Compiler Configuration File

In addition to command line options, configuration commands can be used to apply specific compiler options to specific items within a schema. These options can be applied to specific modules, productions, and elements as well as globally.

Configuration items may be specified in one or two ways (or a combination of both):

- Using special comments embedded directly in an ASN.1 file, or
- Using an external XML configuration file

A simple form of XML is used as the format for the configuration items. XML was chosen because it is fairly well known and provides a natural interface for representing hierarchical data such as the structure of ASN.1 modules and productions.

In the case of embeddeding directives directly in the ASN.1 source file, the directive is included as a comment directly before the item to which it is to be applied. For example:

```
BackupBearerCapOctetGroup5 ::= SEQUENCE {
   octet5 BearerCapOctet5,
   --<end3GExtElem name="octet5.ext"/>
   octet5a BearerCapOctet5a
}
```

In this case, the `end3GExtElem` configuration item would be applied to the `octet5a` element.

An external configuration file would target the item to which the directive is to be applied by specifying module, production, and element in an XML hierarchy. An example of this is as follows:

```
<asn1config>
  <module name=”TS24008IES”>
    <production name=”BackupBearerCapOctetGroup5”>
      <element name=”octet5a”>
        <end3GExtElem name="octet5.ext"/>
      </element>
    </production>
  </module>
</asn1config>
```

At the outer level of the markup is the `<asn1config>`

`</asn1config>` tag pair. Within this tag pair, the specification of global items and modules can be made. Global items are applied to all items in all modules. An example would be the `<storage>` qualifier. A storage class such as dynamic can be specified and applied to all productions in all modules. This will cause dynamic storage (pointers) to be used for any embedded structures within all of the generated code to reduce memory consumption demands.

The specification of a module is done using the `<module></module>` tag pair. This tag pair can only be nested within the top-level `<asn1config>` section. The module is identified by using the required `<name></name>` tag pair or by specifying the name as an attribute (for example, `<module name="MyModule">`). Other attributes specified within the `<module>` section apply only to that module and not to other modules specified within the specification. A complete list of all module attributes is provided in the table at the end of this section.

The specification of an individual production is done using the `<production></production>` tag pair. This tag pair can only be nested within a `<module>` section. The production is identified by using the required <name></name> tag pair or by specifying the name as an attribute (for example, `<production name="MyProd">`). Other attributes within the production section apply only to the referenced production and nothing else. A complete list of attributes that can be applied to individual productions is provided in the table at the end of this section.

When an attribute is specified in more than one section, the most specific application is always used. For example, assume a `<typePrefix>` qualifier is used within a module specification to specify a prefix for all generated types in the module and another one is used to a specify a prefix for a single production. The production with the type prefix will be generated with the type prefix assigned to it and all other generated types will contain the type prefix assigned at the module level.

Values in the different sections can be specified in one of the following ways:

1. Using the `<name>value</name>` form. This assigns the given value to the given name. For example, the following would be used to specify the name of the "H323-MESSAGES" module in a module section:

   ```
   <name>H323-MESSAGES</name>
   ```

2. Flag variables that turn some attribute on or off would be

specified using a single `<name/>` entry. For example, to specify a given production is a PDU, the following would be specified in a production section:

```
<isPDU/>
```

3. An attribute list can be associated with some items. This is normally used as a shorthand form for specifying lists of names. For example, to specify a list of type names to be included in the generated code for a particular module, the following would be used:

```
<include types="TypeName1,TypeName2,TypeName3"/>
```

The following are some examples of configuration specifications:

```
<asn1config><storage>dynamic</storage></asn1config>
```

This specification indicates dynamic storage should be used in all places where its use would result in significant memory usage savings within all modules in the specified source file.

```
<asn1config>
   <module>
      <name>H323-MESSAGES</name>
      <sourceFile>h225.asn</sourceFile>
      <typePrefix>H225</typePrefix>
   </module>
   ...
</asn1config>
```

This specification applies to module ‘H323-MESSAGES’ in the source file being processed. For IMPORT statements involving this module, it indicates that the source file ‘h225.asn’ should be searched for specifications. It also indicates that when C or C++ types are generated, they should be prefixed with ‘H225’. This can help prevent name clashes if one or more modules are involved and they contain productions with common names.

