the numeric part of the message number (e.g., **178**).

Example:

The following ASN.1 notation shows an example using the OSS.SUPPRESS directive. Message **A0178W** warns that the first digit of a number in a module definition should be non-zero.

```
--<OSS.SUPPRESS A0178W>--

Sample DEFINITIONS ::= BEGIN
   Age ::= INTEGER
   legalAge Age ::= 021
END
```

Note that the OSS.SUPPRESS directive does not affect the C-representation. Its only effect is that the following warning message is *not* generated for the above syntax:

```
"filename.asn", line 4 (Sample): A0178W: The first digit should not be
zero unless the number is a single digit.
        legalAge Age ::= 021
                         ^
```

Related Command-Line Options

`-ignoreSuppress` (see section 3.3.41), `-informatoryMessages` (see section 3.3.42), `-suppress` (see section 3.3.69), `-warning` (see section 3.3.79)

4.4.2.48 OSS.TIMESTRUCT

The OSS.TIMESTRUCT directive is used to indicate that a GeneralizedTime or UTCTime type should be represented in the target language as a structure of integers.

By default, GeneralizedTime and UTCTime are represented as a structure of integers if the -helperNames option is not specified, and they have a null-terminated default representation if -helperNames is specified (see section 7.6.9 and 7.6.27). Using the NULLTERM changes the representation to a string. Similarly, using TIMESTRUCT changes the representation to a structure of integers.

This directive has three forms. When specified globally, the *`absoluteReference`* operand should refer to a specific ASN.1 GeneralizedTime or UTCTime type that you want to represent as a structure.

```
--<OSS.TIMESTRUCT absoluteReference>--
```

Alternatively, you can use the *asn1Type* argument to provide the name of an ASN.1 built-in type (GeneralizedTime or UTCTime) whose default representation should be set to the structure. If the *asn1Type* argument is absent, both the GeneralizedTime and UTCTime types will be represented as a structure.

```
--<OSS.TIMESTRUCT [asn1Type]>--
```

When specified locally, the TIMESTRUCT directive takes no operands and affects only the type that it is placed next to.

```
--<TIMESTRUCT>--
```

Examples:

```
--<OSS.TIMESTRUCT Mod.Utc>--
 --<OSS.TIMESTRUCT Mod.Gen>--
Utc ::= UTCTime
Gen ::= GeneralizedTime
```

The generated .h file without -lean or -helperNames will show:

```
typedef UTCTime         Utc;
typedef GeneralizedTime Gen;
```

The generated .h file with -lean or -helperNames will show:

```
typedef char      *Utc;
typedef char      *Gen;
```

Related directives:
OSS.NULLTERM (see section 4.4.2.31)

4.4.2.49 OSS.Truncate

The OSS.Truncate directive was added in version 5.3.0 and allows you to speed up the decoding of certain large PDUs by skipping extra trailing elements in a SET OF or SEQUENCE OF type.

This directive has the following format:

```
--<OSS.Truncate absoluteReference maxLimit>--
```

When specified globally, the `absoluteReference` operand should refer to the SET OF or SEQUENCE OF type for which all elements with an index greater than `maxLimit` should be ignored. `maxLimit` should be an integer number between 1 and 16384.

The OSS.Truncate directive cannot be specified locally.

Example:

The following ASN.1 notation shows an example use of the OSS.Truncate directive:

```
--<OSS.Truncate Mod.Records 3000>--

Mod DEFINITIONS ::= BEGIN
  Records  ::= SEQUENCE OF Employee

  Employee ::= SEQUENCE {
     name IA5String,
     position IA5String,
     salary REAL
  }
END
```

By using the control-table/code-file produced by the above specification, the decoder will only process 3000 elements of type `Records`. This type of truncation is useful when you are sure that the information you need will not be at the tail end of the list of elements or you are sure that the desired information will not be contained in a large list of elements.

4.4.2.50 OSS.TypeIndex

This is an internal directive that is generated into the `.spl` file when the `-splitHeaders` or `-splitForSharing` option is in use (see sections 3.3.67 and 3.3.68).

4.4.2.51 OSS.UNBOUNDED

The OSS.UNBOUNDED directive specifies that the type is to be represented as a length/pointer pair for character string types, the BIT STRING type, and the OCTET STRING type, and as a count/pointer pair for the SETOF and SEQUENCE OF types. **By default:**

Most character string types (i.e. GeneralString, GraphicString, IA5String, NumericString, PrintableString, TeletexString, UTF8String, VideotexString, and VisibleString) are represented as null-terminated strings.

For the BIT STRING type, the default representation is the PADDED representation when the SizeConstraint subtype is specified and/or a NamedBitList is present, and the OSS.UNBOUNDED representation when neither the SizeConstraint subtype nor a NamedBitList is used.

For the OCTET STRING type, the default representation is the VARYING representation when the SizeConstraint subtype is specified; otherwise, the OSS.UNBOUNDED representation is used.

For SET OF and SEQUENCE OF types, the default is the LINKED representation.

This directive has the following format:

```
--<OSS.UNBOUNDED [[absoluteReference] or [asn1Type [,asn1Type]…]]>--
```

When specified globally, the *`absoluteReference`* operand should refer to an ASN.1 character string, BIT STRING, OCTET STRING, SEQUENCE OF, or SET OF type that the user wants to represent using the OSS.UNBOUNDED representation. Alternatively, the user may use the `asn1Type` argument to

provide the name of an ASN.1 built-in type whose default should be set to the UNBOUNDED representation. When specified globally, this directive must take either an absolute reference or one or more ASN.1 built-in type names. However, it cannot take both.

When specified locally, this directive takes no operands and only affects the type that it is placed next to.

Note that the `count` field is always `unsigned int` when the -helperNames option is specified.

Note: If a SizeConstraint subtype was specified (so that the maximum size or number of occurrences of a type is known), the `count` or `length` field of the UNBOUNDED representation is either `short`, `int` or `long`, depending on which is the smallest needed to fit the largest value. In the absence of the SizeConstraint subtype the `count` or `length` field is always `int`.

Example:

The following code excerpt shows the local and global use of the OSS.UNBOUNDED directive:

```
--<OSS.UNBOUNDED Module.SeqOfIntU>--
--<OSS.UNBOUNDED IA5String>--

Module DEFINITIONS ::= BEGIN
        SeqOfInt        ::= [1] SEQUENCE OF INTEGER
        SeqOfIntU       ::= [2] SEQUENCE OF INTEGER
        Ia5Str          ::= [3] IA5String --<NULLTERM>--
        Ia5StrU         ::= [4] IA5String
        UTF8Str         ::= [5] UTF8String
        UTF8StrU        ::= [6] UTF8String --<UNBOUNDED>--
        OctetStrU       ::= [7] OCTET STRING
END
```

Note that the local `--<NULLTERM>--` directive overrides the global default set by `--<OSS.UNBOUNDED IA5String>--`. Also note that the OCTET STRING type's default representation is UNBOUNDED.

After passing the above notation through the Compiler, the following is generated in the header file:

```
typedef struct SeqOfInt {
    struct SeqOfInt *next;
    int             value;
} *SeqOfInt;

typedef struct SeqOfIntU {
    unsigned int    count;
    int             *value;
} SeqOfIntU;

typedef char            *Ia5Str;

typedef struct Ia5StrU {
```

```
    unsigned int    length;
    char            *value;
} Ia5StrU;

typedef char            *UTF8Str;

typedef struct UTF8StrU {
    unsigned int    length;
    char            *value;
} UTF8StrU;

typedef struct OctetStrU {
    unsigned int    length;
    unsigned char   *value;
} OctetStrU;
```

Related directives:
OSS.ARRAY (see section 4.4.2.1), OSS.DLINKED (see section 4.4.2.10), OSS.LINKED (see section 4.4.2.27), OSS.NULLTERM (see section 4.4.2.31), OSS.PADDED (see section 4.4.2.34), OSS.VARYING (see section 4.4.2.55)

4.4.2.52 OSS.UNIVERSALSTRING

The UNIVERSALSTRING directive is used to select the representation of the ASN.1 UTF8String type in C as an unbounded array of four-byte characters (UCS-4). This representation is the same as for the ASN.1 UniversalString type. For more information on the UTF8String type refer to **ITU-T Rec. X.680 (2008), Annex H**. **By default,** the UTF8String is represented as a NULL terminating character string.

This directive has the following format:

```
--<OSS.UNIVERSALSTRING [absoluteReference]>--
```

When specified globally, the OSS.UNIVERSALSTRING directive takes an *absoluteReference* referring to some UTF8String in the ASN.1 syntax.

When specified locally, this directive takes no operands and only affects the type that it is placed next to.

Example:

The following ASN.1 notation shows the local and global use of this directive:

```
--<OSS.UNIVERSALSTRING Module.FourByteGlobal>--

Module DEFINITIONS ::= BEGIN

        Nullterm        ::= [1] UTF8String
        TwoByteLocal    ::= [2] UTF8String --<BMPSTRING>--
        FourByteGlobal  ::= [3] UTF8String
        FourByteLocal   ::= [4] UTF8String --<UNIVERSALSTRING>--

END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef char            *Nullterm;

typedef struct TwoByteLocal {
    unsigned int    length;
    unsigned short  *value;
} TwoByteLocal;

typedef struct FourByteGlobal {
    unsigned int    length;
    int             *value;
} FourByteGlobal;

typedef struct FourByteLocal {
    unsigned int    length;
    int             *value;
} FourByteLocal;
```

Notice that `unsigned short *value` is used for the UTF8String when the OSS.BMPSTRING directive is specified and `int *value` is used when the OSS.UNIVERSALSTRING directive is specified.

Note that the `length` fields in `TwoByteLocal`, `FourByteGlobal`, and `FourByteLocal` specify the number of characters in the string and not the number of bytes.

Related directives:
OSS.BMPSTRING (see section 4.4.2.3)

4.4.2.53 OSS.USERFIELD

The OSS.USERFIELD directive allows the generation of a locally defined user field that is of significance only to a particular application. User fields may only appear in ASN.1 SEQUENCE or SET types and must be either marked as OPTIONAL or DEFAULT. These fields facilitate the handling of compiler-generated C data structures. The OSS.USERFIELD directive instructs the encoder to skip the field and the decoder to perceive an error upon receipt of a value which appears in the same context and has the same tag as the user field. **By default,** no user field is generated.

This directive has the following format:

`--<`**`OSS.USERFIELD`** `[`*`absoluteReference`*`]>--`

When specified globally, the OSS.USERFIELD directive takes an *`absoluteReference`* referring to a field in an ASN.1 SET or SEQUENCE type.

Note: The compiler subjects user fields to the same ASN.1 restrictions as non-user fields. For example, an OPTIONAL element of a SET or SEQUENCE must have a tag that is different from the element which follows it, regardless of whether either of these elements is a user field.

When specified locally, this directive takes no operands and only affects the type that it is placed next to.

Example:

```
--<OSS.USERFIELD Module.DataCard.controlGroup>--

Module DEFINITIONS ::= BEGIN
        DataCard ::= SEQUENCE {
                controlGroup    BOOLEAN DEFAULT FALSE ,
                name            PrintableString,
                numberOfCars    INTEGER (1..5)
        }
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct DataCard {
    unsigned char   bit_mask;
#       define      controlGroup_present 0x80
    ossBoolean      controlGroup;  /* user field */
    char            *name;
    unsigned short  numberOfCars;
} DataCard;
```

The encoder and decoder will behave as if the field `controlGroup` above were not present.

Note that some user fields declared as OPTIONAL may not have bit masks generated for them.

4.4.2.54 OSS.UseThis

The OSS.UseThis directive was introduced in version 5.1.0 and permits you to reduce the ASN.1-compiler-generated header file's size by eliminating semi-duplicate C `typedef`s. Such elimination of similar structures is particularly useful for multiple instances of parameterized and constrained types which do not satisfy all the requirements for typesharing (see glossary definition).

This directive has the following format:

```
--<OSS.UseThis absoluteReference1 absoluteReference2>--
```

The OSS.UseThis directive can only be specified globally and takes two mandatory absolute reference arguments. *`absoluteReference1`* refers to the component of the ASN.1 specification which is to be substituted and *`absoluteReference2`* refers to the component which is to be used in the substitution.

Notes:
1) The time-optimized encoder/decoder code file will still have different encoder/decoder routines produced for the substituted and substituting types.
2) The OSS.UseThis directive cannot be used to substitute types with non-similar representations.

Example:

```
--<OSS.UseThis Module.S1.a Module.S2>--

Module DEFINITIONS ::= BEGIN
   S1 ::= SET {
      a SET OF INTEGER
   }
   S2 ::= SET OF INTEGER
END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct S2 {
   struct S2       *next;
   int             value;
} *S2;

typedef struct S1 {
   struct S2       *a;
} S1;
```

If the OSS.UseThis directive were not specified, the following would be generated in the header file:

```
typedef struct _setof1 {
    struct _setof1  *next;
    int             value;
} *_setof1;

typedef struct S1 {
    struct _setof1  *a;
} S1;

typedef struct S2 {
    struct S2       *next;
    int             value;
} *S2;
```

The added efficiency introduced with the application of the OSS.UseThis directive is obvious in the above example.

Note that the current implementation of OSS.UseThis directive only permits substitution of ASN.1 types (not fields). So, the following directive would be invalid for the above example:

```
--<OSS.UseThis MOD.S2 MOD.S1.a>--
```

4.4.2.55 OSS.VALNAME / OSS.FIELDNAME / OSS.TYPENAME

The OSS.VALNAME, OSS.FIELDNAME and OSS.TYPENAME directives instruct the ASN.1 compiler to generate specific names for the `value` variable, a field in a structure, or a user-defined type. **By default,** the value field has a name of `value` while elements in a structure and user-defined types have names derived from their ASN.1 notation.

These directives have the following format:

```
--<OSS.VALNAME   [absoluteReference] [["]newName["]]>--
--<OSS.FIELDNAME [absoluteReference] [["]newName["]]>--
--<OSS.TYPENAME  [absoluteReference] [["]newName["]>--
```

The OSS.VALNAME directive is used to explicitly specify the variable name that should be generated into the header file in those places where the compiler would otherwise generate the name `value`.

The OSS.FIELDNAME directive is used to explicitly specify the field name that should be generated into the header file in those places where the compiler would otherwise have generated a name derived from the ASN.1 identifier of the corresponding SET, SEQUENCE or CHOICE component.

The OSS.TYPENAME directive is used to explicitly specify a name for a user-defined type that should be generated into the header file in those places where the compiler would otherwise have generated a name derived from the ASN.1 input.

When specified globally, these directives should have an absolute reference referring to the ASN.1 component that is to be renamed in the generated header file. Then, a new name should be provided either in double-quotes or without double quotes.

When specified locally, these directives do not take an absolute reference, but a new name must be provided. Note that when used locally, the new name provided should be in double quotes

In both the local and global case, the new name provided must be a valid C variable name.

Note: The effect of these directives is absolute, meaning that the ASN.1 compiler will not make any attempt to change them, even if they are invalid. If the name that you specify conflicts with another one that the Compiler derived from an ASN.1 identifier or type reference, the Compiler will alter that other name.

Example:

The following notation shows the local and global use of these directives:

```
--<OSS.VALNAME Module.RenamedInt integerValue>--
--<OSS.TYPENAME Module.RenamedStr "asciiStr">--

Module DEFINITIONS ::= BEGIN
    DefaultName         ::= [1] SET OF INTEGER
    RenamedInt          ::= [2] SET OF INTEGER

    DefaultStruct       ::= [3] SEQUENCE {
        lunchPrepared   BOOLEAN,
```

```
        cost            REAL
    }
    RenamedStruct       ::= [4] SEQUENCE {
        lunchPrepared   BOOLEAN --<FIELDNAME "lunchReady">--,
        cost            REAL
    }

    DefaultNameStr     ::= [5] IA5String
    RenamedStr         ::= [6] IA5String

END
```

After passing the above notation through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct DefaultName {
    struct DefaultName *next;
    int             value;
} *DefaultName;

typedef struct RenamedInt {
    struct RenamedInt *next;
    int             integerValue;
} *RenamedInt;

typedef struct DefaultStruct {
    ossBoolean      lunchPrepared;
    double          cost;
} DefaultStruct;

typedef struct RenamedStruct {
    ossBoolean      lunchReady;
    double          cost;
} RenamedStruct;

typedef char            *DefaultNameStr;

typedef char            *asciiStr;
```

The renamed components are underlined in the above C excerpt.

Related directive:
ASN1.Nickname (see section 4.4.1.4)

4.4.2.56 OSS.VALUE / OSS.NOVALUE

The OSS.VALUE and OSS.NOVALUE directives specify whether or not for each ASN.1 value reference a corresponding C initialized variable should be generated in the `.c` file and an associated `extern` statement in the header file. **By default,** the OSS.VALUE directive is implied.

These directives have the following format:

```
--<OSS.VALUE   absoluteReference>--
--<OSS.NOVALUE absoluteReference>--
```

When specified globally, these directives can take an absolute reference to an ASN.1 user-defined type that is used in a value reference. When specified globally without arguments, these directives set a default for all value references within their scope.

When specified locally, these directives also do not take any arguments and only affect the value reference they are placed next to.

Example:

The following ASN.1 notation demonstrates the local and global use of the OSS.VALUE and OSS.NOVALUE directives (Note that local OSS.VALUE directive overrides the global OSS.NOVALUE one):

```
--<OSS.NOVALUE Module.IdInt>--

Module DEFINITIONS ::= BEGIN
        ASCIIStr                        ::= IA5String
        name ASCIIStr                   ::= "John"
        age INTEGER --<SHORT>--         ::= 57
        IdInt                           ::= INTEGER
        idNumAbsent IdInt               ::= 767543
        idNumPresent IdInt --<VALUE>--  ::= 354876
END
```

After passing the above syntax through the Compiler, the following is generated in the header file:

```
typedef char            *ASCIIStr;

typedef int             IdInt;

extern ASCIIStr name;

extern short age;

extern int idNumPresent;
```

In addition, the following initializations are made in the `.c` file.

```
ASCIIStr name = "John";

const short age = 57;

const int idNumPresent = 354876;
```

Notice how there is no `extern` variable nor any initialization for `idNumAbsent`.

4.4.2.57 OSS.VARYING

The OSS.VARYING directive specifies that character string types with a SizeConstraint subtype, BIT STRING types with a NamedBitList, or OCTET STRING types are to be represented as length-prefixed variable-length strings. **By default:**

Most character string types (i.e. GeneralString, GraphicString, IA5String, NumericString, PrintableString, TeletexString, UTF8String, VideotexString, and VisibleString) are represented as null-terminated strings.

For the BIT STRING type, the default representation is the PADDED representation when the SizeConstraint subtype is specified and/or a NamedBitList is present, and the UNBOUNDED representation when neither the SizeConstraint subtype nor a NamedBitList is used.

For the OCTET STRING type, the default representation is the VARYING representation when the SizeConstraint subtype is specified; otherwise, the UNBOUNDED representation is used.

Note that the OSS.VARYING directive is not supported when the -helperNames option is specified.

This directive has the following format:

```
--<OSS.VARYING [[absoluteReference] or [asn1Type [,asn1Type]…]]>--
```

When specified globally, the *absoluteReference* operand should refer to an ASN.1 character string, BIT STRING, or OCTET STRING type that the user wants to represent using the VARYING representation. Alternatively, the user may use the `asn1Type` argument to provide the name of an ASN.1 built-in type whose default representation should be set to a the VARYING representation. When specified globally, this directive must take either an absolute reference or one or more ASN.1 built-in type names. However, it cannot take both.

When specified locally, this directive takes no operands and only affects the type that it is placed next to.

Note: If a SizeConstraint subtype was specified (so that the maximum size or number of occurrences of a type is known), the `length` field of the VARYING representation is either `short`, `int`, or `long`, depending on which is the smallest needed to fit the largest value. In the absence of the SizeConstraint subtype the `length` field is always `short`.

Example:

The following notation shows the global and local use of the OSS.VARYING directive:

```
--<OSS.VARYING Module.VaryingStr>--

Module DEFINITIONS EXPLICIT TAGS ::= BEGIN
        NonVaryingBit   ::= [1] BIT STRING (SIZE(20))
        VaryingBit      ::= [2] BIT STRING (SIZE(20)) --<VARYING>--
        NonVaryingStr   ::= [3] VisibleString (SIZE(10))
        VaryingStr      ::= [4] VisibleString (SIZE(10))
        NonVaryingOct   ::= [5] OCTET STRING (SIZE (24))--<UNBOUNDED>--
        VaryingOct      ::= [6] OCTET STRING (SIZE (24))
END
```

Note that the default representation for OCTET STRING is VARYING.

After passing the above syntax through the ASN.1 compiler, the following is generated in the header file:

```
typedef struct NonVaryingBit {
    unsigned short  length;  /* number of significant bits */
    unsigned char   *value;
} NonVaryingBit;

typedef struct VaryingBit {
    unsigned short  length;  /* number of significant bits */
    unsigned char   value[3];
} VaryingBit;

typedef char            NonVaryingStr[11];

typedef struct VaryingStr {
    unsigned short  length;
    char            value[10];
} VaryingStr;

typedef struct NonVaryingOct {
    unsigned short  length;
    unsigned char   *value;
} NonVaryingOct;

typedef struct VaryingOct {
    unsigned short  length;
    unsigned char   value[24];
} VaryingOct;
```

Related directives:
OSS.DLINKED (see section 4.4.2.10), OSS.LINKED (see section 4.4.2.27), OSS.NULLTERM (see section 4.4.2.31), OSS.PADDED (see section 4.4.2.34), OSS.UNBOUNDED (see section 4.4.2.49)

4.4.2.58 OSS.XEREncodeFunction

The `OSS.XEREncodeFunction` directive was introduced in version 7.0 and allows you to specify your own data formatter for encoding ASN.1 types by XML Encoding Rules (XER Basic or Canonical). The function that you supply is called each time the XER Encoder is about to add the encoding of the type affected by this directive to the output buffer; instead of generating the XER encoding for the type, the encoder passes its C value to the user-defined function and expects the encoding of this value to be prepared and returned by this function.

This directive has the following format:

```
--<OSS.XEREncodeFunction absoluteReference
                         "yourXEREncodeFunctionName"
                         [parameterAbsoluteReference]>--
```

This directive may only be specified globally and takes up to 3 arguments. The first two are an absolute reference and a XER encoding function name; they are mandatory arguments. The last argument, *parameterAbsoluteReference*, is optional. The *`absoluteReference`* argument specifies which type of the ASN.1 syntax is to be encoded in XML with the specified function (it is possible to encode both primitive and constructed types; it is also possible to use the same function for several types, given they

are compatible as C structures). The specified function name, *yourXEREncodeFunctionName*, should be enclosed in double quotes and have **no** parentheses (i.e. "`()`") attached at the end. The last argument, *parameterAbsoluteReference*, if present, specifies an additional ASN.1 type to be used as a parameter of the encoding function; the encoder will submit the last instance of the parameter type found in the input PDU value to the user-provided function each time it is called.

The XER encoding formatter specified must have the following prototype:

```
int DLL_ENTRY yourXEREncodeFunctionName(struct ossGlobal *world,
                  yourType *in, [yourParameter *param,] OssBuf *out,
                  OssSafeMallocp safemalloc);
```

where `OssSafeMallocp` is a function pointer defined as

```
typedef unsigned char * (DLL_ENTRY_FPTR *_System OssSafeMallocp)
              (struct ossGlobal *world, OssBuf *buf);
```

The exact prototype of the encoding formatter function will be generated by the compiler into the generated header file.

If this directive is specified for some ASN.1 type, values of this type will not be encoded in accordance with the ITU-T X.693/ISO 8825-4 standard; instead, each value will be provided to the formatter function for encoding in accordance with custom application requirements. An optional C value for the function parameter may be provided as well (NULL is provided when no parameter values have been found in the input PDU at the time of calling the XER encoding function). The function should prepare the `length` and `value` fields of the `OssBuf *out` structure and return one of the integer values `OSS_FORMAT_OK_STATIC` or `OSS_FORMAT_OK` (these return values mean success) or a positive error code:

```
#define OSS_FORMAT_OK_STATIC  (-1)
#define OSS_FORMAT_OK         0
/* #define OSS_FORMAT_ERROR   any positive integer */
```

In case the function returns `OSS_FORMAT_OK_STATIC` the encoder does not attempt to free the memory referenced by the `value` field of the `OssBuf out` structure; it is assumed that this memory is not allocated on the heap.

In case the function returns `OSS_FORMAT_OK` the encoder will free the memory referenced by the `value` field of the `OssBuf out` structure. It is expected that this memory was allocated as:

```
out->length = estimated_size;
buf = (char *)(*safemalloc)(world, out);
```

This call returns the memory buffer to your application and prepares `out->value` storage for the encoder (in the simplest case they point to the same memory). Note that you should write pieces of the resulting XER encoding to the memory returned by `safemalloc` rather than writing to the `out->value`. Generally, `out->value` may be a handle such as a file descriptor.

In case the encoding formatter function returns a positive error code the encoder will give an error message:

```
E0259S: User application reported an error, error code: value
```

(where `value` is the formatting function return code) and will terminate the encoding process.

Note: The XER encoding function may not use any means of dynamically allocating the output memory other than calling the safemalloc function.
You can also use the safemalloc function to allocate temporary memory; however this is not recommended because this memory will be automatically freed only at the end of the encoding process and this may result in excessive memory usage. It is best to write your formatting function in such a way that it requires no temporary memory.

Example:

Assume that there is a PDU that describes goods in a store; each item is described by its name and price. Price is defined as an INTEGER. There is some utility information to interpret these integers as prices: 'precision' and 'currency' fields of a PDU. Prices should be formatted in accordance with these settings.

```
--<OSS.XEREncodeFunction Module.PDU.goods.*.price "formatAsPrice"
Module.PDU.settings>--
Module DEFINITIONS ::= BEGIN
  Settings ::= SEQUENCE {
      precision   INTEGER (0..8),
      currency    IA5String
  }

  PDU ::= SEQUENCE {
      settings Settings,
      goods SEQUENCE OF item SEQUENCE {
          name IA5String,
          price INTEGER (0..MAX)
      }
  }

  p PDU ::= {
    settings {
      precision 4,
      currency "$"
    },
    goods {
      item {
      name "Table",
      price 200000
      },
      item {
      name "Candy",
      price 2500
      },
      item {
      name "Television",
      price 5000000
      }
```

```
    }

  }

END
```

The compiler produces the following function declaration:

```
/* formatAsPrice is user-defined XER encoding formatter for
 * ASN.1 item(s) Module.PDU.goods.*.price */
extern int DLL_ENTRY formatAsPrice(struct ossGlobal *, unsigned int *, struct
Settings *, OssBuf *, OssSafeMallocp);
```

This function (that is user-written) should look like the following:

```
#include "xerencf.h" /* compiler-generated header file */

#define APP_BAD_ARGS    1 /* application-specific error code */

int DLL_ENTRY_FDEF formatAsPrice(struct ossGlobal *world,
                                 unsigned int *in,
                                 struct Settings *param, OssBuf *out,
                                 OssSafeMallocp safemalloc)
{
    char *buf;
    int prec = 1, i;

    if (!(in && param && param->currency && param->precision))
      return APP_BAD_ARGS;

    for (i = 0; i < param->precision; i++)
      prec *= 10;

    /* Estimate size of the output value. 4-byte integer formatted
       as string of given precision in range [0..8] cannot take more
       than 16 characters. */
    out->length = 16 + strlen (param->currency);
    buf = (char *)(*safemalloc)(world, out);

    /* Now exactly calculate value length and create the value */
    out->length = sprintf(buf, "%s%d.%.*d", param->currency,
            (int)((*in) / prec), param->precision,
            (int)((*in) % prec));

    return OSS_FORMAT_OK;
}
```

After compiling and linking the application, the XER encoder will produce the following encoding for the ‘p’ value from the input ASN.1 syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<PDU>
  <settings>
    <precision>4</precision>
    <currency>$</currency>
  </settings>
  <goods>
```

```
    <item>
      <name>Table</name>
      <price>$20.0000</price>
    </item>
    <item>
      <name>Candy</name>
      <price>$0.2500</price>
    </item>
    <item>
      <name>Television</name>
      <price>$500.0000</price>
    </item>
  </goods>
</PDU>
```

Note that price values are formatted as currency data in accordance with the custom settings rather than the plain INTEGER values.

4.5 Rules for Directives Applied to Parameterized Types

Here are some common rules to be aware of in applying directives to parameterized types and instances of such types:

Applying a directive to a parameterized type definition results in applying that directive to each individual instance of the type.

For example, the OSS.PDU directive applied to a parameterized type makes all instances of this type become PDUs. Likewise, the ASN1.WorkingSet directive applied to a parameterized type forces all instances of the type to be included in the working set. However, the ASN1.Remove directive is an exception; this directive has to be explicitly applied to particular instance or to each instance of a parameterized type.

Applying the OSS.FUNCNAME directive or the ASN1.ConstraintFunction directive to a parameterized assignment results in generation of names for the user-defined constraint functions which are mangled with a number (as a suffix) for each instance of this parameterized type other than the first instance.

Example:

```
--<OSS.FUNCNAME Test.Mouse "doggy">--

Test DEFINITIONS ::= BEGIN

  Mouse {Type} ::= SEQUENCE { t INTEGER } (CONSTRAINED BY {Type})
  Rat1 ::= [2] Mouse{SEQUENCE {i INTEGER, b BOOLEAN}}
  Rat2 ::= [3] Mouse{SEQUENCE {i INTEGER, b BOOLEAN}}

END
```

If the `-userConstraints` command line option is used when ASN.1-compiling the above example, the following is generated in the header file:

```
/* doggy is user-defined constraint function for ASN.1 item Test.Rat1 */
extern int DLL_ENTRY doggy(struct ossGlobal *, Rat1 *, void **);
/* doggy_1 is user-defined constraint function for ASN.1 item Test.Rat2 */
extern int DLL_ENTRY doggy_1(struct ossGlobal *, Rat2 *, void **);
```

Applying a directive to an instance of a parameterized type affects only that particular instance.

Example:

```
--<ASN1.Nickname  Test.S1     "Inst1_Seq">--
--<OSS.TYPENAME   Test.S1.a   "Inst1_Seq_a">--
--<OSS.DefineName Test.S1.a   "Inst1_Seq_a_c">--
--<OSS.DefineName Test.S2.a.c "Inst2_Seq_a_c">--

Test DEFINITIONS ::= BEGIN
   Seq {Type} ::= SEQUENCE { a Type OPTIONAL}
   S1 ::= [2] Seq{SEQUENCE {b INTEGER}}
   S2 ::= [3] Seq{SEQUENCE {c BOOLEAN OPTIONAL}}
END
```

The first three directives cause name changes only for the structure `S1`. The last directive affects the name in the second `#define` of `S2`:

```
typedef struct Inst1_Seq {
    unsigned char   bit_mask;
    #        define      Inst1_Seq_a_c_present 0x80
    struct Inst1_Seq_a {
        int             b;
    } a;/*optional; set in bit_mask Inst1_Seq_a_c_present if present */
} Inst1_Seq;

typedef struct S2 {
    unsigned char   bit_mask;
    #        define      a_present 0x80
    struct {
        unsigned char   bit_mask;
        #        define      Inst2_Seq_a_c_present 0x80
        ossBoolean      c;  /* optional; set in bit_mask
                             * Inst2_Seq_a_c_present if present */
    } a;  /* optional; set in bit_mask a_present if present */
} S2;
```

A directive cannot be applied to a type that is used as a substitution type (in an instance of a parameterized type), to fields of such a type, or to a type that is a parameter in a parameterized type.

Example:

The following directives have no effect:

```
--<ASN1.Nickname Test.T3.bt2.at1 bt2_nickname>--
--<OSS.TYPENAME  Test.T3.bt2 NewName-instance>--
--<OSS.TYPENAME  Test.T2.bt2 NewName-param>--

Test DEFINITIONS ::= BEGIN
   T2 {X} ::= SEQUENCE {at2 BIT STRING, bt2 X OPTIONAL}
   T3 ::= T2 {T1}
   T1 ::= SET {at1 INTEGER, bt1 BOOLEAN}
END
```

Note that the above OSS.TYPENAME directives will succeed if you change the type for `bt2` to be a "non-direct" reference to the parameter `X` (such as `SET OF X`) as shown below:

```
T2 {X} ::= SEQUENCE {at2 BIT STRING, bt2 SET OF X OPTIONAL}
```

5 Configuration files

The OSS ASN.1 compiler has a set of default settings for all global directives. These default settings are given in the previous sections that describe the various ASN.1 compiler directives and options. If you prefer to provide a different set of global directives, you may do so by creating an OSS ASN.1 compiler configuration file. Note however that **there is no need to create a configuration file** unless you prefer to use a different set of default values, or the encoder/decoder is being executed in an environment that is different from that of the compiler, as described below. In the latter case, OSS provides the configuration file.

5.1 asn1dflt default configuration file

The **asn1dflt** configuration file may be passed as the first input file on the command line. The **asn1dflt** file can contain comments and directives. These directives are used in lieu of the default directives that are built into the OSS ASN.1 compiler. Only compiler directives and ASN.1 comments are permitted in the **asn1dflt** file.

The effect of the directives in **asn1dflt** persists for the entire run of the compiler.

5.2 Cross-compiling to a different platform

By default the OSS ASN.1 compiler generates files for use on the operating system where the compiler executable runs. However, it is possible to instruct the ASN.1 compiler to produce files that will be used on a different operating system. This is done by passing a configuration file, `asn1dflt.<`*platform_name*`>`, as the first input file on the command-line.

For example, by default the OSS ASN.1 compiler for Windows generates files with an 8-byte alignment (`/Zp8`) for use with the Microsoft Visual C++ compiler. However, it is possible to use this same OSS ASN.1 compiler to create files that are usable on another platform such as Solaris SPARC by passing the appropriate *asn1dflt* configuration file (included in the shipment of OSS ASN1 Tools for the Solaris target platform) as the first command-line argument. For example, to compile the file `abc.asn` on Windows for use with the Solaris SPARC operating system, we would issue the command:

```
prompt%> asn1 asn1dflt.solaris-sparc abc.asn
```

`asn1dflt.solaris-sparc` is the unique *asn1dflt* filename specific to the Solaris SPARC platform. It contains configuration information that instructs the ASN.1 compiler that the target platform is Solaris SPARC. Each OSS ASN.1 Tools target platform contains one *asn1dflt* file unique to that platform.

After cross-compiling, the generated output files should be copied to the machine on which the application is being developed for the target platform. Subsequently, the generated files can be incorporated into the application as normal.

As mentioned previously, by default the OSS ASN.1 compiler for MS Windows generates files with an 8-byte alignment (`/Zp8`) for use with the Microsoft Visual C++ compiler. However, additional Windows compiler and data alignment options may be used. The following table shows the configuration files available for use along

with the intended compiler and data-alignment. Note that the naming convention of the Windows *asn1dflt* configuration file is slightly different: `asn1dflt.<`*compilerAbbreviation*`[.`*compilerOption*`]>.`

Windows Configuration File	Target Platform	Compiler	Data Alignment
`asn1dflt.ms.zp1`	MS Windows	MS Visual C++	`1-byte (/Zp1)`
`asn1dflt.ms.zp2`	MS Windows	MS Visual C++	`2-byte (/Zp2)`
`asn1dflt.ms.zp4`	MS Windows	MS Visual C++	`4-byte (/Zp4)`
`asn1dflt.ms.zp8`	MS Windows	MS Visual C++	`8-byte (/Zp8)`

The following examples further demonstrate the use of the **asn1dflt** file in cross-compiling:

Examples:

a) To instruct the ASN.1 compiler to compile the file `abc.asn` when the target platform is MS Windows and the C compiler is MS Visual C++ with a default 8-byte alignment (`/Zp8`), we would issue the following command:

```
prompt > asn1 <path-of-asn1dflt-directory>\asn1dflt.ms.zp8 abc.asn
```

The `asn1dflt.ms.zp8` file contains configuration information that instructs the ASN.1 compiler that the target platform is MS Windows, the language compiler in use is Microsoft Visual C++, and the data alignment length is 8-bytes. **Note** that this is the default for the Windows shipment of the OSS ASN.1 Tools.

b) To compile the file `abc.asn` for use with Microsoft Visual C++ (with 1-byte alignment, `/Zp1`) when the target platform is MS Windows platform, we would issue the following command:

```
prompt > asn1 <path-of-asn1dflt-directory>\asn1dflt.ms.zp1 abc.asn
```

The `asn1dflt.ms.zp1` file contains configuration information that instructs the ASN.1 compiler that the target platform is MS Windows, the language compiler in use is Microsoft Visual C++, and the data alignment length is 1-byte.

5.3 Configuration file specification using OSS.INCLUDES directive

One or more configuration files can be processed by means of the OSS.INCLUDES directive (see section 4.4.2.25). The treatment of configuration files specified via the OSS.INCLUDES directive is identical to the treatment of the **asn1dflt** file. Directives that are read from the file specified through the OSS.INCLUDES directive override any conflicting directives in the **asn1dflt** file. However, the effect of the directives obtained via the OSS.INCLUDES directive persists only for the duration of a single ASN.1 module definition.

5.4 Sample configuration file

The following is an example of a configuration file which is required for generating an encoder/decoder C file that is to be processed under the small memory model on MS-DOS.

Example:

```
--<ALIGNMENT 1,2,2,2,2,2,2,2,1,2,2,0,1,1,2,2,2,1,2,2,2,1,2,0,1,2,2,2>--
--<SHORTSIZE 2>--
--<INTSIZE 2>--
--<LONGSIZE 4>--
--<POINTERSIZE 2>--
--<FLOATSIZE 4>--
--<DOUBLESIZE 8>--
--<LONGDOUBLE 10>--
```

Note that the directives found in the OSS-prepared **asn1dflt** files are for internal use only. For more information about these internal directives, refer to Appendix A.

6 Restrictions

This version of the compiler imposes the following restrictions to ASN.1:

6.1 Line size

The maximum number of bytes allowed on a single line of input is 4096.

6.2 Token item size

The maximum number of bytes allowed on a input item such as an identifier, type reference, value reference, module reference, bstring, cstring or hstring is 1024.

