ASN2XML

ASN.1 to XML Translator

Version 2.1

Reference Manual

Objective Systems – July 2010

Author's Contact Information:

Comments, suggestions, and inquiries regarding this document may be submitted via electronic mail to info@obj-sys.com.

Table of Contents

Overview of ASN2XML....4
Using ASN2XML....5
Using the ASN2XML DLL....8
ASN.1 to XML Value Mappings....11
General Mapping without ASN.1 Schema Information....11
General Mapping with ASN.1 Schema Information....12
Specific ASN.1 Type to Value Mappings....12
BOOLEAN....12
INTEGER....13
ENUMERATED....13
BIT STRING....13
OCTET STRING....14
NULL....14
OBJECT IDENTIFIER....15
RELATIVE OID....15
Character String....15
Binary-Coded Decimal (BCD) String....15
REAL....16
SEQUENCE....16
SET....17
SEQUENCE OF / SET OF....17
CHOICE....18
Open Type....19

Overview of ASN2XML

ASN2XML translates an ASN.1 BER/DER or PER encoded binary data file to XML format. BER refers to the ASN.1 Basic Encoding Rules as published in the ITU X.690 standard. DER refers to the Distinguished Encoding Rules as published in the same standard. PER refers to the ASN.1 Packed Encoding Rules as documented in the ITU-T X.691 standard. Both aligned and unaligned basic PER modes are supported.

This tool can decode BER or DER messages with or without an associated ASN.1 definition as follows:

- If ASN.1 definitions are not used, an XML message will be created with tag numbers and decoded values.
- If an ASN.1 source definition is used (via the –schema command-line option), an XML message will be created using element names and other meta-data from the ASN.1 definitions.

It is required when translating PER messages that a schema file be provided. The protocol data unit (PDU) type of the message must also be identified using the -pdu command-line argument.

The translation tool in this package is available in two forms:

- As a command-line executable program (asn2xml.exe), and
- As a dynamic link library (DLL) or shared object that can be embedded within a user application program.

The first section (*Using ASN2XML*) describes the executable program and the following section (*Using the ASN2XML DLL*) describes using the DLL.

Using ASN2XML

To test if the tool was successfully installed, enter asn2xml with no parameters as follows (note: if you have not updated your PATH variable, you will need to enter the full pathname of the asn2xml executable):

```
asn2xml
```

You should observe the following display (or something similar):

```
ASN2XML, Version 2.1.B
ASN.1 to XML translation tool
Copyright (c) 2004-2010 Objective Systems, Inc. All Rights Reserved.

Usage: asn2xml <filename> options

    <filename>             ASN.1 message file name

  options:
    -schema <filename>     ASN.1 definition file name(s)
    -ber                   Use basic encoding rules (BER)
    -per                   Use aligned packed encoding rules (PER)
    -uper                  Use unaligned packed encoding rules (U-PER)
    -pdu <typename>        Message PDU type name
    -o <filename>          Output XML filename
    -ascii                 Print out ASCII for printable hex values
    -emptyoptionals        Insert empty XML elements in place of
                             missing optional elements
    -emptydefault          Insert XML elements with default values in place
                             of missing elements with default values
```

```
-bcdhandling <default|none|bcd|tbcd>
                        Define handling of OCTET STRINGS declared to be
                          binary coded decimal (BCD) strings
-noopentype             Disable automatic open type decoding
-nowhitespace           Remove all whitespace between elements
-paddingbyte <hexbyte> Additional padding byte
-rootElement <element> Root Element Name
-bitsfmt <hex|bin>      BIT STRING content output format
-inputFileType <binary|hextext|base64>
                        Format of data in input file
```

This indicates that to use the translator, at a minimum, the name of a message file containing ASN.1 encoded data must be provided. The filename can be a full pathname or only whatever is necessary to qualify the file. If directory information is not provided, the user's current default directory is assumed.