The following tables specify the list of attributes that can be applied at all of the different levels: global, module, and individual production:

Global Level

These attributes can be applied at the global level by including them within the `<asn1config>` section:

Name	Values	Description
		This configuration item is for use with Event Handling as described in a later

<events> </events>	*defaultValue* keyword.	section in this document. It is used to include a special event that is fired when a PER message is being parsed. This event occurs at the location a value should be present in the message but is not and a default value has been specified in the ASN.1 file for the element. In this case, the normal event sequence (startElement, contents, endElement) is executed using the default value.
<includedir> </includedir>	<Include directory>	This configuration item is used to specify a directory that will be search for IMPORT files. It is equivalent to the -I command-line option.
<protocol> </protocol>	<Protocol identifier>	Specifies a protocol identifier to be associated with the ASN.1 specification set. For C/C++, this specifies a prefix that will be used with generated encode and decode functions. Currently, this only applies to 3GPP Layer 3 functions.
<rootdir> </rootdir>	<ASN1C root directory>	This configuration item is used to specify the root directory of the ASN1C installation for makefile or Visual Studio project generation. It is only needed if generation of these items is done outside of the ASN1C installation.
		If *dynamic* , it indicates that dynamic storage (i.e., pointers) should be used everywhere within the generated types where use could result in lower memory consumption. These places include the array element for sized SEQUENCE OF/SET OF types and all alternative elements within CHOICE constructs. If *static*, it indicates static types should be used in these places. In general, static types are easier to

<storage> </storage>	One of the following keywords: *dynamic*, *static*, *list*, *array*, *dynamicArray*, *std::list*, *std::vector*, *std::deque*.	work with. If *list*, a linked-list type will be used for SEQUENCE OF/SET OF constructs instead of an array type. If *array*, an array type will be used for SEQUENCE OF/SET OF constructs. The maxSize attribute can be used in this case to specify the size of the array variable (for example, <storage maxSize="12"> array </storage>). If *dynamicArray*, a dynamic array will be used for SEQUENCE OF/SET OF constructs. A dynamic array is an array that uses dynamic storage for the array elements. If *std::array* the result is the same as for *array*, except that std::array will be used instead of a plain C/C++ static array. You must specify *-cpp11* on the command line to use this option. If one of *std::list*, *std::vector*, *std::deque*, then the corresponding C++ Standard Library container class will be used. You must specify *-cpp11* on the command line to use one of these options.

Module Level

These attributes can be applied at the module level by including them within a `<module>` section:

Name	Values	Description
<name> </name>	module name	This attribute identifies the module to which this section applies. Either this or the <oid> element/attribute is required.

<oid>	module OID (object identifier)	This attribute provides for an alternate form of module identification for the case when module name is not unique. For example, a given ASN.1 module may have multiple versions. A unique version of the module can be identified using the OID value.
<codename> </codename>	C/C++, Java, or C# name	This item specifies an alternate name for the module to be used in generated code. By default, the module name is used in the form it appears in the ASN.1 specification with hyphens converted to underscores.
<include types="names" values="names"/>	ASN.1 type or value names are specified as an attribute list	This item allows a list of ASN.1 types and/or values to be included in the generated code. By default, the compiler generates code for all types and values within a specification. This allows the user to reduce the size of the generated code base by selecting only a subset of the types/values in a specification for compilation. Note that if a type or value is included that has dependent types or values (for example, the element types in a SEQUENCE, SET, or CHOICE), all of the dependent types will be automatically included as well.
<include encoders="names"/>	ASN.1 type names specified as an attribute	This item allows a list of ASN.1 types to be included in the generated code for which only encode functions will be