6.3 Integer size

The ASN.1 compiler supports integers that are a maximum of 4096 bytes in size by using the `ASN1.HugeInteger` or `OSS.HUGE` directive (see sections 4.4.1.3 and 4.4.2.24). Without the `ASN1.HugeInteger` or `OSS.HUGE` directive, an integer may be 2, 4, or 8 -bytes long (depending on the platform in use).

6.4 Constraints on INTEGER types

Using the MAX or MIN in a constraint on an INTEGER type is interpreted by the OSS ASN.1 compiler as the minimum and maximum value of an "int" in C (which is a 4-byte integer on most machines). To constrain an integer to a range which will use an 8-byte integer, specify the actual values in the constraint rather than using MAX or MIN. For example:

```
A ::= INTEGER (0..MAX)  -- this will become an unsigned int
B ::= INTEGER (MIN..30) -- this will become a int
C ::= INTEGER (-1..3422147483647) -- this will become a long long
```

6.5 REAL value range values

Although the ASN.1 compiler supports the syntax of open ended value range constraint on REAL types, the runtime libraries will not enforce open ended value range constraints (using '<') on REAL types. These are treated as if the '<' were not present in the specification. If the open ended constraint must be enforced use the EXCEPT clause to do so. For example, instead of

```
A ::= REAL (0<..MAX)
B ::= REAL (100 <..< 1000)
```

use

```
A ::= REAL ((0..MAX) EXCEPT 0)
B ::= REAL ((100 <..< 1000) EXCEPT (100 | 1000))
```

Note that for values generated by the ASN.1 compiler for valuereferences, the compiler will not fill in missing DEFAULT values as the decoder does at run-time.

6.6 Command line option: -gendirectives

For the command line option `-gendirectives`, the compiler does not generate either ASN1.Nickname, or OSS.TYPENAME directives for mangled names in definitions of internal OSS types such as ObjectID, External, EmbeddedPDV, ABSTRACT_SYNTAX, TYPE-IDENTIFIER.

For the command-line option `-gendirectives`, the ASN.1 compiler does not generate `OSS.ExtractType` or `OSS.InlineType` directives for elements of COMPONENTS OF and for elements of MACRO definition.

So these directives should be added manually into `.gen` files after the ASN.1 compiler has generated this file. For example, if the ASN.1 syntax and the C representation is:

```
Testfile DEFINITIONS ::= BEGIN
    Testseq ::= SEQUENCE {
        a INTEGER,
        b OCTET STRING
    }
    Testseq2 ::= SEQUENCE {
        g OCTET STRING OPTIONAL,
        COMPONENTS OF Testseq
    }
END

typedef struct _octet1 {
    unsigned int    length;
    unsigned char   *value;
} _octet1;

typedef struct Testseq2 {
    unsigned char   bit_mask;
  #define      g_present 0x80
    _octet1         g;  /* optional; set in bit_mask g_present if present */
    int             a;
    _octet1         b;
} Testseq2;
```

then the directives generated in a `.gen` file would be:

```
--<OSS.TYPENAME Testfile.Testseq2.g "_octet1">--
--<OSS.ExtractType Testfile.Testseq2.g>--
```

As you can see the ASN.1 compiler does not generate similar directives for the field `"b"`, like:

```
--<OSS.TYPENAME Testfile.Testseq.b "_octet1">--
--<OSS.ExtractType Testfile.Testseq.b>--
```

The problem is that if the ASN.1 compiler is run with the same ASN.1 example extended by the directives from the `.gen` file, the OCTET STRING from the `Testseq` and `Testseq2` won't be shared due to the presence of the subtype directive of the second OCTET STRING. So the directives for `Testfile.Testseq.b` should be added manually.

NOTE: The same limitation applies to the command-line option `-keepnames` which is an extension of the compiler option, `-gendirectives`.

6.7 Restrictions imposed when the Lean encoder/decoder (LED) is in use

1) When the `-lean` option is specified, all ASN.1 types have only one C representation except REAL, UTF8String, SEQUENCE OF, and SET OF. Applied constraints and compiler directives do not affect the C-type representation used when the `-lean` option is specified.

2) REAL is represented as an `ossReal` (defined in the header file `leandef.h` which is automatically `#included` in the compiler-generated `.h` file) by default. You can change its representation to a null-terminated character string by applying the OSS.DECIMAL directive.

Example:

ASN.1:

```
Length1 ::= REAL
Length2 ::= REAL --<DECIMAL>--
```

C representation:

```
typedef ossReal Length1;
typedef char   *Length2;
```

(By default, an LED port will not have support for the REAL type. If you require such support, you can request a custom build that includes REAL support. Contact info@oss.com for more information.)

3) All restricted character string types are represented using the UNBOUNDED representation.

4) The UTCTime and GeneralizedTime types are represented as null-terminated character strings.

5) The UTF8String type may be represented as an `ossCharString` (an unbounded string with a UTF8-encoded value), an `ossBMPString` (a two-byte character string), or as an `ossUniversalString` (an unbounded array of four-byte characters. You can obtain each of these representations by using the OSS.UNBOUNDED, OSS.BMPSTRING and OSS.UNIVERSALSTRING directives, respectively.

By default (if none of these directives is present), the compiler chooses the `ossCharString` representation. However, when the `-compat BMPleanUTF8String` option is present, it chooses the ossBMPString representation instead (use this compat flag to restore the default behavior of the pre-7.0 ASN.1 compiler versions).

Example:

ASN.1:

```
Name1 ::= UTF8String
Name2 ::= UTF8String --<BMPSTRING>--
Name3 ::= UTF8String --<UNIVERSALSTRING>--
```

C representation:

```
typedef ossCharString Name1;
typedef ossBMPString Name2;
typedef ossUniversalString Name3;
```

The typedefs for `ossCharString`, `ossBMPString`, and `ossUniversalString` are located in the `leandef.h` header file.

6) By default, SEQUENCE OF and SET OF types are represented with the LINKED representation. However, you can choose you use the UNBOUNDED representation by applying the OSS.UNBOUNDED directive.

Example:

ASN.1:

```
Items1 ::= SEQUENCE OF INTEGER
Items2 ::= SEQUENCE --<UNBOUNDED>-- OF INTEGER
```

C representation:

```
typedef struct Items1 {
   struct Items1    *next;
   OSS_INT32       value;
} *Items1;

typedef struct Items2 {
   unsigned int    count;
   OSS_INT32       *value;
} Items2;
```

7) The index of a CHOICE type is always represented as `OSS_UINT32` (defined in `leandef.h`) regardless of the number of alternatives.

8) The bitmask for OPTIONAL components is represented as `OSS_UINT32` or `char[]` depending on the number of OPTIONAL elements.

9) The Lean encoder supports only definite length encoding. However, the Lean decoder supports any valid BER encoding.

10) When `-lean` is specified, the ASN.1 compiler does not accept the following compiler directives: OSS.ARRAY, OSS.DLINKED, OSS.DOUBLE, OSS.FLOAT, OSS.INT, OSS.LINKED, OSS.LLONG, OSS.LONG, OSS.MIXED, OSS.NULLTERM, OSS.OBJECTID, OSS.ONECHAR, OSS.PADDED, OSS.SHORT, and OSS.VARYING.

Note that all of these above directives are intended to change the ASN.1 type representation while the LED supports only a limited number of representations (i.e. representations that ensure efficiency and fast performance).

11) `-compat` flags related to releases earlier than version 5.1.0 are not supported.

12) The Lean encoder does not support the Canonical Encoding Rules (CER). However, the Lean decoder is able to correctly decode CER encodings.
To learn about the restrictions on the runtime libraries when using the LED, refer to the *Restrictions* chapter of the **OSS ASN.1 API Reference Manual**.

13) The Lean encoder/decoder does not support partial decoding. The partial decoding feature is supported only for the Time Optimized encoder/decoder (TOED).

6.8 Restrictions imposed when using the XER encoder/decoder

The OSS implementation for XER has the following restrictions. If you are hindered by any of these restrictions, you can contact OSS Nokalva.

1. The XER encoder/decoder does not support the OSS.LINKED and OSS.DLINKED directives for ANY:

- ANY --<UNBOUNDED | LINKED>--
- ANY --<UNBOUNDED | DLINKED>--

2. The XER encoder/decoder does not support forward referenced component relation constraints.

3. The XER encoder/decoder currently does not support the ASN1.DeferDecoding, OSS.NOCOPY, and OSS.OBJHANDLE compiler directives.

To learn about the restrictions on the runtime libraries when using the XER, refer to the *Restrictions* chapter of the **OSS ASN.1 API Reference Manual**.

6.9 Limitations imposed when using the Space Optimized encoder/decoder (SOED)

When the SOED encoder/decoder is in use, partial decoding is not supported. The partial decoding feature is supported only for the Time Optimized encoder/decoder (TOED).

6.10 Limitations imposed when using the OER/C-OER encoder/decoder

When the OER/C-OER encoder/decoder is in use, the following ASN.1 compiler directives are not supported: `ASN1.DeferDecoding`, and `OSS.OBJHANDLE(OSS.NOCOPY)`.

6.11 Restrictions when using the helper API

1) When the `-helperNames` option is specified or implied with any other helper-specific option, a limited set of C representations is supported for all ASN.1 types. Applied constraints do not affect the C-type representation used when the `-helperNames` option is specified or implied.

The following directives affecting C representations are supported:

For **INTEGER** type - OSS.SHORT, OSS.INT (default), OSS.LONG, OSS.LONGLONG, OSS.HUGE
For **BIT STRING**, **OCTET STRING**, **ANY** types - OSS.UNBOUNDED
For **CharacterString** types - OSS.NULLTERM (default), OSS.UNBOUNDED
For **GeneralizedTime**, **UTCTime** types - OSS.NULLTERM (default), OSS.TIMESTRUCT
For **OBJECT IDENTIFIER**, **RELATIVE-OID** types - OSS.ENCODED
For **REAL** type - OSS.DOUBLE (default), OSS.DECIMAL
For **SET OF**, **SEQUENCE OF** types - OSS.DLINKED-PLUS (default for types with structured elements), OSS.UNBOUNDED (default for types with simple non-structured elements, ex. SET OF BOOLEAN).
For **UTF8String** types - OSS.NULLTERM (default), OSS.UNBOUNDED, OSS.BMPSTRING, OSS.UNIVERSALSTRING.

The global directives can be used to override C representations set by default. Any local or type-specific directive overrides the default and global settings. The `length` and `count` fields in all generated structures have `unsigned int` types except the unbounded structure for ANY types which has the `length` field defined as `unsigned long`.

2) INTEGER with OSS.HUGE is represented as a typedef to the generated `_HInt` structure.

Example:

ASN.1:

```
HInt ::= INTEGER --<HUGE>--
```

C representation without `-lean`:

```
typedef struct _HugeInt {
    unsigned int    length;
    unsigned char   *value;
} _HugeInt;

typedef _HugeInt HInt;
```

C representation with `-lean` where `ossHugeInt` is defined in the header file `leandef.h` which is automatically `#included` in the compiler-generated `.h` file:

```
typedef ossHugeInt _HugeInt;
typedef _HugeInt HInt;
```

3) BIT STRING, OCTET STRING, and ANY types are represented as typedefs to the generated `_BitStr, _OctStr, and _Any` structures respectively.

Example:

ASN.1:

```
Bit ::= BIT STRING {b1(1), b2(3)}
Oct ::= OCTET STRING
A ::= ANY
```

C representation without `-lean`:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;

typedef struct _Any {
    unsigned long   length;
    unsigned char   *value;
} _Any;


typedef _BitStr Bit;
#define                     b1 0x40
#define                     b1_byte 0
#define                     b2 0x10
#define                     b2_byte 0

typedef _OctStr Oct;
typedef _Any    A;
```

C representation with `-lean` where `ossBitString`, `ossOctetString`, and `ossAny` are defined in the header file `leandef.h` (note that ANY types don't have a special helper-specific typedef):

```
typedef ossBitString _BitStr;
typedef ossOctetString _OctStr;

typedef _BitStr Bit;
#define                     b1 0x40
#define                     b1_byte 0
#define                     b2 0x10
#define                     b2_byte 0

typedef _OctStr Oct;
typedef ossAny  A;
```

4) All restricted character string types with the OSS.UNBOUNDED directive are represented as a typedef to the generated `_UnbCharStr` structure.

Example:

ASN.1:

```
Str ::= PrintableString (SIZE(1..2)) --<UNBOUNDED>--
```

C representation without `-lean`:

```
typedef struct _UnbCharstr {
    unsigned int    length;
    char            *value;
} _UnbCharStr;

typedef _UnbCharStr Str;
```

C representation with `-lean` where `ossCharString` is defined in the header file `leandef.h`:

```
typedef ossCharString _UnbCharStr;
typedef _UnbCharStr Str;
```

5) BMPString and UniversalString types are represented as typedefs to the generated `_BMPStr` and `_UnivStr` structures respectively.

Example:

ASN.1:

```
BStr ::= BMPString
UStr ::= UniversalString
```

C representation without `-lean`:

```
typedef struct _BmpStr {
    unsigned int    length;
    unsigned short  *value;
} _BmpStr;

typedef struct _UnivStr {
    unsigned int    length;
    int             *value;
} _UnivStr;

typedef _BmpStr BStr;
typedef _UnivStr UStr;
```

C representation with `-lean` where `ossBMPString` and `ossUniversalString` are defined in the header file `leandef.h`:

```
typedef ossBMPString _BmpStr;
typedef ossUniversalString _UnivStr;

typedef _BmpStr BStr;
typedef _UnivStr UStr;
```

6) UTF8String with OSS.UNBOUNDED, OSS.BMPSTRING, and OSS.UNIVERSALSTRING types are represented as typedefs to the generated `_UTF8Str, BMPStr, and _UnivStr` structures respectively.

Example:

ASN.1:

```
UnbUtf8 ::= UTF8String --<UNBOUNDED>--
BmpUtf8 ::= UTF8String --<BMPSTRING>--
UnivUtf8 ::= UTF8String --<UNIVERSALSTRING>--
```

C representation without `-lean`:

```
typedef struct _BmpStr {
    unsigned int    length;
    unsigned short  *value;
} _BmpStr;

typedef struct _UnivStr {
    unsigned int    length;
    int             *value;
} _UnivStr;

typedef struct _UTF8Str {
    unsigned int    length;
    unsigned char   *value;
} _UTF8Str;

typedef _UTF8Str UnbUtf8;
typedef _BmpStr  BmpUtf8;
typedef _UnivStr UnivUtf8;
```

C representation with `-lean` where `ossBMPString, ossUniversalString, ossCharString` are defined in the header file `leandef.h`:

```
typedef ossBMPString _BmpStr;
typedef ossUniversalString _UnivStr;
typedef ossCharString _UTF8Str;

typedef _UTF8Str UnbUtf8;
typedef _BmpStr BmpUtf8;
typedef _UnivStr UnivUtf8;
```

7) The UTCTime and GeneralizedTime with OSS.TIMESTRUCT types are represented as typedefs to `UTCTime` and `GeneralizedTime` structures defined in the shipped header file `asn1hdr.h`

Example:

ASN.1:

```
--<OSS.TIMESTRUCT Mod.Utc>--
--<OSS.TIMESTRUCT Mod.Gen>--

Utc ::= UTCTime
Gen ::= GeneralizedTime
```

C representation without `-lean`:

```
typedef UTCTime         Utc;
typedef GeneralizedTime Gen;
```

C representation with `-lean` is always null-terminated:

```
typedef char            *Utc;
typedef char            *Gen;
```

8) OBJECT IDENTIFIER and RELATIVE-OID types are represented as typedefs to the generated `_OID` and `_RelOID` structures respectively.

Example:

ASN.1:

```
Oid ::= OBJECT IDENTIFIER
RelOid ::= RELATIVE-OID
```

C representation without `-lean`:

```
typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

typedef struct _RelOID {
    unsigned int    length;
    unsigned char   *value;
} _RelOID;

typedef _OID            Oid;

typedef _RelOID         RelOid;
```

C representation with `-lean` where `ossObjectID` is defined in the header file `leandef.h`:

```
typedef ossObjectID _OID;
typedef ossObjectID _RelOID;

typedef _OID            Oid;
typedef _RelOID         RelOid;
```

9) By default, SEQUENCE OF and SET OF types with structured or pointered non-structured elements are represented with the DLINKED-PLUS representation. However, you can choose to use the UNBOUNDED representation by applying the type-specific or local OSS.UNBOUNDED directive or the global --<UNBOUNDED SET OF, SEQUENCE OF>-- directive.

By default, SEQUENCE OF and SET OF types with simple non-pointered non-structured elements are represented with the UNBOUNDED representation. The local or type-specific or global OSS.DLINKED-PLUS directive can be used to represent such types with DLINKED-PLUS structures. Note that the built-in type-specific directive like --<OSS.UNBOUNDED SEQUENCE OF>-- overrides the global DLINKED-PLUS

directive without parameters. But if both built-in type-specific directives --<OSS.UNBOUNDED SET OF>-- and --<OSS.DLINKED-PLUS SET OF>-- are present, each such directive specified next in the input ASN.1 syntax overrides the C representation previously set by another global directive.

If a type-specific OSS.HelperListAPI is applied to the SET OF/SEQUENCE OF type, it must have the DLINKED-PLUS representation. An error is issued if the representation is UNBOUNDED.

The -helperListAPI option does not affect SET OF/SEQUENCE OF representations. If it is specified, the helper list API functions are generated for all SET OF and SEQUENCE OF types that have the DLINKED-PLUS representation implied by default or set by local or global DLINKED-PLUS directives.

The -noHelperListAPI option, the global and the local OSS.NoHelperListAPI don't affect SET OF/SEQUENCE OF representations set by default or due to application of DLINKED-PLUS or UNBOUNDED directives.

Example:

ASN.1:

```
SetofDP ::= SET OF BIT STRING
SetofUnb ::= SET --<UNBOUNDED>-- OF BIT STRING
SeqofBP ::= SEQUENCE OF BOOLEAN --<POINTER>--
SeqofI ::= SEQUENCE OF INTEGER
```

C representation:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef struct SetofDP {
    struct SetofDP_node *head;
    struct SetofDP_node *tail;
    unsigned int    count;
} SetofDP;

typedef struct SetofDP_node {
    struct SetofDP_node *next;
    struct SetofDP_node *prev;
    struct _BitStr  *value;
} SetofDP_node;

typedef struct SetofUnb {
    unsigned int    count;
    _BitStr         *value;
} SetofUnb;

typedef struct SeqofBP {
    struct SeqofBP_node *head;
    struct SeqofBP_node *tail;
    unsigned int    count;
} SeqofBP;

typedef struct SeqofBP_node {
    struct SeqofBP_node *next;
```

```
    struct SeqofBP_node *prev;
    ossBoolean      *value;
} SeqofBP_node;

typedef struct Seqof {
    unsigned int    count;
    ossBoolean      *value;
} Seqof;
```

10) The POINTER directive is supported only for simple non-structured types: BOOLEAN, INTEGER without HUGE the directive, ENUMERATED, REAL with the DOUBLE directive, NULL.

11) All nested built-in complex types or nested built-in simple types that are represented by a structure (e.g., BIT STRING) are represented as extracted compiler-generated C structures with names fabricated by the compiler using so-called context-based naming conventions (See section 7.2.2 for more details). The OSS.InlineType directive is supported and forces the compiler to generate such structures inline.

Note that built-in types specified for valuereferences don't get context-based names. These names are created using the general method described in section 7.2.1.

Example:

ASN.1:

```
val SET OF IA5String ::= {"abc", "def"}
```

C representation:

```
typedef struct _setof1 {
    struct _setof1_node *head;
    struct _setof1_node *tail;
    unsigned int    count;
} _setof1;

typedef struct _setof1_node {
    struct _setof1_node *next;
    struct _setof1_node *prev;
    char            *value;
} _setof1_node;
extern _setof1 val;
```

12) All references to C structures derived from built-in complex ASN.1 types or simple types represented by a structure are always pointered. The NOPOINTER directive is not supported for such types. Tags of structures are generated without extra pointers.

Example:

ASN.1:

```
S ::= SEQUENCE {
    a SET OF INTEGER
}
```

C representation:

```
typedef struct S {
    struct S_a      *a;
} S;

typedef struct S_a {
    unsigned int    count;
    int             *value;
} S_a;
```

Note that OSS.UseThis directives are supported under the same conditions as if no `-helperNames` were specified. In the example below the OSS.UseThis directive is incorrectly applied along with the OSS.TYPENAME directive. The ASN.1 compiler issues a warning and generates the following structure.

Example:

ASN.1:

```
--<OSS.TYPENAME Mod.B.a MyType>--
--<OSS.UseThis Mod.B.a Mod.A.a>--

Mod DEFINITIONS ::= BEGIN
A ::= SET {a SEQUENCE OF INTEGER}
B ::= SEQUENCE {a SEQUENCE OF INTEGER }
END
```

C representation:

```
typedef struct A {
    struct A_a      *a;
} A;

typedef struct A_a {
    unsigned int    count;
    int             *value;
} A_a;

typedef struct MyType {
    unsigned int    count;
    int             *value;
} MyType;

typedef struct B {
    struct MyType   *a;
} B;
```

13) General type sharing procedures are disabled for non-parameterized types except in some cases where for example the OSS.TYPENAME directives are specified with the same "name" parameter for the structures that may be shared. In such cases, type sharing may occur to avoid structures with duplicate names being generated in the header file.

Example:

ASN.1:

```
--<OSS.TYPENAME Mod1.SetSetof.a "MyType">--
--<OSS.TYPENAME Mod1.ChSetof.a "MyType">--

SetSetof ::= SET {a SET OF INTEGER}
ChSetof ::= CHOICE {a SET OF INTEGER}
```

C representation:

```
typedef struct SetSetof {
    struct MyType   *a;
} SetSetof;

typedef struct ChSetof {
    unsigned short  choice;
#       define      a_chosen 1
    union {
        struct MyType   *a;  /* to choose, set choice to a_chosen */
    } u;
} ChSetof;

typedef struct MyType {
    unsigned int    count;
    int             *value;
} MyType;
```

14) Parameterized types sharing is preserved except for some very complex cases. If instances of a parameterized type can be shared an extra typedef is created for the parameterized type and typedefs with context-based names are generated for each instance.

Example:

ASN.1:

```
Mod DEFINITIONS ::= BEGIN

EXTENSION ::= CLASS {
    &id         OBJECT IDENTIFIER UNIQUE,
    &critical   BOOLEAN DEFAULT FALSE,
    &ExtenType
}
 WITH SYNTAX {
    SYNTAX          &ExtenType
    [ CRITICAL      &critical ]
    IDENTIFIED BY  &id
}


Extension { EXTENSION : InfoObjectSet } ::= SEQUENCE {
    extnID     EXTENSION.&id ({InfoObjectSet}),
    critical   EXTENSION.&critical ({InfoObjectSet}{@extnID})
}
```

```
Extensions {EXTENSION :InfoObjectSet } ::= SEQUENCE OF Extension {{
InfoObjectSe
t }}

Type1 ::= SEQUENCE {
     type1Extensions        [0] Extensions {{CRLExtensions1 }}
}

Type2 ::= SEQUENCE {
     type2Extensions        [0] Extensions {{CRLExtensions2 }}
}

CRLExtensions1 EXTENSION ::= { ...}
CRLExtensions2 EXTENSION ::= { ...}


END
```

C representation:

```
typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

typedef struct EXTENSION {
    unsigned char   bit_mask;
#       define      critical_present 0x80
    struct _OID     *id;
    ossBoolean      critical;  /* critical_present not set in bit_mask
implies
                                * value is FALSE */
    unsigned short  ExtenType;
} EXTENSION;

typedef struct Type1 {
    struct Extensions *type1Extensions;
} Type1;

typedef struct Type2 {
    struct Extensions *type2Extensions;
} Type2;

typedef struct Extension {
    struct _OID     *extnID;
    ossBoolean      critical;
} Extension;

typedef Extension Extensions_seq;

typedef struct Extensions {
    struct Extensions_node *head;
    struct Extensions_node *tail;
    unsigned int    count;
} Extensions;

typedef struct Extensions_node {
    struct Extensions_node *next;
```

```
        struct Extensions_node *prev;
        struct Extension *value;
    } Extensions_node;

    typedef Extensions Type1_type1Extensions;

    typedef Extensions_node Type1_type1Extensions_node;

    typedef Extensions Type2_type2Extensions;

    typedef Extensions_node Type2_type2Extensions_node;
```

To learn about the restrictions on the runtime libraries when using the Helper API, refer to the *Restrictions* chapter of the **OSS ASN.1/C Runtime API Reference Manual**.

7 C representations of ASN.1 notation

The following sections describe the representation of the ASN.1 type notation in C:

Section 7.1, **Translation rules for C representations**, describes the general method for translating ASN.1 into C data types.

Section 7.2, **Generating names for types, variables, and constants**, outlines the method of generating type variable, and constant names in the header file.

Section 7.3, **Effect of subtypes on generated data types**, illustrates how each ASN.1 subtype alters the C representation.

Section 7.4, **Representation of the ASN.1 macro notation in C**, outlines the means of translating ASN.1 macro notation into C.

Section 7.5, **Representation of the ASN.1 value notation in C**, outlines the means of translating ASN.1 Value Notation into C.

Section 7.6, **ASN.1 types C representation reference**, illustrates for each ASN.1 type, the default C representation as well as the representations that result after certain directives or ASN.1 specifications are applied.

Section 7.7, **Effect of type extensibility**, describes how the presence of extensible elements affects the C representation of ASN.1 types.

Section 7.8, **Representing information object classes**, gives details about how the OSS ASN.1 compiler represents information object classes.

Section 7.9, **Representing parameterized types**, discusses how the OSS ASN.1 compiler handles parameterized types.

Section 7.10, **Representing ASN.1 comments**, informs you about the general method used to preserve input ASN.1 comment information in the generated header file.

7.1 Translation rules for C representations

7.1.1 General rules for types

The means for representing ASN.1 data types in C is generally to:

- Generate a `typedef` for each type reference. **For example:**

  ASN.1:

  ```
  Age ::= INTEGER
  ```

  C language default representation:

  ```
  typedef int Age;
  ```

- Generate a `struct` for ASN.1 constructor types SET, and SEQUENCE. **For example:**

  ASN.1:

  ```
  Personal ::=  SEQUENCE {
      age     INTEGER,
      weight  INTEGER
  }
  ```

  C language default representation:

  ```
  typedef struct Personal {
      int age;
      int weight;
  } Personal;
  ```

- For OPTIONAL elements in a SET or SEQUENCE, generate a `bit_mask` field to indicate their presence or absence and generate a `#define` constant for each of these elements (for testing their presence or absence). **For example:**

  ASN.1:

  ```
  Personal ::= SEQUENCE {
      married  BOOLEAN,
      age      INTEGER,
      weight   INTEGER OPTIONAL,
      name     IA5String OPTIONAL
  }
  ```

  C language default representation:

  ```
  typedef struct Personal {
      unsigned char   bit_mask;
  #       define      weight_present 0x80
      ossBoolean      married;
  ```

```
    int             age;
    int             weight;  /* optional; set in bit_mask
                                weight_present if * present */
    char            *name;  /* NULL for not present */
} Personal;
```

Note: Bit mask values are generally not needed for optional elements that have pointers, since their absence is typically indicated by a NULL pointer.

- For the CHOICE type, generate a `struct` consisting of a selector field and a `union` containing the alternatives (represented as fundamental C data types). For each element in the CHOICE, generate a `#define` to be used in indicating which element is present. **For example:**

ASN.1:

```
Personal ::= CHOICE {
    married   BOOLEAN,
    age       INTEGER
}
```

C language default representation:

```
typedef struct Personal {
    unsigned short  choice;
#       define      married_chosen 1
#       define      age_chosen 2
    union {
        ossBoolean      married;  /* to choose, set choice to
                                     married_chosen */
        int             age;  /* to choose, set choice to
                                 age_chosen */
    } u;
} Personal;
```

- Generate a `#define` for each element of a named number list or named bit list. **For example:**

ASN.1:

```
Name ::= BIT STRING {red(0), white(1), blue(2)}
```

C language default representation:

```
typedef unsigned char   Name;
#define                   red 0x80
#define                   white 0x40
#define                   blue 0x20
```

7.1.2 Effects of ASN.1 constraints, keywords and OSS compiler directives

The C data types generated for ASN.1 types can be modified by the use of:

- The ASN.1 SizeConstraint, SingleValue, ValueRange, Contained, and Inner Subtype notations. **For example:**

ASN.1:

```
Name ::= BIT STRING
```

C language default representation:

```
typedef struct Name {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} Name;
```

The same notation with a SizeConstraint will yield:

ASN.1:

```
Name ::= BIT STRING (SIZE (5))
```

C language default representation if no -helperNames is specified or implied:

```
typedef struct Name {
    unsigned short  length;  /* number of significant bits */
    unsigned char   *value;
} Name;
```

Cross-reference: The effect of subtypes on generated data types is described in more detail in section 7.3.

- The ASN.1 contents constraint notation. See section 7.6.5 for more details.

- ASN.1 OPTIONAL and DEFAULT keywords cause bitmasks and `#define` constants to be generated (if the field does not have a pointer).

- OSS ASN.1 compiler directives alter the representation of the type they are applied to. For example, if you specify the OSS.POINTER directive on the INTEGER type, a pointer to an integer is generated, rather than an inline integer.

ASN.1:

```
Age ::= INTEGER --<POINTER>--
```

C language representation:

```
typedef int *Age;
```

Cross-reference: The OSS ASN.1 compiler directives are discussed in more detail in section 4.1.

Notes:
1) Neither subtype constraints nor compiler directives affect how data is encoded when BER, CER, or DER encoding rules are being used.
2) More than one compiler directive may be applied to one ASN.1 type. However when mutually exclusive directives are applied to the same type, the last one specified is used.

7.1.3 Recursive ASN.1 types

Recursively defined ASN.1 types are represented in C as types that point to an instance of themselves. **For example:**

ASN.1:

```
TreeNode ::= SEQUENCE {
        text        VisibleString (SIZE(1..255)),
        left    [1] TreeNode OPTIONAL,
        right   [2] TreeNode OPTIONAL
}
```

The following representation assumes that the OSS.PDU directive (see section 4.4.2.35) is specified.

C language representation:

```
typedef struct TreeNode {
    char            text[256];
    struct TreeNode *left;  /* NULL for not present */
    struct TreeNode *right;  /* NULL for not present */
} TreeNode;
```

Notice above how each `TreeNode` type has two pointers to other `TreeNode` types.

7.2 Generating names for types, variables, and constants

7.2.1 General method

Names from the input ASN.1 definitions are preserved in the generated header files. When element identifiers are missing (such as for CHOICE, SEQUENCE, and set types in **ASN.1:1990**), the ASN.1 compiler generates names derived from the type name. When the element of a SET OF or SEQUENCE OF is not a simple defined type (but is a SEQUENCE OF or SET OF), the compiler generates names in the header output for these structures, as needed; these names are of the form `_setof#` or `_seqof#` where '#' is an integer value. **For example:**

ASN.1:

```
Age  ::= INTEGER

Info ::= SEQUENCE {
                Age,
    married     BOOLEAN,
    names       SEQUENCE OF NameInfo
}

NameInfo ::= SEQUENCE {
     firstName VisibleString (SIZE(20)),
     lastName  VisibleString (SIZE(20))
}
```

C representation:

```
typedef int             Age;

typedef struct NameInfo {
    char            firstName[21];
    char            lastName[21];
} NameInfo;

typedef struct Info {
    Age             age;
    ossBoolean      married;
    struct _seqof1 {
        struct _seqof1  *next;
        NameInfo        value;
    } *names;
} Info;
```

Notice how the Compiler switches the order of the `Info` and `NameInfo` structures so that `NameInfo` can be referenced from within `Info`. Also notice how the ASN.1 SEQUENCE OF is named `struct _seqof1` in the C header file. Finally, note that the other names in the ASN.1 input are passed by default to the header file without change.

7.2.2 Context-based method

The `-helperNames` option instructs the ASN.1 compiler to use the new method in fabricating names for nested C structures derived from built-in ASN.1 types.

1) Built-in simple types that are represented by a C structure have simple intuitive names derived from corresponding ASN.1 type names (see Table of simple type names below). Note that all names are prefixed with an underscore except OpenType, UTCTime, GeneralizedTime that are defined in the shipped asn1hdr.h file.

Table of simple type names

ASN.1 type	C structure name
BIT STRING	_BitStr
BMPString	_BMPStr
All character string types with --<UNBOUNDED>-- directive applied	_UnbCharStr
GeneralizedTime --<TIMESTRUCT>--	GeneralizedTime
INTEGER --<HUGE>--	_HugeInt
OBJECT IDENTIFIER	_OID
OCTET STRING	_OctStr
Open type	OpenType or generated 'typedef' name
RELATIVE-OID	_RelOID
UniversalString	_UnivStr
UTCTime --<TIMESTRUCT>--	UTCTime
UTF8String --<UNBOUNDED>--	_UTF8Str

For example:

ASN.1:

```
B ::= BIT STRING {bit(1)}
S ::= SET OF BIT STRING
```

C representation:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef _BitStr B;
#define                     bit 0x40
#define                     bit_byte 0

typedef struct S {
    struct S_node   *head;
    struct S_node   *tail;
    unsigned int    count;
} S;
```

```
typedef struct S_node {
    struct S_node   *next;
    struct S_node   *prev;
    struct _BitStr *value;
} S_node;
```

2) Compiler-fabricated context-based names for structures that are not derived from built-in simple types are created based on the position of the built-in type within a complex type with a user-defined name. Such names have the following format:

`TypedefName_componentId1``[componentId2[…]][number]`

where:

"**TypedefName**" is the name generated in a typedef, usually derived from an ASN.1 typereference name.

"**componentId**1" is the identifier within the topmost structure whose type is built-in and whose C representation is a structure.

"**componentId2**", a field within "componentId1", is added in order to name a structure that appears at the second level of nesting if its ASN.1 type is a built-in type and its C representation is a structure, and so on.

"***number***" can be appended to the compiler-fabricated name in some rare cases if it conflicts with a user-defined name (ex. a user-defined name is “S-id1” and there is a typereference “S” that has the field “id1” whose type is structured) or with another compiler-fabricated name.

In the case of the SET OF and SEQUENCE OF types whose elements don't have identifiers and have built-in types represented by a C structure, the structures for such elements have names that include one of the following component names instead of identifiers:

`set` - for SEQUENCE OF SET
`seq` - for SEQUENCE OF SEQUENCE
`setof` - for SEQUENCE OF SET OF
`seqof` - for SEQUENCE OF SEQUENCE OF
`enum` - for SEQUENCE OF ENUMERATED

For example:

ASN.1:

```
S ::= SEQUENCE {
    s1 SEQUENCE OF SET {
        a1 INTEGER,
        a2 BOOLEAN
    }
}
```

C representation:

```
typedef struct S {
    struct S_s1     *s1;
} S;

typedef struct S_s1_set {
    int             a1;
    ossBoolean      a2;
} S_s1_set;

typedef struct S_s1 {
    struct S_s1_node *head;
    struct S_s1_node *tail;
    unsigned int    count;
} S_s1;

typedef struct S_s1_node {
    struct S_s1_node *next;
    struct S_s1_node *prev;
    struct S_s1_set *value;
} S_s1_node;
```

One of keywords: `seq`, `set`, `choice`, `setof`, `seqof`, `enum`, `any`, `boolean`, `null`, `integer`, `real` can be added to the name of the structure that appears within CONTAINING or CONSTRAINED BY constraints.

For example:

ASN.1:

```
SeqBit ::= SEQUENCE {
    a BIT STRING (CONTAINING INTEGER)
}
```

C representation:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef struct SeqBit {
    struct SeqBit_a *a;
} SeqBit;

typedef int             SeqBit_integer;

typedef struct SeqBit_a {
    /* ContentsConstraint is applied to SeqBit_a */
    _BitStr         encoded;
    SeqBit_integer  *decoded;
} SeqBit_a;
```

ASN.1:

```
I ::= INTEGER (CONSTRAINED BY {SEQUENCE {a INTEGER}})
```

C representation:

```
typedef int             I;

typedef struct I_seq {
    int             a;
} I_seq;
```

7.2.3 Exceptional cases

7.2.3.1 Handling long names

The compiler command-line option, `-shortenNames`, and the compiler directive, SHORTENNAMES, allows you to limit all variable names generated by the OSS Language Translator to a maximum of 31 characters. The compiler shortens user-supplied and compiler-generated names that are too long, and guarantees that they are unique.

7.2.3.2 Disambiguation of names

The compiler applies a disambiguation algorithm to guarantee that all variable names generated are unique within the current scope of reference. Additionally, it guarantees that no generated name conflicts with any C reserved name.

If a name generated by ASN.1 compiler results in ambiguity and the type being named is a member of a structure (CHOICE, SET, SET OF, SEQUENCE or SEQUENCE OF), then the name is prefixed with the containing-structure's name. If the name is still ambiguous, it is further prefixed with the next higher-level containing-structure's name (if present), and so on.

If an ENUMERATED type, or an INTEGER or BIT STRING with named values would generate an ambiguous name, the compiler will modify that name by prefixing it with the name of the type being defined. **For example:**

ASN.1:

```
PrimaryColors ::= INTEGER {blue(0), yellow(1), red(2)}
FlagColors ::= INTEGER {red(0), white(1), blue(2)}
```

C representation:

```
typedef int    PrimaryColors;
#define        PrimaryColors_blue 0
#define        yellow 1
#define        PrimaryColors_red 2
```

```
typedef int    FlagColors;
#define        FlagColors_red 0
#define        white 1
#define        FlagColors_blue 2
```

Notice how the shared colors, `red` and `blue`, have their parent-type's name prefixed to them with an underscore.

7.3 Effect of subtypes on generated data types

The OSS ASN.1 compiler supports all subtypes by, minimally, checking the syntax of the subtypes usage and verifying the context in which they are being used. Subtype specifications can determine the C representation of the ASN.1 type. The effect of subtypes being used on ASN.1 user-defined types is the same as on the ASN.1 built-in types.