The message file must contain binary, hexadecimal text, or base64 encoded data. It must contain data formatted using any of the following ASN.1 encoding rules:

- BER (Basic Encoding Rules)
- DER (Distinguished Encoding Rules)
- CER (Canonical Encoding Rules)
- PER (Packed Encoding Rules, basic aligned variant)
- U-PER (Packed Encoding Rules, basic unaligned variant)

The following table lists all of the command line options and what they are used for:

Option	Argument	Description
-ascii		Scan data in untyped fields and if all bytes contain values within the ASCII character range, display as standard text. Otherwise display as formatted hexadecimal text. Note that his option only has meaning if BER/DER/CER data is being decoded and no schema file is specified.
-bcdhandling	default | none | bcd | tbcd	Determines how binary coded decimal (BCD) strings are handled. These refer to OCTET STRING's defined within the ASN.1 schema in the following form: BCDSTRING ::= OCTET STRING TBCDSTRING ::= OCTET STRING These are not standard ASN.1 types, yet they are widely used in telephony applications. This command line option provides a way to force a particular type of handling of the data values. The options are as follows: default : Handle BCDSTRING (and similar) as a BCD string (no digit swapping) and handle TBCDSTRING (and similar) as a TBCD string (swap digits). This is what is done if this argument is not specified. none : Do not do any special conversion of these types of OCTET STRING values. bcd : Force conversion of all of these types of declared values to BCD format (i.e. do not swap digits).

		tbcd : Force conversion of all of these types to TBCD format (i.e. swap digits). Further details on the conversion formats is provided in the OCTET STRING data mapping section.
-ber		Indicates input file contains BER encoded data or one of the BER subsets (DER or CER).
-bitsfmt	hex | bin	Specifies how BIT STRING values should be displayed. The options are hex (hexadecimal formatted text) or bin (binary formatted text – i.e. 1's and 0's).
-emptyDefault		Insert an element with a default value as specified in the schema at the location of a missing element in the instance.
-emptyOptionals		Insert an empty element at the location of a missing element in the schema that was declared to be optional.
-inputfiletype	binary | hextext | base64	Specifies the format of the input file (binary, hextext, or base64). If hextext, whitespace between characters will be ignored.
-noopentype		This option suppresses the attempted further decoding of embedded open type values with a message. By default, the translator will try to use tag information, table constraints, or type constraint information to try and further decode open type data. If this option is set, this is not done and the data is formatted as hexadecimal text.
-nowhitespace		Do not generate any whitespace (blanks and newline characters) between elements. This make the generated XML document more compact at the expense of reducing readability.
-o	<filename>	This option defines the output XML filename. The XML message will be printed to this file. By default, the XML message will be printed to standard output.
-paddingbyte	<hex byte value>	This option allows specification of a padding byte that may occur between records in a file containing multiple ASN.1 encoded data records. By default, zero (0x00) is assumed to be a padding byte since this is not a legal tag value that may be used to start an ASN.1 BER encoded message. Bytes of this value will be skipped when they are encountered at the end of a record and the next encoded record will be assumed to start at the position of the fist byte not having this value.
-pdu	<typename>	This option explicitly defines the PDU (Protocol Data Unit) type name for the binary message. By default, the *asn2xml*program will try to determine the PDU of the message using a tag-matching algorithm in the case of BER/DER/CER. In the case of PER, the protocol data unit must be specified.
-per		Indicates the input file contains data encoded in Packed Encoding Rules (PER) basic or canonical aligned format as defined in the ITU-T X.691 standard.
-rootelement	<name>	This option is used to specify a root element name that will be used to wrap the entire XML message at the outer level. This makes it possible to create

		an XML document for an ASN.1 file containing multiple individually encoded binary messages (this is common for some Call Detail Record (CDR) ASN.1 formats).
-schema	<filenames>	This option defines the ASN.1 files used to encode the binary message. This file must strictly adhere to the syntax specified in ASN.1 standard ITU X.680. More than one ASN.1 filename may be specified by separating the names with spaces (e.g. asn2xml message.dat -schema X.asn Y.asn Z.asn) The specified ASN.1 file(s) must contain the definition of the root PDU type used to encode the binary message.
-uper		Indicates input file contains data encoded in Packed Encoding Rules (PER) basic unaligned format as defined in the ITU-T X.691 standard.

Using the ASN2XML DLL

The ASN2XML DLL allows the ASN.1-to-XML translator to be embedded into an application program. For example, it is possible to create a GUI interface for displaying the XML output instead of having it go to standard output or to a file.

The DLL is used from a C++ program by including the *Asn2Xml.h* header file and using the *Asn2Xml* class. This class provides methods for setting most of the options defined in the command-line version of the tool. The basic usage pattern is as follows:

- Declare an Asn2Xml class variable in your program and instantiate it using the default constructor.
- Use the various setter methods to set all of the required options for the translation. This includes encoding rules types, ASN.1 schema files, input sources, and other options.
- Invoke one of the *toXML* methods to translate the input to XML.