	list.	generated.
<include decoders="names"/>	ASN.1 type names specified as an attribute list.	This item allows a list of ASN.1 types to be included in the generated code for which only decode functions will be generated.
<include memfree="names"/>	ASN.1 type names specified as an attribute list.	This item allows a list of ASN.1 types to be included in the generated code for which only memory free functions will be generated.
<include importsFrom= "name"/>	ASN.1 module name(s) specified as an attribute list.	This form of the include directive tells the compiler to only include types and/or values in the generated code that are imported by the given module(s).
<exclude types="names" values="names"/>	ASN.1 type or values names are specified as an attribute list	This item allows a list of ASN.1 types and/or values to be excluded in the generated code. By default, the compiler generates code for all types and values within a specification. This is generally not as useful as in *include* directive because most types in a specification are referenced by other types. If an attempt is made to exclude a type or value referenced by another item, the directive will be ignored.
<storage> </storage>	One of the following keywords: *dynamic*, *static*, *list*, *array*, *dynamicArray*, *std::list*, *std::vector*,	The definition is the same as for the global case except that the specified storage type will only be applied to generated C and C++ types from the given module.

	std::deque.	
<sourceFile> </sourceFile>	source file name	Indicates the given module is contained within the given ASN.1 source file. This is used on IMPORTs to instruct the compiler where to look for imported definitions.
<prefix> </prefix>	prefix text	This is used to specify a general prefix that will be applied to all generated C and C++ names (note: for C++ types, the prefix is applied after the standard 'ASN1T_' prefix). This can be used to prevent name clashes if multiple modules are involved in a compilation and they all contain common names.
<typePrefix> </typePrefix>	prefix text	This is used to specify a prefix that will be applied to all generated C and C++ typedef names (note: for C++, the prefix is applied after the standard 'ASN1T_' prefix). This can be used to prevent name clashes if multiple modules are involved in a compilation and they all contain common names.
<enumPrefix> </enumPrefix>	prefix text	This is used to specify a prefix that will be applied to all generated enumerated identifiers within a module. This can be used to prevent name clashes if multiple modules are involved in a compilation. (note: this attribute is normally not needed for C++ enumerated identifiers because they are already wrapped in a structure to allows the type

		name to be used as an additional identifier).
<valuePrefix> </valuePrefix>	prefix text	This is used to specify a prefix that will be applied to all generated value constants within a module. This can be used to prevent name clashes if multiple modules are involved that use a common name for two or more different value declarations.
<classPrefix> </classPrefix>	prefix text	This is used to specify a prefix that will be applied to all generated items in a module derived from an ASN.1 CLASS definition.
<objectPrefix> </objectPrefix>	prefix text	This is used to specify a prefix that will be applied to all generated items in a module derived from an ASN.1 Information Object definition.
<objectsetPrefix> </objectsetPrefix>	prefix text	This is used to specify a prefix that will be applied to all generated items in a module derived from an ASN.1 Information Object Set definition.
<noPDU/>	n/a	Indicates that this module contains no PDU definitions. This is normally true in modules that are imported to get common type definitions (for example, InformationFramework). This will prevent the C++ version of the compiler from generating any control class definitions for the types in the module.
	byte, int16,	This is used to specify a specific C integer type be

<intCType>	uint16, int32, uint32, int64, string	used for all unconstrained integer types. By default, ASN1C will use the int32 (32-bit integer) type for all unconstrained integers.
<arcCType>	int32, int64	The is used to specify a specific C integer type be used for the arc types in Object Identifier definitions. By default, int32 (32-bit integer arc values) are generated.
<namespace> </namespace>	namespace URI	This is used to specify the target namespace for the given module when generating XSD and/or XML code. By default, the compiler will not include a targetNamespace directive in the generated XSD code (i.e. all items will not be assigned to any namespace). This option only has meaning when used with the - xml / -xsd command line options.
<hFile> </hFile>	C/C++ header filename	This is used to specify the name of a C/C++ header file to be used to store generated definitions for the module. By default, the header file name is set to the ASN.1 name of the module with '.h' appended to the end.
<alias asn1name="name" codename="name"/>	ASN.1 to computer language name mapping	This item allows a name in the ASN.1 specification being compiled to be mapped to an alternate name in the generated computer language files. The primary use is to allow shorter names to be used in places where a combination of names may be

		very long. In this release, the only names that can be used in the alias statement are information object set names.