For example, suppose that the following appeared in an ASN.1 module:

```
Name ::= VisibleString (SIZE (1..32))
```

The following C data type would be generated, assuming that the VisibleString is to be represented as a null-terminated string:

```
typedef char      Name[33];
```

If on the other hand we used a user-defined type as follow:

```
Name ::= VisibleString
First-name ::= Name (SIZE (1..32))
```

the following C data type would be generated in the header file:

```
typedef char      First_name[33];
```

Note that in presence of -helperNames option, subtype constraints do not affect generated C data types and the most generic type is always produced; for the last example above it is a `char *` type:

```
typedef char      *First_name;
```

7.3.1 SingleValue subtype

The OSS ASN.1 compiler syntax checks SingleValue subtypes and verifies that the values specified are valid in the context in which they are being used.

Additionally, when the SingleValue subtype is used with the INTEGER type and the OSS.SHORT/OSS.INT/OSS.LONG/LONGLONG local directives are not specified and the -helperNames option is not specified or implied, the ASN.1 compiler uses the specified SingleValue in determining the

size of the associated C variable. This is done by ensuring that the C variable is large enough to hold the value specified in the SingleValue subtype.

Cross-reference: See section 7.6.11 for more information on how the SingleValue Subtype affects the INTEGER type.

7.3.2 ValueRange subtype

In the absence of the OSS.SHORT/OSS.INT/OSS.LONG/LONGLONG local directive and the -helperNames option, the OSS ASN.1 compiler uses the ValueRange subtype to determine the size of the integer C data types. When a ValueRange subtype is specified, the compiler generates a C integer data type that is large enough to hold the minimum and maximum values indicated in the ValueRange.

At run-time the decoder makes sure that the value it is decoding is not too large to fit into the C data type associated with the ValueRange.

Cross-reference: See section 7.6.11 for examples of how the ValueRange subtype affects the INTEGER type.

Note: The ValueRange subtype has no effect on the representation of the REAL type.

7.3.3 SizeConstraint subtype

The OSS ASN.1 compiler uses the SizeConstraint subtype to determine the size of the C data types associated with BIT STRING, OCTET STRING, Character String types, SET OF and SEQUENCE OF if -helperNames option is not specified or implied. At run-time the decoder checks that the value it is decoding is not too large to fit into the C data type associated with the SizeConstraint. Likewise, the encoder ensures that the size or number of occurrences of the data it is encoding lies within the uppermost and lowermost bounds of the SizeConstraint.

Cross-reference: See BIT STRING, OCTET STRING, Character String types, SET OF and SEQUENCE OF in section 7.6 for examples on how they are affected by the SizeConstraint subtype.

7.3.4 PermittedAlphabet subtype

The OSS ASN.1 compiler syntax checks PermittedAlphabet subtypes and verifies that they are valid in the context in which they are being used. However, PermittedAlphabet subtypes have no effect on the C representation.

7.3.5 Contained subtype

The representation of a type with a Contained subtype is the same as that of the type it "INCLUDES".

For example, suppose the following is defined in an ASN.1 module (the global OSS.PDU directive is assumed):

```
NormalBit ::= BIT STRING --<VARYING>--
Flags     ::= BIT STRING (SIZE(1..61)) --<VARYING>--
MoreFlags ::= BIT STRING (INCLUDES Flags) --<VARYING>--
```

The following C data types are generated by the ASN.1 compiler for the above:

```
typedef struct NormalBit {
    unsigned short  length;  /* number of significant bits */
    unsigned char   value[1];  /* first element of the array */
} *NormalBit;

typedef struct Flags {
    unsigned short  length;  /* number of significant bits */
    unsigned char   value[8];
} Flags;

typedef struct MoreFlags {
    unsigned short  length;  /* number of significant bits */
    unsigned char   value[8];
} MoreFlags;
```

In this case, the SizeConstraint subtype `(SIZE(1..61))` affects the C data type produced for the `Flags` structure (i.e. 64 bits (8 bytes) are allocated for the `value` field). Additionally, the Contained subtype `(INCLUDES Flags)` above instructs the compiler to treat `MoreFlags` in the same manner as `Flags`.

However, as can be seen in the following example, if the subtype has no effect on the data type to be "INCLUDED", the type that "INCLUDES" it will be unaffected. Given the following ASN.1 notation:

```
Flags ::= BIT STRING (SIZE(1..61)) --<UNBOUNDED>--
MoreFlags ::= BIT STRING (INCLUDES Flags) --<UNBOUNDED>--
```

The following C data types is generated by the ASN.1 compiler:

```
typedef struct Flags {
   short           length;  /* number of significant bits */
   unsigned char  *value;
} Flags;

typedef struct MoreFlags {
   short       length;  /* number of significant bits */
   unsigned char  *value;
} MoreFlags;
```

Since the OSS.UNBOUNDED representation is not affected by the SizeConstraint, the Contained subtype has no effect on the `MoreFlags` structure. However, note that at run-time the encoder/decoder nonetheless enforces the SizeConstraint for both `Flags` and `MoreFlags`.

7.3.6 Inner subtype

The use of Inner subtypes has the same effect on compiler-generated data as if the subtype information which follows the "WITH COMPONENT(S)" had been specified in the definition of the element(s) of the parent type.

For example, suppose the following was defined in an ASN.1 module:

```
Flags ::= BIT STRING (SIZE(1..61)) --<VARYING>--
ManyFlags ::= SET OF Flags
LessFlags ::= ManyFlags (WITH COMPONENT (SIZE(1..51)))
```

For the above, the following C data types would be generated by the ASN.1 compiler if -helperNames option is not specified or implied:

```
typedef struct Flags {
   unsigned short length;  /* number of significant bits */
   unsigned char  value[8];
} Flags;

typedef struct ManyFlags {
   struct ManyFlags *next;  /* number of significant bits */
   Flags            value;
} *ManyFlags;

typedef struct LessFlags {
   struct LessFlags *next;
   struct {
      short          length;  /* number of significant bits */
      unsigned char  value[7];
   } value;
} *LessFlags;
```

In this case, the Inner subtype of `LessFlags` (i.e., `WITH COMPONENT(SIZE(1..51))`) affects the C data type produced (i.e., `value` is of size 7 instead of 8). The Inner subtype affects the C data type produced not simply because it was an Inner subtype, but also because the Compiler's rules for VARYING bit strings allow the SizeConstraint subtype to affect the C representation.

If however, a different directive (e.g., OSS.UNBOUNDED) for `Flags` had been used which did not allow SizeConstraints to have an effect on the C representaion, a SizeConstraint subtype applied to `LessFlags` would also have no affect on the C representation. This is illustrated by the following ASN.1 definition (the global OSS.PDU directive is assumed):

```
Flags ::= BIT STRING (SIZE(1..61)) --<UNBOUNDED>--
ManyFlags ::= SET OF Flags
LessFlags ::= ManyFlags (WITH COMPONENT (SIZE(1..51)))
```

For the above, the following C data types would be generated by the ASN.1 compiler:

```
typedef struct Flags {
   short       length;  /* number of significant bits */
```

```
   unsigned char  *value;
} Flags;

typedef struct ManyFlags {
   struct ManyFlags *next;
   Flags            value;
} *ManyFlags;

typedef struct LessFlags {
   struct LessFlags *next;
   struct {
      short         length;  /* number of significant bits */
      unsigned char *value;
   } value;
} *LessFlags;
```

Notice how the `value` fields for `Flags` and `LessFlags` are both represented by `unsigned char*` .

7.3.7 PATTERN constraint

Starting with version 5.4.0, the OSS ASN.1 Tools supports the PATTERN constraint notation. This notation allows you to restrict a character string type to be of a preset sequence of acceptable character patterns. This constraint notation has the general format:

<TypeName> **::=** *<RestrictedCharType>* **(PATTERN** *<RegularExpression>***)**

For example:

```
NoZeros ::= NumericString (PATTERN noZero)
noZero  UniversalString ::= "[^0]"
```

Note that the *RegularExpression* must be of type UniversalString. You can refer to Annex A "ASN.1 regular expressions" of the new ASN.1 standard to learn the intricate language for specifying character patterns using a regular expression.

PATTERN constraints are only enforced by the Space-optimized, and Lean encoders/decoders at runtime. The Time-optimized encoders/decoders do not support the checking of PATTERN constraints for the purposes of speed and efficiency.

The header file output of the ASN.1 compiler usually does not change by the application of a PATTERN constraint.

7.3.8 Contents constraint

Refer to section 7.6.5 to learn more about how the OSS ASN.1 Tools handle contents constraints.

7.4 Representation of ASN.1 macro notation in C

The representation of ASN.1 macro instances in C is determined by the type returned by the macro instance. Whatever the C representation of the returned type is, that is the representation of the ASN.1 macro.

For example, suppose an ASN.1 macro, ERROR, was defined as follows:

```
ERROR MACRO ::=
BEGIN
    TYPE NOTATION  ::= "PARAMETER" NamedType | empty
    VALUE NOTATION ::= value(VALUE INTEGER)
    NamedType      ::= identifier type | type
END
```

and then was used as follows:

```
ErrorRecord ::= SEQUENCE {
    Rectype  ERROR PARAMETER VisibleString,
    Sector   INTEGER
}
```

The following is the resulting C representation:

```
typedef struct ErrorRecord {
    int    rectype;
    int    sector;
} ErrorRecord;
```

The return type in the macro and the type used in the C representation have been underlined.

7.5 Representation of ASN.1 value notation in C

The representation of the ASN.1 value notation in C is that of an initialized variable whose name is derived from the ASN.1 value reference and whose type is defined from the associated type reference.

For each ASN.1 value reference in the root module(s) or referenced by it, a C variable is initialized in the `.c` file, and an `extern` declaration of that variable is made in the header file. You can use the OSS.NOVALUE (see section 4.4.2.54) directive to suppress the generation of values at ASN.1 compile time to have the C compiler ignore all initialized variables in the header and C files.

For example, if the following is defined:

```
age INTEGER ::= 5
```

The following is generated into the header file:

```
extern int age;
```

And the following is generated near the top of the `.c` file:

```
int age = 5;
```

7.6 Representation of ASN.1 Type Notation in C

This section outlines for each ASN.1 type:

- the ASN.1 specifications that can affect the representation (e.g., subtype constraints, NamedBitLists, etc.),
- the directive specifications that affect the representation (e.g., OSS.POINTER, OSS.ARRAY, etc.),
- the default representations for the ASN.1 type in the absence of the two items above,
- the effect on the representations of any applicable ASN.1 specifications or directives (e.g., how the OSS.SHORT directive affects an INTEGER type)
- helper macros and list API functions for each type (generated utility routines helping to create, access and modify values of an ASN.1 data type in a particular C representation)

The following ASN.1 type notations are listed in alphabetical and numerical order.

7.6.1 ANY/ANY DEFINED BY

The following specifications affect the representation of the ASN.1 ANY type:

ASN.1 specifications:
None.

Directive specifications without -helperNames:
OSS.DLINKED
OSS.LINKED
OSS.OBJHANDLE
OSS.POINTER
OSS.UNBOUNDED

Directive specifications with -helperNames:
OSS.HelperMacro
OSS.NoHelperMacro
OSS.UNBOUNDED

By default, the UNBOUNDED representation is implied:

ASN.1:
```
Any ::= ANY
```

C representation:

```
typedef struct _Any {
    unsigned long   length;
    unsigned char   *value;
} _Any;

typedef _Any            Any;
```

Refer to section 4.4.2.49 for more information about the UNBOUNDED representation.

Effect of the OSS.DLINKED directive on the ANY type

The OSS.DLINKED directive allows you to represent the ANY type as a doubly linked list of values which when concatenated forms the entire value of the Any type. The OSS.UNBOUNDED directive is always implied when OSS.DLINKED specified.

ASN.1:

```
AnyD ::= ANY --<DLINKED>--
```

C representation:

```
typedef struct AnyD {
    struct AnyD     *next;
    struct AnyD     *prev;
    unsigned long   length;
    unsigned char   *value;
} *AnyD;
```

Effect of the OSS.LINKED directive on the ANY type

The OSS.LINKED directive allows you to represent the ANY type as a linked list of values which when concatenated forms the entire value of the ANY type. The OSS.UNBOUNDED directive is always implied when the OSS.LINKED directive is specified.

ASN.1:

```
AnyL ::= ANY --<LINKED>--
```

C representation:

```
typedef struct AnyL {
    struct AnyL     *next;
    unsigned long   length;
    unsigned char   *value;
} *AnyL;
```

Effect of OSS.POINTER on the ANY type

The sole effect of the OSS.POINTER directive on the representation of ANY (or any other ASN.1 type) is to introduce a level of indirection in referencing the data.

ASN.1:

```
AnyP ::= ANY --<POINTER>--
```

C representation:

```
typedef _Any            *AnyP;
```

Notes:
1) The sole effect of the OSS.POINTER directive on an ASN.1 type is to introduce a level of indirection by declaring a C pointer to the type. Since this effect does not vary from ASN.1 type to ASN.1 type. We will not mention the effect of this directive on the remaining types in this section.
2) The conversion of the ASN.1 value notation to C is supported for all types except ANY. If you need such a feature, inform OSS Nokalva.

Effect of OSS.HelperMacro on the ANY type

The OSS.HelperMacro directive applied to an ANY results in generating the following helper macros:

1) `_new` or `_new_pdu` macro (if ANY is a PDU type) to allocate an empty structure for ANY
2) `_copy` or `_copy_pdu` macro to create a structure and copy input data to it
3) `_setv` macro to fill in an existing structure with input data

Note that 'main' helper macros are produced for generated `_Any` types and then referenced by macros for each user-defined ANY type. The below example demonstrates this:

ASN.1:

```
AnyHM ::= ANY
```

Helper macro:

```
typedef struct _Any {
    unsigned long   length;
    unsigned char   *value;
} _Any;

typedef _Any            AnyHM;

/* allocates memory for AnyHM_PDU */
#define oss_AnyHM_new_pdu(world) \
    (AnyHM *)ossGetInitializedMemory(world, sizeof(AnyHM))

/* allocates memory for AnyHM_PDU and initializes it by copy of an input
 * string */
#define oss_AnyHM_copy_pdu(world, value_, length_) \
    (AnyHM *)oss__UnbCharString_copy(world, (char *)value_, length_)

/* initializes 'value' and 'length' fields of a structure by given values */
#define oss_AnyHM_setv(outp, value_, length_) \
    { \
        (outp)->value = (value_); \
        (outp)->length = (length_); \
    }

/* initializes 'value' and 'length' fields of a structure by given values */
#define oss_AnyHM_setv(outp, value_, length_) \
    oss__Any_setv(outp, value_, length_)
```

7.6.2 BIT STRING

The following specifications affect the representation of the ASN.1 BIT STRING type:

ASN.1 specifications:

NamedBitList
SizeConstraint subtype

Directive specifications without -helperNames:

OSS.OBJHANDLE
OSS.PADDED
OSS.POINTER
OSS.PrintfunctionName
OSS.UNBOUNDED
OSS.VARYING

Directive specifications with -helperNames:

OSS.HelperMacro
OSS.NoHelperMacro
OSS.OBJHANDLE
OSS.PrintfunctionName
OSS.UNBOUNDED

Note that the OSS.PADDED, OSS.UNBOUNDED , and OSS.VARYING directives are mutually exclusive to each other.

By default, the UNBOUNDED representation is implied:

ASN.1:

```
BitStr ::= BIT STRING
```

C representation:

```
typedef struct BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} BitStr;
```

Under the UNBOUNDED representation , a pointer to a `char*` field is generated, preceded by a `length` field of type `unsigned int`.

Effect of NamedBitList on the BIT STRING type

When a NamedBitList is present, constants with a suffix of `_byte` are generated to free the user from worrying about which specific byte a NamedBit is located in. For example:

ASN.1:

```
BitStrNbB ::= BIT STRING {red(0), white(1), blue(50)}
```

C representation without -helperNames:

```
typedef struct BitStrNbB {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} BitStrNbB;
#define                     red 0x80
#define                     red_byte 0
#define                     white 0x40
#define                     white_byte 0
#define                     blue 0x20
#define                     blue_byte 6
```

C representation with -helperNames:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef _BitStr BitStrNbA;
#define                     a 0x80
#define                     a_byte 0
#define                     b 0x40
#define                     b_byte 0
#define                     c 0x08
#define                     c_byte 0
```

When a NamedBitList generates an array of characters, constants with a suffix of `_byte` are generated to free the user from worrying about which specific byte a NamedBit is located in. For example:

ASN.1:

```
BitStrNbB ::= BIT STRING {red(0), white(1), blue(50)}
```

C representation without -helperNames:

```
typedef unsigned char   BitStrNbB[7];
#define                     red 0x80
#define                     red_byte 0
#define                     white 0x40
#define                     white_byte 0
#define                     blue 0x20
#define                     blue_byte 6
```

C representation with -helperNames:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef _BitStr BitStrNbB;
#define                     red 0x80
#define                     red_byte 0
#define                     white 0x40
#define                     white_byte 0
#define                     blue 0x20
#define                     blue_byte 6
```

In the following example, the application programmer could easily set the `blue` bit in the variable `v` (which is of type `BitStrNbB`), without having to figure out which byte in the array of chars is to be set. For example:

```
Name1 v;
v[blue_byte] |= blue;
```

Effect of SizeConstraint on the BIT STRING type

The compiler uses the SizeConstraint subtype to determine the maximum number of bytes that there are in a BIT STRING. Each byte is packed with as many bits as can fit. In the absence of directives and the -helperNames option, an `unsigned char`, `short`, `int`, `long` or an array of `unsigned chars` is generated (i.e., the PADDED representation), depending on the number of bytes needed to hold the bit string. If the -helperNames option is specified, an `unsigned int` is always generated.

ASN.1:

```
BitStrSz ::= BIT STRING (SIZE(50))
```

C representation without -helperNames:

```
typedef struct BitStrSz {
    unsigned short  length;  /* number of significant bits */
    unsigned char   *value;
} BitStrSz;
```

C representation with -helperNames:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef _BitStr BitStrSz;
```

Note: Remember that the SizeConstraint for BIT STRING types is specified in bits.

Effect of the OSS.PADDED directive on the BIT STRING type

The OSS.PADDED directive requires the presence of a NamedBitList or the SizeConstraint subtype, because the compiler must be able to determine the maximum length of the bit string. An `unsigned char`, `short`, `int`, `long` or array of `unsigned chars` is generated, depending on the number of bytes needed to satisfy the SizeConstraint or accommodate the largest named bit. If the OSS.PADDED directive is specified without the required length indicators, the compiler issues an error message.

ASN.1:

```
BitStrPa  ::= BIT STRING --<PADDED>-- (SIZE(10))
```

C representation:

```
typedef unsigned short  BitStrPa;
```

Effect of the OSS.VARYING directive on the BIT STRING type

If the OSS.VARYING directive alone is specified, the result is a pointer to a structure containing a length field of type `unsigned short` and a character field of unspecified length. The effect is the same as when the OSS.VARYING and OSS.POINTER directives are specified together (i.e., OSS.POINTER is assumed because the array length is indeterminable by the compiler).

ASN.1:

```
BitStrVa ::= BIT STRING --<VARYING>--
```

C representation:

```
typedef struct BitStrVa {
    unsigned short  length;  /* number of significant bits */
    unsigned char   value[1];  /* first element of the array */
} *BitStrVa;
```

Effect of OSS.HelperMacro on the BIT STRING type

The OSS.HelperMacro directive applied to a BIT STRING results in generating the following helper macros:

1) `_new` or `_new_pdu` macro (if BIT STRING is a PDU type) to allocate an empty structure
2) `_copy` or `_copy_pdu` macro to create a structure and copy input data to it
3) `_setv` macro to fill in an existing structure with input data

Note that 'main' helper macros are produced for generated `_BitStr` types and then referenced by macros for each user-defined BIT STRING type. The below example demonstrates this:

ASN.1:

```
BitStrHM  ::= BIT STRING
```

Helper macro:

```
typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

/* allocates memory for an empty string */
#define oss__BitStr_new(world) \
    (_BitStr *)ossGetInitializedMemory(world, sizeof(_BitStr))

/* allocates memory for the string and initializes it by copying the input
 * value, length_ is number of bits */
#define oss__BitStr_copy(world, value_, length_) \
    (_BitStr *)oss__UnbBitStr_copy(world, value_, length_)

/* initializes 'value' and 'length' fields of a string by given values, length_
 * is number of bits */
#define oss__BitStr_setv(outp, value_, length_) \
    { \
    (outp)->value = (value_); \
    (outp)->length = (length_); \
    }

typedef _BitStr BitStrHM;

/* allocates memory for BitStrHM_PDU */
#define oss_BitStrHM_new_pdu(world) \
```

```
    oss__BitStr_new(world)

/* allocates memory for BitStrHM_PDU and initializes it by copy of an input
 * string */
#define oss_BitStrHM_copy_pdu(world, value_, length_) \
    oss__BitStr_copy(world, value_, length_)

/* initializes 'value' and 'length' fields of a string by given values, length_
 * is number of bits */
#define oss_BitStrHM_setv(outp, value_, length_) \
    oss__BitStr_setv(outp, value_, length_)
```

7.6.3 BOOLEAN

The following specification affects the representation of the ASN.1 BOOLEAN type:

ASN.1 specifications:
None.

Directive specification:
OSS.POINTER

Directive specification with -helperNames for BOOLEAN types with POINTER directive or marked as PDUs:
OSS.HelperMacro
OSS.NoHelperMacro

By default, the C representation for the BOOLEAN type is:

ASN.1:

```
Married ::= BOOLEAN
```

C representation:

```
typedef ossBoolean      Married;
```

where `ossBoolean` is defined in file `asn1hdr.h` as:

```
typedef char ossBoolean;
```

Effect of OSS.HelperMacro on the BOOLEAN type

The OSS.HelperMacro directive applied to a BOOLEAN results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the value and set it to FALSE
2) `_copy` or `_copy_pdu` macro to allocate memory and copy input data to it

Macros for a BOOLEAN type are produced only if this type is a PDU or it is referenced by pointer. If a BOOLEAN type is a non-typereference field of a SET or SEQUENCE, then its parent type should have its own macros produced in order to produce macros for BOOLEAN (the same rule is true for any non-typereference ASN.1 type in place of BOOLEAN). An example follows.

ASN.1:

```
--<OSS.HelperMacro Mod.BoolHM>--
--<OSS.HelperMacro Mod.SetBoolHM>--
--<OSS.HelperMacro Mod.SetBoolHM.bool>--
Mod DEFINITIONS ::= BEGIN
BoolHM  ::= BOOLEAN --<PDU>--
SetBoolHM  ::= SET {bool BOOLEAN --<POINTER>--}
END
```

Helper macro:

```
/* allocates memory for BoolHM_PDU */
#define oss_BoolHM_new_pdu(world) \
    (BoolHM *)ossGetMemory(world, sizeof(BoolHM))

/* allocates memory for BoolHM_PDU and initializes it by given value */
#define oss_BoolHM_copy_pdu(world, value_) \
    oss__Bool_copy(world, value_)

typedef struct SetBoolHM {
    ossBoolean      *bool;
} SetBoolHM;

…

/* allocates memory for an instance of the boolean type */
#define oss_SetBoolHM_bool_new(world) \
    (ossBoolean *)ossGetMemory(world, sizeof(ossBoolean))

/* allocates memory for an instance of the boolean type and initializes it by
 * given value */
#define oss_SetBoolHM_bool_copy(world, value_) \
    oss__Bool_copy(world, value_)
```

7.6.4 CHOICE

The following specifications affect the representation of the ASN.1 CHOICE type:

ASN.1 specification:

Inner subtype

Directive specification without -helperNames:

OSS.DefineName
OSS.ExtractType
OSS.FIELDNAME
OSS.InlineType
OSS.POINTER

Directive specification with -helperNames:

OSS.DefineName
OSS.ExtractType
OSS.FIELDNAME
OSS.InlineType

OSS.HelperMacro
OSS.NoHelperMacro

By default, the C representation is of the form:

```
typedef struct Name1 {
   unsigned short choice;     /* selector for the union */
#       define      alternative1_chosen;
#       define      alternative2_chosen;
                .
                .
                .
#       define      alternativen_chosen;
   union {
      type1   alternative1;
      type2   alternative2;
           .
           .
           .
      typen   alternativen;
     } u;
  } Name1;
```

The alternatives (e.g., `alternative1`) are the identifiers associated with the alternatives of the CHOICE type. If an identifier is missing, the compiler will generate a name based on the name of the alternative.

The compiler generates manifest constants to identify which alternative is contained within the union. The names generated for these manifest constants are the identifier plus a suffix of `_chosen`.

ASN.1:

```
ProductDesignator ::= CHOICE {
departmentNo        INTEGER,
description     [0] IMPLICIT VisibleString,
inventoryNo     [1] IMPLICIT INTEGER
}
```

C representation:

```
typedef struct ProductDesignator {
unsigned short  _choice;
#       define      departmentNo_chosen 1
#       define      description_chosen 2
#       define      inventoryNo_chosen 3
    union {
        int             departmentNo;  /* to choose, set _choice to
                                         * departmentNo_chosen */
        char            *description;  /* to choose, set _choice to
                                         * description_chosen */
        int             inventoryNo;  /* to choose, set _choice to
                                         * inventoryNo_chosen */
    } u;
} ProductDesignator;
```

Effect of Inner Subtype on the CHOICE type

See sections 7.3.6 for a full discussion on the effect of this subtype notation.

The Inner Subtype specification for the CHOICE type is supported as defined in ISO 8824 | X.208 (1990). The ASN.1 compiler checks the syntax of Inner Subtype notation for the CHOICE type; however, no visible effect is produced in the generated files.

Effect of OSS.HelperMacro on the CHOICE type

The OSS.HelperMacro directive applied to a CHOICE results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the structure
2) `_which` macro to identify a CHOICE alternative currently selected
3) `_<identifier>_get` macro to determine a value of that alternative
4) `_<identifier>_set` macro to set a particular CHOICE alternative value

An example follows.

ASN.1:

```
C ::= CHOICE {
    a1 INTEGER,
    a2 C
} --<PDU>--
```

Helper macros:

```
typedef struct C {
    unsigned short  choice;
#       define      a1_chosen 1
#       define      a2_chosen 2
    union {
        int             a1;  /* to choose, set choice to a1_chosen */
        struct C        *a2;  /* to choose, set choice to a2_chosen */
    } u;
} C;

/* allocates memory for an empty instance of the choice type */
#define oss_C_new(world) \
    (C *)ossGetInitializedMemory(world, sizeof(C))

/* allocates memory for C_PDU */
#define oss_C_new_pdu(world) \
    oss_C_new(world)

/* gets index of currently selected alternative */
#define oss_C_which(inp) \
    (inp)->choice

/* gets "a1" alternative value */
#define oss_C_a1_get(inp) \
    (inp)->u.a1

/* selects "a1" alternative and set its value */
#define oss_C_a1_set(outp, a1_) \
    { \
    (outp)->choice = a1_chosen; \
    (outp)->u.a1 = (a1_); \
    }
```

```
/* gets "a2" alternative value */
#define oss_C_a2_get(inp) \
    (inp)->u.a2

/* selects "a2" alternative and set its value */
#define oss_C_a2_set(outp, a2_) \
    { \
    (outp)->choice = a2_chosen; \
    (outp)->u.a2 = (a2_); \
    }
```

7.6.5 Contents constraint

Starting from version 5.4.0, the CONTAINING and ENCODED BY subtype constraint notation (defined in X.682) is supported by the ASN.1 compiler and runtime libraries. With this notation you may further define the contents of the BIT STRING and OCTET STRING types.

With this new notation, you can explicitly specify a simple or structured ASN.1 value as being contained in a bit stream or octet stream. You can also indicate which encoding rules were used to produce the value contained in the bit or octet stream.

Two examples of this new notation are:

```
Ex1 ::= OCTET STRING (
   CONTAINING INTEGER
   ENCODED BY {joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0)
               aligned(0)}
)

Ex2 ::= BIT STRING (
   CONTAINING IA5String (SIZE(0..8))
   ENCODED BY {joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0)
               aligned(0)}
)
```

The first example above illustrates how to constrain an octet stream to contain INTEGER values which are encoded with PER (Packed Encoding Rules).

The second example above illustrates how to constrain a bit stream to contain IA5String values which are between 0 and 8 bytes long encoded with PER (Packed Encoding Rules).

Note how the encoding rules specified after the ENCODED BY reserved words are specified using an OBJECT IDENTIFIER value. This OBJECT IDENTIFIER value must refer to a valid node in the object identifier tree for a set of encoding rules for ASN.1 data structures. However, these encoding rules may be different from the standard encoding rules officially defined by the ASN.1 standard.

If the OBJECT IDENTIFIER value used for ENCODED BY does not match any of the permitted encoding rules supported by the OSS tools then the ASN.1 compiler issues the corresponding warning and ENCODED BY clause is ignored by the ASN.1 compiler.

For example, for the following notation:

```
OS ::= OCTET STRING (
   CONTAINING INTEGER
```

```
        ENCODED BY {1 2 3}
    )
```

this warning will be issued:

```
"sample.asn", line 3 (Sample): C0673W: The object identifier value
specified in the ENCODED BY clause of ContentsConstraint for 'OS' does
not match any of the permitted encoding rules.  Refer to OSS
documentation for a list of permitted values.
```

Thus, the type OS will be encoded/decoded as if the ENCODED BY clause is absent.

For purposes of convenience and readability, you may pre-define values for the encoding rules you wish to use:

```
enc-BER OBJECT IDENTIFIER ::=
        {joint-iso-itu-t(2) asn1(1) basic-encoding(1)}
enc-DER OBJECT IDENTIFIER ::=
        { joint-iso-itu-t(2) asn1(1) ber-derived(2)
distinguished-encoding(1)}
enc-CER OBJECT IDENTIFIER(2) ::=
        { joint-iso-itu-t(2) asn1(1) ber-derived(2) canonical-encoding(0)}
enc-PER-Aligned OBJECT IDENTIFIER ::=
        {joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0)
aligned(0)}
enc-PER-Unaligned OBJECT IDENTIFIER ::=
        {joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0)
unaligned(1)}
enc-XER-Basic OBJECT IDENTIFIER ::=
        {joint-iso-itu-t(2) asn1(1) xml-encoding(5) basic(0)}
enc-XER-Canonical OBJECT IDENTIFIER ::=
        {joint-iso-itu-t(2) asn1(1) xml-encoding(5) canonical(1)}
```

This allows for a much more readable contents constraint:

```
-- Pre-defined values for OBJECT IDENTIFIER
--
enc-PER-Aligned OBJECT IDENTIFIER ::=
    {joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0)
aligned(0)}

-- Contents constraint definition
--
Ex1 ::= OCTET STRING (
  CONTAINING INTEGER
  ENCODED BY enc-PER-Aligned
)
```

7.6.5.1 C representation of types with contents constraints applied without -helperNames

After passing the above notation through the ASN.1 compiler, the following data structures are generated in the header file:

```
#define          Ex1_PDU 1
#define          Ex1_integer_PDU 2
```

```
typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} ObjectID;

typedef int             Ex1_integer;

/* Contents constraint definition*/
/**/
typedef struct Ex1 {
    /* ContentsConstraint is applied to Ex1 */
    struct {
        unsigned int    length;
        unsigned char   *value;
    } encoded;
    Ex1_integer     *decoded;
} Ex1;
```

Notice how both the containing OCTET STRING and also the contained integer have PDU constants generated for them. [Note that in the case that the containing type is not specified, decoded is declared as void*.]

[Note if the global or local OSS.NOPDU is applied to the contained type or the `-noPdusForContainingTypes` compiler option is specified then a PDU constant is not generated for such types].

7.6.5.2 C representation of types with contents constraints applied with -helperNames

If the -helperNames option is specified, the “encoded” field gets the extracted but still non-pointered type. The following data structures are generated in the header file for the same notation:

```
#define          Ex1_PDU 1
#define          Ex1_integer_PDU 2

typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;

typedef int             Ex1_integer;

    /* Contents constraint definition */
    /**/
typedef struct Ex1 {
    /* ContentsConstraint is applied to Ex1 */
    _OctStr         encoded;
    Ex1_integer     *decoded;
} Ex1;
```

Effect of OSS.HelperMacro on types with contents constraints

The OSS.HelperMacro directive applied to a type with a contents constraint results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the structure and initialize it to zeroes
2) `_set_encoded` macro to fill the `encoded` representation of a type by reference to some encoded data
3) `_copy_encoded` or `_copy_encoded_pdu` macro to allocate memory for the structure and fill its `encoded` representation by a copy of some encoded data
4) `_set_decoded` macro to fill the `decoded` representation of a type by reference to some C value
5) `_copy_decoded` or `_copy_decoded_pdu` macro to allocate memory for the structure and fill its `decoded` representation by a copy of some C value (implies a call to `ossCpyValue()` API function)

An example follows.

ASN.1:

```
--<OSS.HelperMacro M.CC.*>--
M DEFINITIONS ::= BEGIN
CC ::= SEQUENCE OF OCTET STRING (CONTAINING NULL)
END
```

Helper macros:

…

```
typedef struct CC_seq {
    /* ContentsConstraint is applied to CC_seq */
    _OctStr         encoded;
    CC_null         *decoded;
} CC_seq;

/* allocates memory for the octet string with a ContentsConstraint applied */
#define oss_CC_seq_new(world) \
    (CC_seq *)ossGetInitializedMemory(world, sizeof(CC_seq))

/* allocates memory for the octet string with a ContentsConstraint applied and
 * initializes it by copying the encoded value */
#define oss_CC_seq_copy_encoded(world, value_, length_) \
    (CC_seq *)oss__OctStrCC_copy_encoded(world, value_, length_)

/* allocates memory for the octet string with a ContentsConstraint applied and
 * initializes it by copying the decoded value */
#define oss_CC_seq_copy_decoded(world, decoded_) \
    (CC_seq *)oss__CC_copy_decoded(world, decoded_, CC_null_PDU)

/* sets encoded value of the unbounded octet string */
#define oss_CC_seq_set_encoded(outp, value_, length_) \
    { \
    (outp)->encoded.value = (value_); \
    (outp)->encoded.length = (length_); \
    (outp)->decoded = NULL; \
    }
```

```
/* sets decoded value of the type within CONTAINING clause */
#define oss_CC_seq_set_decoded(outp, decoded_) \
    (outp)->decoded = (decoded_)
```

7.6.5.3 Encoding types with contents constraints applied

If you want the encoder to automatically carry out the encoding of the contained type, you should set the `encoded` field of the generated structure to NULL and the `decoded` field to point to the decoded value of the contained type. If you do that, the encoder will encode the contained value using the encoding rules specified with the ENCODED BY notation (or current encoding rules in use if the ENCODED BY notation was omitted) and place it in the encoded field. If the encoding rules which corresponds to the OBJECT IDENTIFIER value used for ENCODED BY is not specified (either implicitly or explicitly) during ASN.1-compiling, then automatic encoding of contained types is not possible and the encoder issues the following error message:

`"E0103S: The requested encoding rules were not linked".`

If the automatic encoding of the `decoded` field is successful, the encoder will again encode the contained value but this time treating it as the containing OCTET STRING or BIT STRING. Thus, the `encoded` field will be encoded as a normal OCTET STRING value or BIT STRING value.

In the case when the CONTAINING keyword is present and the contents of the `encoded` field is not empty, the contents of the `decoded` field are ignored. In such a case, the encoder skips the automatic encoding of the contained type and immediately passes over to encoding the value of the encoded field. For example, you may manually encode the contained type and place it in the `encoded` field. Then when you call the encoder, it will simply encode your preset octet stream or bit stream as a BIT STRING or OCTET STRING type.

[Note that if the global or local `OSS.NOPDU` directive is applied to the contained type, or if the global `OSS.NoConstrain` directive is applied to the BIT STRING or OCTET STRING with a contents constraint, or when the `-noPdusForContainingTypes` compiler option is specified, automatic encoding/decoding will not be performed at runtime.]

7.6.5.4 Decoding types with contents constraints applied

Upon receipt of an encoded PDU corresponding to a type with the contents constraint applied, the decoder proceeds to decode the PDU as a normal OCTET STRING or BIT STRING value and places the result in the `encoded` field of the generated structure. Then, if the CONTAINING keyword is present, a second decode is automatically done on the contents of the encoded field using the encoding rules specified with the ENCODED BY clause and the result is placed in the `decoded` field of the structure generated to represent the type with the contents constraint, at the same time the encoder discards the contents of the `encoded` field and sets it to an empty value.

If the encoding rule that corresponds to the OBJECT IDENTIFIER value used for ENCODED BY clause is not linked during ASN.1-compiling then the automatic decoding will be impossible and the decoder will issue the following error message: `D0103S: The requested encoding rules were not linked.`

The above paragraph assumes that a full contents constraint with the CONTAINING and ENCODED BY keywords is used. If the CONTAINING keyword is omitted, then the second decode is not automatically done. If the ENCODED BY clause is omitted then the current encoded rules in use are employed.

7.6.5.5 Contents constraint value notation

Value notation of OCTET STRING and BIT STRING types with contents constraints is supported.

The value notation associated with a BIT STRING or OCTET STRING type defined with the contents constraint can either correspond to a value for the contained type or (if the ENCODED BY clause is present) correspond to a bit or octet stream. In the first case, the keyword CONTAINING can be inserted before the assigned value to indicate that the value corresponds to the contained type. For example, the following type definition:

```
A ::= OCTET STRING (
    CONTAINING INTEGER
    ENCODED BY
        {joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0)
aligned(0)}
)
```

allows for the following two value notation alternatives:

```
a1 A ::= '1A25'H -- specifies a value for containing OCTET STRING
a2 A ::= CONTAINING 22 -- specifies a value for the contained type
```

7.6.5.6 Current limitations on contents constraint implementation

1) The OCTET STRING or BIT STRING type in question must have the UNBOUNDED representation, which is the default. Applying POINTER or VARYING directives the OCTET STRING type or the POINTER, VARYING or PADDED directives to the BIT STRING type in question is not valid.

7.6.6 EMBEDDED PDV

The EMBEDDED PDV type is defined in ASN.1 as:

```
SEQUENCE {
  identification CHOICE {
    syntaxes SEQUENCE {
      abstract OBJECT IDENTIFIER,
      transfer OBJECT IDENTIFIER }
      -- Abstract and transfer syntax object identifiers --,
    syntax OBJECT IDENTIFIER
      -- A single object identifier for identification of the class and encoding --,
    presentation-context-id INTEGER
      -- (Applicable only to OSI environments)
      -- The negotiated presentation context identifies the class of the value and its encoding --,
    context-negotiation SEQUENCE {
      presentation-context-id INTEGER,
      transfer-syntax OBJECT IDENTIFIER }
      -- (Applicable only to OSI environments)
```