The following is a summary of the Asn2Xml class definition showing all of the public methods:

```
class Asn2Xml {
 public:
   enum EncodingRules { Unknown, BER, PER, UPER } ;

   enum EncodingOptions {
      EmptyOptional, EmptyDefault, NoWhitespace, Ascii
   };

   enum BinaryDataFormat { Bin, Hex } ;

   /**
    * The default constructor initializes all variables to their
    * null or default state.
    */
```

```
Asn2Xml();

/**
 * The destructor frees any internal memory in use by the instance.
 */
virtual ~Asn2Xml();

/**
 * This method is used to add an ASN.1 schema to the schema file list.
 *
 * @param filepath  Full path to the schema file.
 */
void addAsn1Schema (const char* filepath);

/**
 * This method clears all existing ASN.1 schemas from the internal
 * schema file list.
 */
void clearAsn1Schemas ();

/**
 * This methods sets the executable path which is normally argv[0].
 * This parameter is used by license checking to allow the executable
 * directory to be searched for the license file.  It is not required
 * that this be set; other methods can be used to find the license
 * file (for example, setting the OSLICDIR environment variable or
 * putting the license file in the PATH).
 *
 * @param value  Executable path value (normally argv[0])
 */
inline void setExePath (const char* value) {
   mpExePath = value;
}

/**
 * This method sets a file containing binary data as the data source.
 *
 * @param filepath Full path to file containing binary ASN.1 encoded
 * data to be parsed.
 * @return Status of operation: 0 for success, negative number for error.
 */
int setFileDataSource (const char* filepath);

/**
 * This method sets a memory buffer containing binary data as the
 * data source.
 *
 * @param buffer Data buffer (byte array) containing data to be parsed.
 * @param bufsiz Size of the data buffer.
 * @return Status of operation: 0 for success, negative number for error.
 */
int setMemDataSource (const OSOCTET* buffer, size_t bufsize);

/**
 * This method sets the ASN.1 encoding rules that were used to encode
 * the data specified in the data source.
 *
 * @param encrules ASN.1 encoding rules enumerated identifier
```

```
 */
void setEncodingRules (EncodingRules encrules);

/**
 * This method sets the given encoding option.
 *
 * @param option Enumerated encoding option.
 */
void setEncodingOption (EncodingOptions option);

/**
 * This method sets the name of the Protocol Data Unit (PDU) to be
 * used for the top-level element in the ASN.1-to-XML translation.
 *
 * @param typeName  Name of the PDU data type.
 */
void setPDUType (const char* typeName);

/**
 * This method sets the format in which BIT STRING data content is
 * to be displayed in the XML markup.  The binary ('Bin') option
 * results in '0' and 0's in the output whereas hex ('Hex') results
 * in hexadecimal text.
 *
 * @param format  Format in which data is to be displayed.
 */
void setBitStringFormat (BinaryDataFormat format);

/**
 * This method sets text that will be inserted as an XML comment
 * immediately after the XML header has been output.  It can be used
 * for purposes such as providing information on the application that
 * created the document or providing general information on the
 * contents of the document.
 *
 * @param comment Comment text to be inserted.  This text would not
 *   include the XML comment delimitters (<!-- and -->).
 */
void setStartComment (const OSUTF8CHAR* comment);

/**
 * This method translates the binary data specified in the data source
 * to XML and writes the output to the given output stream.
 *
 * @param ostream Output stream to which XML data is to be written.
 * @return Status of translation operation: 0 is success; a negative
 *    status value indicates an error occurred.
 */
int toXML (OSRTOutputStream& ostream);

/**
 * This is a convenience method for writing the XML output directly
 * to an output file.  This can also be accomplished by using the
 * toXML method with an OSRTFileOutputStream object.
 *
 * @param filepath Full path to file to which XML data is to be written.
 *    If null, output will be written to stdout.
 * @return Status of translation operation: 0 is success; a negative
```

```
     *    status value indicates an error occurred.
     */
    int toXMLFile (const char* filepath);

} ;
```

ASN.1 to XML Value Mappings

This section defines the mapping between ASN.1 encoded data values and XML for each of the ASN.1 types defined in the X.680 standard.

General Mapping without ASN.1 Schema Information

A BER, DER, or CER encoded data stream may be translated to XML format without providing associated ASN.1 schema information. In this case, XML element names are derived from built-in ASN.1 tag information contained within the message and values are encoded as either hexadecimal text, ASCII text, or as specific data-typed values if universal tag information is present.