Production Level

These attributes can be applied at the production level by including them within a `<production>` section:

Name	Values	Description
<name> </name>	production name	This attribute identifies the production (type) to which this section applies. It is required.
<addarg name="name" type="type" func="encode|decode"/>	Argument name, type, and function specified using attributes.	This item adds an argument to the generated C encode and/or decode function. The name and C type of the argument are specified in the name and type attributes respectively. The func attribute is optional and only required if the argument should be added to either the encode or decode function only. By default, the argument is added to both the encode and decode function. This item is only supported for C 3GPP layer 3 code generation.
<aligned/>	n/a	This item is used to specify that byte alignment is to be done after encoding or decoding an instance of the targeted type. This item is only supported for C 3GPP layer 3 code generation.

<cDecFuncName> </cDecFuncName>	<C source file name	This item is used to substitute the C source code contained within the given file for what would have been generated for the C decode function for the given type. The current include path is searched for the given filename. This item is only supported for C 3GPP layer 3 code generation.
<cEncFuncName> </cEncFuncName>	<C source file name	This item is used to substitute the C source code contained within the given file for what would have been generated for the C encode function for the given type. The current include path is searched for the given filename. This item is only supported for C 3GPP layer 3 code generation.
<ctype>	byte, int16, uint16, int32, uint32, int64, string, chararray	This is used to specify a specific C integer or character string type be used in place of the default definition generated by ASN1C. In the case of integers, ASN1C will normally try and use the smallest integer type available based on the value or value range constraint on the integer type. If the integer is not constrained, the int32 (32-bit integer) type will be used. For character string, ASN1C will use a character string pointer (char*) by default. The 'chararray'

		item can be used on strings with size constrains to specify a static character array variable be used.
<enumPrefix> </enumPrefix>	prefix text	This is used to specify a prefix that will be applied to all generated enumerated identifiers within a module. This can be used to prevent name clashes if multiple modules are involved in a compilation. (note: this attribute is normally not needed for C++ enumerated identifiers because they are already wrapped in a structure to allows the type name to be used as an additional identifier).
<format> </format>	base64, hex, xmllist	This is used to set format options specific to XER encoding. The base64 or hex alternative is used to set the output format that binary data in OCTET STRING variables is displayed in in XML markup. The xmllist alternative is used with SEQUENCE OF or SET OF types to denote that items should be displayed in XML space-separated list format as opposed to a using a separate element for each list item.
		This item specifies that this production will be modelled as a 3G extended list. This can only be

<is3GExtList pre-eol="0|1" post-eol="0|1"/>	n/a	applied to SEQUENCE OF productions. It is used in 3G layer 3 messages when the extension of a repeating type is controlled by an extension bit that occurs either before or after the record. If the pre-eol attribute (short for "preceding end-of-list") is specified, it indicate a bit before the record signals whether another record follows. The value (0 or 1) indicates which bit value signal end-of-list. The post-eol attribute is the same except that it indicates the control bit follows after the record. This item is only supported for C 3GPP layer 3 code generation.
<is3GMessage/>	n/a	This item specifies that this production represents a 3G layer 3 message type as opposed to a 3G layer 3 information element (IE). This item is only supported for C 3GPP layer 3 code generation.
<isBigInteger/>	n/a	This item specifies that this production will be used to store an integer larger than the C or C++ int type on the given system (normally 32 bits). A C string type (char*) will be used to hold a textual representation of the value. This qualifier can be applied to either an integer or constructed

		type. If constructed, all integer elements within the constructed type are flagged as big integers.
<isOpenType/>	n/a	This item is used to indicate that any element of this type will be decoded as an open type (i.e. skipped). Refer to the section on deferred decoding for further information. Note that this variable can only be used with BER, CER, or DER encoding rules.
<isPDU/>	n/a	This item is used to indicate that this production represents a Protocol Data Unit (PDU). This is defined as a production that will be encoded or decoded from within the application code. This attribute only makes a difference in the generation of C++ classes. Control classes that are only used in the application code are only generated for types with this attribute set.
<isTBCDString/>	n/a	This item is used to indicate that this production is to be encoded and decoded as a telephony binary coded string (TBCD). This is type is not part of the ASN.1 standards but is a widely used encoding format in telephony applications.