```
    -- Context-negotiation in progress for a context to identify the class of the value
    -- and its encoding --,
  transfer-syntax OBJECT IDENTIFIER
    -- The class of the value (for example, specification that it is the value of an ASN.1 type)
    -- is fixed by the application designer (and hence known to both sender and receiver). This
    -- case is provided primarily to support selective-field-encryption (or other encoding
    -- transformations) of an ASN.1 type --,
  fixed NULL
    -- The data value is the value of a fixed ASN.1 type (and hence known to both sender
    -- and receiver) -- },
 data-value-descriptor ObjectDescriptor OPTIONAL
    -- This provides human-readable identification of the class of the value --,
 data-value OCTET STRING }
( WITH COMPONENTS {
 ... ,
 data-value-descriptor ABSENT } )
```

The following specifications affect the representation of the ASN.1 EMBEDDED PDV type:

ASN.1 specifications:

None.

Directive specification without -helperNames:

OSS.POINTER

Directive specification with -helperNames:

OSS.HelperMacro
OSS.NoHelperMacro

By default without -helperNames, the C representation is:

ASN.1:

```
DefaultEPDV        ::= EMBEDDED PDV
```

C representation:

```
typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} ObjectID;

typedef struct EmbeddedPDV {
    struct {
        unsigned short  choice;
#           define      syntaxes_chosen 1
#           define      syntax_chosen 2
#           define      presentation_context_id_chosen 3
#           define      context_negotiation_chosen 4
#           define      transfer_syntax_chosen 5
#           define      fixed_chosen 6
        union {
            struct EmbeddedPDV_syntaxes {
                ObjectID        abstract;
                ObjectID        transfer;
            } syntaxes;   /* to choose, set choice to syntaxes_chosen */
```

```
                ObjectID        syntax;   /* to choose, set choice to syntax_chosen */
                int             presentation_context_id;   /* to choose, set
                                        choice to presentation_context_id_chosen */
                struct EmbeddedPDV_negotiation {
                    int             presentation_context_id;
                    ObjectID        transfer_syntax;
                } context_negotiation;   /* to choose, set choice to
                                        context_negotiation_chosen */
                ObjectID        transfer_syntax;   /* to choose, set choice
                                        to transfer_syntax_chosen */
                Nulltype        fixed;   /* to choose, set choice to fixed_chosen */
            } u;
        } identification;
        struct {
            unsigned int    length;
            unsigned char   *value;
        } data_value;
    } EmbeddedPDV;

    typedef EmbeddedPDV     DefaultEPDV;
```

By default with -helperNames, the C representation is:

ASN.1:

```
DefaultEPDV        ::= EMBEDDED PDV
```

C representation:

```
typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;

typedef struct EmbeddedPDV {
    struct EmbeddedPDV_identification *identification;
     _OctStr  *data_value;
} EmbeddedPDV;

typedef EmbeddedPDV     DefaultEPDV;

typedef struct EmbeddedPDV_syntaxes {
    struct _OID     *abstract;
    struct _OID     *transfer;
} EmbeddedPDV_syntaxes;

typedef struct EmbeddedPDV_negotiation {
    int             presentation_context_id;
    struct _OID     *transfer_syntax;
} EmbeddedPDV_negotiation;

typedef struct EmbeddedPDV_identification {
    unsigned short  choice;
#       define      syntaxes_chosen 1
```

```
#       define      syntax_chosen 2
#       define      presentation_context_id_chosen 3
#       define      context_negotiation_chosen 4
#       define      transfer_syntax_chosen 5
#       define      fixed_chosen 6
    union {
        struct EmbeddedPDV_syntaxes *syntaxes;  /* to choose, set choice
                                                 * to syntaxes chosen */
       struct _OID     *syntax;/*to choose, set choice to syntax_chosen */
        int             presentation_context_id;  /* to choose, set choice
                                    to presentation_context_id_chosen */
        struct EmbeddedPDV_negotiation *context_negotiation;  /* to
                    choose, set choice to context_negotiation_chosen */
        struct _OID     *transfer_syntax;  /* to choose, set choice to
                                             * transfer_syntax_chosen */
        Nulltype        fixed; /* to choose, set choice to fixed_chosen */
    } u;
} EmbeddedPDV_identification;
```

Effect of OSS.HelperMacro on the EXTERNAL type with -helperNames:

The EMBEDDED PDV type is defined in ASN.1 as a SEQUENCE type containing CHOICE, OCTET STRING and other types. Correspondingly, helper macros for it are produced in accordance with the rules for SEQUENCE, CHOICE, OCTET STRING etc.

7.6.7 ENUMERATED

The following specifications affect the representation of the ASN.1 ENUMERATED type:

ASN.1 specification:

Value reference in the NamedNumber of the enumeration

Directive specification:

OSS.POINTER
OSS.PrintfunctionName

Directive specification with -helperNames for ENUMERATED types with POINTER directive or marked as PDUs:

OSS.HelperMacro
OSS.NoHelperMacro

By default, the C-representation of ENUMERATED is of the form:

```
enum Name1 {
    identifier1=value1,
    identifier2=value2,
    .
    .
    .
```

```
        identifiern=valuen
    } Name1;
```

The identifiers (e.g., `identifier1`) are the identifiers associated with each ASN.1 NamedNumber in the ENUMERATED type, and the values (e.g., `value1`) are the numeric values associated with the NamedNumbers.

ASN.1:

```
Colors ::= ENUMERATED {red(1), white(2), blue(3)}
```

C representation:

```
typedef enum Colors {
    red = 1,
    white = 2,
    blue = 3
} Colors;
```

Effect on the ENUMERATED type of a value reference in the NamedNumber

If an identifier of a NamedNumber in the ENUMERATED type has an associated value reference instead of a SignedNumber, the value reference will be represented in C by the SignedNumber that it signifies:

ASN.1:

```
Colors ::= ENUMERATED {red(1), white(2), blue(purple)}
purple INTEGER ::= 3
```

C representation:

```
typedef enum Colors {
    red = 1,
    white = 2,
    blue = 3
} Colors;

extern int purple;
```

Associated C initialization:

```
int purple = 3;
```

Note that the value reference, `purple`, was replaced by `3` in the C representation.

NOTE: When assigning values to enumerated types, you should always use the defined enumerator and not the integer equivalent. For example use:

```
Color myColor = blue;
```

and not:

```
Color myColor = 3;
```

Effect on the ENUMERATED type of a unknown ENUMERATED value

Note that when storing unknown ENUMERATED types in relay-safe mode, the value is preserved in enciphered form which is represented locally as `(MAXINT-`*`enumValue`*`)`. This means that:

a) unknown enumerated values should not be encoded,

b) unknown enumerated values may, however, be decoded and re-encoded (for relay purposes),

c) unknown enumerated values which are decoded will have the enciphered form of the value which appears in the C structure. This value (MAXINT-n) is not intended for interpretation, but only for re-encoding for relaying. You should not try to interpret it.

Effect of OSS.HelperMacro on the ENUMERATED type

The OSS.HelperMacro directive applied to an ENUMERATED results in generating the following helper macros:

1) _new or _new_pdu macro to allocate memory for the value (memory stays uninitialized)
2) _copy or _copy_pdu macro to allocate memory and copy input data to it

Macros for the ENUMERATED type are produced only if this type is a PDU or it is referenced by pointer. If the ENUMERATED type is a non-typereference field of a SET or SEQUENCE, then its parent type should have its own macros produced in order to produce macros for the ENUMERATED. An example follows.

ASN.1:

```
--<OSS.HelperMacro Mod.EnumHM>--
--<OSS.HelperMacro Mod.SetEnumHM>--
--<OSS.HelperMacro Mod.SetEnumHM.en>--
Mod DEFINITIONS ::= BEGIN
EnumHM  ::= ENUMERATED {first (1), second (2)} --<PDU>--
SetEnumHM  ::= SET {
    en ENUMERATED {one(1), two(2), three (3)} --<POINTER>--
}
END
```

Helper macro:

```
typedef enum EnumHM {
    first = 1,
    second = 2
} EnumHM;

/* allocates memory for EnumHM_PDU */
#define oss_EnumHM_new_pdu(world) \
    (EnumHM *)ossGetMemory(world, sizeof(EnumHM))

/* allocates memory for EnumHM_PDU and initializes it by given value */
#define oss_EnumHM_copy_pdu(world, value_) \
    (EnumHM *)oss__UInt_copy(world, (unsigned int)value_)

typedef enum SetEnumHM_en {
    one = 1,
    two = 2,
    three = 3
} SetEnumHM_en;

/* allocates memory for an instance of the enumerated type */
#define oss_SetEnumHM_en_new(world) \
    (SetEnumHM_en *)ossGetMemory(world, sizeof(enum SetEnumHM_en))
```

```
/* allocates memory for an instance of the enumerated type and
 * initializes it by given value */
#define oss_SetEnumHM_en_copy(world, value_) \
    (SetEnumHM_en *)oss__UInt_copy(world, (unsigned int)value_)

typedef struct SetEnumHM {
    SetEnumHM_en    *en;
} SetEnumHM;

…
```

7.6.8 EXTERNAL

The EXTERNAL type is defined in ASN.1 as:

```
SEQUENCE {
  identification CHOICE {
    syntaxes SEQUENCE {
      abstract OBJECT IDENTIFIER,
      transfer OBJECT IDENTIFIER }
      -- Abstract and transfer syntax object identifiers --,
    syntax OBJECT IDENTIFIER
      -- A single object identifier for identification of the class and encoding --,
    presentation-context-id INTEGER
      -- (Applicable only to OSI environments)
      -- The negotiated presentation context identifies the class of the value and its encoding --,
    context-negotiation SEQUENCE {
      presentation-context-id INTEGER
      transfer-syntax OBJECT IDENTIFIER }
      -- (Applicable only to OSI environments)
      -- Context-negotiation in progress for a context to identify the class of the value
      -- and its encoding --,
    transfer-syntax OBJECT IDENTIFIER
      -- The class of the value (for example, specification that it is the value of an ASN.1 type)
      -- is fixed by the application designer (and hence known to both sender and receiver). This
      -- case is provided primarily to support selective-field-encryption (or other encoding
      -- transformations) of an ASN.1 type --,
    fixed NULL
      -- The data value is the value of a fixed ASN.1 type (and hence known to both sender
      -- and receiver) -- },
    data-value-descriptor ObjectDescriptor OPTIONAL
      -- This provides human-readable identification of the class of the value --,
    data-value OCTET STRING } }
  (WITH COMPONENTS {
    ... ,
    identification (WITH COMPONENTS {
      ... ,
      syntaxes ABSENT,
      transfer-syntax ABSENT,
      fixed ABSENT } ) } )
```

The following specifications affect the representation of the ASN.1 EXTERNAL type:

ASN.1 specifications:

None.

Directive specification without -helperNames:

OSS.POINTER

Directive specification with -helperNames:

OSS.HelperMacro
OSS.NoHelperMacro

By default without -helperNames, the C representation is:

ASN.1:

```
MyExternal        ::= EXTERNAL
```

C representation:

```
typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} ObjectID;

typedef struct External {
    unsigned char   bit_mask;
#       define      direct_reference_present 0x80
#       define      indirect_reference_present 0x40
    ObjectID        direct_reference;   /* optional; set in bit_mask
                                           direct_reference_present if present */
    int             indirect_reference; /* optional; set in bit_mask
                                           indirect_reference_present if  present */
    char            *data_value_descriptor;   /* NULL for not present */
    struct {
        unsigned short  choice;
#           define      single_ASN1_type_chosen 1
#           define      octet_aligned_chosen 2
#           define      arbitrary_chosen 3
        union {
            OpenType        single_ASN1_type;   /* to choose, set choice to
                                                   single_ASN1_type_chosen */
            struct External_octet_aligned {
                unsigned int    length;
                unsigned char   *value;
            } octet_aligned;  /* to choose, set choice to octet_aligned_chosen */
            struct External_arbitrary {
                unsigned int    length;   /* number of significant bits */
                unsigned char   *value;
            } arbitrary;   /* to choose, set choice to arbitrary_chosen */
        } u;
    } encoding;
} External;
```

By default with -helperNames, the C representation is:

ASN.1:

```
MyExternal        ::= EXTERNAL
```

C representation:

```
typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

typedef struct _BitStr {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} _BitStr;

typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;

typedef struct External {
    unsigned char   bit_mask;
#       define      indirect_reference_present 0x80
    struct _OID     *direct_reference;  /* NULL for not present */
    int             indirect_reference;  /* optional; set in bit_mask
                                          * indirect_reference_present if
                                          * present */
    char            *data_value_descriptor;  /* NULL for not present */
    struct External_encoding *encoding;
} External;

typedef External        MyExternal;

typedef _OctStr         External_octet_aligned;

typedef _BitStr         External_arbitrary;

typedef struct External_encoding {
    unsigned short  choice;
#       define      single_ASN1_type_chosen 1
#       define      octet_aligned_chosen 2
#       define      arbitrary_chosen 3
    union {
        OpenType        *single_ASN1_type;  /* to choose, set choice to
                                             * single_ASN1_type_chosen */
        External_octet_aligned *octet_aligned;  /* to choose, set choice to
                                                 * octet_aligned_chosen */
        External_arbitrary *arbitrary; /* to choose, set choice to
                                        * arbitrary_chosen */
    } u;
} External_encoding;
```

Usage Notes:

The C representation of the components of the EXTERNAL type is affected by global directives. For example, the representation for ObjectID can be altered by the global OSS.UNBOUNDED directive.

EXTERNAL is used whenever there is a need to carry a value that uses an entirely different encoding scheme other than the one in use. For example, if you are using BER and the value that you wish to carry is encoded using PER, you cannot simply switch to PER from one field to the next. Instead, you would have to encode the

value in PER and then put the encoded value in the `External.encoding` field. Subsequently, you would set `direct_reference` or `indirect_reference` to indicate what the type/encoding of the value in `External.encoding` is of.

Another case when in which you would use EXTERNAL is when the application that you are implementing has a "container" type. For example, suppose you have a message which is capable of carrying a value of any ASN.1 specification in existence, or one which has not been dreamed of yet. How would you encode it since you don't know what the ASN.1 definition of it is? You do so by having some other application that knows the ASN.1 definition of the type do the encoding. Then, they pass you the value in the already encoded form for you to put into the `External.encoding` field. They would also provide you with a `direct_reference` or an `indirect_reference` which indicates to the communications peer what the type of the value is that is contained in `External.encoding`.

Effect of OSS.HelperMacro on the EXTERNAL type

The EXTERNAL type is defined in ASN.1 as a SEQUENCE type containing CHOICE, OCTET STRING and other types. Correspondingly, helper macros for it are produced in accordance with the rules for SEQUENCE, CHOICE, OCTET STRING etc.

7.6.8.1 Customizing the representation of the EXTERNAL type

In the event that the above C representation of the EXTERNAL type does not suit your needs, you may create a custom representation of EXTERNAL. To do so, create a type reference, External, whose tag is UNIVERSAL 8, and whose elements mirror those of the UNIVERSAL type as defined in ISO 8824 | X.208. You may then add whatever local directives you wish to the individual elements. For example:

```
External ::= [UNIVERSAL 8] IMPLICIT SEQUENCE {
    direct-reference          OBJECT IDENTIFIER --<OBJECTID 10>-- OPTIONAL,
    indirect-reference        INTEGER OPTIONAL,
    data-value-descriptor     ObjectDescriptor OPTIONAL,
    encoding CHOICE {
        single-ASN1-type   [0] ANY --<LINKED>--,
        octet-aligned      [1] IMPLICIT OCTET STRING,
        arbitrary          [2] IMPLICIT BIT STRING}}
```

This results in the following reminder that may be safely ignored:

```
A0307W: OSS has relaxed its implementation of the standards to allow the
definition of a type with tag [UNIVERSAL 8].  This is normally invalid ASN.1.
```

The OSS ASN.1 compiler does not allow you to explicitly specify the UNIVERSAL class in any other context.

7.6.9 GeneralizedTime

The following specifications affect the representation of the ASN.1 GeneralizedTime type:

ASN.1 specifications:

None.

Directive specification:

OSS.NULLTERM
OSS.POINTER
OSS.PrintfunctionName
OSS.TIMESTRUCT

Directive specification with -helperNames:

OSS.NULLTERM
OSS.PrintfunctionName
OSS.TIMESTRUCT
OSS.HelperMacro
OSS.NoHelperMacro

By default, the C representation is:

ASN.1:

```
Generaltime     ::= GeneralizedTime
```

C representation if -helperNames is not specified:

```
typedef GeneralizedTime Generaltime;
```

where `GeneralizedTime` is defined in file `asn1hdr.h` as

```
typedef struct {
  short          year;      /* YYYY format when used for GeneralizedTime */
                            /* YY format when used for UTCTime */
  short          month;
  short          day;
  short          hour;
  short          minute;
  short          second;
  short          millisec;
  short          mindiff;      /* UTC +/- minute differential     */
  ossBoolean     utc;          /* TRUE means UTC time             */
} GeneralizedTime;
```

The GeneralizedTime type can be represented either as GeneralizedTime structs (i.e., { short year; short month; etc }) or as strings. Strings are used whenever the -lean option or -helperNames option or the NULLTERM directive is used.

Effect of NULLTERM on the GeneralizedTime type

When the NULLTERM directive is used or the -lean option or the -helperNames option is specified, the GeneralizedTime type is represented as strings.

ASN.1:

```
Generaltime     ::= GeneralizedTime --<NULLTERM>--
t Generaltime ::= "19851106210627.3-0500"
-- local time 6 minutes, 27.3 seconds after 9 pm on 6 November 1985 with
-- local time 5 hours retarded in relation to coordinated universal time
```

C representation:

```
typedef char            *Generaltime;

Generaltime t = "19851106210627.3-0500";
```

Effect of OSS.HelperMacro on the GeneralizedTime type

The OSS.HelperMacro directive applied to a GeneralizedTime with NULLTERM representation results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the string of given length and initialize it to zeroes
2) `_copy` or `_copy_pdu` macro to allocate memory and copy given input null terminated time value to it

The OSS.HelperMacro directive applied to a GeneralizedTime with TIMESTRUCT directive results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the time structure and initialize it to zeroes
2) `_copy_nullterm` or `_copy_nullterm_pdu` macro to allocate memory for the time structure and initialize it by parsing a string containing a time value
3) `_setv_nullterm` macro to initialize existing time structure by parsing a string containing a time value

ASN.1:

```
--<OSS.HelperMacro Mod.SeqTime>--
--<OSS.HelperMacro Mod.SeqTime.n-time>--
--<OSS.TIMESTRUCT  Mod.SeqTime.s-time>--
--<OSS.HelperMacro Mod.SeqTime.s-time>--
Mod DEFINITIONS ::= BEGIN
SeqTime ::= SEQUENCE {
    n-time GeneralizedTime,
    s-time GeneralizedTime
}
END
```

Helper macros:

```
typedef struct SeqTime {
    char            *n_time;
    GeneralizedTime *s_time;
} SeqTime;

…

/* allocates memory for a string of given length */
#define oss_SeqTime_n_time_new(world, length_) \
    oss__CharStr_new(world, length_)

/* allocates memory and returns a copy of an input string */
#define oss_SeqTime_n_time_copy(world, value_) \
    oss__CharStr_copy(world, value_)
```

```
…

/* allocates memory for the time structure */
#define oss_SeqTime_s_time_new(world) \
    (GeneralizedTime *)ossGetInitializedMemory(world, sizeof(GeneralizedTime))

/* allocates memory for the time structure and initializes it by parsing a
 * string containing a GeneralizedTime value */
#define oss_SeqTime_s_time_copy_nullterm(world, value_) \
    oss__GeneralizedTime_copy_nullterm(world, value_)

/* initializes the structure by parsing a string containing a GeneralizedTime
 * value */
#define oss_SeqTime_s_time_setv_nullterm(world, outp, value_) \
    oss__GeneralizedTime_setv_nullterm(world, outp, value_)
```

Also refer to the following notebox for more information about the GeneralizedTime type:

Note: : As specified in ITU-T Rec. X.680 | ISO/IEC 8824-1, and indicated in the above comments, `year` is to be specified in YYYY format. The time differential, `mindiff`, indicates the minute differential from Coordinated Universal Time (UTC time or Greenwich Mean Time); `mindiff` may have either a positive or negative value. The minute differential can have an absolute value greater than 59, so to express a time differential of one hour and thirty minutes, you would set `mindiff` to 90. A `utc` value of `TRUE` indicates that the time contained in the `GeneralizedTime` structure is the UTC time. If `utc` is `TRUE`, `mindiff` is ignored. If `utc` is `FALSE`, and `mindiff` contains zero, the time in the GeneralizedTime structure is the local time. If `utc` is `FALSE`, and `mindiff` is non-zero, the hour, minute, and second minus the minute differential yields

7.6.10 INSTANCE OF

The INSTANCE OF type is equivalent to an EXTERNAL type that is used to specify an attribute-id / attribute-value pair.

The following specifications affect the representation of the ASN.1 INSTANCE OF type:

ASN.1 specifications:

None.

Directive specifications:

None. The only directive allowed on the INSTANCE OF type is OSS.OBJHANDLE and it does not affect the C representation in the header file.

By default, the C representation is:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   MHS-BODY     ::= TYPE-IDENTIFIER
   InstanceOfMHS        ::= INSTANCE OF MHS-BODY
END
```

where TYPE-IDENTIFIER is internally defined as:

```
TYPE-IDENTIFIER ::= CLASS
{
   &id OBJECT IDENTIFIER UNIQUE,
   &Type
}
```

and `IntanceOfMHS` has the following implied associated SEQUENCE:

```
InstanceOfMHS ::= SEQUENCE
{
   type-id        MHS-BODY.&id,
   value      [0] MHS-BODY.&Type
}
```

C representation without -helperNames:

```
#define          InstanceOfMHS_PDU 1

typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} ObjectID;

typedef struct InstanceOfMHS {
    ObjectID        type_id;
    OpenType        value;
} InstanceOfMHS;
```

C representation with -helperNames:

```
#define          InstanceOfMHS_PDU 1

typedef struct _OID {
    unsigned int  length;
    unsigned char   *value;
} _OID;

typedef struct InstanceOfMHS {
    struct _OID     *type_id;
    OpenType        *value;
} InstanceOfMHS;
```

Refer to section 7.6.16 for more information about the `OpenType` type.

7.6.11 INTEGER

The following specifications affect the representation of the ASN.1 INTEGER type:

ASN.1 specifications:

NamedNumberList
SingleValue subtype

ValueRange subtype

Directive specifications:

ASN1.HugeInteger / OSS.HUGE
OSS.INT
OSS.LONG
OSS.LONGLONG
OSS.POINTER
OSS.PrintfunctionName
OSS.SHORT

Additional directive specifications with -helperNames for INTEGER types without HUGE directive and with POINTER directive or marked as PDUs:

OSS.HelperMacro
OSS.NoHelperMacro

By default, the INT representation is used:

ASN.1:

```
RegInt       ::= INTEGER
```

C representation:

```
typedef int             RegInt;
```

The OSS.LONGLONG directive should be used only if the target platform supports the `long long int` type.

Effect of NamedNumberList on the INTEGER type

When a NamedNumberList is present, the compiler generates an integer type, and a manifest constant for each identifier present.

ASN.1:

```
NumLstInt    ::= INTEGER {one(1), two(2), three(3)}
```

C representation:

```
typedef int             NumLstInt;
#define                     one 1
#define                     two 2
#define                     three 3
```

Effect of SingleValue on the INTEGER type

The compiler uses the SingleValue subtype to determine the number of bytes needed to hold the smallest or largest allowed integer value; a `short`, `int`, `long`, or `long long` is generated.

ASN.1:

```
SngValInt    ::= INTEGER (1|2|-1234567890)
```

C representation:

```
typedef int             SngValInt;
```

Effect of ValueRange on the INTEGER type

The compiler uses the ValueRange subtype to determine the number of bytes needed to hold the smallest or largest allowed integer value; a `short`, `int`, `long`, or `long long` is generated.

ASN.1:

```
SizeInt      ::= INTEGER (1..123456789)
```

C representation:

```
typedef unsigned int    SizeInt;
```

Effect of ASN1.HugeInteger / OSS.HUGE on the INTEGER type

Refer to sections 4.4.1.3 and 4.4.2.24 for more information on the effect of these directives on the INTEGER type.

Effect of OSS.SHORT / OSS.INT / OSS.LONG / LONGLONG on the INTEGER type

In the absence of the SingleValue/ValueRange subtypes or a NamedNumberList, the representation of INTEGER is determined by the OSS.SHORT/OSS.INT/OSS.LONG/LONGLONG directive which can be global or local. The global default is OSS.INT; it can be overridden locally, at the module-level, or globally by OSS.SHORT, OSS.LONG, or OSS.LONGLONG.

ASN.1:

```
LongInt      ::= INTEGER --<LONG>--
```

C representation:

```
typedef long            LongInt;
```

Illustrations of specification combinations:

Effect of SingleValue / ValueRange and OSS.SHORT / OSS.INT / OSS.LONG / LONGLONG on the INTEGER type

When a ValueRange or SingleValue subtype conflicts with a local OSS.SHORT / OSS.INT / OSS.LONG / LONGLONG directive (i.e., the subtype value is too large/small to fit in the type indicated by the directive), the directive takes precedence and the compiler issues a warning similar to the following:

```
C0255W: Number (123456789) is too large to fit in 2 bytes. It is being
truncated to 65535.
```

ASN.1:

```
ShrtSzInt    ::= INTEGER (1..123456789) --<SHORT>--
```

C representation:

```
typedef unsigned short  ShrtSzInt;
```

Effect of OSS.HelperMacro on the INTEGER type without HUGE directive

The OSS.HelperMacro directive applied to an INTEGER results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the value (memory stays uninitialized)
2) `_copy` or `_copy_pdu` macro to allocate memory and copy input integer value to it

Macros for an INTEGER type are produced only if this type is a PDU or it is referenced by pointer. If an INTEGER type is a non-typereference field of a SET or SEQUENCE, then its parent type should have its own macros produced in order to produce macros for the INTEGER. An example follows.

ASN.1:

```
--<OSS.HelperMacro Mod.IntHM>--
--<OSS.HelperMacro Mod.SetIntHM>--
--<OSS.HelperMacro Mod.SetIntHM.integ>--
Mod DEFINITIONS ::= BEGIN
IntHM  ::= INTEGER --<LONGLONG|PDU>--
SetIntHM  ::= SET {
    integ INTEGER (0..16384) --<SHORT|POINTER>--
}
END
```

Helper macro:

```
typedef LONG_LONG       IntHM;

/* allocates memory for IntHM_PDU */
#define oss_IntHM_new_pdu(world) \
    (IntHM *)ossGetMemory(world, sizeof(IntHM))

/* allocates memory for IntHM_PDU and initializes it by given value */
#define oss_IntHM_copy_pdu(world, value_) \
    oss__LongLong_copy(world, value_)

typedef struct SetIntHM {
    unsigned short  *integ;
} SetIntHM;

…

/* allocates memory for an instance of the integer type */
#define oss_SetIntHM_integ_new(world) \
    (unsigned short *)ossGetMemory(world, sizeof(unsigned short))

/* allocates memory for an instance of the integer type and initializes it by
 * given value */
#define oss_SetIntHM_integ_copy(world, value_) \
    oss__UShort_copy(world, value_)
```

7.6.12 NULL

The following specifications affect the representation of the ASN.1 NULL type:

ASN.1 specifications:
None.

Directive specification:
OSS.POINTER

Directive specification with -helperNames for NULL types with POINTER directive or marked as PDUs:
OSS.HelperMacro
OSS.NoHelperMacro

By default, the C representation is:

ASN.1:

```
Name1 ::= NULL
```

C representation:

```
typedef Nulltype Name1;
```

where `Nulltype` is defined in file `asn1hdr.h` as

```
typedef char Nulltype;
```

Effect of OSS.HelperMacro on the NULL type

The OSS.HelperMacro directive applied to NULL results in generating the `_new` or `_new_pdu` macro to allocate memory for the value (memory stays uninitialized because its contents are not taken into account by the OSS runtime when encoding or decoding). Macros for the NULL type are produced only if this type is a PDU or it is referenced by pointer. An example follows.

ASN.1:

```
N ::= NULL --<PDU>--
```

Helper macro:

```
typedef Nulltype        N;

/* allocates memory for N_PDU */
#define oss_N_new_pdu(world) \
    (N *)ossGetMemory(world, sizeof(N))
```

7.6.13 OBJECT IDENTIFIER

The following specifications affect the representation of the ASN.1 OBJECT IDENTIFIER type:

ASN.1 specifications:

None.

Directive specifications without -helperNames:

OSS.ARRAY
OSS.LINKED
OSS.OBJECTID *length*
OSS.OBJECTID ENCODED / ENCODED
OSS.POINTER
OSS.SHORT / OSS.INT / OSS.LONG (when applied locally)
OSS.UNBOUNDED

Directive specification with -helperNames:

OSS.ENCODED
OSS.HelperMacro
OSS.NoHelperMacro

By default, the ENCODED representation is used:

ASN.1:

```
DefaultOID   ::= OBJECT IDENTIFIER
```

C representation without -helperNames:

```
typedef struct ObjectID {
    unsigned short  length;
    unsigned char   *value;
} ObjectID;

typedef ObjectID        DefaultOID;
```

C representation with -helperNames:

```
typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

typedef _OID            DefaultOID;
```

The ENCODED and OSS.OBJECTID ENCODED directives allow the representation of the OBJECT IDENTIFIER type as a structure which contains a length in octets and the address of an array of characters containing the BER-encoded contents of the OBJECT IDENTIFIER (e.g., {1 2 3 4 5} would be stored as: {0x2A, 0x03, 0x04, 0x05}). This permits the representation of OBJECT IDENTIFIER types with node values exceeding the maximum integer representation on a given machine. The encoded value is derived according to the **ITU-T Rec.X.690 (2008)** document.

Effect of the OSS.ARRAY directive on the OBJECT IDENTIFIER type

If the OSS.ARRAY directive alone is specified, then an array of `short` integers is generated, preceded by a count field of the number of occurrences that follow. The OSS.POINTER directive is implied when the OSS.ARRAY directive is used but the OSS.OBJECTID directive is missing.

ASN.1:

```
ArrayOID     ::= OBJECT IDENTIFIER --<ARRAY>--
```

C representation:

```
typedef struct ArrayOID {
    unsigned short  count;
    unsigned short  value[1];
} *ArrayOID;
```

Under this representation, the `value` field contains an array of `short` integers containing the individual node values of the OBJECT IDENTIFIER.

Effect of the OSS.LINKED directive on the OBJECT IDENTIFIER type

If the OSS.LINKED directive alone is specified, then a pointer to linked list of values is generated.

ASN.1:

```
LinkedOID    ::= OBJECT IDENTIFIER --<LINKED>--
```

C representation:

```
typedef struct LinkedOID {
    struct LinkedOID *next;
    unsigned short  value;
} *LinkedOID;
```

Under this representation, each successive `value` field contains the corresponding node value of the OBJECT IDENTIFIER.

Effect of the OSS.OBJECTID directive followed by a *length* operand on the OBJECT IDENTIFIER type

If the OSS.OBJECTID directive is specified along with a length operand, then a fixed-size character string is generated preceded by a `length` field giving the size of the character string.

ASN.1:

```
ObjIdOID     ::= OBJECT IDENTIFIER --<OBJECTID 8>--
```

C representation:

```
typedef struct ObjIdOID {
    unsigned short  length;
    unsigned char   value[8];
} ObjIdOID;
```

Under this representation, the `value` field contains the BER-encoded contents of the OBJECT IDENTIFIER (e.g., `{1 2 3 4 5}` would be stored as: `{0x2A, 0x03, 0x04, 0x05}`).

Effect of the OSS.SHORT/OSS.INT/OSS.LONG local directives on the OBJECT IDENTIFIER type

The OSS.SHORT, OSS.INT, or OSS.LONG local directives do not affect the representation of the OBJECT IDENTIFIER type if the ENCODED representation is used (which is the default). For example:

ASN.1:

```
LongOID      ::= OBJECT IDENTIFIER --<LONG>--
```

C representation:

```
typedef struct LongOID {
    unsigned short  length;
    unsigned char   *value;
} LongOID;
```

However when the UNBOUNDED, ARRAY, or LINKED representation is being used for the OBJECT IDENTIFIER type, then these local directives do have effect. For example:

ASN.1:

```
UnboundedLongOID ::= OBJECT IDENTIFIER --<UNBOUNDED|LONG>--
```

C representation:

```
typedef struct UnboundedLongOID {
    unsigned short  count;
    unsigned long   *value;
} UnboundedLongOID;
```

Effect of the OSS.UNBOUNDED directive on the OBJECT IDENTIFIER type

If the OSS.UNBOUNDED directive alone is specified, a pointer to an array of `short` integers is generated, preceded by a `count` field containing the number of elements in the `value` array.

ASN.1:

```
UnboundedOID ::= OBJECT IDENTIFIER --<UNBOUNDED>--
```

C representation:

```
typedef struct UnboundedOID {
    unsigned short  count;
    unsigned short  *value;
} UnboundedOID;
```

Under this representation, the `value` field contains an array of `short` integers containing the individual node values of the OBJECT IDENTIFIER.

Illustrations of specification combinations:

Effect of OSS.OBJECTID with OSS.ARRAY on the OBJECT IDENTIFIER type

When the OSS.ARRAY directive is combined with OSS.OBJECTID the ARRAY representation is used. A `short` integer always gets generated for the count field in the ARRAY representation of an OBJECT IDENTIFIER.

ASN.1:

```
OIDArrayOID  ::= OBJECT IDENTIFIER --<OBJECTID 8 | ARRAY>--
```

C representation:

```
typedef struct OIDArrayOID {
    unsigned short  count;
    unsigned short  value[8];
} OIDArrayOID;
```

Effect of OSS.OBJECTID with OSS.UNBOUNDED on the OBJECT IDENTIFIER type

When the OSS.UNBOUNDED directive is combined with OSS.OBJECTID the UNBOUNDED representation is used. A `short` integer always gets generated for the `count` field in the UNBOUNDED representation of an OBJECT IDENTIFIER.

ASN.1:

```
OIDUnbndOID  ::= OBJECT IDENTIFIER --<OBJECTID 10 | UNBOUNDED>--
```

C representation:

```
typedef struct OIDUnbndOID {
    unsigned short  count;
    unsigned short  *value;
} OIDUnbndOID;
```

Effect of OSS.OBJECTID with OSS.LINKED and OSS.POINTER on the OBJECT IDENTIFIER type

OSS.OBJECTID has no effect on the OSS.LINKED representation. OSS.POINTER simply introduces an additional level of indirection.

ASN.1:

```
OIDLnkPtrOID ::= OBJECT IDENTIFIER --<OBJECTID 8 | LINKED | POINTER>--
```

C representation:

```
typedef struct OIDLnkPtrOID {
    struct OIDLnkPtrOID *next;
```

```
        unsigned short  value;
    } **OIDLnkPtrOID;
```

Effect of OSS.HelperMacro on the OBJECT IDENTIFIER type with -helperNames:
The OSS.HelperMacro directive applied to an OBJECT IDENTIFIER results in generating the following helper macros:

1) _new or _new_pdu macro to allocate an empty structure
2) _copy or _copy_pdu macro to create a structure and copy input data to it
3) _setv macro to fill in an existing structure with input data

Note that 'main' helper macros are produced for generated _OID types and then referenced by macros for each user-defined OBJECT IDENTIFIER type. The below example demonstrates this:

ASN.1:

```
MyOID ::= OBJECT IDENTIFIER
```

Helper macros:

```
typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

/* allocates memory for an empty structure */
#define oss__OID_new(world) \
    (_OID *)ossGetInitializedMemory(world, sizeof(_OID))

/* allocates memory for the structure and initializes it by copying the input
 * value */
#define oss__OID_copy(world, value_, length_) \
    (_OID *)oss__UnbCharString_copy(world, (char *)value_, length_)

/* initializes 'value' and 'length' fields of a structure by given values */
#define oss__OID_setv(outp, value_, length_) \
    { \
    (outp)->value = (value_); \
    (outp)->length = (length_); \
    }

typedef _OID            MyOID;

/* allocates memory for MyOID_PDU */
#define oss_MyOID_new_pdu(world) \
    oss__OID_new(world)

/* allocates memory for MyOID_PDU and initializes it by copy of an input
 * string */
#define oss_MyOID_copy_pdu(world, value_, length_) \
    oss__OID_copy(world, value_, length_)

/* initializes 'value' and 'length' fields of a structure by given values */
#define oss_MyOID_setv(outp, value_, length_) \
    oss__OID_setv(outp, value_, length_)
```

7.6.14 ObjectDescriptor

Since the ObjectDescriptor type is simply a tagged GraphicString, it has been dealt with in section 7.6.19.1 with the other restricted character strings.

7.6.15 OCTET STRING

The following specifications affect the representation of the ASN.1 OCTET STRING type:

ASN.1 specification:

SizeConstraint subtype

Directive specifications without -helperNames:

OSS.OBJHANDLE
OSS.POINTER
OSS.PrintfunctionName
OSS.UNBOUNDED
OSS.VARYING
OSS.PADDED

Directive specifications with -helperNames:

OSS.HelperMacro
OSS.NoHelperMacro
OSS.OBJHANDLE
OSS.PrintfunctionName
OSS.UNBOUNDED

Note: Note that the OSS.VARYING and OSS.UNBOUNDED directives are mutually exclusive of each other, as are OSS.LINKED and OSS.DLINKED.

By default, the UNBOUNDED representation is used:

ASN.1:

```
DefaultOctStr        ::= OCTET STRING
```

C representation:

```
typedef struct DefaultOctStr {
    unsigned int    length;
    unsigned char   *value;
} DefaultOctStr;
```

Under this representation, the `value` field contains an array of 8-bit characters which form the OCTET STRING value. The `length` field hold the size of the `value` character array.

Effect of SizeConstraint on the OCTET STRING type

The compiler uses the SizeConstraint subtype to determine the maximum number of bytes allowed in an OCTET STRING. In the absence of directives, the VARYING representation is generated. Specifically, an `unsigned short`, `int`, or `long` (depending on the string's maximum length) is generated to indicate the length of the string, followed by an `unsigned char` array to hold the value.