XML element names derived from ASN.1 tag names for all tags except known universal tags is in the following general form:

```
<TagClass_TagValue>
```

where TagClass is the tag class name (APPLICATION, CONTEXT, or PRIVATE) and TagValue is the numeric tag value. For example, an [APPLICATION 1] tag would be printed as <APPLICATION_1> and a [0] tag (context-specific zero) would be printed as <CONTEXT_0>.

In the case of known universal tags, the tag value is derived using the name of the known type. In general, this is the type name defined in the ASN.1 standard with an underscore character used in place of embedded whitespace if it exists. The following table shows the XML tag names for the known types:

Tag	XML Element Name
UNIVERSAL 1	BOOLEAN
UNIVERSAL 2	INTEGER
UNIVERSAL 3	BIT_STRING
UNIVERSAL 4	OCTET_STRING
UNIVERSAL 5	NULL
UNIVERSAL 6	OBJECT_IDENTIFIER
UNIVERSAL 7	OBJECT_DESCRIPTOR
UNIVERSAL 8	EXTERNAL
UNIVERSAL 9	REAL
UNIVERSAL 10	ENUMERATED
UNIVERSAL 11	EMBEDDED_PDV
UNIVERSAL 12	UTF8STRING
UNIVERSAL 13	RELATIVE_OID
UNIVERSAL 16	SEQUENCE
UNIVERSAL 17	SET
UNIVERSAL 18-22, 25-30	Character string

UNIVERSAL 23	UTCTIME
UNIVERSAL 24	GENERALIZEDTIME

Element content will be formatted in one of three ways:

- Hexadecimal text
- ASCII (character) text
- Specific-typed value

Hexadecimal text is the default format for untyped content. ASCII text will be used if the -ascii command-line switch is specified and all byte values within a particular field are determined to be printable ASCII characters. A specific-type value encoding will be done if a known universal tag is found. The mapping in this case will be as described in the "Specific ASN.1 Type to XML Value Mapping" section below.

General Mapping with ASN.1 Schema Information

ASN.1 schema information is used if one or more ASN.1 schema files are specified on the command-line using the -schema command-line switch. In this case, element names as specified in the schema file are used for the XML element names and the content is decoded based on the specific type.

It is possible to use the –pdu command-line switch to force the association of a type within the specification to the message. This is only necessary if the ASN.1 files contain multiple types with the same start tag as the message type. Otherwise, the program will be able to determine on its own which type to use by matching tags. Note that this is true for BER/DER/CER messages only - for PER, it is necessary to specify the PDU type along with the schema.

Specific ASN.1 Type to Value Mappings

This section defines the type-to-value mapping for each of the specific ASN.1 types. Note that these mappings are not in the form defined in the ASN.1 XML Encoding Rules (XER) standard (X.693). There are some differences to make the XML align better to an XML schema document (XSD) that might be used to describe the encoding. An XSD that can be used to validate the translated XML document can be generated using the Objective System's ASN2XSD tool.

BOOLEAN

An ASN.1 boolean value is transformed into the keyword 'true' or 'false'. If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <BOOLEAN> tag is added.

Example

```
b BOOLEAN ::= TRUE
```

maps to:

```
<BOOLEAN>true</BOOLEAN>
```

INTEGER

An ASN.1 integer value is transformed into numeric text. The one exception to this rule is if named number identifiers are specified for the integer type. In this case, if the number matches one of the declared identifiers, the identifier text is used.

If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, an <INTEGER> tag is added.

Example

```
i INTEGER ::= 35

maps to:

<INTEGER>35</INTEGER>
```

ENUMERATED

An ASN.1 enumerated value is transformed into the enumerated identifier text value. If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, an <ENUMERATED> tag is added.

Example

```
colors ENUMERATED { red green blue } ::= green

maps to:

<ENUMERATED>green</ENUMERATED>
```

BIT STRING

An ASN.1 bit string value is transformed into one of three forms:

- Binary text ('1's and '0's)
- Hexadecimal text
- Named bit identifiers

Binary text is the default output format. This is used if the bit string type contains no named bit identifiers and if specification of hexadecimal output was not specified on the asn2xml command-line.

Hexadecimal text is displayed when the '-bitsfmt hex' command-line option is used. Any unused bits in the last octet are set to zero. Note that the other bits are displayed in most-significant bit order as they appear in the string in the last byte (i.e. they are not right shifted). For example, if the last byte contains a bit string value of 1010xxxx (where x denotes an unused bit), the string is displayed as A0 in the XML output, not 0A.