<length fixed-size="number"/>	n/a	This item is used to configure a length field in an OCTET STRING type for 3GPP layer 3 messages. By default, a length field is a single byte, but there are occasions where the field width may be different. This allows a fixed-size encoded field width to be specified. The most common values are 0 (no length field) or 2.
<noDecoder/>	n/a	Indicates that no decode function should be generated for this production. This item is only supported for C 3GPP layer 3 code generation.
<noEncoder/>	n/a	Indicates that no encode function should be generated for this production. This item is only supported for C 3GPP layer 3 code generation.
<setvar name="name" value="value"/>	n/a	This item is used within encode and decode functions to set a given variable within a generated structure to the given value. Normally it is used in conjunction with the 'addarg' configuration item to set a variable to value of an additional argument passed into a function. This item is only supported for C 3GPP layer 3 code generation.
	One of the following keywords:	The definition is the same

<storage> </storage>	*dynamic*, *static*, *list*, *array*, *dynamicArray*, *std::list*, *std::vector*, *std::deque*.	as for the global case except that the specified storage type will only be applied to the generated C or C++ type for the given production.
<typePrefix> </typePrefix>	prefix text	This is used to specify a prefix that will be applied to all generated C and C++ typedef names (note: for C++, the prefix is applied after the standard 'ASN1T_' prefix). This can be used to prevent name clashes if multiple modules are involved in a compilation and they all contain common names.

Element Level

These attributes can be applied at the element level by including them within an `<element>` section:

Name	Values	Description
<name> </name>	element name	This attribute identifies the element within a SEQUENCE, SET, or CHOICE construct to which this section applies. It is required.
<addarg name="name"	Argument name, function	This item adds an argument to the generated C encode and/or decode function that is invoked to encode or decode the element. The name attribute specified the value to be passed. The func attribute is optional and only required if the

func="encode|decode"/>	specified using attributes.	argument should be added to either the encode or decode function only. By default, the argument is added to both the encode and decode function. This item is only supported for C 3GPP layer 3 code generation.
<aligned/>	n/a	This item is used to specify that byte alignment is to be done after encoding or decoding this element. This item is only supported for C 3GPP layer 3 code generation.
<cDecSrcName> </cDecSrcName>	<C source file name	This item is used to substitute the C source code contained within the given file for what would have been generated for decoding the element. The code in this case is not a complete function but rather a snippet to be inserted within a larger function. The current include path is searched for the given filename. This item is only supported for C 3GPP layer 3 code generation.
<cEncSrcName> </cEncSrcName>	<C source file name	This item is used to substitute the C source code contained within the given file for what would have been generated for encoding the element. The code in this case is not a complete function but rather a snippet to be

		inserted within a larger function. The current include path is searched for the given filename. This item is only supported for C 3GPP layer 3 code generation.
<ctype>	chararray	This is used to specify a specific C type be used in place of the default definition generated by ASN1C. In the case of elements, the only supported customization is for character string types which would normally be represented by a character pointer type (char*) to be changed to use static character arrays. This can only be done if the string type contains a size constraint.
<end3GExtElem name="element name"/>	<Element name> attribute	This item is used to delimit a group of optional elements that start with an 'ext' boolean element. A common pattern in the specification of 3GPP IE's is to include an extension bit to signal the presence or absence of group of elements (these normally comprise a single octet). This is essentially an alternative way to specify an optional element group in ASN.1. This item is only supported for C 3GPP layer 3 code generation.
		This item is used to indicate an element is part