ASN.1:

```
SizeOctStr           ::= OCTET STRING (SIZE(10))
```

C representation:

```
typedef struct SizeOctStr {
    unsigned short  length;
    unsigned char   value[10];
} SizeOctStr;
```

Note that starting with version 5.0.1, the OCTET STRING type is represented using the UNBOUNDED representation whenever a size constraint larger than 256 is used. Refer to section 3.5.91 for more details. If you would like the VARYING representation be generated for such OCTET STRING types, you should use the OSS.VARYING directive or the `-compat v4.2octetStringDefault` command-line option.

Effect of the OSS.VARYING directive on the OCTET STRING type

If the OSS.VARYING directive is specified alone, the result is a pointer to a structure containing a `length` field of type `short` and an `unsigned char` array of unspecified length. The effect is the same as when the OSS.VARYING and OSS.POINTER directives are specified together (i.e., OSS.POINTER is assumed because the array length is indeterminable by the compiler).

ASN.1:

```
VaryingOctStr        ::= OCTET STRING --<VARYING>--
```

C representation:

```
typedef struct VaryingOctStr {
    unsigned short  length;
    unsigned char   value[1];  /* first element of the array */
} *VaryingOctStr;
```

Effect of the OSS.PADDED directive on the OCTET STRING type

The OSS.PADDED directive requires the presence of the SizeConstraint subtype, because the compiler must be able to determine the length of the OCTET STRING. If the OSS.PADDED directive is specified without SizeConstraint, the compiler issues an error message.

If the directive is used globally as `--<OSS.PADDED OCTET STRING>--`, then it will affect all OCTET STRING types constrained to a fixed size.

ASN.1:

```
PaddedOctStr  ::= OCTET STRING (SIZE(10)) --<PADDED>--
```

C representation:

```
typedef unsigned char  PaddedOctStr[10];
```

Illustrations of specification combinations

Effect of SizeConstraint with OSS.UNBOUNDED on the OCTET STRING type

When the OSS.UNBOUNDED directive is combined with SizeConstraint the UNBOUNDED representation is used, but the `length` field varies depending on the size needed to accommodate the longest possible octet string.

ASN.1:

```
SizeUnbndOctStr      ::= OCTET STRING (SIZE(1..7)) --<UNBOUNDED>--
```

C representation:

```
typedef struct SizeUnbndOctStr {
    unsigned short  length;
    unsigned char   *value;
} SizeUnbndOctStr;
```

Effect of SizeConstraint with OSS.VARYING on the OCTET STRING type

When the OSS.VARYING directive is combined with SizeConstraint the VARYING representation is used, but the `length` field varies depending on the size needed to accommodate the longest possible octet string.

ASN.1:

```
SizeVaryOctStr       ::= OCTET STRING (SIZE(50000)) --<VARYING>--
```

C representation:

```
typedef struct SizeVaryOctStr {
    unsigned short  length;
    unsigned char   value[50000];
} SizeVaryOctStr;
```

Effect of OSS.HelperMacro on the OCTET STRING type with -helperNames:

The OSS.HelperMacro directive applied to an OCTET STRING results in generating the following helper macros:

1) `_new` or `_new_pdu` macro (if OCTET STRING is a PDU type) to allocate an empty structure
2) `_copy` or `_copy_pdu` macro to create a structure and copy input data to it
3) `_setv` macro to fill in an existing structure with input data

Note that 'main' helper macros are produced for generated `_OctStr` types and then referenced by macros for each user-defined OCTET STRING type. The below example demonstrates this:

ASN.1:

```
SixOctets ::= OCTET STRING (SIZE(6))
```

Helper macros:

```
typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;

/* allocates memory for an empty string */
```

```
#define oss__OctStr_new(world) \
    (_OctStr *)ossGetInitializedMemory(world, sizeof(_OctStr))

/* allocates memory for the string and initializes it by copying the input
 * value */
#define oss__OctStr_copy(world, value_, length_) \
    (_OctStr *)oss__UnbCharString_copy(world, (char *)value_, length_)

/* initializes 'value' and 'length' fields of a string by given values */
#define oss__OctStr_setv(outp, value_, length_) \
    { \
    (outp)->value = (value_); \
    (outp)->length = (length_); \
    }

typedef _OctStr SixOctets;

/* allocates memory for SixOctets_PDU */
#define oss_SixOctets_new_pdu(world) \
    oss__OctStr_new(world)

/* allocates memory for SixOctets_PDU and initializes it by copy of an input
 * string */
#define oss_SixOctets_copy_pdu(world, value_, length_) \
    oss__OctStr_copy(world, value_, length_)

/* initializes 'value' and 'length' fields of a string by given values */
#define oss_SixOctets_setv(outp, value_, length_) \
    oss__OctStr_setv(outp, value_, length_)
```

7.6.16 Open type

Open types in **ASN.1:2008** are the equivalent to ANY and ANY DEFINED BY in **ASN.1:1990**. Unlike the other types in this section, the open type does not have a reserved identifier in the ASN.1 notation which causes it to be generated. Rather, it results when information object notation (see section 7.8) is written in a way that renders ambiguous the exact type intended for a certain field. Note that the open type is particularly useful for embedding a separately encoded ASN.1 type within some other ASN.1 type. The representation for open type contains information about an ASN.1 type and its decoded value or stores encoded data of the type.

The following specifications affect the representation of the ASN.1 open type:

ASN.1 specifications:

constraint (or Component relation constraint) is applied to the ASN.1 open type.

Directive specifications:

OSS.POINTER

OSS.NonUnionOpenType

The OSS.OBJHANDLE / OSS.NOCOPY directives are allowed on this type, but have no effect on the representation in the header file.)

7.6.16.1 Representation with the decoded value as a union of PDU type alternatives

An open type is represented this way if the following three conditions hold:

- Table constraints are applied to the type.
- The OSS.NonUnionOpenType directive is not applied to the type.
- The `-compat noUnionRepresentationForOpenTypes` command-line option is not specified in the ASN.1 compiler command-line.

Otherwise, the predefined OpenType structure is used to represent an ASN.1 open type (see 7.6.16.2).

Example:

ASN.1:

```
DEFINITIONS ::= BEGIN
    C ::= CLASS {
        &code INTEGER,
        &Type
    }

    Object C ::= {
        { &code 1, &Type INTEGER } |
        { &code 2, &Type UTF8String }
    }

    S ::= SEQUENCE {
        key C.&code ({Object}),
        value C.&Type ({Object}{@key})
    }
END
```

C representation:

```
#define          S_PDU 1
#define          Object_integer_PDU 2
#define          Object_UTF8String_PDU 3

typedef int             Object_integer;

typedef unsigned char   *Object_UTF8String;

enum Object_Type_PDUs {
    PDU_Object_Type_UNKNOWN        = 0,
    PDU_Object_Type_integer        = Object_integer_PDU,
    PDU_Object_Type_UTF8String     = Object_UTF8String_PDU
};

union Object_Type_union {
    Object_integer  *pdu_Object_integer;  /* PDU_Object_Type_integer */
    Object_UTF8String *pdu_Object_UTF8String;  /* PDU_Object_Type_UTF8String */
};

typedef struct Object_Type {
    enum Object_Type_PDUs pduNum;
    OssBuf          encoded;
    union Object_Type_union decoded;
} Object_Type;
```

```
typedef struct S {
    int             key;
    Object_Type     value;
} S;
```

The generated `Object_Type` type consists of the following fields:

1. The `pduNum` field contains the PDU identification constant of the decoded value. All constants possible for the field are defined in the `enum Object_Type_PDUs`. When there is an extensible ObjectSet, the `PDU_Object_Type_MAX` constant is also generated in `enum Object_Type_PDUs` and serves as a placeholder to ensure that `pduNum` has a broad enough numeric range to store the PDU numbers beside the enum constant set.
   The `PDU_Object_Type_UNKNOWN` constant means that nothing is stored in the `decoded` field and other PDU identification constants are equal to the PDU numbers.

2. The `encoded` field optionally contains data in encoded form. The C-type of the `encoded` field is OssBuf, defined as follows:

   ```
   typedef struct {
       long           length;
       unsigned char *value;
   } OssBuf;
   ```

   A non-zero value of both the `length` and `value` fields indicates that encoded data is present for OpenType. The address of the `encoded` field can be passed directly to the `ossDecode()`and `ossEncode()` OSS ASN.1/C Runtime API functions as the `input` and `output` parameters respectively.

3. The `decoded` field contains the decoded value of the PDU type identified by the `pduNum` field. The address of the PDU type value can be taken from the field of `union Object_Type_union` that is composed of all the PDU types allowed by the ObjectSet. Each field of `union Object_Type_union` is a pointer type, so all fields are compatible with the untyped pointer. When there is an extensible ObjectSet, the `other` field is also generated in `union Object_Type_union` and serves as storage for a PDU type value for a PDU type that is not in the ObjectSet.

In the above structure, the `pduNum` and `decoded` fields are only used if the `AUTOMATIC_ENCDEC` flag, which specifies the automatic encoding/decoding of open types, is set using the `ossSetEncodingFlags()` and `ossSetDecodingFlags()` functions. Specifically, the `pduNum` field should reference the PDU identification constant of the data structure to be contained in the open type and the `decoded` field should reference the value-filled compiler-generated data structure to be encoded. Finally, when `AUTOMATIC_ENCDEC` is specified, the `length` and `value` fields in the `encoded` field should be set to zero and NULL respectively, before calling the encoder.

When the `AUTOMATIC_ENCDEC` flag is *not* set, the `pduNum` and `decoded` fields should be set to UNKNOWN and NULL, respectively, while the `value` and `length` fields in the `encoded` field should be set to the pre-encoded data and its length, respectively.

7.6.16.2 Legacy representation with the untyped pointer to the decoded value

This C representation uses a predefined OpenType structure. This was the only possible representation prior to version 9.0 of the ASN.1 compiler. Starting with version 9.0, this representation is used

a) For non-constrained ASN.1 open types
b) When the ASN1.DeferDecoding or the OSS.ENCODABLE or the OSS.NonUnionOpenType directive is applied to the type
c) When the `-compat noUnionRepresentationForOpenTypes` command-line option is specified in the ASN.1 compiler command line.

ASN.1:

```
Module DEFINITIONS ::= BEGIN
                A ::= TYPE-IDENTIFIER.&Type
END
```

where `TYPE-IDENTIFIER` is defined as:

```
TYPE-IDENTIFIER ::= CLASS
{
   &id      OBJECT IDENTIFIER UNIQUE,
   &Type
}
```

C representation:

```
#define          A_PDU 1

typedef OpenType        A;
```

Note that the `OpenType` type has a declaration of the following form:

```
typedef struct {
       int            pduNum;
       long           length;          /* length of "encoded" */
       void          *encoded;
       void          *decoded;
} OpenType;
```

The `pduNum` field can contain the PDU identification number of the decoded structure/value. The `length` field can contain the length of the encoded value returned by the encoder. The `encoded` field can contain the address of the encoded value returned by the encoder. The `decoded` field can contain the address of the decoded structure/value.

In the above structure, the `pduNum` and `decoded` fields are only used if the `AUTOMATIC_ENCDEC` flag (specifies the automatic encoding/decoding of open types) is set using the `ossSetEncodingFlags()` and `ossSetDecodingFlags()` functions. Specifically, the `pduNum` field should reference the compiler-generated PDU identification number of the data structure to be contained in the open type and the `decoded` field should reference the value-filled compiler-generated data structure to be encoded. Finally when `AUTOMATIC_ENCDEC` is specified, the `length` and `encoded` fields should be set to zero and NULL respectively, before calling the encoder.

When the AUTOMATIC_ENCDEC flag is *not* set, the pduNum and decoded fields should be set to zero and NULL, respectively, while the encoded and length fields should be set to the pre-encoded data and its length, respectively.

Effect of OSS.HelperMacro on the open type with -helperNames:

The OSS.HelperMacro directive applied to an open type results in generating the following helper macros:

1) _new or _new_pdu macro to allocate memory for the OpenType structure and initialize it to zeroes
2) _set_encoded macro to fill the encoded representation of a type by reference to some encoded data
3) _copy_encoded or _copy_encoded_pdu macro to allocate memory for the OpenType structure and fill its encoded representation by a copy of some encoded data
4) _set_decoded macro to fill the decoded representation of a type by reference to some C value
5) _copy_decoded or _copy_decoded_pdu macro to allocate memory for the OpenType structure and fill its decoded representation by a copy of some C value (implies a call to ossCpyValue() API function)

An example follows.

ASN.1:

```
ArbitraryType ::= [0] TYPE-IDENTIFIER.&Type
```

Helper macros:

```
typedef OpenType        ArbitraryType;

/* allocates memory for ArbitraryType_PDU */
#define oss_ArbitraryType_new_pdu(world) \
    (ArbitraryType *)ossGetInitializedMemory(world, sizeof(ArbitraryType))

/* allocates memory for ArbitraryType_PDU and initializes it by copying the
 * encoded value */
#define oss_ArbitraryType_copy_encoded_pdu(world, value_, length_) \
    (ArbitraryType *)oss__OpenType_copy_encoded(world, value_, length_,
sizeof(ArbitraryType))

/* allocates memory for ArbitraryType_PDU and initializes it by copying the
 * decoded value */
#define oss_ArbitraryType_copy_decoded_pdu(world, decoded_, pdunum_) \
    (ArbitraryType *)oss__OpenType_copy_decoded(world, decoded_, pdunum_,
sizeof(ArbitraryType))

/* sets encoded value of the OpenType */
#define oss_ArbitraryType_set_encoded(outp, value_, length_) \
    { \
    (outp)->encoded = (value_); \
    (outp)->length = (length_); \
    (outp)->decoded = NULL; \
    }

/* sets decoded value of the OpenType */
#define oss_ArbitraryType_set_decoded(outp, decoded_, pdunum_) \
    { \
    (outp)->decoded = (decoded_); \
    (outp)->pduNum = (pdunum_); \
    }
```

For more information on encoding and decoding open types, refer to section 4.2 in the **OSS ASN.1/C API Reference Manual**.

Notes:
1) An `OpenType` is also generated for elements of CHOICE, SEQUENCE, and SET types that have the ASN1.DeferDecoding or OSS.ENCODABLE directive applied to them.
2) If the `-extendOpenType` compiler option is specified, a user-specific field (`userField`) is generated within the `OpenType` structure which is not encoded by the encoder. This field has the following format:

```
void            userField;
```

7.6.17 REAL

The following specifications affect the representation of the ASN.1 REAL type:

ASN.1 specification:
None.

Directive specifications without -helperNames:
OSS.FLOAT
OSS.DECIMAL
OSS.DOUBLE
OSS.MIXED
OSS.POINTER
OSS.PrintfunctionName

Directive specifications with -helperNames:
OSS.DECIMAL
OSS.DOUBLE
OSS.POINTER
OSS.PrintfunctionName

Directive specification with -helperNames for REAL types with POINTER directive or marked as PDUs:
OSS.HelperMacro
OSS.NoHelperMacro

By default, the DOUBLE representation is used:

ASN.1:

```
DefaultReal          ::= REAL
```

C representation:

```
typedef double          DefaultReal;
```

Effect of the OSS.FLOAT / OSS.DOUBLE directive on the REAL type

The OSS.FLOAT / OSS.DOUBLE directives determine the representation of the REAL ASN.1 type. By default, the DOUBLE representation is assumed, but this can be overridden by use of a OSS.FLOAT directive.

ASN.1:

```
FloatReal            ::= REAL --<FLOAT>--
```

C representation:

```
typedef float           FloatReal;
```

Effect of the OSS.DECIMAL and OSS.MIXED directives on the REAL

The following ASN.1 notation along with its resultant header file declarations illustrates the use of the OSS.DECIMAL and OSS.MIXED directive:

ASN.1:

```
DecimalReal            ::= REAL --<DECIMAL>--
MyMixReal              ::= REAL --<MIXED>--
```

C representation:

```
typedef char            *DecimalReal;
typedef MixedReal       MyMixReal;
```

For more information about the OSS.DECIMAL and OSS.MIXED directives, refer to section 4.4.2.6.

Effect of OSS.HelperMacro on the REAL type

The OSS.HelperMacro directive applied to a REAL results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the value (a double variable is allocated for binary REAL and a string of given length for decimal type)
2) `_copy` or `_copy_pdu` macro to allocate memory and copy input data to it (copies either a double constant or a null terminated string value)

Macros for binary REAL types are produced only if this type is a PDU or it is referenced by pointer. Macros for decimal REAL are produced without restrictions since it is always pointered. If a REAL type is a non-typereference field of a SET or SEQUENCE, then its parent type should have its own macros produced in order to produce macros for REAL. An example follows.

ASN.1:

```
--<OSS.HelperMacro Mod.MyMixed>--
--<OSS.HelperMacro Mod.MyMixed.binary>--
--<OSS.HelperMacro Mod.MyMixed.decimal>--
Mod DEFINITIONS ::= BEGIN
MyMixed ::= CHOICE {
    binary  [0] REAL --<POINTER>--,
    decimal [1] REAL --<DECIMAL>--
}
END
```

Helper macro:

```
typedef struct MyMixed {
    unsigned short  choice;
#       define      binary_chosen 1
#       define      decimal_chosen 2
    union {
        double          *binary;  /* to choose, set choice to binary_chosen */
        char            *decimal;  /* to choose, set choice to decimal_chosen */
    } u;
} MyMixed;

…

/* allocates memory for an instance of the real type */
#define oss_MyMixed_binary_new(world) \
    (double *)ossGetMemory(world, sizeof(double))

/* allocates memory for an instance of the real type and initializes it by given
 * value */
#define oss_MyMixed_binary_copy(world, value_) \
    oss__Double_copy(world, value_)

…

/* allocates memory for a string of given length */
#define oss_MyMixed_decimal_new(world, length_) \
    oss__CharStr_new(world, length_)

/* allocates memory and returns a copy of an input string */
#define oss_MyMixed_decimal_copy(world, value_) \
    oss__CharStr_copy(world, value_)
```

7.6.18 RELATIVE-OID type

Support for the ASN.1 type RELATIVE-OID (relative object identifier) was added in version 5.1. A relative object identifier can be used to identify an object in the object identifier tree relative to some other known object identifier. Thus, the relative object identifier type can be used to transmit the trailing component or components of an object identifier value. This leads to a bandwidth savings over transmitting the entire object identifier value.

The following is a simple example of how the RELATIVE-OID type may be used:

```
DefaultROID ::= RELATIVE-OID

myUniversity  OBJECT IDENTIFIER ::=
    {iso member-body country(29) universities(56) universityOfNokalva(32)}

relativeDept DefaultROID ::=  {engineering(4) digitalSignalProcessing(3)}

dspDept OBJECT IDENTIFIER ::= { myUniversity relativeDept }
```

In the above example, `relativeDept` may be transmitted alone if both sender and receiver are aware that `engineering(4)` is an offshoot (branch) of `{iso member-body country(29) universities(56) universityOfNokalva(32)}`

Note that the following initializations are generated in the `.c` file for the above value notation:

```
static unsigned char _v0[] = { 0x2A, 0x1D, 0x38, 0x20 };
ObjectID myUniversity = {
    4,      /* length in octets */
    _v0     /* BER-encoded value */
};

static unsigned char _v1[] = { 0x04, 0x03 };
DefaultROID relativeDept = {
    2,      /* length in octets */
    _v1     /* BER-encoded value */
};

static unsigned char _v2[] = { 0x2A, 0x1D, 0x38, 0x20, 0x04, 0x03 };
ObjectID dspDept = {
    6,      /* length in octets */
    _v2     /* BER-encoded value */
};
```

Note how the `value` field contents are simply the BER-encoded octets of the object identifier value.

The following specifications affect the representation of the ASN.1 RELATIVE-OID type:

ASN.1 specification:

None.

Directive specifications:

OSS.OBJECTID *length*
OSS.OBJECTID ENCODED / ENCODED
OSS.POINTER

Directive specification with -helperNames:

OSS.ENCODED
OSS.HelperMacro
OSS.NoHelperMacro

By default, the ENCODED representation is used:

ASN.1:

```
DefaultROID ::= RELATIVE-OID
```

C representation without -helperNames:

```
typedef struct OssRelativeOID {
    unsigned short  length;
    unsigned char   *value;
} OssRelativeOID;

typedef OssRelativeOID  DefaultROID;
```

C representation with -helperNames:

```
typedef struct _RelOID {
    unsigned int    length;
    unsigned char   *value;
```

```
} _RelOID;

typedef _RelOID         DefaultROID;
```

Effect of the OSS.OBJECTID directive followed by a *length* operand on the OBJECT IDENTIFIER type

If the OSS.OBJECTID directive is specified along with a length operand, then a fixed-size character string is generated preceded by a `length` field giving the size in octets of the character string.

ASN.1:

```
BigROID ::= RELATIVE-OID --<OBJECTID 80>--
```

C representation:

```
typedef struct BigROID {
    unsigned short  length;
    unsigned char   value[80];
} BigROID;
```

Under this representation, the `value` field contains the BER-encoded contents of the OBJECT IDENTIFIER (e.g., `{1 2 3 4 5}` would be stored as: `{0x2A, 0x03, 0x04, 0x05}`).

Effect of OSS.HelperMacro on the RELATIVE-OID type with -helperNames:

The OSS.HelperMacro directive applied to a RELATIVE-OID results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate an empty structure
2) `_copy` or `_copy_pdu` macro to create a structure and copy input data to it
3) `_setv` macro to fill in an existing structure with input data

Note that 'main' helper macros are produced for generated _RelOID types and then referenced by macros for each user-defined RELATIVE-OID type. The below example demonstrates this:

ASN.1:

```
OID-Tail ::= RELATIVE-OID
```

Helper macros:

```
typedef struct _RelOID {
    unsigned int    length;
    unsigned char   *value;
} _RelOID;

/* allocates memory for an empty structure */
#define oss__RelOID_new(world) \
    (_RelOID *)ossGetInitializedMemory(world, sizeof(_RelOID))

/* allocates memory for the structure and initializes it by copying the input
 * value */
#define oss__RelOID_copy(world, value_, length_) \
    (_RelOID *)oss__UnbCharString_copy(world, (char *)value_, length_)

/* initializes 'value' and 'length' fields of a structure by given values */
#define oss__RelOID_setv(outp, value_, length_) \
    { \
    (outp)->value = (value_); \
```

```
    (outp)->length = (length_); \
    }

typedef _RelOID         OID_Tail;

/* allocates memory for OID_Tail_PDU */
#define oss_OID_Tail_new_pdu(world) \
    oss__RelOID_new(world)

/* allocates memory for OID_Tail_PDU and initializes it by copy of an input
 * string */
#define oss_OID_Tail_copy_pdu(world, value_, length_) \
    oss__RelOID_copy(world, value_, length_)

/* initializes 'value' and 'length' fields of a structure by given values */
#define oss_OID_Tail_setv(outp, value_, length_) \
    oss__RelOID_setv(outp, value_, length_)
```

7.6.19 Character string types

7.6.19.1 Restricted character string types

The character string types described in this section refer to the ASN.1 CharacterString types and all types derived from them by tagging or subtyping. Specifically, the following ASN.1 built-in types are considered to be restricted character strings:

BMPString
GeneralString
GraphicString (ObjectDescriptor)
IA5String
ISO646String
NumericString
PrintableString
TeletexString
T61String
UniversalString
UTF8String
VideotexString
VisibleString

The **GeneralizedTime** and **UTCTime** ASN.1 types are described later in this representations section.

The following specifications affect the representation of the ASN.1 CharacterString types:

ASN.1 specification:

SizeConstraint subtype

Directive specifications without -helperNames:

OSS.NULLTERM
OSS.OBJHANDLE
OSS.ONECHAR
OSS.PADDED

OSS.POINTER
OSS.PrintfunctionName
OSS.UNBOUNDED
OSS.VARYING

Note that the OSS.NULLTERM, OSS.PADDED, OSS.UNBOUNDED, and OSS.VARYING directives are mutually exclusive with each other.

Note also that the OSS.VARYING directive is not supported with BMPString and UniversalString.

Directive specifications with -helperNames:

OSS.HelperMacro
OSS.NoHelperMacro
OSS.NULLTERM
OSS.OBJHANDLE
OSS.PrintfunctionName
OSS.UNBOUNDED

By default, the C representation is NULLTERM-POINTER for all of the restricted character string types, except BMPString and UniversalString which have a default representation of UNBOUNDED. Refer to sections 4.4.2.3 and 4.4.2.50 for more information about the available representations for the BMPString, UniversalString, and UTF8String types.

ASN.1:

```
Name1 ::= VisibleString
```

C representation:

```
typedef char *Name1;
```

Effect of SizeConstraint of less than 256 bytes on restricted character strings

The compiler uses the SizeConstraint subtype to determine the maximum number of bytes that there are in a CharacterString. In the absence of directives, the NULLTERM representation is generated.

ASN.1:

```
Name1 ::= VisibleString (SIZE(10))
```

C representation:

```
typedef char Name1[11];
```

Note the extra character space allocated above for the null-terminated character.

Effect of SizeConstraint greater than or equal to 256 bytes on restricted character strings

When a SizeConstraint is larger than 256 bytes, the compiler automatically uses the UNBOUNDED representation in the absence of other compiler directives. If the upper bound of the SizeConstraint is MAX then the representation is the same as for unconstrained strings as described earlier in this section.

ASN.1:

```
PStringSzBig ::= PrintableString (SIZE(257))
```

C representation:

```
typedef struct PStringSzBig {
    unsigned short  length;
    char            *value;
} PStringSzBig;
```

Effect of the OSS.NULLTERM directive on restricted character strings

The OSS.NULLTERM directive varies in effect depending on the presence/absence of the SizeConstraint subtype. If a SizeConstraint is present, the compiler will generate a char array to accommodate the constraint.

ASN.1:

```
Name1 ::= VisibleString (SIZE(10)) --<NULLTERM>--
```

C representation:

```
typedef char            Name1[11];
```

If the SizeConstraint is absent, a pointer to a null-terminated array of characters is generated. OSS.POINTER is assumed because the string length is indeterminable.

ASN.1:

```
Name1 ::= VisibleString  --<NULLTERM>--
```

C representation:

```
typedef char           *Name1;
```

Warning: When using the null-terminated representation, the user should ensure that the character string does not contain any embedded null characters. The presence of an embedded null character will cause the encoder to mis-encode the sent string. Likewise, the decoder will prematurely exit on such a string issuing an informatory message about the embedded null character that it found.

Effect of the OSS.PADDED directive on restricted character strings

The OSS.PADDED directive requires the presence of the SizeConstraint subtype, because the compiler must be able to determine the length of the CharacterString. If the OSS.PADDED directive is specified without SizeConstraint, the compiler issues an error message.

ASN.1:

```
Name1 ::= VisibleString (SIZE(10)) --<PADDED>--
```

C representation:

```
typedef char            Name1[10];
```

Effect of the OSS.UNBOUNDED directive on restricted character strings

If the OSS.UNBOUNDED directive is specified alone, a pointer to a character string is generated, preceded by a length field of size `unsigned int`.

ASN.1:

```
Name1 ::= VisibleString --<UNBOUNDED>--
```

C representation without -helperNames:

```
typedef struct Name1 {
    unsigned int    length;
    char            *value;
} Name1;
```

C representation with -helperNames:

```
typedef struct _UnbCharStr {
    unsigned int    length;
    char            *value;
} _UnbCharStr;

typedef _UnbCharStr Name1;
```

Effect of the OSS.VARYING directive on restricted character strings

If the OSS.VARYING directive is specified alone, the result is a pointer to a structure containing a `length` field of size `unsigned short` and a character field of unspecified length. The effect is the same as when the OSS.POINTER and OSS.VARYING directives are specified together (i.e., OSS.POINTER is assumed because the array length is indeterminable by the compiler).

ASN.1:

```
Name1 ::= VisibleString --<VARYING>--
```

C representation:

```
typedef struct Name1 {
    unsigned short  length;
    char            value[1]; /* first element of the array */
} *Name1;
```

Illustrations of specification combinations

Effect of SizeConstraint and OSS.UNBOUNDED on restricted character strings

When the OSS.UNBOUNDED directive is combined with SizeConstraint the UNBOUNDED representation is used, but the `length` field varies depending on the size needed to accommodate the longest CharacterString.

ASN.1:

```
Name1 ::= VisibleString (SIZE(1..99999)) --<UNBOUNDED>--
```

C representation:

```
typedef struct Name1 {
    int             length;
    char            *value;
} Name1;
```

Effect of SizeConstraint and OSS.VARYING on restricted character strings

When the OSS.VARYING directive is combined with SizeConstraint, the VARYING representation is used, but the `length` field varies depending on the size needed to accommodate the longest CharacterString.

ASN.1:

```
Name1 ::= VisibleString (SIZE(99999)) --<VARYING>--
```

C representation:

```
typedef struct Name1 {
    long            length;
    char            value[99999];
} Name1;
```

Effect of OSS.HelperMacro on restricted character string types with -helperNames:

The OSS.HelperMacro directive applied to a restricted character string with NULLTERM representation results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the string of given length and initialize it with zeroes
2) `_copy` or `_copy_pdu` macro to allocate memory and copy given input null terminated time value to it

The OSS.HelperMacro directive applied to a restricted character string with the UNBOUNDED representation to an instance of _BMPStr (2-byte character string) or to an instance of _UnivStr (4-byte character string) results in the generation of the following helper macros:

1) `_new` or `_new_pdu` macro to allocate an empty structure
2) `_copy` or `_copy_pdu` macro to create a structure and copy input data to it
3) `_copy` or `_copy_pdu` macro to create a structure and copy input null-terminated data to it (length of data is calculated within a macro)
4) `_setv` macro to fill in an existing structure with input data
5) `_setv_nullterm_` macro to fill in an existing structure with input null-terminated data (length of data is calculated within a macro)

Note that 'main' helper macros are produced for generated types like `_CharStr` and then referenced by macros for each user-defined restricted character string type. An example that demonstrates macros for NULLTERM, UNBOUNDED (UTF8) and 4byte character strings follows.

ASN.1:

```
StrData ::= CHOICE {
    ia5 IA5String,
    utf8 UTF8String --<UNBOUNDED>--,
    bmp BMPString
}
```

Helper macros:

```
typedef struct _BmpStr {
    unsigned int    length;
    unsigned short  *value;
} _BmpStr;

/* allocates memory an empty string */
#define oss_BmpStr_new(world) \
    (_BmpStr *)ossGetInitializedMemory(world, sizeof(_BmpStr))

/* allocates memory for the string and initializes it by copying the input
```

```
 * value */
#define oss__BmpStr_copy(world, value_, length_) \
    oss__BmpString_copy(world, value_, length_)

/* initializes 'value' and 'length' fields of a string by given values */
#define oss__BmpStr_setv(outp, value_, length_) \
    { \
    (outp)->value = value_; \
    (outp)->length = length_; \
    }

typedef struct _UTF8Str {
    unsigned int    length;
    unsigned char   *value;
} _UTF8Str;

/* allocates memory for an empty string */
#define oss__UTF8Str_new(world) \
    (_UTF8Str *)ossGetInitializedMemory(world, sizeof(_UTF8Str))

/* allocates memory for the string and initializes it by copying the input
 * value */
#define oss__UTF8Str_copy(world, value_, length_) \
    (_UTF8Str *)oss__UnbCharString_copy(world, (char *)value_, length_)

/* allocates memory for the string and initializes it by copying the
 * null-terminated value */
#define oss__UTF8Str_copy_nullterm(world, value_) \
    (_UTF8Str *)oss__UnbCharString_copy_nullterm(world, (char *)value_)

/* initializes 'value' and 'length' fields of a string by given values */
#define oss__UTF8Str_setv(outp, value_, length_) \
    { \
    (outp)->value = (value_); \
    (outp)->length = (length_); \
    }

/* initializes the string using the input null-terminated value without copying
 * it */
#define oss__UTF8Str_setv_nullterm(outp, value_) \
    { \
    (outp)->value = (value_); \
    (outp)->length = strlen(value_); \
    }

typedef struct StrData {
    unsigned short  choice;
#       define      utf8_chosen 1
#       define      ia5_chosen 2
#       define      bmp_chosen 3
    union {
        char            *ia5;  /* to choose, set choice to ia5_chosen */
        struct _UTF8Str *utf8;  /* to choose, set choice to utf8_chosen */
        struct _BmpStr  *bmp;  /* to choose, set choice to bmp_chosen */
    } u;
} StrData;

/* allocates memory for a string of given length */
#define oss_StrData_ia5_new(world, length_) \
    oss__CharStr_new(world, length_)

/* allocates memory and returns a copy of an input string */
#define oss_StrData_ia5_copy(world, value_) \
    oss__CharStr_copy(world, value_)
…
```

7.6.19.2 Unrestricted CharacterString type

The (unrestricted) CHARACTER STRING type has the following definition in ASN.1 notation:

```
SEQUENCE {
  identification CHOICE {
    syntaxes SEQUENCE {
      abstract OBJECT IDENTIFIER,
      transfer OBJECT IDENTIFIER }
      -- Abstract and transfer syntax object identifiers --,
    syntax OBJECT IDENTIFIER
      -- A single object identifier for identification of the class and encoding --,
    presentation-context-id INTEGER
      -- (Applicable only to OSI environments)
      -- The negotiated presentation context identifies the class of the value and its encoding --,
    context-negotiation SEQUENCE {
      presentation-context-id INTEGER,
      transfer-syntax OBJECT IDENTIFIER }
      -- (Applicable only to OSI environments)
      -- Context-negotiation in progress for a context to identify the class of the value
      -- and its encoding --,
    transfer-syntax OBJECT IDENTIFIER
      -- The class of the value (for example, specification that it is the value of an ASN.1 type)
      -- is fixed by the application designer (and hence known to both sender and receiver). This
      -- case is provided primarily to support selective-field-encryption (or other encoding
      -- transformations) of an ASN.1 type --,
    fixed NULL
      -- The data value is the value of a fixed ASN.1 type (and hence known to both sender
      -- and receiver) -- },
  data-value-descriptor ObjectDescriptor OPTIONAL
      -- This provides human-readable identification of the class of the value --,
  string-value OCTET STRING }
( WITH COMPONENTS {
  ... ,
data-value-descriptor ABSENT } )
```

The following specifications affect the representation of the ASN.1 CHARACTER STRING type:

ASN.1 specifications:

None.

Directive specifications without -helperNames:

OSS.POINTER

Directive specification with -helperNames:

OSS.HelperMacro
OSS.NoHelperMacro

By default, the C representation is very similar to that of the EMBEDDED PDV and EXTERNAL types.

C representations

In the absence of the OSS.POINTER directive, the C representation is:

ASN.1:

```
Characterstring ::= CHARACTER STRING
```

C representation without -helperNames:

```
typedef UnrestrictedChar Characterstring;
```

where `UnrestrictedChar` is defined in the generated header file as:

```
typedef struct UnrestrictedChar {
    struct {
        unsigned short  choice;
#define      UnrestrictedChar_identification_syntaxes_chosen 1
#define      UnrestrictedChar_identification_syntax_chosen 2
#define      UnrestrictedChar_identification_presentation_context_id_chosen 3
#define      UnrestrictedChar_identification_context_negotiation_chosen 4
#define      UnrestrictedChar_identification_transfer_syntax_chosen 5
#define      UnrestrictedChar_identification_fixed_chosen 6
        union {
            struct UnrestrictedChar_syntaxes {
                ObjectID        abstract;
                ObjectID        transfer;
            } syntaxes; /* to choose, set choice to UnrestrictedChar_identification_syntaxes_chosen */
            ObjectID        syntax;   /* to choose, set choice to
                                         UnrestrictedChar_identification_syntaxes_chosen */
            int             presentation_context_id;    /* to choose, set choice to
                                         UnrestrictedChar_identification_presentation_context_id_chosen */
            struct UnrestrictedChar_negotiation {
                int             presentation_context_id;
                ObjectID        transfer_syntax;
            } context_negotiation;   /* to choose, set choice to
                                         UnrestrictedChar_identification_context_negotiation_chosen */
            ObjectID        transfer_syntax;   /* to choose, set choice to
                                         UnrestrictedChar_identification_transfer_syntax_chosen */
            Nulltype        fixed;   /* to choose, set choice to
                                         UnrestrictedChar_identification_fixed_chosen */
        } u;
    } identification;
    struct {
        unsigned int    length;
        unsigned char   *value;
    } string_value;
} UnrestrictedChar;
```

C representation with -helperNames:

```
typedef struct _OID {
    unsigned int    length;
    unsigned char   *value;
} _OID;

typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;
```

```
typedef struct UnrestrictedChar {
    struct UnrestrictedChar_identification *identification;
     _OctStr  *string_value;
} UnrestrictedChar;

typedef UnrestrictedChar Characterstring;

typedef struct UnrestrictedChar_syntaxes {
    struct _OID     *abstract;
    struct _OID     *transfer;
} UnrestrictedChar_syntaxes;

typedef struct UnrestrictedChar_negotiation {
    int             presentation_context_id;
    struct _OID     *transfer_syntax;
} UnrestrictedChar_negotiation;

typedef struct UnrestrictedChar_identification {
    unsigned short  choice;
#       define      syntaxes_chosen 1
#       define      syntax_chosen 2
#       define      presentation_context_id_chosen 3
#       define      context_negotiation_chosen 4
#       define      transfer_syntax_chosen 5
#       define      fixed_chosen 6
    union {
        struct UnrestrictedChar_syntaxes *syntaxes;  /* to choose, set
                                            * choice to syntaxes_chosen */
        struct _OID     *syntax;  /* to choose, set choice to
                                   * syntax_chosen */
        int             presentation_context_id;  /* to choose, set choice
                                   * to presentation_context_id_chosen */
        struct UnrestrictedChar_negotiation *context_negotiation;  /* to
                                    * choose, set choice to
                                    * context_negotiation_chosen */
        struct _OID     *transfer_syntax;  /* to choose, set choice to
                                     * transfer_syntax_chosen */
        Nulltype        fixed; /* to choose, set choice to fixed_chosen */
    } u;
} UnrestrictedChar_identification;
```

Effect of OSS.HelperMacro on the UnrestrictedCharacterString type with -helperNames:

The CHARACTER STRING type is defined in ASN.1 as a SEQUENCE type containing CHOICE, OCTET STRING and other types. Correspondingly, helper macros for it are produced in accordance with the rules for SEQUENCE, CHOICE, OCTET STRING etc.