Named bit identifiers are used in the case of a bit string declared with identifiers. In this case, the XML content is a space-separated list of identifier values corresponding to the bits that are set. It is assumed

that bits in the string all have corresponding identifier values.

If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <BIT_STRING> tag is added.

Example

```
BS3 BIT STRING { zero(0), a(1), b(3), c(5) } ::= '100100'B
```

maps to:

```
<BIT_STRING>zero b</BIT_STRING>
```

OCTET STRING

An ASN.1 octet string value is transformed into one of two forms:

- Hexadecimal text
- ASCII character text

Hexadecimal text is the default display type. ASCII text will be used for the content when the '-ascii' command-line option is used and the field contains are printable ASCII characters.

A special case of OCTET STRING handling is for declared binary-coded decimal (BCD) data types. This is discussed in a later section.

If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <OCTET_STRING> tag is added.

Example

```
OS OCTET STRING ::= '3031'H
```

maps to:

```
<OCTET_STRING>3031</OCTET_STRING>
```

if -ascii is specified:

```
<OCTET_STRING>01</OCTET_STRING>
```

NULL

An ASN.1 null value is displayed as an empty XML element. If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <NULL> tag is added.

Example

```
n NULL ::= NULL
```

maps to:

```
<NULL/>
```

OBJECT IDENTIFIER

An ASN.1 object identifier value is mapped into space-separated list of identifiers in numeric and/or named number format. The identifiers are enclosed in curly braces ({}). Numeric identifiers are simply numbers. Named number format is a textual identifier followed by the corresponding numeric identifier in parentheses. The named number format is used in cases where the identifier can be determined from the schema or is a well known identifier as specified in the ASN.1 standard.

If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <OBJECT_IDENTIFIER> tag is added.

Example

```
OID OBJECT IDENTIFIER ::= { 1 2 840 113549 1 1 2  }
```

maps to:

```
<OBJECT_IDENTIFIER>{ 1 2 840 113549 1 1 2  }</OBJECT_IDENTIFIER>
```

RELATIVE OID

The mapping of the RELATIVE OID data type is identical to that for OBJECT IDENTIFIER

Character String

An ASN.1 value of any of the known character string types is transformed into the character string text in whatever the default encoding for that type is. For example, an IA5String would contain an ASCII text value whereas a BMPString would contain a Unicode value.

If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a tag is added which is the name of the character string type as defined in the ASN.1 standard in angle brackets. For example, the default tag for a UTF8String type would be <UTF8String>.

Example

```
str UTF8String ::= 'this is a test'
```

maps to:

```
<UTF8String>this is a test<UTF8String>
```

Binary-Coded Decimal (BCD) String

Binary-Coded Decimal (BCD) strings and Telephony Binary-Coded Decimal (TBCD) strings are not part of the ASN.1 standard, but their use is prevalent in many telephony-related ASN.1 specifications. Conversion of these types into standard numeric text strings is supported.

In general, BCD strings pack 2 numeric digits into a single byte value by using a 4-bit nibble to hold each digit. The digits are sometimes reversed in the bytes with the convention being that bytes are reversed in TBCD strings and not reversed in BCD strings. But since this is not standardized, different

specifications will not always follow this convention. In this case, the -bcdhandling command-line option can be used to force a certain type of conversion. The default handling is to reverse digits in string determined to be TBCD strings and not reverse digits in BCD strings. The 'bcd' option is used to for no reversal of digits in all of these types of strings. The 'tbcd' option causes digits to always be reversed. Finally, the 'none' option turns off this type of conversion altogether.

OCTET STRINGS's are determined to be BCD strings based on how the types are declared in the ASN.1 schema document. The following declaration is recognized as a BCD string type:

```
BCDString ::= OCTET STRING
```

Similarly, the following declaration is recognized as a TBCD string:

```
TBCDString ::= OCTET STRING
```

Some variations of these keywords are recognized as well including all upper case (BCDSTRING).

REAL

An ASN.1 real value is transformed into numeric text in exponential number format. If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <REAL> tag is added.

SEQUENCE

An ASN.1 sequence value is transformed into an XML value containing an element wrapper with each of the XML element encoded values inside.

If an element is declared to be optional in the schema and is not present in the encoded value being translated, it is simply not added to the translated XML. The exception to this rule is if the -emptyOptionals command-line argument is used. In this case, an empty XML element declaration will be added to the XML instance.

If an element is declared to have a default value and the element is not present in the encoded value being translated, an entry is not added to the translated XML. The exception to this rule is if the -emptyDefaults command-line argument is used. In this case, an XML element with the declared default value is added to the XML instance.