<iei format="t|tv|tlv" length="length"/>	IEI hex value	of the non-imperative part 3GPP layer 3 message. These are optional elements with single byte tags. The tag is the IEI hex value specified at the value of the item. The format attribute specifies if the item is a tag (t), tag/value (tv), or tag/length/value (tlv). The length attribute is only required if format if tv to specify the length of the value. This item is only supported for C 3GPP layer 3 code generation.
<inline/>	n/a	This item is used to indicate that code generated for a nested item within a constructed type should be expanded inline rather than pulled out to create a separate new type.
<is3GExtList pre-eol="0|1" post-eol="0|1"/>	n/a	This item specifies that this element will be modelled as a 3G extended list. This can only be applied to elements of type SEQUENCE OF. It is used in 3G layer 3 messages when the extension of a repeating type is controlled by an extension bit that occurs either before or after the record. If the pre-eol attribute (short for "preceding end-of-list") is specified, it indicate a bit before the record signals whether another record

		follows. The value (0 or 1) indicates which bit value signal end-of-list. The post-eol attribute is the same except that it indicates the control bit follows after the record. This item is only supported for C 3GPP layer 3 code generation.
<is3GLength bitFieldSize="nbits" units="bits|bytes"/>	n/a	This item is used to mark an element as a 3GPP length field element. Normally this an element with the name 'length' of type INTEGER that is the first element in a SEQUENCE. This indicates special processing should be done on the element. On encode, any value populated in this field will be ignored and the actual length of the encoded data will be calculated and populated in this field after encoding is complete. On decode, this element is used to determine when end of message occurs. The 'bitFieldLength' attribute is used to specify the field size if it not an even octet (8 bits). The 'units' attribute specifies the units stored in the length field (bits or bytes). This item is only supported for C 3GPP layer 3 code generation.
		This item specifies that this element will be

<is3GVarLenList lengthElem="name"/>	n/a	modelled as a 3G variable length list. This can only be applied to elements of type SEQUENCE OF. It is used in 3G layer 3 messages when a length element is used to determine the number of items in the list. The 'lengthElem' attribute specifies the element within the structure that contains this count. This item is only supported for C 3GPP layer 3 code generation.
<isBigInteger/>	n/a	This item specifies that this element will be used to store an integer larger than the C or C++ int type on the given system (normally 32 bits). A C string type (char*) will be used to hold a textual representation of the value. This qualifier can be applied to either an integer or constructed type. If constructed, all integer elements within the constructed type are flagged as big integers.
<isOpenType/>	n/a	This flag variable specifies that this element will be decoded as an open type (i.e. skipped). Refer to the section on deferred decoding for further information. Note that this variable can only be used with BER, CER, or DER encoding rules.

<length fixed-size="number"/>	n/a	This item is used to configure a length field in an OCTET STRING type for 3GPP layer 3 messages. By default, a length field is a single byte, but there are occasions where the field width may be different. This allows a fixed-size encoded field width to be specified. The most common values are 0 (no length field) or 2. This item is only supported for C 3GPP layer 3 code generation.
<notUsed/>	n/a	This flag variable specifies that this element will not be used at all in the generated code. It can only be applied to optional elements within a SEQUENCE or SET, or to elements within a CHOICE. Its purpose is for production of more compact code by allowing users to configure out items that are of no interest to them.
<perEncoding></perEncoding>	hex data	This variable allows a user to substitute a known binary PER encoding for the given element. This encoding will be inserted into the encoded data stream on encoding and skipped over on decoding. Its purpose is the production of more compact and faster code for PER by bypassing run-

		time calculations needed to encode or decode variable data.
<selector element="name" value="value"/>	n/a	This item is used to configure an element within a CHOICE in a 3GPP layer 3 message. It specifies the value of another element within the container type which selects this element. The 'element' field specifies the name of the element within the container type and 'value' specifies the value. are 0 (no length field) or 2. This item is only supported for C 3GPP layer 3 code generation.
<setvar name="name" value="value"/>	n/a	This item is used within encode and decode functions to set a given variable within a generated structure to the given value. Normally it is used in conjunction with the 'addarg' configuration item to set a variable to value of an additional argument passed into a function. This item is only supported for C 3GPP layer 3 code generation.
<storage> </storage>	One of the following keywords: *dynamic*, *static*, *list*, *array*, *dynamicArray*, *std::list*, *std::vector*,	The definition is the same as for the global case except that the specified storage type will only be applied to the generated C or C++ type for this element.

	std::deque.	

[Prev](#) [Up](#) [Next](#)
Porting Run-time Code to Other Platforms [Home](#) Compiler Error Reporting