7.6.20 Selection

The representation in C of a Selection type is that of the type selected.

For example,

ASN.1:

```
Info1 ::= CHOICE
       {name   VisibleString,
        age    INTEGER} --<PDU>--

Info2 ::= SEQUENCE
       {birthdate INTEGER,
        name < Info1}
```

C representation:

```
typedef struct Info1 {
    unsigned short  choice;
#       define      name_chosen 1
#       define      age_chosen 2
    union {
        char            *name;
        int             age;
    } u;
} Info1;

typedef struct Info2 {
    int             birthdate;
    char            *name;
} Info2;
```

The Selection type of name `< Info1` has the same representation as name `VisibleString`..

7.6.21 SEQUENCE

The following specifications affect the representation of the ASN.1 SEQUENCE type:

ASN.1 specifications:

COMPONENTS OF notation
Contained subtype
DEFAULT keyword
Inner subtype
OPTIONAL keyword

Directive specifications without -helperNames:

OSS.DefineName
OSS.ExtractType
OSS.FIELDNAME
OSS.InlineType
OSS.POINTER

Directive specifications with -helperNames:

OSS.DefineName
OSS.ExtractType
OSS.FIELDNAME
OSS.InlineType
OSS.HelperMacro
OSS.NoHelperMacro

By default, C representation is of the form:

```
typedef struct Name1 {
        type1   element1;
                .
                .
                .
        typen   elementn;
} Name1;
```

The elements (e.g., `element1`) are the identifiers associated with each component of the SEQUENCE type. If an identifier is missing, the compiler will generate a C identifier based on the ASN.1 type.

ASN.1:

```
NamesOfOfficers ::= SEQUENCE {
        president               VisibleString (SIZE(1..32)),
        vicePresident           VisibleString (SIZE(1..32)),
        secretary               VisibleString (SIZE(1..32)),
                                VisibleString (SIZE(1..32))}
```

C representation without -helperNames:

```
typedef struct NamesOfOfficers {
    char            president[33];
    char            vicePresident[33];
    char            secretary[33];
    char            visibleString[33];
} NamesOfOfficers;
```

C representation with -helperNames:

```
typedef struct NamesOfOfficers {
    char            *president;
    char            *vicePresident;
    char            *secretary;
    char            *visibleString;
} NamesOfOfficers;
```

Note: Leaving out identifiers from CHOICE, SEQUENCE, and SET types was valid in the **ASN.1:1990** standard but is no longer considered so.

Effect of OPTIONAL and DEFAULT within the SEQUENCE type

If any component(s) of the SEQUENCE is OPTIONAL or has a DEFAULT value, the C representation is of the form:

```
typedef struct Name1{
    unsigned int    bit_mask;
#       define      elementj_present  0x80
    type1           element1;
                    .
                    .
                    .
    typen           elementn;
} Name1;
```

The variable, `bit_mask`, is used to indicate the presence or absence of SEQUENCE components that are OPTIONAL or have a DEFAULT value, but which are not referenced via a pointer. For those components that are referenced via a pointer, a non-NULL or NULL pointer value indicates the component's presence or absence.

The `bit_mask` type (`char`, `short`, `int`, `long`, or `unsigned char` array) that the compiler generates depends on the smallest type that can accommodate all optional components that are not referenced via a pointer.

For each SEQUENCE component represented in the `bit_mask`, the compiler generates a `bit_mask` manifest constant whose name is the component's name with a suffix of `_present`. The application programmer should perform a logical ***AND*** with this manifest constant and the `bit_mask` field to determine if a specific component is present.

ASN.1:

```
NamesOfOfficers2 ::= SEQUENCE {
        president               VisibleString (SIZE(1..32)) OPTIONAL,
        vicePresident       [1] VisibleString (SIZE(1..32)),
        treasurer           [2] VisibleString OPTIONAL,
        secretary               VisibleString (SIZE(1..32)) OPTIONAL}
```

C representation without -helperNames:

```
typedef struct NamesOfOfficers2 {
    unsigned char   bit_mask;
#       define      president_present 0x80
#       define      secretary_present 0x40
    char            president[33];  /* optional; set in bit_mask
                                     * president_present if present */
    char            vicePresident[33];
    char            *treasurer;  /* NULL for not present */
    char            secretary[33];  /* optional; set in bit_mask
                                     * secretary_present if present */
} NamesOfOfficers2;
```

Note: Although field `treasurer` is OPTIONAL, it is not represented in the bitmask since pointered elements can have their pointers set to NULL when not present.

C representation with -helperNames:

```
typedef struct NamesOfOfficers2 {
    char            *president;  /* NULL for not present */
    char            *vicePresident;
    char            *treasurer;  /* NULL for not present */
    char            *secretary;  /* NULL for not present */
} NamesOfOfficers2;
```

Effect of the COMPONENTS OF notation within the SEQUENCE type

When the COMPONENTS OF notation is used in a SEQUENCE, the elements of the referenced SEQUENCE are copied inline.

ASN.1:

```
NamesOfOfficers3 ::= SEQUENCE {
    president     VisibleString (SIZE(1..32)),
    vicePresident VisibleString (SIZE(1..32)),
    COMPONENTS OF MoreOfficers}

MoreOfficers ::= [1] SEQUENCE {
secretary     VisibleString (SIZE(1..32)),
           VisibleString (SIZE(1..32))}
```

C representation:

```
typedef struct NamesOfOfficers3 {
    char            president[33];
    char            vicePresident[33];
    char            secretary[33];
    char            visibleString[33];
} NamesOfOfficers3;
```

Effect of the Contained and Inner subtype notations

See sections 7.3.5 and 7.3.6 for a full discussion on the effect of these subtype notations.

Effect of OSS.HelperMacro on the SEQUENCE type with -helperNames:

The OSS.HelperMacro directive applied to a SEQUENCE results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the structure
2) `_<identifier>_set` macro for each field of a SEQUENCE (it takes into account the possible need to set an optional presence indicator)
3) `_<identifier>_get` macro for each field of a SEQUENCE (presence of a field should be known in advance)
4) `_<identifier>_is_present` macro for each optional field
5) `_<identifier>_omit` macro to mark given optional field as absent (for the fields not having bits in a bit mask, it sets the field to NULL)
6) `_<identifier>_default` macro for each field having a `DEFAULT` value; it indicates that default value for the field should be used;

An example follows.

ASN.1:

```
Mod DEFINITIONS AUTOMATIC TAGS ::= BEGIN
Person ::= SEQUENCE {
    fullName IA5String,
    spource Person OPTIONAL,
    nOfChildren INTEGER DEFAULT 0
} --<PDU>--
END
```

Helper macros:

```
typedef struct Person {
    unsigned char   bit_mask;
#       define      nOfChildren_present 0x80
    char            *fullName;
    struct Person   *spource;  /* NULL for not present */
    int             nOfChildren;  /* nOfChildren_present not set in bit_mask
                                   * implies value is 0 */
} Person;

/* allocates memory for an empty instance of the sequence type */
#define oss_Person_new(world) \
    (Person *)ossGetInitializedMemory(world, sizeof(Person))

/* allocates memory for Person_PDU */
#define oss_Person_new_pdu(world) \
    oss_Person_new(world)

/* gets "fullName" field value */
#define oss_Person_fullName_get(inp) \
    (inp)->fullName

/* sets "fullName" field value */
#define oss_Person_fullName_set(outp, fullName_) \
    (outp)->fullName = (fullName_)

…

/* gets "spource" field value */
#define oss_Person_spource_get(inp) \
    (inp)->spource

/* sets "spource" field value */
#define oss_Person_spource_set(outp, spource_) \
    (outp)->spource = (spource_)

/* checks if "spource" field value is present */
#define oss_Person_spource_is_present(inp) \
    ((inp)->spource != NULL)

/* indicates that "spource" field value is absent */
#define oss_Person_spource_omit(outp) \
    (outp)->spource = NULL

/* Macros for "spource" field operate with 'Person' type; you may use macros
 * after this type declaration to create its values */

/* gets "nOfChildren" field value */
#define oss_Person_nOfChildren_get(inp) \
    (inp)->nOfChildren

/* sets "nOfChildren" field value */
#define oss_Person_nOfChildren_set(outp, nOfChildren_) \
    { \
    (outp)->bit_mask |= nOfChildren_present; \
    (outp)->nOfChildren = (nOfChildren_); \
    }

/* checks if "nOfChildren" field value is present */
#define oss_Person_nOfChildren_is_present(inp) \
    ((inp)->bit_mask & nOfChildren_present)
```

```
/* indicates that "nOfChildren" field should be set to default */
#define oss_Person_nOfChildren_default(outp) \
    (outp)->bit_mask &= ~nOfChildren_present
```

7.6.22 SEQUENCE OF

The following specifications affect the representation of the ASN.1 SEQUENCE OF type:

ASN.1 specifications:

Contained subtype
Inner subtype
SizeConstraint subtype

Directive specifications without -helperNames:

OSS.ARRAY
OSS.DLINKED
OSS.ExtractType
OSS.InlineType
OSS.LINKED
OSS.POINTER
OSS.UNBOUNDED
OSS.DLINKED-PLUS

Directive specifications with -helperNames:

OSS.ExtractType
OSS.InlineType
OSS.UNBOUNDED
OSS.DLINKED-PLUS
OSS.HelperMacro
OSS.NoHelperMacro

Note that the OSS.ARRAY, OSS.LINKED, OSS.DLINKED, OSS.UNBOUNDED, and OSS.DLINKED-PLUS directives are mutually exclusive.

By default without -helperNames, the LINKED representation is OSS.LINKED, and is of the form:

```
typedef struct Name1 {
   struct Name1   *next;
   Type1          value;
} *Name1;
```

Notice above how the entire structure is declared as a pointer for purposes of allocation as an item in a list.

By default with -helperNames, the DLINKED-PLUS representation is generated for SEQUENCE OF types with structured or pointered non-structured elements, and is of the form:

```
typedef struct Name1 {
   struct Name1   *next;
   Type1          value;
} *Name1;
typedef struct Name1 {
```

```
    struct Name1_node *head;
    struct Name1_node *tail;
    unsigned int    count;
} Name1;

typedef struct Name1_node {
    struct Name1_node *next;
    struct Name1_node *prev;
    Type1             *value;
} Name1_node;
```

The UNBOUNDED representation is generated for SEQUENCE OF types with simple non-pointered elements, and is of the form:

```
typedef struct Name1 {
   unsigned int   count;
   ossBoolean     *value;
} Name1;
```

A note about SEQUENCE OF representation:

SEQUENCE OF and SET OF are typically represented by a linked list. The ASN.1

```
X ::= SEQUENCE OF INTEGER
```

turns into something like

```
typedef struct X {
       struct X  *next;
       int       value;
}*X;
```

The important thing to note is that the first link (let's call it link0) does not contain the first value. Instead the first link points to another link (let's call it link1) containing the first value. By way of example, if we were trying to populate X so that it contains a sequence of three integers

```
x X ::= { 1, 2, 3 }
```

we would do it like this

```
struct X link0, link1, link2, link3;

link0.next=&link1;
link1.value=1;
link1.next=&link2;
link2.value=2;
link2.next=&link3;
link3.value=3;
link3.next=NULL;
```

not like this

```
link0.value=1;
link0.next=&link1;
link1.value=2;
link1.next=&link2;
```

```
link2.value=3;
link2.next=NULL;
```

Note the extra level of indirection. The point of link0, since it clearly does not contain a value, is to differentiate between an empty `SEQUENCE OF`

```
x X ::= {}
```

and one containing values. If `link0.next=NULL`, the `SEQUENCE OF` is empty; if it's not `NULL`, the `SEQUENCE OF` contains values.

Effect of the Contained and Inner subtype notations

See sections 7.3.5 and 7.3.6 for a full discussion on the effect of these subtype notations.

Effect of SizeConstraint on the SEQUENCE OF type

The compiler uses the SizeConstraint subtype to determine the maximum number of occurrences allowed in a SEQUENCE OF. In the absence of directives, the LINKED representation is generated if no -helperNames option is specified or implied. Otherwise, the SizeConstraint has no effect. SEQUENCE OF types with structured or pointered non-structured elements are represented with the DLINKED-PLUS representation. SEQUENCE OF types with simple non-pointered non-structured elements are represented with the UNBOUNDED representation.

ASN.1:

```
SeqSzOfInt   ::= SEQUENCE SIZE(5) OF INTEGER
```

C representation:

```
typedef struct SeqSzOfInt {
    struct SeqSzOfInt *next;
    int             value;
} *SeqSzOfInt;
```

Effect of the OSS.ARRAY directive on the SEQUENCE OF type

If the OSS.ARRAY directive is specified alone, an array of types is generated, preceded by a `count` field holding the number of occurrences that follow. The OSS.POINTER directive is implied when the OSS.ARRAY directive is used.

ASN.1:

```
SeqArrOfInt  ::= SEQUENCE --<ARRAY>-- OF INTEGER
```

C representation:

```
typedef struct SeqArrOfInt {
    unsigned int    count;
    int             value[1];  /* first element of the array */
} *SeqArrOfInt;
```

Effect of the OSS.DLINKED directive on the SEQUENCE OF type

If the OSS.DLINKED directive is specified alone, then a pointer to a doubly linked list of values is generated.

ASN.1:

```
SeqDlnkOfInt ::= SEQUENCE --<DLINKED>-- OF INTEGER
```

C representation:

```
typedef struct SeqDlnkOfInt {
    struct SeqDlnkOfInt *next;
    struct SeqDlnkOfInt *prev;
    int             value;
} *SeqDlnkOfInt;
```

Effect of the OSS.LINKED directive on the SEQUENCE OF type

If the OSS.LINKED directive is specified alone, then a pointer to linked list of values is generated.

ASN.1:

```
SeqLnkOfInt  ::= SEQUENCE --<LINKED>-- OF INTEGER
```

C representation:

```
typedef struct SeqLnkOfInt {
    struct SeqLnkOfInt *next;
    int             value;
} *SeqLnkOfInt;
```

Effect of the OSS.UNBOUNDED directive on the SEQUENCE OF type

If the OSS.UNBOUNDED directive is specified alone, then a pointer to an array of types is generated, preceded by a count of the number of occurrences pointed to.

ASN.1:

```
SeqUnbdOfInt ::= SEQUENCE --<UNBOUNDED>-- OF INTEGER
```

C representation:

```
typedef struct SeqUnbdOfInt {
    unsigned int    count;
    int             *value;
} SeqUnbdOfInt;
```

Effect of the OSS.DLINKED-PLUS directive on the SEQUENCE OF type

If the OSS.DLINKED-PLUS directive is specified, then two structures are generated. One structure whose name has the suffix `_node` is a doubly linked list of values similar to the DLINKED representation. The second structure keeps pointers to the head and to the tail nodes in the doubly-linked list as well as the count field where the number of nodes in the list is stored. This structure is used to facilitate linked-list manipulation in the helper list API.

ASN.1:

```
SeqDPlusOfInt ::= SEQUENCE --<DLINKED-PLUS>-- OF INTEGER
```

C representation:

```
typedef struct SeqDPlusOfInt {
    struct SeqDPlusOfInt_node *head;
    struct SeqDPlusOfInt_node *tail;
    unsigned int    count;
} SeqDPlusOfInt;

typedef struct SeqDPlusOfInt_node {
    struct SeqDPlusOfInt_node *next;
    struct SeqDPlusOfInt_node *prev;
    int             value;
```

```
} SeqDPlusOfInt_node;
```

Illustrations of specification combinations:

Effect of SizeConstraint with OSS.UNBOUNDED and without -helperNames on the SEQUENCE OF type

When the OSS.UNBOUNDED directive is combined with SizeConstraint the UNBOUNDED representation is used, but the count field varies depending on the size needed to accommodate the maximum number of occurrences in the SEQUENCE OF. Without the SizeConstraint, an int always gets generated for the UNBOUNDED representation.

ASN.1:

```
SeqSzUnOfInt ::= SEQUENCE SIZE(33) --<UNBOUNDED>-- OF INTEGER
```

C representation:

```
typedef struct SeqSzUnOfInt {
    unsigned short  count;
    int             *value;
} SeqSzUnOfInt;
```

Effect of SizeConstraint with ARRAY on the SEQUENCE OF type

When the OSS.ARRAY directive is combined with SizeConstraint, the ARRAY representation is used. Note that the `count` field varies depending on the size needed to accommodate the maximum number of occurrences in the SEQUENCE OF. Without the SizeConstraint, an `int` always gets generated for the `count` field in the ARRAY representation.

ASN.1:

```
SeqSzArOfInt ::= SEQUENCE SIZE(33000) --<ARRAY>-- OF INTEGER
```

C representation:

```
typedef struct SeqSzArOfInt {
    unsigned short  count;
    int             value[33000];
} SeqSzArOfInt;
```

Effect of SizeConstraint with OSS.LINKED and OSS.POINTER on the SEQUENCE OF type

SizeConstraint has no effect on the LINKED representation. OSS.POINTER simply introduces an additional level of indirection.

ASN.1:

```
SeqSzLPOfInt ::= SEQUENCE SIZE(5) --<LINKED|POINTER>-- OF INTEGER
```

C representation:

```
typedef struct SeqSzLPOfInt {
    struct SeqSzLPOfInt *next;
    int             value;
} **SeqSzLPOfInt;
```

The following examples illustrate the effect of a SizeConstraint on TypeX *(where the Sequence-Of type is of the form* Street-addresses ::= SEQUENCE OF TypeX*) -*

Effect of SizeConstraint and ARRAY on SEQUENCE OF and SizeConstraint on TypeX

ASN.1:

```
Street-addresses ::= SEQUENCE SIZE (1..20) --<ARRAY>-- OF
                     VisibleString (SIZE (1..64))
```

C representation:

```
typedef struct Street_addresses {
    unsigned short  count;
    char            value[20][65];
} Street_addresses;
```

The `count` field above will hold the number VisibleString types contained in the `Street_addresses` structure.

Effect of SizeConstraint and OSS.LINKED on SEQUENCE OF and SizeConstraint on TypeX

ASN.1:

```
Street-addresses ::= SEQUENCE SIZE (1..20) --<LINKED>-- OF
                        VisibleString (SIZE (1..64))
```

C representation:

```
typedef struct Street_addresses {
    struct Street_addresses *next;
    char            value[65];
} *Street_addresses;
```

Notice how the SizeConstraint on the SEQUENCE OF has no effect, due to the OSS.LINKED directive.

Effect of SizeConstraint and OSS.UNBOUNDED on SEQUENCE OF and SizeConstraint on TypeX

ASN.1:

```
Street-addresses ::= SEQUENCE SIZE (1..20) --<UNBOUNDED>-- OF
                VisibleString (SIZE (1..64))
```

C representation:

```
typedef struct Street_addresses {
    unsigned short  count;
    char            (*value)[65];
} Street_addresses;
```

Illustrations of nested SEQUENCE OF types:

Nested SEQUENCE OF with no directives

ASN.1:

```
Street-addresses ::= SEQUENCE OF SEQUENCE OF
                VisibleString (SIZE (1..64))
```

C representation (using default set of directives without -helperNames):

```
typedef struct Street_addresses {
    struct Street_addresses *next;
    struct _seqof1 {
        struct _seqof1  *next;
        char            value[65];
    } *value;
} *Street_addresses;
```

C representation (using default sequence of directives and with -helperNames):

```
typedef struct Street_addresses {
    struct Street_addresses_node *head;
    struct Street_addresses_node *tail;
    unsigned int    count;
} Street_addresses;

typedef struct Street_addresses_node {
    struct Street_addresses_node *next;
    struct Street_addresses_node *prev;
    struct Street_addresses_seqof *value;
} Street_addresses_node;

typedef struct Street_addresses_seqof {
    struct Street_addresses_seqof_node *head;
    struct Street_addresses_seqof_node *tail;
    unsigned int    count;
} Street_addresses_seqof;

typedef struct Street_addresses_seqof_node {
    struct Street_addresses_seqof_node *next;
    struct Street_addresses_seqof_node *prev;
    char            *value;
} Street_addresses_seqof_node;
```

Nested SEQUENCE OF, OSS.LINKED directive,and OSS.UNBOUNDED directive

ASN.1:

```
Street-addresses ::= SEQUENCE --<LINKED>-- OF
                          SEQUENCE --<UNBOUNDED>-- OF
                              VisibleString (SIZE (1..64))
```

C representation:

```
typedef struct Street_addresses {
    struct Street_addresses *next;
    struct {
        unsigned int    count;
        char            (*value)[65];
    } value;
} *Street_addresses;
```

Effect of OSS.HelperMacro on the SEQUENCE OF type with -helperNames:

The OSS.HelperMacro directive applied to a SEQUENCE OF with UNBOUNDED representation results in generating the following helper macros:

1) _new or _new_pdu macro to allocate memory for structure and for its value field (if needed)
2) _copy or _copy_pdu macro to allocate memory and copy given input array of given length to the output storage
3) _setv macro to fill in an existing unbounded structure with input data (set reference to an array of a given length)

The OSS.HelperMacro directive applied to a SEQUENCE OF with DLINKED-PLUS representation results in generating the following helper macros:

1) _new or _new_pdu macro to allocate an empty structure
2) _node_new macro for DLINKED-PLUS node to allocate an empty structure for it
3) _head macro to get the first DLINKED-PLUS node in the list
4) _tail macro to get the last DLINKED-PLUS node in the list
5) _count macro to get the number of the DLINKED-PLUS nodes in the list
6) _node_next macro to get the pointer to the next DLINKED-PLUS node in the list
7) _node_prev macro to get the pointer to the previous DLINKED-PLUS node

More sophisticated macros for DLINKED-PLUS are not produced. It is recommended to use DLINKED-PLUS helper list API (see 3.3.35) instead of macros when working with this C representation of a SEQUENCE OF.

An example for both C representations follows.

ASN.1:

```
Data ::= SEQUENCE {
    pdu-numbers SEQUENCE OF INTEGER,
    encodings SEQUENCE OF OCTET STRING
}
```

Helper macros:

```
typedef struct _OctStr {
    unsigned int    length;
    unsigned char   *value;
} _OctStr;

…

typedef struct Data {
    struct Data_pdu_numbers *pdu_numbers;
    struct Data_encodings *encodings;
} Data;

…

typedef struct Data_pdu_numbers {
    unsigned int    count;
    int             *value;
} Data_pdu_numbers;

/* allocates memory for the sequence-of type and for its 'value' field (only if
 * count_ > 0) */
#define oss_Data_pdu_numbers_new(world, count_) \
    (Data_pdu_numbers *)oss__UnbSeqOf_new(world, count_, sizeof(int), FALSE)
```

```
/* allocates memory for the structure and initializes it by copying the input
 * value */
#define oss_Data_pdu_numbers_copy(world, value_, count_) \
    (Data_pdu_numbers *)oss__UnbSeqOf_copy(world, value_, count_, sizeof(int))

/* initializes 'value' and 'count' fields of a structure by given values */
#define oss_Data_pdu_numbers_setv(outp, value_, count_) \
    { \
    (outp)->value = (value_); \
    (outp)->count = (count_); \
    }

typedef struct Data_encodings {
    struct Data_encodings_node *head;
    struct Data_encodings_node *tail;
    unsigned int    count;
} Data_encodings;

typedef struct Data_encodings_node {
    struct Data_encodings_node *next;
    struct Data_encodings_node *prev;
    struct _OctStr  *value;
} Data_encodings_node;

…

/* allocates memory for an empty instance of the sequence-of type */
#define oss_Data_encodings_new(world) \
    (Data_encodings *)ossGetInitializedMemory(world, sizeof(Data_encodings))

/* allocates memory for an empty node of the list */
#define oss_Data_encodings_node_new(world) \
    (Data_encodings_node *)ossGetInitializedMemory(world, sizeof(Data_encodings_node))

/* get pointer to the first node in the list (the 'head' of the list) */
#define oss_Data_encodings_head(_list) \
    (_list)->head

/* get pointer to the last node in the list (the 'tail' of the list) */
#define oss_Data_encodings_tail(_list) \
    (_list)->tail

/* get number of nodes in the list */
#define oss_Data_encodings_count(_list) \
    (_list)->count

/* get pointer to the 'next' node */
#define oss_Data_encodings_node_next(_node) \
    (_node)->next

/* get pointer to the 'previous' node */
#define oss_Data_encodings_node_prev(_node) \
    (_node)->prev

…
```

7.6.23 SET

The following specifications affect the representation of the ASN.1 SET type:

ASN.1 specifications:

COMPONENTS OF notation
Contained subtype
DEFAULT keyword
Inner subtype
OPTIONAL keyword

Directive specification without -helperNames:

OSS.DefineName
OSS.ExtractType
OSS.FIELDNAME
OSS.InlineType
OSS.POINTER

Directive specifications with -helperNames:

OSS.DefineName
OSS.ExtractType
OSS.FIELDNAME
OSS.InlineType
OSS.HelperMacro
OSS.NoHelperMacro

By default, the C representation is of the form:

```
typedef struct Name1 {
        type1   element1;
                .
                .
                .
        typen   elementn;
} Name1;
```

The elements (e.g., `element1`) are the identifiers associated with each component of the SET type. If an identifier is missing, the compiler will generate a C identifier based on the ASN.1 type.

ASN.1:

```
NamesOfOfficers1 ::= SET {
        president         [0]   VisibleString (SIZE(1..32)),
        vicePresident     [1]   VisibleString (SIZE(1..32)),
        secretary         [2]   VisibleString (SIZE(1..32)),
                          [3]   VisibleString (SIZE(1..32)) }
```

C representation without -helperNames:

```
typedef struct NamesOfOfficers1 {
    char            president[33];
    char            vicePresident[33];
    char            secretary[33];
    char            treasurer[33];
} NamesOfOfficers1;
```

C representation without -helperNames:

```
typedef struct NamesOfOfficers1 {
    char            *president;
    char            *vicePresident;
    char            *secretary;
    char            *visibleString;
} NamesOfOfficers1;
```

Effect of the COMPONENTS OF Notation on the SET type

When the COMPONENTS OF notation is used in a SET, the elements of the referenced SET are copied inline.

For example,

ASN.1:

```
NamesOfOfficers3 ::= SET {
        president           [0] VisibleString (SIZE(1..32)),
        vicePresident       [1] VisibleString (SIZE(1..32)),
        COMPONENTS OF MoreOfficers }

MoreOfficers ::= [1] SET {
     secretary     [2] VisibleString (SIZE(1..32)),
     treasurer     [3] VisibleString (SIZE(1..32))}
```

C representation:

```
typedef struct NamesOfOfficers3 {
    char            president[33];
    char            vicePresident[33];
    char            secretary[33];
    char            treasurer[33];
} NamesOfOfficers3;
```

Effect of OPTIONAL and DEFAULT within the SET type

If any component(s) of the SET is OPTIONAL or has a DEFAULT value, the C representation is of the form:

```
typedef struct Name1{
    unsigned char   bit_mask;
#       define      elementj_present  0x80
    type1           element1;
                    .
                    .
                    .
    typen           elementn;
} Name1;
```

The variable, `bit_mask`, is used to indicate the presence or absence of SET components that are **OPTIONAL** or have a **DEFAULT** value, but which are not referenced via a pointer. (For those components that are referenced via a pointer, a non-NULL or NULL pointer value indicates the component's presence or absence.)

The type of the variable `bit_mask` (`char`, `short`, `int`, `long`, or `unsigned char*` array) depends on the smallest type that can accommodate all optional components that are not referenced via a pointer.

For each SET component represented in the bitmask, the compiler generates a bitmask manifest constant whose name is the component's name with a suffix of `_present`. The application programmer should perform a logical ***AND*** with this manifest constant and the `bit_mask` variable to determine if a specific component is present.

For example,

ASN.1:

```
NamesOfOfficers2::= SET {
        president           [0] VisibleString (SIZE(1..32)) OPTIONAL,
        vicePresident       [1] VisibleString (SIZE(1..32)),
        treasurer           [2] VisibleString OPTIONAL,
        secretary           [3] VisibleString (SIZE(1..32)) OPTIONAL }
```

C representation without -helperNames:

```
typedef struct NamesOfOfficers2 {
    unsigned char   bit_mask;
#       define      president_present 0x80
#       define      secretary_present 0x40
    char            president[33];  /* optional; set in bit_mask
                                     * president_present if present */
    char            vicePresident[33];
    char            *treasurer;  /* NULL for not present */
    char            secretary[33];  /* optional; set in bit_mask
                                     * secretary_present if present */
} NamesOfOfficers2;
```

Note that although field `treasurer` is OPTIONAL, it is not represented in the bitmask since its pointer can simply be set to the address of a character string or to NULL to indicate its presence or absence, respectively.

C representation with -helperNames:

```
typedef struct NamesOfOfficers2 {
    char            *president;  /* NULL for not present */
    char            *vicePresident;
    char            *treasurer;  /* NULL for not present */
    char            *secretary;  /* NULL for not present */
} NamesOfOfficers2;
```

Effect of the Contained and Inner subtype notations on the SET type

See sections 7.3.5 and 7.3.6 for a full discussion on the effect of these subtype notations.

Effect of OSS.HelperMacro on the SET type with -helperNames:

Macros generated for SET are fully identical to ones for corresponding SEQUENCE type. Refer to 7.6.21.

7.6.24 SET OF

The following specifications affect the representation of the ASN.1 SET OF type:

ASN.1 specifications:

Contained subtype
Inner subtype
SizeConstraint subtype

Directive specifications without -helperNames:

OSS.ARRAY
OSS.DLINKED
OSS.DLINKED-PLUS
OSS.ExtractType
OSS.InlineType
OSS.LINKED
OSS.POINTER
OSS.UNBOUNDED

The OSS.ARRAY, OSS.LINKED, OSS.DLINKED and OSS.UNBOUNDED directives are mutually exclusive.

Directive specifications with -helperNames:

OSS.DLINKED-PLUS
OSS.ExtractType
OSS.InlineType
OSS.HelperMacro
OSS.NoHelperMacro
OSS.NULLTERM
OSS.OBJHANDLE
OSS.PrintfunctionName
OSS.UNBOUNDED

By default without -helperNames, the LINKED representation is used:

```
typedef struct Name1 {
   struct Name1  *next;
   Type1         value;
} *Name1;
```

By default with -helperNames, the DLINKED-PLUS representation is generated for SET OF types with structured or pointered non-structured elements, and is of the form:

```
typedef struct Name1 {
   struct Name1   *next;
   Type1          value;
} *Name1;
typedef struct Name1 {
    struct Name1_node *head;
    struct Name1_node *tail;
    unsigned int    count;
} Name1;

typedef struct Name1_node {
```

```
    struct Name1_node *next;
    struct Name1_node *prev;
    Type1             *value;
} Name1_node;
```

The UNBOUNDED representation is generated for SET OF types with simple non-pointered elements, and is of the form:

```
typedef struct Name1 {
   unsigned int   count;
   ossBoolean     *value;
} Name1;
```

See A note about SEQUENCE OF representation: for more information about SET OF representation.

Effect of SizeConstraint on the SET OF type

In the absence of directives the SizeConstraint does not affect the representation.

ASN.1:

```
SetSzOfInt           ::= SET SIZE(5) OF INTEGER
```

C representation:

```
typedef struct SetSzOfInt {
    struct SetSzOfInt *next;
    int             value;
} *SetSzOfInt;
```

Effect of the OSS.ARRAY directive on the SET OF type

If the OSS.ARRAY directive is specified alone, then an array of types is generated, preceded by a count of the number of occurrences that follow. The OSS.POINTER directive is implied when the OSS.ARRAY directive is used. If SizeConstraint is also specified then, depending on the array size, a short or int is generated to indicate the size of the array.

ASN.1:

```
SetArOfInt           ::= SET --<ARRAY>-- OF INTEGER
```

C representation:

```
typedef struct SetArOfInt {
    unsigned int    count;
    int             value[1];  /* first element of the array */
} *SetArOfInt;
```

Effect of the OSS.DLINKED directive on the SET OF type

If the OSS.DLINKED directive is specified alone, then a pointer to a doubly linked list of values is generated.

ASN.1:

```
SetDlOfInt           ::= SET --<DLINKED>-- OF INTEGER
```

C representation:

```
typedef struct SetDlOfInt {
```

```
    struct SetDlOfInt *next;
    struct SetDlOfInt *prev;
    int             value;
} *SetDlOfInt;
```

Effect of the OSS.LINKED directive on the SET OF type

If the OSS.LINKED directive is specified alone, then a pointer to a linked list of values is generated.

ASN.1:

```
SetLnOfInt           ::= SET --<LINKED>-- OF INTEGER
```

C representation:

```
typedef struct SetLnOfInt {
    struct SetLnOfInt *next;
    int             value;
} *SetLnOfInt;
```

Effect of the OSS.UNBOUNDED directive on the SET OF type

If the OSS.UNBOUNDED directive is specified alone, then a pointer to a string of types is generated, preceded by a count of the number of occurrences pointed to.

ASN.1:

```
SetUnOfInt           ::= SET --<UNBOUNDED>-- OF INTEGER
```

C representation:

```
typedef struct SetUnOfInt {
    unsigned int    count;
    int             *value;
} SetUnOfInt;
```

Effect of the OSS.DLINKED-PLUS directive on the SET OF type

If the OSS.DLINKED-PLUS directive is specified, then two structures are generated. One structure whose name has the suffix `_node` is a doubly linked list of values similar to the DLINKED representation. The second structure keeps pointers to the head and to the tail nodes in the doubly-linked list as well as the count field where the number of nodes in the list is stored. This structure is used to facilitate linked-list manipulation in the helper list API.

ASN.1:

```
SetDPlusOfInt ::= SET --<DLINKED-PLUS>-- OF INTEGER
```

C representation:

```
typedef struct SetDPlusOfInt {
    struct SetDPlusOfInt_node *head;
    struct SetDPlusOfInt_node *tail;
    unsigned int    count;
} SetDPlusOfInt;

typedef struct SetDPlusOfInt_node {
    struct SetDPlusOfInt_node *next;
    struct SetDPlusOfInt_node *prev;
    int             value;
} SetDPlusOfInt_node;
```

Illustrations of specification combinations

Effect of SizeConstraint with OSS.UNBOUNDED on the SET OF type

When the OSS.UNBOUNDED directive is combined with SizeConstraint, the UNBOUNDED representation is used. However if -helperNames is not specified or implied, the type of the `count` field varies depending on the size needed to accommodate the maximum number of occurrences in the SET OF. Without the SizeConstraint, an `int` always gets generated for the `count` field for the UNBOUNDED representation.

ASN.1:

```
SetSzUnOfInt          ::= SET SIZE(33) --<UNBOUNDED>-- OF INTEGER
```

C representation without -helperNames:

```
typedef struct SetSzUnOfInt {
    unsigned short  count;
    int             *value;
} SetSzUnOfInt;
```

C representation with -helperNames:

```
typedef struct SetSzUnOfInt {
    unsigned int  count;
    int           *value;
} SetSzUnOfInt;
```

Effect of SizeConstraint with OSS.ARRAY on the SET OF type

When the OSS.ARRAY directive is combined with SizeConstraint the ARRAY representation is used, but the type of the `count` field varies depending on the size needed to accommodate the maximum number of occurrences in the SET OF. Without the SizeConstraint, an `int` always gets generated for the `count` field under the ARRAY representation.

ASN.1:

```
SetSzArOfInt          ::= SET SIZE(33000) --<ARRAY>-- OF INTEGER
```

C representation:

```
typedef struct SetSzArOfInt {
    unsigned short  count;
    int             value[33000];
} SetSzArOfInt;
```

Effect of SizeConstraint with OSS.LINKED and OSS.POINTER on the SET OF type

SizeConstraint has no effect on the LINKED representation. OSS.POINTER simply introduces an additional level of indirection.

ASN.1:

```
SetSzLnPtOfInt        ::= SET SIZE(5) --<LINKED|POINTER>-- OF INTEGER
```

C representation:

```
typedef struct SetSzLnPtOfInt {
    struct SetSzLnPtOfInt *next;
    int             value;
} **SetSzLnPtOfInt;
```

The following examples illustrate the effect of a SizeConstraint on TypeX *(where the Set-Of type is of the* form Street-addresses ::= SET OF TypeX*):*

Effect of SizeConstraint and OSS.ARRAY on SET OF and SizeConstraint on TypeX

ASN.1:

```
Street-addresses1 ::= SET SIZE (1..20) --<ARRAY>-- OF
                          VisibleString (SIZE (1..64))
```

C representation:

```
typedef struct Street_addresses1 {
    unsigned short  count;
    char            value[20][65];
} Street_addresses1;
```

Effect of SizeConstraint and OSS.LINKED on SET OF and SizeConstraint on TypeX

ASN.1:

```
Street-addresses2 ::= SET SIZE (1..20) --<LINKED>-- OF
                          VisibleString (SIZE (1..64))
```

C representation:

```
typedef struct Street_addresses2 {
    struct Street_addresses2 *next;
    char            value[65];
} *Street_addresses2;
```

Effect of SizeConstraint and OSS.UNBOUNDED for SET OF and SizeConstraint on TypeX

ASN.1:

```
Street-addresses3 ::= SET SIZE (1..20) --<UNBOUNDED>-- OF
                          VisibleString (SIZE (1..64))
```

C representation:

```
typedef struct Street_addresses3 {
    unsigned short  count;
    char            (*value)[65];
} Street_addresses3;
```

Illustrations of nested SET OF types:

Nested SET OF with no directives

ASN.1:

```
Street-addresses4 ::= SET OF SET OF
         VisibleString (SIZE (1..64))
```

C representation (using default set of directives and without -helperNames):

```
typedef struct Street_addresses4 {
    struct Street_addresses4 *next;
    struct _setof1 {
```

```
        struct _setof1  *next;
        char            value[65];
    } *value;
} *Street_addresses4;
```

C representation (using default set of directives and with -helperNames):

```
typedef struct Street_addresses4 {
    struct Street_addresses4_node *head;
    struct Street_addresses4_node *tail;
    unsigned int    count;
} Street_addresses4;

typedef struct Street_addresses4_node {
    struct Street_addresses4_node *next;
    struct Street_addresses4_node *prev;
    struct Street_addresses4_setof *value;
} Street_addresses4_node;

typedef struct Street_addresses4_setof {
    struct Street_addresses4_setof_node *head;
    struct Street_addresses4_setof_node *tail;
    unsigned int    count;
} Street_addresses4_setof;

typedef struct Street_addresses4_setof_node {
    struct Street_addresses4_setof_node *next;
    struct Street_addresses4_setof_node *prev;
    char            *value;
} Street_addresses4_setof_node;
```

Nested SET OF, the OSS.LINKED directive, and the OSS.UNBOUNDED directive

ASN.1:

```
Street-addresses5 ::= SET --<LINKED>-- OF
                SET --<UNBOUNDED>-- OF
                    VisibleString (SIZE (1..64))
```

C representation:

```
typedef struct Street_addresses5 {
    struct Street_addresses5 *next;
    struct {
        unsigned int    count;
        char            (*value)[65];
    } value;
} *Street_addresses5;
```

Effect of OSS.HelperMacro on the SET OF type with -helperNames:

Macros generated for SET OF are fully identical to ones for corresponding SEQUENCE OF type. Refer to 7.6.22.