If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <SEQUENCE> tag is added.

Example

```
Name ::= SEQUENCE {
   first UTF8String,
   middle UTF8String OPTIONAL,
   last UTF8String
}
myName Name ::= { first 'Joe', last 'Jones' }
```

Maps to:

```
<myName>
   <first>Joe</first>
   <last>Jones</last>
</myName>
```

if -emptyOptionals is specified:

```
<myName>
   <first>Joe</first>
   <middle/>
   <last>Jones</last>
</myName>
```

SET

The mapping of an ASN.1 SET value is identical to that of SEQUENCE.

If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <SET> tag is added.

SEQUENCE OF / SET OF

The representation of a repeating value in XML varies depending on the type of the element value.

If the value being translated is a sequence of an atomic primitive type, the XML content is a space-separated list of values. The definition of 'atomic primitive type' is any primitive type whose value may not contain embedded whitespace. This includes BOOLEAN, INTEGER, ENUMERATED, REAL, BIT STRING, and OCTET STRING values.

If the value being translated is a constructed type or if it may contain whitespace, the value is wrapped in a tag which is either the name of the encapsulating type (defined or built-in) or the SEQUENCE OF element name if this form of the type was used.

If BER/DER/CER data is being decoded without a schema and the universal tag for this type is parsed, a <SEQUENCE> or <SET> tag is added. That is because the tag value (hex 30 or 31) is the same for SEQUENCE OF or SET OF as it is for SEQUENCE or SET.

The following are several different examples that show an encoding of specific repeating types:

Example – Space-separated List

```
mySeqOfInt SEQUENCE OF INTEGER ::= { 1, 2, 3 }
```

maps to:

<mySeqOfInt>1 2 3</mySeqOfInt>

Example – List of Strings

```
mySeqOfStrings SEQUENCE OF UTF8String ::= {
```

```
   'this is a test', 'this is another test'
}
```

maps to:

```
<mySeqOfString>
   <UTF8String>this is a test</UTF8String>
   <UTF8String>this is another test</UTF8String>
</mySeqOfString>
```

Example – List of Constructed Type

```
Name ::= SEQUENCE {
   first UTF8String,
   middle UTF8String OPTIONAL,
   last UTF8String
}

myListOfNames SEQUENCE OF Name ::= {
   { first 'Joe', last 'Jones' },
   { first 'John', middle 'P', last 'Smith' }
}
```

Maps to:

```
<myListOfNames>
   <Name>
      <first>Joe</first>
      <last>Jones</last>
   </Name>
   <Name>
      <first>John</first>
      <middle>P</middle>
      <last>Smith</last>
   </Name>
</myListOfNames>
```

CHOICE

The mapping of an ASN.1 CHOICE value is the alternative element tag followed by the value translated to XML format.

Example

```
MyChoice ::= CHOICE {
   a  INTEGER,
   b  OCTET STRING,
   c  UTF8String
}
choiceValue MyChoice ::= { c : 'my choice value' }
```

maps to:

```
<choiceValue>
   <c>my choice value</c>
```

```
</choiceValue>
```

Open Type

The mapping of an ASN.1 open type value depends on whether the actual type used to represent the value can be determined. ASN2XML attempts to determine the actual type using the following methods (in this order):

- Table constraints
- Tag lookup in all defined schema types (BER/DER/CER only)
- Universal tag lookup (BER/DER/CER only)

If the type can be determined, an XML element tag containing the type name is first added followed by the translated content of the value.

If the type cannot be determined, the open type content is translated into hexadecimal text from of the encoded value. This will also be done if the -noopentype command-line switch is used.

Example:

As an example, consider the AlgorithmIdentifier type used in the AuthenticationFramework and other related security specifications:

```
          AlgorithmIdentifier ::= SEQUENCE {
             algorithm   ALGORITHM.&id({SupportedAlgorithms}),
             parameters  ALGORITHM.&Type({SupportedAlgorithms}{@algorithm})
OPTIONAL
          }
```

In this case, the parameters element references an open type that is tied to a type value based on the value of the algorithm key. Without getting into the details of the use of the accompanying information object sets, it is known that for an algorithm value of object identifier { 1 2 840 113549 1 1 2 }, the type of the parameters field is NULL (i.e. there are no associated parameters). The XML translation in this case will be the following:

```
<algorithm>{ 1 2 840 113549 1 1 2  }</algorithm>
<parameters>
   <NULL/>
</parameters>
```