7.6.25 Tagged

The representation in C of a tagged type is the same as that of the type being tagged. ASN.1 tags do not affect a type's C language representation.

ASN.1:

```
TaggedIntegerType      ::= INTEGER
MySeq ::= SEQUENCE {
     builtInInt        INTEGER,
     tagggedInt        TaggedIntegerType
}
```

C representation:

```
typedef int             TaggedIntegerType;

typedef struct MySeq {
    int             builtInInt;
    TaggedIntegerType tagggedInt;
} MySeq;
```

Note that `int` and `TaggedIntegerType` are equated above using a `typedef` statement.

7.6.26 TIME / DATE / TIME-OF-DAY / DATE-TIME / DURATION

Starting with versions 8.1.3 and 8.2, the ASN.1 compiler supports ISO 8601 time types by way of the ASN.1 built-in TIME type and the useful time types (DATE, DATE-TIME, TIME-OF_DAY, DURATION) along with the new subtypes; PropertySettings, TimePointRange, DurationRange, and RecurrenceRange, according to ITU-T Rec. X.680:2008 | ISO/IEC 8824-1:2008. The following specifications affect the representation of the ASN.1 TIME type and the useful time types:

ASN.1 specifications:

None.

Directive specification:

OSS.NULLTERM
OSS.POINTER
OSS.PrintFunctionName

Directive specification with -helperNames:

OSS.NULLTERM
OSS.PrintFunctionName
OSS.HelperMacro
OSS.NoHelperMacro

The C representation of the new time types is always a null-terminated string:

ASN.1:

```
Time      ::= TIME
t Time ::= "R/P12Y"
```

```
Date ::= DATE
d Date ::= "1999-10-12"
```

C representation:

```
typedef char *Time;
typedef char *Date;

Time t = "R/P12Y";
Date d = "1999-10-12";
```

Effect of OSS.HelperMacro on the TIME type

The OSS.HelperMacro directive applied to a TIME or other useful time type results in generation of the following helper macros:

1. the `_new` or `_new_pdu` macro to allocate memory for the string of a given length and initialize it to zeroes,
2. the `_copy` or `_copy_pdu` macro to allocate memory and copy a given input null terminated time value to it.

ASN.1:

```
--<OSS.HelperMacro Mod.Time>--
Mod DEFINITIONS ::= BEGIN
    Time ::= TIME
END
```

Helper macros:

```
typedef char    *Time;

/* allocates memory for Time_PDU */
#define oss_Time_new_pdu(world, length_) \
    (Time *)oss__Ptr_copy(world, oss__CharStr_new(world, length_))

/* allocates memory for Time_PDU and initializes it by copy of an input string */
#define oss_Time_copy_pdu(world, value_) \
    (Time *)oss__Ptr_copy(world, oss__CharStr_copy(world, value_))
```

7.6.27 UTCTime

The following specifications affect the representation of the ASN.1 UTCTime type:

ASN.1 specifications:

None.

Directive specification without -helperNames:

OSS.NULLTERM
OSS.POINTER
OSS.PrintfunctionName

Directive specification with -helperNames:

OSS.NULLTERM
OSS.PrintfunctionName
OSS.TIMESTRUCT
OSS.HelperMacro
OSS.NoHelperMacro

By default, the C representation is:

ASN.1:

```
Name1 ::= UTCTime
```

C representation without -helperNames:

```
typedef UTCTime Name1;
```

where UTCTime is defined in file `asn1hdr.h` as

```
typedef GeneralizedTime UTCTime;
```

C representation with -helperNames:

```
typedef char            *Name1;
```

Although UTCTime uses the same C structure as GeneralizedTime (see section 7.6.9), the semantics of the two structures differ in that when used to represent UTCTime, the `year` field is in YY format, and the `millisec` field is ignored by the encoder and set to zero by the decoder.

The UTCTime type can be represented either as UTCTime structs (i.e., { short year; short month; etc }) or as strings. Strings are used whenever the -lean option or the NULLTERM directive is used.

Effect of NULLTERM on the UTCTime type

When the NULLTERM directive is used, the UTCTime type is represented as strings. The -lean or -helperNames option has the same effect as NULLTERM.

ASN.1:

```
Name1     ::= UTCTime --<NULLTERM>--

 -- If local time is 7am on 2 January 1982 and coordinated universal
 -- time is 12 noon on 2 January 1982, the value of UTCTime is either of:
t1 Name1 ::= "8201021200Z"
t2 Name1 ::= "8201020700-0500"
```

C representation:

```
typedef char            *Name1;

Name1 t1 = "8201021200Z";
Name1 t2 = "8201020700-0500";
```

Effect of OSS.HelperMacro on the UTCTime type

The OSS.HelperMacro directive applied to a UTCTime with NULLTERM representation results in generating the following helper macros:

1) `_new` or `_new_pdu` macro to allocate memory for the string of given length and initialize it with zeroes

2) _copy or _copy_pdu macro to allocate memory and copy given input null terminated time value to it

The OSS.HelperMacro directive applied to a UTCTime with TIMESTRUCT directive results in generating the following helper macros:

1) _new or _new_pdu macro to allocate memory for the time structure and initialize it with zeroes
2) _copy_nullterm or _copy_nullterm_pdu macro to allocate memory for the time structure and initialize it by parsing a string containing a time value
3) _setv_nullterm macro to initialize existing time structure by parsing a string containing a time value

ASN.1:

```
--<OSS.HelperMacro Mod.SeqUTime>--
--<OSS.HelperMacro Mod.SeqUTime.n-time>--
--<OSS.TIMESTRUCT  Mod.SeqUTime.s-time>--
--<OSS.HelperMacro Mod.SeqUTime.s-time>--
Mod DEFINITIONS ::= BEGIN
SeqUTime ::= SEQUENCE {
    n-time UTCTime,
    s-time UTCTime
}
END
```

Note: When using a two digit year field, it is currently the industry standard to allow digits 50-99 represent the years 1950-1999 and the digits 00-49 to represent the years 2000-2049. Therefore when doing a comparison in your application, note that 50-99 is smaller than 00-49.

Helper macros:

```
typedef struct SeqUTime {
    char            *n_time;
    UTCTime         *s_time;
} SeqUTime;

…

/* allocates memory for a string of given length */
#define oss_SeqUTime_n_time_new(world, length_) \
    oss__CharStr_new(world, length_)

/* allocates memory and returns a copy of an input string */
#define oss_SeqUTime_n_time_copy(world, value_) \
    oss__CharStr_copy(world, value_)

…

/* allocates memory for the time structure */
#define oss_SeqUTime_s_time_new(world) \
    (UTCTime *)ossGetInitializedMemory(world, sizeof(UTCTime))

/* allocates memory for the time structure and initializes it by parsing a
 * string containing an UTCTime value */
#define oss_SeqUTime_s_time_copy_nullterm(world, value_) \
    oss__UTCTime_copy_nullterm(world, value_)
```

```
/* initializes the structure by parsing a string containing an UTCTime value */
#define oss_SeqUTime_s_time_setv_nullterm(world, outp, value_) \
    oss__UTCTime_setv_nullterm(world, outp, value_)
```

7.6.28 OID-IRI / RELATIVE-OID-IRI

Starting with version 9.0, the ASN.1 compiler supports the built-in OID-IRI and RELATIVE-OID-IRI types according to ITU-T Rec. X.680 (11/2008) | ISO/IEC 8824-1:2008.

ASN.1 specifications:

None

Directive specification without `-helperNames`:

OSS.NULLTERM
OSS.UNBOUNDED
OSS.POINTER

Note that the OSS.NULLTERM and OSS.UNBOUNDED directives are mutually exclusive, and the OSS.POINTER directive has an effect on the UNBOUNDED representation only.

Directive specification with -helperNames:

OSS.NULLTERM
OSS.UNBOUNDED
OSS.HelperMacro
OSS.NoHelperMacro

By default, the C representation is NULLTERM-POINTER.

Effect of NULLTERM on the OID-IRI type

ASN.1:

```
ID ::= OID-IRI --<NULLTERM>--
```

C representation:

```
typedef char            *ID;
```

Effect of UNBOUNDED on the OID-IRI type

ASN.1:

```
ID ::= OID-IRI --<UNBOUNDED>--
```

C representation without -helperNames:

```
typedef struct ID {
        unsigned int    length;
        char            *value;
} ID;
```

C representation with -helperNames:

```
typedef struct _UnbCharStr {
        unsigned int    length;
        char            *value;
```

```
} _UnbCharStr;

typedef _UnbCharStr ID;
```

The effect of OSS.HelperMacro on the OID-IRI and RELATIVE-OID-IRI types is the same as on the restricted character string types (see item 8.6.19.1 in this manual).

7.7 Effect of Type Extensibility

When writing ASN.1 modules, it is possible to provide mechanisms to ensure compatibility between the current version of the definitions and future versions. This is done through the use of the extensibility marker (…) or by using the OSS-specific OSS.EXTENSIBLE directive. In either case, the decoder is given special handling instructions for received components that are not present in the receiver's version of the ASN.1 data definitions. Specifically for each PDU, the decoder is told to:

- Ignore all elements of a CHOICE, SEQUENCE, or SET whose tags are not defined in the abstract syntax definition that was input to the compiler.
- Treat any bit as insignificant, if no name is assigned to it in the original ASN.1 source file (i.e. If a BIT STRING (or OCTET STRING) defines named bits and bits other than the named bits are encountered in the input stream, the decoder should ignore them).
- Ignore all unrecognized elements in a NamedNumberList of ENUMERATED and INTEGER types.
- Allow values not defined in the original constrained INTEGER type.
- Allow array sizes not specified in the original definitions of SEQUENCE OF and SET OF types.
- Allow unrecognized sizes and values for constrained character string types (i.e. IA5String / NumericString / PrintableString / BMPString / UniversalString).

7.7.1 Effect of the Extensibility Marker on Compiler Generated Files

The ASN.1 extensibility marker (`...`) and the OSS.EXTENSIBLE directive cause code to be generated into the `.c` file informing the encoder/decoder about the possibility of extensible elements.

If any elements are defined after the extensibility marker, a bitmask is generated for them indicating their presence or absence:

ASN.1:

```
NamesOfOfficers4 ::= SEQUENCE {
                president               VisibleString (SIZE(1..32)),
                vicePresident           VisibleString (SIZE(1..32)),
                secretary               VisibleString (SIZE(1..32)),
                ...
        }
```

```
NamesOfOfficers5 ::= SEQUENCE {
        president               VisibleString (SIZE(1..32)),
        vicePresident           VisibleString (SIZE(1..32)),
        secretary               VisibleString (SIZE(1..32)),
        ...,
        treasurer               VisibleString (SIZE(1..32))
}
```

C representation without -helperNames:

```
typedef struct NamesOfOfficers4 {
    char            president[33];
    char            vicePresident[33];
    char            secretary[33];
} NamesOfOfficers4;

typedef struct NamesOfOfficers5 {
    unsigned char   bit_mask;
#       define      treasurer_present 0x80
    char            president[33];
    char            vicePresident[33];
    char            secretary[33];
    char            treasurer[33];  /* extension #1; set in bit_mask
                                     * treasurer_present if present */
} NamesOfOfficers5;
```

C representation with -helperNames:

```
typedef struct NamesOfOfficers4 {
    char            *president;
    char            *vicePresident;
    char            *secretary;
} NamesOfOfficers4;

typedef struct NamesOfOfficers5 {
    char            *president;
    char            *vicePresident;
    char            *secretary;
    char            *treasurer;  /* extension #1; NULL for not present */
} NamesOfOfficers5;
```

7.8 Representing Information Object Classes

The following example is based on clauses 11 and 12 of **ITU-T Rec. X681 (2008) | ISO/IEC 8824-2 : 2008**

As noted in the ASN.1 standard, information object classes contain three common kinds of fields: (1) type fields, (2) object set fields, and (3) value fields. The ASN.1 compiler generates an `unsigned short` for each type field, an `ObjectSetEntry` structure (see below) for each object set field, and a variable representation (as outlined in section 7.6) for each value field. In the example below, `&ArgumentType` and `&ResultType` are type fields, `&Errors` and `&Linked` are object set fields, and `&resultReturned` and `&operationCode` are value fields.

The type fields are used to identify the specific type associated with the information object. By default, the compiler assigns ascending integers (starting from 1) to types declared for information objects.

C representations

The `ObjectSetEntry` structure can be used traverse a series of information objects which together make up an information object set. Each `object` field points to an information object. The first item in the `ObjectSetEntry` doubly-linked list has its `prev` pointer set to NULL while the last element has its `next` pointer set to NULL.

Notes:

1) The `ObjectSetEntry` structure is defined in `asn1hdr.h` as follows:

```
typedef struct ObjectSetEntry {
  struct ObjectSetEntry *next;
  void                  *object;
  struct ObjectSetEntry *prev;
  char                  *object_name;
} ObjectSetEntry;
```

2) Each declared information object set is has a unique identifier generated for it. These identifiers are `#defined` constants declared in the header file of the form

```
#define InfoObjSetName_OSET 1
```

The first set encountered gets a value of 1, the second set encountered gets a

Example:

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   OPERATION ::= CLASS
   {
      &ArgumentType OPTIONAL,
      &ResultType OPTIONAL,
      &Errors ERROR OPTIONAL,
      &Linked OPERATION OPTIONAL,
      &resultReturned BOOLEAN DEFAULT TRUE,
      &operationCode INTEGER UNIQUE
   }
   ERROR ::= CLASS
   {
      &ParameterType OPTIONAL,
      &errorCode INTEGER UNIQUE
   }

   intIsValid OPERATION ::=
   {
      &ArgumentType INTEGER,
      &ResultType BOOLEAN,
      &Errors {intIsNegative},
      &operationCode 7
   }
   intIsNegative ERROR ::=
   {
      &errorCode 1
   }
   intIsZero ERROR ::=
   {
```

```
      &errorCode 2
   }
   NegZeroInfoObjSet ERROR ::= {
      intIsNegative |
      intIsZero,
      ...
   }
END
```

C representation:

```
#define          IntIsValid_integer_PDU 1
#define          IntIsValid_boolean_PDU 2
#define          NegZeroInfoObjSet_OSET 1

typedef struct OPERATION {
    unsigned char   bit_mask;
#       define      ArgumentType_present 0x80
#       define      ResultType_present 0x40
#       define      resultReturned_present 0x20
    unsigned short  ArgumentType;  /* optional; set in bit_mask
                                    * ArgumentType_present if present */
    unsigned short  ResultType;  /* optional; set in bit_mask ResultType_present
                                  * if present */
    ObjectSetEntry  *Errors;  /* NULL for not present */
    ObjectSetEntry  *Linked;  /* NULL for not present */
    ossBoolean      resultReturned;  /* resultReturned_present not set in
                                      * bit_mask implies value is TRUE */
    int             operationCode;
} OPERATION;

typedef struct ERROR {
    unsigned char   bit_mask;
#       define      ParameterType_present 0x80
    unsigned short  ParameterType;  /* optional; set in bit_mask
                                     * ParameterType_present if present */
    int             errorCode;
} ERROR;

typedef int             IntIsValid_integer;

typedef ossBoolean      IntIsValid_boolean;

extern OPERATION intIsValid;

extern ERROR intIsNegative;

extern ERROR intIsZero;
```

Note above how bitmasks along with #`defined` constants were generated for the OPTIONAL and DEFAULT fields in the information object classes.

7.9 Representing Parameterized Types

The representation of parameterized types is simply that of the CHOICE, SEQUENCE or SET structure which results after the substitution of the specified parameter into the fields that refer to it.

ASN.1:

```
Module DEFINITIONS ::= BEGIN
   DesiredType ::= INTEGER

   Record {TypeForSubstitution} ::= SET
   {
      myTypeVar    TypeForSubstitution,
      filled       BOOLEAN
   }

   MyRec ::= Record {DesiredType}
END
```

C representation:

```
#define          MyRec_PDU 1

typedef int             DesiredType;

typedef struct MyRec {
    DesiredType     myTypeVar;
    ossBoolean      filled;
} MyRec;
```

Refer to section 4.5 for information about applying directives to parameterized types.

7.10 Representing ASN.1 comments

Starting with version 5.4.0, the ASN.1 compiler **by default** preserves ASN.1 comment content in the generated header file. To turn off this new default behavior, you can use the `-compat noASN.1Comments` option. Starting with version 8.1.0, the `–noComments` command line option can be used for the same purpose.

Default C representation:

The content of input ASN.1 comments are transferred to valid C comment constructs. Thus, the generated comments begin with '`/*`' and end with '`*/`'. Now if the `-c++` option was specified, single-line comment will begin with the '`//`' recognized C++ comment token. Refer to the examples below to gain a better understanding of the comment translation algorithm used.

Notes: Starting with version 5.4.0, the ASN.1 compiler now supports the new multi-line ASN.1 comment defined in the revised standard. To specify such a multi-line ASN.1 comment, simply start it with the '/*' token and end it with the '*/' token similar to how you specify comments in the C programming language.

a) Comments located on a line without any ASN.1 expression are transferred to the header file as they are:

ASN.1:

```
     -- pretty --
   /* integer
           type */

    -- one --  /* more
                   comment */

  Foo ::= INTEGER
```

C representation:

```
     /* pretty */
   /* integer
           type */
    /* one */  /* more
                   comment */
 typedef int Foo;
```

b) Comments located on a line containing an ASN.1 type, value assignment, etc. before the '::=' token are generated on a separate line right before the associated C declaration in the header file. The first line of such a comment will have the same indent as the C declaration associated with the comment:

ASN.1:

```
   /* This is a
     pretty */ Foo -- integer -- ::= INTEGER

-- This is a pretty -- f Foo -- value -- ::= 10
```

C representation:

```
 /* This is a
     pretty */ /* integer */
 typedef int Foo;

 /* This is a pretty */ /* value */
extern const Foo f;
```

c) Comments located on a line containing a type/value/etc. assignment after the completion of the assignment or after the '::=' token are generated immediately after the associated C-declaration on the same line in the header file:

ASN.1:

```
Foo ::= /* pretty
            integer */
         INTEGER    -- type
```

C representation:

```
typedef int Foo; /* pretty
            integer */
   /* type */
```

d) If comments are located immediately below a type/value/etc. assignment with the same indent as a comment from the previous line, there become associated with the C-declaration matching the given type/value/etc. assignment:

ASN.1:

```
a Atype ::= 10 -- pretty
               -- integer
               -- value
-- integer type
Atype ::= INTEGER
```

C representation:

```
/* integer type */
 typedef int Atype;

 extern Atype a; /* pretty */
                 /* integer */
                 /* value */
```

e) Comments located on a line after a curly brace '{}' appear in the generated header file right after the respective brace:

ASN.1:

```
        Foo ::= SEQUENCE { -- Comment before component
                foo     EMBEDDED PDV,
                bar     BOOLEAN
        } /* After
                item */
```

C representation:

```
        typedef struct Foo { /* Comment before component */
                EmbeddedPDV     foo;
                ossBoolean      bar;
```

```
        } Foo; /* After
                item */
```

f) Comments located before or after a module will appear in the generated header file before the `#include` statements or alternatively right before the end of the file:

ASN.1:

```
/* comment
                before D2 */

D2 DEFINITIONS ::=
...
END

/* comment after D2 */
```

C representation:

```
...
#ifndef OSS_ex
#define OSS_ex

/* comment
                before D2 */

#include "ossasn1.h"
...

#endif /* OSS_ex */

/* comment after D2 */
```

g) Comments associated with ASN.1 expressions from a non-root ASN.1 module having no referenced C-declarations in the generated header file are ignored by the ASN.1 compiler (see the "ignored comment" in the example below):

ASN.1:

```
D1 DEFINITIONS ::= BEGIN
        B ::= BOOLEAN -- this comment will be ignored by default
END

D2 DEFINITIONS ::= BEGIN
        A ::= BOOLEAN -- this comment will be transferred by default
END
```

h) If the ASN.1 compiler is invoked with the `-splitHeaders` option each comment appears only in the header file generated for the module where the given comment is located.

7.11 XML value notation in input ASN.1 files

Starting with version 6.0, the ASN.1 compiler is now capable of processing ASN.1 value references defined using XML value notation as defined in the ASN.1:2008 standard. The C representations for equivalent ASN.1 and XML value notations are identical.

In other words, XML value notation may now appear in the same input ASN.1 file as the other ASN.1 components in your specification.

An example of equivalent ASN.1 and XML value notations follow:

ASN.1 value notation:

```
AModule DEFINITIONS ::= BEGIN
   StringType ::= UTF8String
   IntType  ::= INTEGER

   studentName StringType ::= "John H. Doe"
   studentAge IntType ::= 23
END
```

is equivalent to the XML-based value notation:

```
XModule DEFINITIONS ::= BEGIN
   StringType ::= UTF8String
   IntType  ::= INTEGER

   studentName ::= <StringType>John H. Doe</StringType>
   studentAge ::= <IntType>23</IntType>
END
```

A more complex example of XML-based value notation may be found in shippable samples directory, `basic/xml/xsl/person.asn`.

For a detailed explanation of the mapping rules from ASN.1 to XML and vice versa, refer to the ASN.1:2008 standards document.

7.12 E-XER and character-encodable types

An ASN.1 type is called character-encodable if all of its possible values contain sequences of legal characters. A character is an atomic unit of text as specified by ISO/IEC 10646. Legal characters are tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646.

There are certain E-XER encoding instructions that can only be assigned to character-encodable ASN.1 types. For example, the ATTRIBUTE encoding instruction can only be assigned to character-encodable types. Similarly, the LIST encoding instruction can only be applied to the SET OF/ SEQUENCE OF types if all the components of the SET OF / SEQUENCE OF type are character-encodable, the USE-

UNION encoding instruction can only be applied to the CHOICE types if all the alternatives of the CHOICE are character-encodable.

The OSS ASN.1 compiler requires the user to specify either a Single Value Constraint or a Permitted Alphabet Constraint for character-encodable types. If there is any other constraint present on such types, an error is reported by the compiler when "`-exer`" compiler option is specified. This error can be ignored using "`-ignoreError`" compiler option.

It is recommended to define a separate character-encodable type, so that the character-encodable constraint definition is not repeated too many times in your ASN.1 syntax. For example, you can create another type "XMLCompatibleString" as shown below, and use this type in your ASN.1 syntax wherever a character-encodable type is needed.

Example:

```
XMLCompatibleString ::= UTF8String (FROM(
        {0, 0, 0, 9} |
        {0, 0, 0, 10} |
        {0, 0, 0, 13} |
        {0, 0, 0, 32} .. {0, 0, 215, 255} |
        {0, 0, 224, 0} .. {0, 0, 255, 253} |
        {0, 1, 0, 0} .. {0, 16, 255, 253}))

Attr ::= XMLCompatibleString

Color ::= XMLCompatibleString ("red" | "blue" | "green")

S ::= [LIST] SEQUENCE OF Color

C ::= [USE-UNION] CHOICE {
    c1  [TAG:0] XMLCompatibleString,
    c2  [TAG:1] XMLCompatibleString,
    c3  [TAG:2] XMLCompatibleString,
    c4  [TAG:3] XMLCompatibleString
}
```

8 The OSS macro processor

8.1 Introduction

The OSS macro expander/syntax checker expands all macro notation, performs full syntax checking of ASN.1 source files, and produces source listings (with all imported items expanded) and diagnostic messages.

As with all the OSS ASN.1 Tools, no preprocessing (e.g., adding delimiters, etc.) of the ASN.1 source is required before invoking the OSS macro expander/syntax checker.

The OSS macro processor is part of the OSS ASN.1 compiler. No separate program has to be run to check and expand macros.

8.2 Expansion of ASN.1 macro notation

- The ASN.1 type that the macro instance returns (i.e. the type of the value that the macro assigns to the local value reference, VALUE) is used to determine how the macro should be expanded. If a single assignment is made to VALUE in an instance of the macro, the type associated with that value is returned.

- A CHOICE type is generated when the type returned by the macro instance is indeterminate due to there being zero, or more than one, assignment to VALUE in the macro instance. If no assignment is made to VALUE in the macro instance, the resulting CHOICE contains each possible type that the macro is capable of returning. If more than one assignment is made to VALUE in the macro instance, the resulting CHOICE contains the type associated with each VALUE to which an assignment was made.

- Generate an ANY type when the type returned by the macro instance is completely indeterminate; such as when the value assigned to VALUE is an argument that is present only in the macro's VALUE NOTATION, but the macro is used strictly as a type with no value being specified.

Any macro instance which has an indeterminate returned type is non-standard and its use is discouraged. The compiler tolerates these types of macros only so as to support any such already existing macros. A warning message is issued for all non-standard macro usage.

8.3 Restrictions on the ASN.1 macro notation

The OSS compiler supports user-defined ASN.1 macros, with the following restrictions:

- If a macro is defined by another macro, that other macro must first be defined.

  Example:

```
ABSTRACT-OPERATION MACRO ::= OPERATION
```

In this example OPERATION must be defined before it can be used in defining ABSTRACT-OPERATION.

- The character '[' is not supported in macros as an astring.

- All macros must be defined before they are used if, in the instance of use of that macro, arguments are specified.

  Example:

```
A ::= ERROR

B ::= ERROR PARAMETER BOOLEAN

ERROR MACRO ::=
BEGIN
    TYPE NOTATION  ::= Parameter
    VALUE NOTATION ::= value(VALUE CHOICE{
                localValue  INTEGER,
                globalValue OBJECT IDENTIFIER})

    Parameter      ::= "PARAMETER" NamedType | empty
    NamedType      ::= identifier type | type
END

C ::= ERROR PARAMETER BOOLEAN
```

A is acceptably defined, since in this instance ERROR is not being passed an argument. C is acceptably defined, although it is being passed arguments, since the ERROR macro is defined before it is referenced by C. However, B is unacceptably defined, since ERROR is referenced (by B) before it is defined and because arguments are being passed to it.

- Macro arguments and values that do not have an effect on the returned type are checked by the compiler for correct usage, but have no effect on the compiler output. The compiler treats such parameters and values as formal comments whose meaning is understood only by the user. As such, it validates their usage but does not attempt to interpret their meaning.

  For example, using the ERROR definition which is specified above in this section, one could create an instance of ERROR as follows:

```
C ::= ERROR PARAMETER BOOLEAN
```

Note that in the macro definition the type of the value returned by ERROR is `CHOICE {localValue INTEGER, globalValue OBJECT IDENTIFIER}`; this type is independent of whatever parameter was passed to ERROR. As such, the parameters are syntax checked by the compiler to ensure that if the word PARAMETER occurs immediately after ERROR, it is followed by a valid ASN.1 type. These parameters are then forgotten and have no further effect. (Note: In some instances this behavior is contrary to the type/value compatibility rules which are poorly implied in ASN.1).

8.4 Some examples of macro use

The following examples are mostly drawn from the ITU-T Recommendations X.208 and X.410:

Example 1. Macro whose Returned Type is Independent of the Instance of the Type Notation and the Value Notation

Suppose an ASN.1 macro, ERROR, was defined as follows:

```
ERROR MACRO ::=
BEGIN
    TYPE NOTATION  ::= "PARAMETER" NamedType | empty
    VALUE NOTATION ::= value(VALUE INTEGER)
    NamedType      ::= identifier type | type
END
```

And then it was used as follows:

```
ErrorRecord ::= SEQUENCE {
    rectype    ERROR PARAMETER VisibleString,
    sector     INTEGER
}
```

The following definition is always implied, regardless of what parameter was passed to the macro, since the type that the macro returns is always INTEGER::

```
ErrorRecord ::= SEQUENCE {
     rectype    INTEGER,
     sector     INTEGER
}
```

Example 2. Another macro whose Returned Type is Independent of the Instance of the Type Notation and the Value Notation

Suppose an ASN.1 macro, OPERATION, was defined as follows:

```
OPERATION MACRO ::=
BEGIN
    TYPE NOTATION  ::= "ARGUMENT" NamedType
                       Result Errors | empty
    VALUE NOTATION    ::= value(VALUE INTEGER)
    Result     ::= empty | "RESULT" NamedType
    Errors     ::= empty | "ERRORS" "{"ErrorNames"}"
    NamedType         ::= identifier type | type
    ErrorNames        ::= empty | IdentifierList
    IdentifierList    ::= identifier | IdentifierList
                              "," identifier
END
```

And then it was used as follows:

```
cancel OPERATION
    ARGUMENT   jobname IA5String
    RESULT jobCancelled NULL
    ERRORS {jobNotFound, unauthorizedCancel}
::= 1

Message ::= SEQUENCE {
    invokeID   INTEGER,
           OPERATION,
    argument   ANY
}
```

The following is always generated, regardless of what parameters were passed to the macro, since the type that the macro returns is always INTEGER:

```
lookup INTEGER ::= 1

Message ::= SEQUENCE {
    invokeID   INTEGER,
           INTEGER,
    argument   ANY
}
```

Example 3. Macro Whose Returned Type is Independent of the Instance of the Value Notation, but is Dependent on the Instance of the Type Notation

Suppose an ASN.1 macro, PAIR, was defined as follows:

```
PAIR MACRO ::=
BEGIN
  TYPE NOTATION ::=
      "TYPEX" "=" type(LocalType1)
      "TYPEY" "=" type(LocalType2)
  VALUE NOTATION ::=
      "("
      "X" "=" value(LocalValue1 LocalType1)
      ","
      "Y" "=" value(LocalValue2 LocalType2)
      <VALUE SEQUENCE {LocalType1, LocalType2} ::=
             {LocalValue1, LocalValue2}>
      ")"
END
```

And then it was used as follows:

```
AgeAndMarried ::= PAIR
    TYPEX = INTEGER
    TYPEY = BOOLEAN

serena AgeAndMarried ::= (X = 2, Y = FALSE)
```

The following is generated. Note that in this example the generated types are based on the macro parameters.

```
AgeAndMarried ::= SEQUENCE {
    INTEGER,
```

```
        BOOLEAN
    }

    serena AgeAndMarried ::= {2, FALSE}
```

Example 4. Macro Whose Returned Type is Dependent on the Instance of both the Type and Value Notations (Contains Multiple Assignments to VALUE)

Suppose an ASN.1 macro, BIND, was defined as follows:

```
BIND MACRO ::=
BEGIN
    TYPE NOTATION  ::= Argument  Result  Error
    VALUE NOTATION ::= Argument-value|Result-value|Error-value

    Argument       ::= empty | "ARGUMENT" Name type(Argument-type)
Result     ::= empty | "RESULT" Name type (Result-type)
    Error          ::= empty | "BIND-ERROR" Name type(Error-type)
    Name           ::= empty | identifier

    Argument-value ::= empty | "ARGUMENT"
                   value(Arg-value Argument-type)
    <VALUE [16] EXPLICIT Argument-type ::= Arg-value>

    Result-value   ::= empty | "RESULT"
                   value(Res-value Result-type)
    <VALUE [17] EXPLICIT Result-type   ::= Res-value>

    Error-value       ::= empty | "ERROR"
                   value(Err-value Error-type)
    <VALUE [18] EXPLICIT Error-type    ::= Err-value>
END
```

And then it was used as follows:

```
    BindA ::= BIND
    BindB ::= BIND RESULT INTEGER
    b BIND ARGUMENT INTEGER ::= ARGUMENT 2
```

A warning message is issued for BindA and BindB, and the following is generated:

```
    BindA ::= CHOICE {
        [16] ANY,
        [17] ANY,
        [18] ANY
    }

    BindB ::= [17] INTEGER

    b [16] INTEGER ::= 2
```

Appendices

A. Configuration directives

The configuration directives are solely for OSS use and are listed below:

```
ALIGNMENT n,n,...,n
CHARENUMSIZE nn
CHARSIZET nn
DOUBLESIZE nn
FLOATSIZE nn
GENERICENUMSIZE nn
INTMAX nn
INTMIN nn
INTSIZE nn
LLONGSUFFIX nn
LONGDBLSIZE nn
LONGENUMSIZE nn
LONGLONGMAX nn
LONGLONGMIN nn
LONGLONGSIZE nn
LONGMAX nn
LONGMIN nn
LONGSIZE nn
MACHINETYPE ss
MANDATORYOPTIONS ss
MAXSIZET nn
OPERATINGSYSTEM ss
POINTERSIZE nn
PRAGMA "s1", "s2", ..., "sn"
SHORTENUMSIZE nn
SHORTMAX nn
SHORTMIN nn
SHORTSIZE nn
TARGET ss
UALIGN nn
UCHARENUMSIZE nn
UINTMAX nn
ULLONGSUFFIX nn
ULONGLONGMAX nn
ULONGMAX nn
USHORTENUMSIZE nn
USHORTMAX nn
```

B. Demonstrative examples

Use of the -genDirectives Command-line Option

As a first example, given the ASN.1 syntax:

```
Module DEFINITIONS ::= BEGIN
       Type1 ::= BIT STRING {ambig(0), field(1)}
       Type2 ::= SEQUENCE {
       ambig SET OF INTEGER
    }
END
```

the compiler would generate the following structure:

```
typedef struct Type1 {
    unsigned int    length;  /* number of significant bits */
    unsigned char   *value;
} Type1;
#define                 Type1_ambig 0x80
#define                 Type1_ambig_byte 0
#define                 field 0x40
#define                 field_byte 0

typedef struct Type2 {
    struct _setof1 {
        struct _setof1  *next;
        int             value;
    } *ambig;
} Type2;
```

If you compiled the above ASN.1 syntax with the -genDirectives option and then later modified the ASN.1 syntax to add a new SET OF type before the existing one shown above, the original name of the existing SET OF type remains unchanged and a new name is generated for the new SET OF type. Had you not compiled with the -genDirectives option, the new SET OF type would get the name generated for the existing SET OF type and the existing SET OF type would get a new name; thus, forcing you to modify your code accordingly.

The generated directive is

```
--<OSS.TYPENAME Module.Type2.ambig "_setof1">--
```

As a second example, given the ASN.1 syntax:

```
Mod DEFINITIONS ::= BEGIN
       Type1 ::= SET OF SET OF INTEGER
END
```

The ASN.1 compiler would create the following .gen file:

```
-- Directives captured from the ASN.1 input:
-- none
```

```
    -- Directives artificially generated by the compiler:
    --<OSS.TYPENAME Mod.Type1.* "_setof1">--
    --<OSS.InlineType Mod.Type1.*>--
```

Note that anonymous type name (the `SET OF`) is replaced by "*" according to the X/Open ASN.1/C++ API. For more information on the X/Open ASN.1/C++ API, see section 4.3.1.

As a third example, given the ASN.1 syntax in the file `def.asn`:

```
    --<ASN1.FileNameSuffix "Test">--

    Module DEFINITIONS --<ASN1VER 1994>-- ::= BEGIN
        X ::= INTEGER --<HUGE>--
        BigInteger ::= INTEGER
            RecTypes ::= BIT STRING {primes(0), odds(1), evens(2)}
        RecordContainingSetOfPrimes ::= SEQUENCE {
            primes SET --<FIELDNAME "odds" | TYPENAME "P" | VALNAME "v">-- OF
INTEGER
        }
        T ::= SET OF SEQUENCE OF SET OF BOOLEAN
    END
```

The ASN.1 compiler is invoked by the command:

```
    asn1 def.asn -gen def.gen
```

The following informatory message is printed:

```
    C0064I: There are unused standard or OSS-specific directives.  Look at all
    lines containing "WARNING:" in the .gen file.
```

This means that one or more directives in ASN.1 syntax were not used and thus have no effect. Each of the directives that have no effect are commented in the file `def.gen`. The contents of the file `def.gen` are shown below::

```
    -- Directives captured from the ASN.1 input:

    -- <ASN1.FileNameSuffix "Test"> -- -- WARNING: could not use the directive
    --<OSS.ASN1VER Module 1994>--
    --<OSS.HUGE Module.X>--
    --<OSS.FIELDNAME Module.RecordContainingSetOfPrimes.primes "odds">--
    --<OSS.TYPENAME Module.RecordContainingSetOfPrimes.primes "P">--
    --<OSS.VALNAME Module.RecordContainingSetOfPrimes.primes "v">--

    -- Directives artificially generated by the compiler:

    --<ASN1.Nickname Module.RecTypes.odds RecTypes_odds>--
    --<OSS.InlineType Module.RecordContainingSetOfPrimes.primes>--
    --<OSS.TYPENAME Module.T.* "_seqof1">--
    --<OSS.InlineType Module.T.*>--
    --<OSS.TYPENAME Module.T.*.* "_setof1">--
    --<OSS.InlineType Module.T.*.*>--
```

Note the message next to the `ASN1.FileNameSuffix` directive:

```
    "-- WARNING: could not use the directive".
```

This message simply states that the directive has no effect.

The directive

```
--<OSS.ASN1VER Module 1994>--
```

is a module OSS-specific directive. An absolute reference in the directive is the module name `Module` and an operand is `1994`.

The directive

```
--<OSS.HUGE Module.X>--
```

is a local OSS-specific directive. An absolute reference in the directive is a type reference specified with the module name `Module.X`.

In the above example, the ASN.1 compiler also generated two directives for the names which could be changed by the ASN.1 compiler in case recompiling is done. Thus, for the type

```
T ::= SET OF SEQUENCE OF SET OF BOOLEAN
```

the directive

```
--<OSS.TYPENAME Module.T.*.* "_setof1">--
```

is generated. The first "`*`" corresponds to `SEQUENCE OF SET OF BOOLEAN` and the second to `SET OF BOOLEAN`.

As a last example (a more complicated one), given the following ASN.1 syntax:

```
Mod DEFINITIONS ::= BEGIN
    Type ::=  SET OF SET OF SEQUENCE OF SEQUENCE (SIZE(1..20)) OF
            CHOICE {
                a1 SEQUENCE {
                    b1 CHOICE {
                            c1 [1] SET {
                            d1 [1] SET OF INTEGER,
                                d2 [2] SET OF SET {
                                    f1 INTEGER,
                                    f2 REAL
                                }
                          },
                        c2 [2] SET OF SET {
                            e1 INTEGER,
                                e2 REAL
                            }
                    },
                    b2 SET OF SET {
                            d1 INTEGER,
                        d2 REAL
                    }
            },
            a2 SET OF SET {
```

```
                    c1 INTEGER,
                    c2 REAL
                }
            }
END
```

The compiler will produce the following `.gen` file:

```
-- Directives artificially generated by the compiler:

--<OSS.TYPENAME Mod.Type.* "_setof6">--
--<OSS.InlineType Mod.Type.*>--
--<OSS.TYPENAME Mod.Type.*.* "_seqof2">--
--<OSS.InlineType Mod.Type.*.*>--
--<OSS.TYPENAME Mod.Type.*.*.* "_seqof1">--
--<OSS.InlineType Mod.Type.*.*.*>--
--<OSS.TYPENAME Mod.Type.*.*.*.*.a1 "_seq1">--
--<OSS.InlineType Mod.Type.*.*.*.*.a1>--
--<OSS.TYPENAME Mod.Type.*.*.*.*.a1.b1.c1 "_set1">--
--<OSS.InlineType Mod.Type.*.*.*.*.a1.b1.c1>--
--<OSS.TYPENAME Mod.Type.*.*.*.*.a1.b1.c1.d1 "_setof1">--
--<OSS.InlineType Mod.Type.*.*.*.*.a1.b1.c1.d1>--
--<OSS.TYPENAME Mod.Type.*.*.*.*.a1.b1.c1.d2 "_setof2">--
--<OSS.InlineType Mod.Type.*.*.*.*.a1.b1.c1.d2>--
--<OSS.TYPENAME Mod.Type.*.*.*.*.a1.b1.c2 "_setof3">--
--<OSS.InlineType Mod.Type.*.*.*.*.a1.b1.c2>--
--<OSS.TYPENAME Mod.Type.*.*.*.*.a1.b2 "_setof4">--
--<OSS.InlineType Mod.Type.*.*.*.*.a1.b2>--
--<OSS.TYPENAME Mod.Type.*.*.*.*.a2 "_setof5">--
--<OSS.InlineType Mod.Type.*.*.*.*.a2>--
```

Tip: If you want to replace all the OSS-specific old-style directives with the new OSS-specific ones and write them into a separate file, you should first run the ASN.1 compilerASN.1 compiler with the `-genDirectives` option. You may then remove all the directives from your ASN.1 syntax and invoke the ASN.1 compiler with the generated `.gen` file as the first parameter on the command line.

C. The OSS BER Type-Length-Value Utility (OSSTLV.EXE)

Note: OSSTLV.EXE is only available on common platforms like Linux, Windows, and Solaris, and may not be available for your embedded system port. If you are interested in OSSTLV.EXE for your platform, contact OSS Nokalva Sales at info@oss.com.

The following excerpt is the manual page for the OSS TLV utility:

```
osstlv(1)                 OSS TLV Print Utility                    osstlv(1)

NAME
     OSS Nokalva TLV Print Utility  Version 1.0.1



SYNOPSIS

     osstlv [-b|-t] <input>  [+off[,num]] [-dec|-hex|-syn]
               [-out <output>]  [-notitles]  [-noaddition]
               [-pad  <byte> ]  [-noshort ]  [-help]


DESCRIPTION

     The OSS TLV Print Utility, osstlv, is a utility program that takes a BER
     or DER encoding in ASCII or binary format from an input file and writes
     it to an output file in one of three different type-length-value (TLV)
     formats.  All references below to BER apply equally to DER.

     BER encodings can be printed in any one of the following formats:
     hexadecimal-TLV, decomposed-TLV or syntax-TLV.  These formats are
     described below under "Description of TLV Formats."


OPTIONS

     Keyword options, except -help, can be abbreviated to the fewest number
     of letters needed to uniquely identify them (e.g., -binary can be
     abbreviated -b).

     -binary infile    Input file, infile, contains a BER/DER encoding in
                       binary format.  If infile is a dash (e.g., -b -) then
                       the input is read from stdin.  -binary is the default
                       input format if the input file name is specified without
                       being preceded by either the -binary or -text option.

     -text infile      Input file, infile, contains a BER/DER encoding in
                       hexadecimal format.  If infile is a dash (e.g., -t -)
                       then the input is read from stdin.

     +offset,num       Print a number of bytes, num, of a BER/DER encoding from
                       an input file starting from the position, offset.
```

```
-hexadecimal      Print a BER/DER encoding in hexadecimal-TLV format.
                  This is the default output format.

-decomposed       Print a BER/DER encoding in decomposed-TLV format.

-noshort          Don't truncate output lines that would otherwise be
                  longer than a line.  By default, such lines are
                  truncated to fit one line without wrapping.

-notitles         Don't print titles "Type", "Length", and "Contents"
                  when printing a BER encoding in the syntax-TLV format.
                  By default, titles are printed.

-noaddition       Don't print additional clarifying values for INTEGER and
                  REAL types when printing in the decomposed-TLV or
                  syntax-TLV formats.  (see example 3 below.)  Use this
                  option if the output is to be parsed and you wish to
                  have output lines that follow the same pattern
                  regardless of type.  By default, additional values
                  are printed.

-output outfile   The output is to be written to file "outfile".  Output
                  is written to stdout if the -output option is omitted.
                  If outfile is a dash (e.g., -o -) then the output file
                  name is read from stdin.

-pad byte         Skip possible padding areas between concatenated records
                  filled with <byte>s.  <byte> represents a
                  hexadecimal number (e.g., 0x00 for hex 00, 0xFF for hex FF)
                  or a non-negative decimal number from 0 to 255 (e.g. 10 or
                  128).

-syntax           Print a BER/DER encoding in syntax-TLV format.

-help             Print help information.


Examples:

1. Print the BER binary file, foo.dat, in hexadecimal-TLV format to
   out.txt without truncating output lines:

   osstlv -b foo.dat -nos -o out.txt

2. Print the BER text file, foo.txt, in syntax-TLV format to stdout
   without printing titles:

   osstlv -t foo.txt -s -not

3. Print the BER binary file, foo.dat, in decomposed-TLV format to
   stdout without printing additional hexadecimal values for INTEGER
   and REAL types:

   osstlv -b foo.dat -d -noa

   Without -noa a line may be printed as:

       18:d= 3 hl=2 l=   9 prim: REAL       :{2365,10,28} (2.365000E+031)
```

```
        With -noa the same line would be printed as:

            18:d= 3 hl=2 l=   9 prim: REAL       :{2365,10,28}

     4. To skip any number of zero padding bytes between concatenated records,
        specify the following command line:

            osstlv in.ber -pad 0x00

Description of TLV Formats
--------------------------

     BER encodings can be printed in any one of the following formats:

     - Hexadecimal-TLV
     - Decomposed-TLV
     - Syntax-TLV

     1. The hexadecimal-TLV format allows a BER/DER encoding to be viewed in
        hexadecimal format with type, length, and value information separated
        from each other by a space or newline.  All nested encodings are printed
        in an indented manner.  Lengths of value 80 (hex) are not the true
        length of the value, but indicate that the encoding is indefinite
        in length, and that the occurrence of matching end-of-contents octets
        (00 00) indicates the end of the value.

        Example:

        Definite length encoding:

           30 2A
               30 06
                   02 01 0A
                   01 01 FF
               30 00
               30 1E
                   09 09 803509540A2BE520DF
                   17 11 393230343233313734333334 2D30373030

        Indefinite length encoding:

           30 80
               30 80
                   02 01 0A
                   01 01 FF
                   00 00
               30 80
                   00 00
               30 80
                   09 09 803509540A2BE520DF
                   17 11 393230343233313734333334 2D30373030
                   00 00
               00 00

     2. The decomposed-TLV format allows a BER/DER encoding to be viewed
        in a format that lends itself to post-processing.  The structure
        of each output line is:
```

```
<offset>:d=<lvl> hl=<head> l=<len> {prim | cons}: <type> :<contents>

with the following meanings:

    offset   is the offset from the beginning of the message.

    lvl      is the level of nesting, 1 - for the outermost level.

    head     is the number of octets occupied by the header information
             (type and length fields) of an encoding.  If an input
             value is misencoded and the length of the tag-length pair
             is greater than the remaining data length, three question
             marks, "???", are printed in this field.

    len      is the decimal length of an encoding, -128 for encodings
             with an indefinite length.  In the case of misencoded
             values when a value in the length field exceeds the
             remaining data length, the remaining data length with a
             question mark is printed (e.g., "?35").  If an input
             value is misencoded so that the length cannot be found,
             three question marks are printed: "???".

    type     is the built-in name of an ASN.1 type, if the type
             information contains a UNIVERSAL class, else it is the
             tag class and number in brackets.  The tag class is
             printed only for the APPLICATION and PRIVATE classes;
             it is not printed for context-specific classes.  For
             encodings with an indefinite length "END-OF-CONTENTS" is
             printed when the end of the current encoding is reached.
             The type information is prefixed with either "prim:",
             indicating that the type is a primitive one, or "cons:",
             indicating that it is a constructed one (i.e., one that
             contains zero or more types, lengths and values nested
             within.)

    contents is the value component of the encoding.  The contents
             field is present only if the encoding is primitive.  If
             an error occurs when converting a primitive encoding
             value, its contents is printed as a hexadecimal string.

Example:

Definite length encoding:

    0:d= 1 hl=2 l=  42 cons: SET
    2:d= 2 hl=2 l=   6 cons: SET
    4:d= 3 hl=2 l=   1 prim: INTEGER      :10 (0x0a)
    7:d= 3 hl=2 l=   1 prim: BOOLEAN      :TRUE
   10:d= 2 hl=2 l=   0 cons: SET
   12:d= 2 hl=2 l=  30 cons: SET
   14:d= 3 hl=2 l=   9 prim: REAL         :{2365,10,28} (2.365000E+031)
   25:d= 3 hl=2 l=  17 prim: UTCTime      :920423174334-0700

Indefinite length encoding:

    0:d= 1 hl=2 l=-128 cons: SET
    2:d= 2 hl=2 l=-128 cons: SET
```

```
 4:d= 3 hl=2 l=   1 prim: INTEGER      :10 (0x0a)
 7:d= 3 hl=2 l=   1 prim: BOOLEAN      :TRUE
10:d= 3 hl=2 l=   0 prim: END-OF-CONTENTS
12:d= 2 hl=2 l=-128 cons: SET
14:d= 3 hl=2 l=   0 prim: END-OF-CONTENTS
16:d= 2 hl=2 l=-128 cons: SET
18:d= 3 hl=2 l=   9 prim: REAL         :{2365,10,28} (2.365000E+031)
29:d= 3 hl=2 l=  17 prim: UTCTime      :920423174334-0700
48:d= 3 hl=2 l=   0 prim: END-OF-CONTENTS
50:d= 2 hl=2 l=   0 prim: END-OF-CONTENTS
```

3. The syntax-TLV format includes three fields: "Type", "Length", "Contents", similar to the format used in X.690 to show examples of encodings, where:

    Type — is the name of the ASN.1 type, if this information can be determined, else the tag class and number of the encoding.

    Length — is the length of the encoding or "indef" for indefinite length encodings. In the case of misencoded values when a value in the length field exceeds the remaining data length, the remaining data length with a question mark is printed, for example "?35", where 35 is not a valid length. If an input value is misencoded so that the length cannot be found, three question marks are printed: "???".

    Contents is the contents if the encoding is primitive, or empty for constructed encodings.

Example:

Definite length encoding:

```
Type         Length  Contents
SET          42
    Type         Length  Contents
    SET          6
         Type         Length  Contents
         INTEGER      1       10 (0x0a)
         BOOLEAN      1       TRUE
    SET          0
    SET          30
        Type         Length  Contents
        REAL         9       {2365,10,28} (2.365000E+031)
        UTCTime      17      920423174334-0700
```

Indefinite length encoding:

```
Type         Length  Contents
SET          indef
    Type         Length  Contents
    SET          indef
        Type         Length  Contents
        INTEGER      1       10 (0x0a)
        BOOLEAN      1       TRUE
        END-OF-CONTENTS
    SET          indef
```

```
                    END-OF-CONTENTS
               SET          indef
                   Type         Length  Contents
                   REAL         9       {2365,10,28} (2.365000E+031)
                   UTCTime      17      920423174334-0700
                   END-OF-CONTENTS
               END-OF-CONTENTS
```

Error messages

One of the following error messages is generated if an error occurs while parsing the command line:

"Error opening input file, foo.txt. No such file or directory."

"Error opening output file, foo.out."

"Error reading input file, foo.dat."

"Error reading, stdin."

"Input file name is missing."

"Too many input file names."

"Output file name is missing."

"Too many output file names."

"Input text file, foo.txt, contains non-hexadecimal characters."

"Invalid number of bytes to be read from input file, 12bb45."

"Invalid offset into input file, 12aa67."

"Missing mandatory input file name."

"Output file name is missing."

"Unidentified input parameter, -foo."

One of the following error messages is generated if an error occurrs while parsing an input BER encoding:

"Value misencoded: premature end of input encoding encountered."

"Value misencoded, length, nnn (0xff), longer than remaining data length, mmm (0xgg); continuing as if length were %ld:"

"Value misencoded: expected EOC octets not found for indefinite length encoding."

"Value misencoded, EOC with non-zero second octet encountered."

"Value misencoded, length of a tag-length pair, nnn (0xff), longer than remaining data length, mmm (0xgg):"

“Value misencoded, a tag of '00' encountered.”

D. OSS Nokalva version information utility

Note: `osswhat` is only available on common platforms like Linux, Windows, and Solaris, and may not be available for your embedded system port. If you are interested in `osswhat` for your platform, contact OSS Nokalva Sales at info@oss.com.

```
NAME
   osswhat - extracts SCCS version information from a specified file.  This
             utility helps users examine the internal version information
             of the OSS Nokalva executables and runtime libraries.  Users
             can also use this utility to determine the version of the OSS
             libraries which were used to build their own statically
             application executable.
             Such information is often helpful to the OSS Nokalva technical
             support team and to users with multiple versions of the OSS
             ASN.1 Tools residing on their system.

SYNOPSIS
   osswhat [-spattern] filename ...

where "filename" is the name of the user's application executable or the
name of an OSS Nokalva library file.

OPTIONS
   -spattern   :  Stop after the first occurrence of the pattern

EXAMPLE
   osswhat asn1.exe
```

E. Glossary

abstract character :

The set of information, usually including one or more graphic symbols, character names, or related function definitions, that are associated with a cell in a table used to define a character repertoire.
(see X.680, 3.8.1)

abstract syntax value :

Synonymous with "abstract value". A value of an abstract syntax (defined as the set of values of a single ASN.1 type), which is to be encoded by BER, CER, DER, PER, XER, OER, or C-OER, or which is to be generated by BER, CER, DER, PER, XER, OER, or C-OER decoding.
(see X.691, 3.8.2)

abstract value :

A value defined only on the basis of its type, and not dependent on its representation by any encoding rule. Note that when the term "abstract value" is used, it commonly indicates that what is being described varies by the encoding rule being used.
(see X.680, 3.8.2)

absolute reference

A single or multiple part dot-notation used for uniquely specifying one or more components in an ASN.1 syntax. Refer to the definition box in section 4.4.1 for more information about this term.

Abstract syntax notation one (ASN.1)

A formal language for abstractly defining messages to be exchanged between distributed computer systems.

ANSI

American National Standards Institute
(see X.680, 4)

ASN.1:1990

The version of ASN.1 standardized by CCITT in 1988 and by ISO/IEC in 1990. It consists of the documents [X.208] and [X.209].

ASN.1:1994

The version of ASN.1 standardized by ITU-T and ISO/IEC in 1994 and 1995, as well as all associated corrigenda. It consists of the documents [X.680], [X.681], [X.682], [X.683], [X.690] and [X.691].

ASN.1:2008

The version of ASN.1 standardized by ITU-T and ISO/IEC in 2008. It consists of the documents [X.680], [X.681], [X.682], [X.683], [X.690] and [X.691].

ASN.1 C++ API :

A C++API that supports access to abstract data in the form of Abstract Syntax Notation One (ASN.1) types and values.

ASN.1 compiler (OSS)

A program that translates one or more ASN.1 modules into a C header file and either a time-optimized encoder/decoder or a control table to be used by the space-optimized encoder/decoder.

ASN.1 module

A grouping of ASN.1 data types, values, information objects, etc. that is used to control their scope. Modules can be complete and self-contained; however, they often refer to data types, values, information objects, etc. defined in other modules. This is achieved by the `IMPORTS` and `EXPORTS` statements.

ASN.1 specification

One or more modules that completely describe the application messages of a given protocol exchanged between communicating peers.

ASN.1 type

A built-in or user-defined type of the ASN.1 language. For example, INTEGER, BOOLEAN and IA5String are all ASN.1 types.

ASN.1 value

The value of an ASN.1 type. For example, 33, TRUE and “Hello” correspond to the types INTEGER, BOOLEAN and IA5String.

API :

Application Program Interface
(see X.680, 4)

associated type :

An ASN.1 type used only for defining the value and subtype notation of some other type. Associated types are defined in standards as necessary to make it clear that there may be significant differences between the type ASN.1 defines and the way it is encoded. Associated types are not part of user specifications.
(see X.680, 3.8.5)

Basic Encoding Rules (BER)

A set of rules for encoding (and decoding) which uses a tag and length to identify every value, and which allows a one-to-many mapping between the value of an ASN.1 type and its encoded representation. For example, a BOOLEAN value of TRUE is encoded by BER as any non-zero value contained in one octet.

BIT STRING type:

A simple type whose distinguished values are an ordered sequence of zero or more bits.
(see X.680, 3.8.6)

Blue API :

(see ASN.1 C++ API)

BOOLEAN type:

A simple type that has two distinguished values: true and false.
(see X.680, 3.8.7)

BMP:

Basic Multilingual Plane. The ASN.1 type BMPString gets its name from this abbreviation.
(see ISO/IEC 10646-1)

Canonical Encoding Rules (CER)

A set of rules for encoding (and decoding) that is a subset of the Basic Encoding Rules (BER) that eliminates some of the extra flexibility provided by BER.

code file

A compiler-generated C source file containing mostly executable statements, and which implements the time-optimized encoder/decoder.

CHOICE type:

A type that is defined by referencing a list of distinct types, so that each value of the choice type is derived from the value of one of the component types.
(see X.680, 3.8.13)

compiler

A reference to the OSS ASN.1 compiler (not the C compiler).

compiler-generated

Something generated by the OSS ASN.1 compiler.

compiler-only shipment

A packaging of the OSS ASN.1 Tools that includes only the OSS ASN.1 compiler; the run-time libraries are missing.

component:

One of the fields that is included in the definition of the CHOICE, SET, SEQUENCE, SET-OF, or SEQUENCE-OF types.
(see X.680, 3.8.14)

component relation constraint:

A constraint on a component of a SET or SEQUENCE that establishes a relationship between the value or type of that component (the referencing component) and the value or type of one or more other components of the same SET or SEQUENCE (referenced components).
(see X.682, 3.4.1)

composite listing

A listing of the input ASN.1 specification produced by the OSS ASN.1 compiler when the `-listingfile` compiler option is specified. It is useful when reviewing an ASN.1 specification that consists of multiple modules since it presents the abstract syntax in its entirety as a single module with no external references and with macros and parameterized types expanded to what they return.

composite type:

A SET, SEQUENCE, SET-OF, SEQUENCE-OF, CHOICE, EMBEDDED-PDV, EXTERNAL or unrestricted character-string type.
(see X.691, 3.8.5)

concrete syntax

Those aspects of the rules used in the formal specification of data which embody a specific representation of that data in some system environment. Within a real open system, data defined in terms of an abstract syntax will be represented within the local system environment by a local concrete syntax. In communication between real open systems, there are three concrete syntax versions of the data: the concrete syntax used by the sender, the concrete syntax used by the receiver, and the concrete syntax used

between the sender and receiver (transfer syntax). In practice, all three of these local concrete syntax versions may be identical, or transformations between these syntax versions may be required.
(see X.200, 7.2.1.1)

constrained type
The innermost "Type" which contains the referencing component and all of the referenced components of a component relation constraint.
(see X.682, 3.4.2)

constraining set
The information object set referenced in a component relation constraint.
(see X.682, 3.4.3)

constraint
A notation used in association with some type, to define a subtype of that type.
(see X.680, 3.8.15)

control file
A compiler-generated C source file containing mostly initialized C variables, and which are used by the space-optimized encoder/decoder to determine how to encode/decode values.

control table
See Control file.

cross-compiling
The compilation of ASN.1 specifications on a particular computer platform to obtain `.c` and `.h` files for use on a different platform (e.g., compiling ASN.1 specifications on Windows to generate an encoder/decoder that will be utilized by an application running on a SUN workstation).

decoded value:
A value that has been decoded and is currently in an OSS internal format. Contrast with **encoded value**.

decoder:
A compiler-generated or OSS-supplied function used in your application to convert encoded messages into data mapped onto C structures generated by the compiler.

development shipment
See **full shipment**.

Distinguished Encoding Rules (DER)
A set of rules for encoding (and decoding) which uses a tag and length to identify every value, and which allows a one-to-one mapping between the value of an ASN.1 type and its encoded representation. For example, a BOOLEAN value of TRUE is encoded by DER as one octet with all eight bits set to one. The DER is a subset of the BER; it is used in secure protocols in detecting the tampering of messages.

EMBEDDED PDV type
A type whose set of values is the union of the sets of values in all possible abstract syntaxes. Such types are part of an ASN.1 specification, though they may carry values whose types may be defined externally to that ASN.1 specification.
(see X.680, 3.8.21)

encoder

A compiler-generated or OSS-supplied function used in your application to convert data mapped by C structures generated by the compiler into an encoded form.

encoding (encoded value)

The bit-pattern resulting from the application of a set of encoding rules to a value of a specific abstract syntax.
(see X.680, 3.8.22)

encoding rules

Sets of rules used to transform data described by the ASN.1 language into a standard format that can be decoded by any system that has a decoder based on the same set of rules.
(see X.680, 3.8.23)

external object

An ASN.1 type whose underlying local representation is defined "external" to the encoder/decoder. The representation of such types are known only to your application and the memory manager in use by the encoder/decoder. Such types either have the OSS.OBJHANDLE directive specified in the ASN.1 syntax or are marked as "external" at run-time by using one of the IAAPI functions. Only types that can take on the UNBOUNDED representantion may be marked as external objects. Note: only the following ASN.1 types can be made external objects: character strings, BIT STRINGs, OCTET STRINGs, ANY and open types.

EXTERNAL type

A type which is a part of an ASN.1 specification that carries a value whose type may be defined externally to that ASN.1 specification.
(see X.680, 3.8.25)

full shipment

A packaging of the OSS ASN.1 Tools that includes both the OSS ASN.1 compiler and run-time libraries.

GUI

Graphical User Interface

IEC

International Electrotechnical Commission
(see X.680, 4)

information object

An instance of some information object class, being composed of a set of fields which conform to the field specifications of the class.
(see X.681, 3.4.7)

information object class

A set of fields, forming a template for the definition of a potentially unbounded collection of information objects, the instances of the class.
(see X.681, 3.4.8)

INSTANCE OF type

A type, defined by referencing an information object class which associates object identifiers with types.
(see X.681, 3.4.12)

INTEGER type

A simple type with distinguished values which are the positive and negative whole numbers, including zero (as a single value).
(see X.680, 3.8.29)

ISO

International Organization for Standardization
(see X.680, 4)

ITU-T

Telecommunication Standardization Sector of the International Telecommunication Union
(see X.680, 4)

Lean encoder/decoder (LED)

A table-driven (interpretive) encoder/decoder that aims to minimize the use of memory and CPU cycles. This encoder/decoder is especially attractive for those applications that run in memory-constrained environments (e.g., embedded systems) with time critical data.

length determinant

A count (of bits, octets, characters, or components) determining the length of part or all of a PER encoding.
(see X.691, 3.8.17)

macro reference

A name associated uniquely with a ASN.1 macro within some context.

mangling (names)

The method of changing/deleting certain letters in an ASN.1 identifier to avoid name conflicts in the produced header and `.c` files.

marked object

A value of an external object marked with one of the following types: OSS_FILE, OSS_SOCKET, OSS_UNKNOWN_OBJECT. It is used to hold a filename or a TCP/IP socket id instead of the actual value.

MHS

Message Handling Systems
(see X.400)

module

One or more instances of the use of the ASN.1 notation for type, value, etc., encapsulated using the ASN.1 module notation.
(see X.680, 3.8.31)

module listing

A listing of the input ASN.1 modules produced by the OSS ASN.1 compiler when the `-modlistingfile` compiler option is specified. Each module is reproduced with macros expanded but with parameterized items not expanded. It is used as a diagnostic option to ensure that the compiler's view of the input modules is correct.

module reference

A name associated uniquely with a module within some context.

named item
A component of a SET, SEQUENCE or CHOICE type or a bit in a BIT STRING with a NamedBitList, an enumerator in an ENUMERATED type, or a number in an INTEGER type with a NamedNumberList.

NULL type
A simple type consisting of a single value (i.e. NULL).
(see X.680, 3.8.32)

object class reference
A name associated uniquely with an information object class within some context.

object set reference
A name associated uniquely with an information object set within some context.

OBJECT IDENTIFIER type
A simple type whose distinguished values are the set of all object identifiers allocated in accordance with the rules of the X.680 standard.
(see X.680, 3.8.36)

Octet Encoding Rules (OER)
The fastest set of rules for encoding (and decoding) ASN.1 values. The Octet Encoding Rules (OER), like the Packed Encoding Rules (PER), produce compact encodings by taking advantage of information present in the ASN.1 schema to limit the amount of information included in each encoded message.

OCTET STRING type
A simple type whose distinguished values are an ordered sequence of zero or more octets, each octet being an ordered sequence of eight bits.
(see X.680, 3.8.37)

open type notation
An ASN.1 notation used to denote a set of values from more than one ASN.1 type.
(see X.6.80, 3.8.38)

OS
Operating System or Operations System

OSI
Open Systems Interconnection

Packed Encoding Rules (PER)
A set of rules for encoding (and decoding) ASN.1 values which aims to minimize the size of the encodings. Mostly, PER allows a one-to-one mapping between the value of an ASN.1 type and its encoded representation.

parameterized type
A type defined using a parameterized type assignment and thus whose components are incomplete definitions which must be supplied with actual parameters when the type is used.
(see X.683, 3.4.3)

parent type (of a subtype)

A type that is being constrained when defining a subtype of a parent type, which may itself be a subtype of some other type.
(see X.680, 3.8.39)

protocol data unit (PDU)

A message for transfer that can be encoded/decoded as one unit. PDUs can be represented as simple variables or complex sructures. Also refer to the definition box shown under the `-root` compiler option.

REAL type

A simple type whose distinguished values are members of the set of real numbers.
(see X.680, 3.8.41)

restricted character string type

A character string type whose values are zero or more characters taken from a fixed collection of characters. The collection is a repertoire identified in a type specification. The restricted character string types are BMPString, GeneralString, GraphicString, IA5String, ISO646String, NumericString, PrintableString, TeletexString, T61String, UniversalString, UTF8String, VideotexString, and VisibleString.
(see X.680, 3.8.43)

run-time-only copy

See **target shipment**.

space-optimized encoder/decoder (SOED)

A table-driven (interpretive) encoder/decoder that aims to minimize the use of memory when the ASN.1 specification is large or complex.

selection types

Types defined by reference to a component type of a choice type, and whose values are precisely the values of that component type.
(see X.680, 3.8.44)

SEQUENCE types

Types defined by referencing an ordered list of types (some of which may be declared to be optional); each value of the sequence type is an ordered list of values, one from each component type.
(see X.680, 3.8.45)

SEQUENCE-OF types

Types defined by referencing a single component type; each value in the sequence-of type is an ordered list of zero or more values of the component type.
(see X.680, 3.8.46)

SET types

Types defined by referencing a fixed, unordered, list of distinct types (some of which may be declared to be optional); each value in the set type is an unordered list of values, one from each of the component types.
(see X.680, 3.8.47)

SET-OF types
Types defined by referencing a single component type; each value in the set-of type is an unordered list of zero, one or more values of the component type.
(see X.680, 3.8.48)

simple type
Any type that is not a composite type.
(see X.691, 3.8.25)

subtype (of a parent type)
A type whose values are a subset (or the complete set) of the values of some parent type.
(see X.680, 3.8.51)

structured type
A SET, SEQUENCE, SET OF, SEQUENCE OF, or CHOICE ASN.1 type. These types are represented using a `struct` or `union` of components/alternatives in C.

table constraint
A constraint applied to an object class field type which demands that its values conform to the contents of the appropriate column of some table.
(see X.682, 3.4.8)

tag
A type denotation which is associated with every ASN.1 type.
(see X.680, 3.8.52)

tagged types
A type defined by referencing a single existing type and a tag; the new type is isomorphic to the existing type, but is distinct from it.
(see X.680, 3.8.53)

tagging
Replacing the existing (possibly the default) tag of a type by a specified tag.
(see X.680, 3.8.54)

target platform
The computer platform on which your application that uses the encoder/decoder will execute.

target shipment
A packaging of the OSS ASN.1 Tools that includes only the run-time libraries; the OSS ASN.1 compiler is missing.

time-optimized encoder/decoder (TOED)
An encoder/decoder that aims to minimize the use of CPU; it is most useful when the ASN.1 specification is not very large or complex and there is no need for detailed diagnostics.

transfer syntax
The abstract syntax and concrete syntax used in the transfer of data between open systems. Note that the term "transfer syntax" is sometimes used to mean encoding rules, and sometimes used to mean the representation of bits in data while in transit.
(see X.200:1994 7.2.1.1)

type

A named set of values.
(see X.680, 3.8.56)

type reference

A name associated uniquely with a type within some context.
(see X.680, 3.8.57)

typesharing

The ability of the ASN.1 compiler to generate one structure in the header file which is used to define multiple types by taking advantage of shared similarities. By default, typesharing is enabled. To turn it off, specify the `-compat noSharedTypes` option.

Unicode

The values expressed as two-octet codes by the ASN.1 BMPString type. Unicode, maintained by the Unicode Consortium, is a system for the coding of 16-bit characters. Most of the worlds written languages can be interchanged, processed and displayed using BMPString.

user-defined constraint

A constraint which requires a more complicated statement than can be accommodated by the other forms of constraint, and which must therefore involve specification by some means outside of ASN.1.
(see X.682, 3.4.9)

UTC (Coordinated Universal Time)

The time scale maintained by the Bureau Internationale de l'Heure (International Time Bureau) that forms the basis of a coordinated dissemination of standard frequencies and time signals.
(see X.680, 3.8.17)

value reference

A name associated uniquely with a value within some context.
(see X.680, 3.8.61)

white-space

Any formatting action that yields a space on a printed page, such as the SPACE or TAB character, or multiple uses of such characters.
(see X.680, 3.8.63)

XML Encoding Rules (XER)

A set of encoding rules which are intended for use in circumstances where displaying values and/or processing them using commonly available XML tools (e.g., internet web browsers) is the major concern. These rules allow users to efficiently map (back-and-forth) verbose XML text data to/from compact binary ASN.1 representations and to seamlessly integrate their XML and ASN.1 applications.

X.200

ITU-T Recommendation X.200 | ISO/IEC 7498-1, Information Technology - Open Systems Interconnection- Basic Reference Model: The basic model

X.207

ITU-T Recommendation X.207, Open Systems Interconnection- Application Layer Structure

X.208

CCITT Recommendation X.208, Open Systems Interconnection- Specification of Abstract Syntax Notation One (ASN.1) (no longer the ASN.1 standard)

X.209

CCITT Recommendation X.209, Open Systems Interconnection- Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) (no longer the BER standard)

X.215

ITU-T Recommendation X.215, Open Systems Interconnection- Session Service Definition

X.216

ITU-T Recommendation X.216, Open Systems Interconnection- Presentation Service Definition

X.217

ITU-T Recommendation X.217, Open Systems Interconnection- Service Definition for the Association Control Service Element

X.226

ITU-T Recommendation X.226, Open Systems Interconnection- Connection-oriented presentation protocol: Protocol specification

X.680

ITU-T Recommendation X.680 | ISO/IEC 8824-1, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation

X.681

ITU-T Recommendation X.681 | ISO/IEC 8824-2, Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification

X.682

ITU-T Recommendation X.682 | ISO/IEC 8824-3, Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification

X.683

ITU-T Recommendation X.683 | ISO/IEC 8824-4, Information Technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications

X.690

ITU-T Recommendation X.690 | ISO/IEC 8825-1, Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

X.691

ITU-T Recommendation X.691 (1994) | ISO/IEC 8825-2:1995, Information Technology - ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER)

X.693

ITU-T Recommendation X.693 | ISO/IEC 8825-4: 2001
Information Technology - ASN.1 Encoding Rules: Specification of XML Encoding Rules (XER)

Appendices

X.694

ITU-T Recommendation X.694 | ISO/IEC 8825-4: 2001
Information Technology - ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1

X.700

ITU-T Recommendation X.700 | ISO/IEC 7498-4 Information Technology - Management Framework for OSI.

X.701

ITU-T Recommendation X.701 | ISO/IEC 7498-4 Information Technology - Systems Management Overview (SMO)

F. Frequently asked questions

If you cannot find the answer to your question here, also try to refer to the *Frequently asked questions* chapter in the **OSS ASN.1 API Reference Manual**.

When I ASN.1-compile my syntax, I get warning messages about duplicate PDU tags. How do I get rid of these warning messages?

You can try specifying the `-noUniquePDU` command-line option when ASN.1-compiling your syntax. Additionally, note that these messages are only issued when BER/DER is in use. If your ASN.1 specification will be transmitted using PER, you should specify the `-per` option.

Do I have to manually add tags to CHOICE and SET components which have the same ASN.1 type? Or is there a way to automatically add the needed tags?

You can add the AUTOMATIC TAGS keywords to your module definition statement to instruct the ASN.1 compiler to automatically add the needed tags for component differentiation.

The following excerpt illustrates the use of the AUTOMATIC TAGS keywords:

```
ModuleName DEFINITIONS AUTOMATIC TAGS ::= BEGIN
   ChoiceA ::= CHOICE {
      a INTEGER,
      b INTEGER,
      c INTEGER
   }
END
```

Note how the elements of the CHOICE above do not have any context-sensitive tags applied to them. The OSS ASN.1 compiler provides all necessary tags when the AUTOMATIC TAGS keywords are present.

Is the order of the input files important? Which input files should I place first? Which input files should I place last?

The order of the input files on the command-line may be important. Files containing global compiler directives (e.g., *asn1dflt* files) should be placed before all ASN.1 input files. Additionally, files that contain definitions referenced by other files should be placed first (left to right) on the command line. For example, if the file `myModules.asn` references macro definitions in `macro.asn`, you would invoke the compiler as:

```
prompt%> asn1 macro.asn myModules.asn
```

Also note that the last ASN.1 file specified on the command-line has a special role. The name of this file determines the default name-prefix of the output file. Additionally, if neither the `-root` command-line option (see section 3.3.61) nor the global OSS.ROOT directive (see section 4.4.2.39) is specified, then the last (or only) module in this right-most ASN.1 input file is taken as the central ASN.1 module in the specification. In such a case, only this last module has PDUs generated for it into the header file; definitions from other files are only included if they are referenced from within this last module.

How do I make the ASN.1 compiler generate C representations for all the types in all the input modules? Currently, only the types in the last input module are being generated into the produced header file.

If you wish for the ASN.1 compiler to generate C representations for all the types in all the input modules, you should either specify the `-root` command-line option (see section 3.3.61) when ASN.1-compiling or declare the global OSS.ROOT directive (see section 4.4.2.39) in your specification.

Is the `-noUnique` option synonymous with `-noUniquePDU`?

Yes. Any command line keyword can be abbreviated as much as you like, so long as the resulting abbreviation is unique. If the resulting abbreviation is not unique then the ASN.1 compiler will list all the keywords that the abbreviation matches so that you can choose a unique abbreviation.

Can I control the name that gets generated for variables or constants in the produced header file?

Yes. See the ASN1.Nickname directive in section 4.4.1.4 for more details.

Is there a way to only have a header file generated without a C file?

Yes. You can specify the `-noControlFile` option (see section 3.3.15) when ASN.1-compiling to do this.

G. Getting help from OSS

Mailing address:
OSS Nokalva, Inc.
300 Atrium Drive, Suite 402
Somerset, N.J. 08873
U.S.A.

Phone numbers:
Toll free: 1-888-OSS-ASN1 (USA and Canada)
International: 1-732-302-0750
Technical Support : 1-732-302-9669
Fax: 1-732-302-0023

Electronic mail / Internet:
Tech support email: support@oss.com
General information: info@oss.com
ASN.1 discussion list: asn1@oss.com
ASN.1 discussion list subscription: asn1-request@oss.com
URL: http://www.oss.com

If you have any problem with the **OSS® ASN.1 Tools**, you can phone OSS Nokalva, Inc., at the toll free or technical support numbers given above. You can also send the following to support@oss.com:

A description of the problem.

Your product **license number**. You can find this number in the comment at the top of the ASN.1-compiler-generated header and `.c` files.

The **version** and **platform** of the OSS ASN.1 Tools that you are using (you can find this information on the first three lines of the `README.TXT` file included in your shipment).

If the problem is compiler or run-time related, the following:

- All your `.asn` files, either `tar`red and compressed or zipped.
- The ASN.1 compiler command-line options that you used in compiling.
- Any other pertinent files, such as snippets of application code or unexpected output.

If the problem is decoder related, the following:

- A copy of the message that you are having problems decoding.
- The PDU number (e.g., `Bcas_PDU`) that you passed to the decoder.

If the problem is encoder related, a small program that illustrates the problem you are having.