



# **ASN1 DECODER v3.1**

*User's Manual*



The software described in this document is furnished under a license agreement and may be used only in accordance with the terms of this agreement.

### **Copyright Notice**

Copyright ©2017–2021 Objective Systems, Inc. All rights reserved.

This document may be distributed in any form, electronic or otherwise, provided that it is distributed in its entirety and that the copyright and this notice are included.

### **Author's Contact Information**

Comments, suggestions, and inquiries regarding this product may be submitted via electronic mail to [info@obj-sys.com](mailto:info@obj-sys.com).



## Table of Contents

Revision History.....	7
Overview of ASN1DECODER.....	9
Using ASN1DECODER.....	10
Installation.....	10
Installing on a Windows System.....	10
Installing on a UNIX System.....	11
Command-line Options.....	11
Input and Output Encoding Options.....	13
Common Options.....	13
XML/JSON Options.....	15
License File Options.....	16
Configuration Files.....	17
Licensing.....	19
ASN.1 to XML Type Mappings.....	20
General Mapping without ASN.1 Schema Information.....	20
General Mapping with ASN.1 Schema Information.....	21
Specific ASN.1 Type to Value Mappings.....	22
ASN.1 to JSON Type Mappings.....	31



## Revision History

- February 2018 — Initial release of documentation (2.7 – to match ASN1VE version it is bundled with)
- April 2018 – Minor updates and corrections.
- January 2019 – Version and copyright updates. Add reference to X.697 JSON Encoding Rules (JER).
- January 2020 – Version and copyright updates. Add option for named INTEGERS.
- January 2021 – Version and copyright updates.





## Overview of ASN1DECODER

ASN1DECODER is a command-line tool that translates ASN.1 data encoded in the Basic, Canonical, Distinguished, or Packed encoding rules into a few text formats suitable for ingestion into text processing tools. ASN1DECODER supports converting ASN.1 data to XML, XER, and JSON data formats.

Conversions to XML support both an Objective Systems custom format as well as the XML Encoding Rules standard as described in ITU-T standard X.693. Conversions from all ASN.1 binary encodings (BER, CER, DER, PER) are supported by ASN1DECODER.

Conversions to JSON conform to ITU-T X.697 JSON Encoding Rules (JER). (Note: earlier versions followed a proprietary format, which is documented, along with the differences from X.697, at

<https://www.obj-sys.com/docs/JSONEncodingRules.pdf>.)

## Using ASN1DECODER

### *Installation*

ASN1DECODER is available either as part of an ASN1VE package (also known as ASN1VE Pro) or as a standalone kit. Either way, it is contained as part of an executable installation program for Windows or a `.tar.gz` archive for UNIX systems. The package is comprised of the following directory tree (using version 3.0.0 as an example):

```
asn1ve_v300 (when bundled with ASN1VE)
  OR
asn1dec_v300 (when packaged standalone)
|
+-asn1specs
|
+-bin
|
+-doc
|
+-sample
|
+-sample_per
```

The `bin` subdirectory contains the `asn1dec` executable. The `asn1specs` directory contains specifications used by some of the sample programs in the `sample` directory. This document is found in the `doc` directory.

### *Installing on a Windows System*

To install ASN1DECODER on a Windows system, simply double-click the executable installer program (ASN1VE Pro or ASN1DECODER standalone). Selecting the default installation options will install ASN1DECODER in either `c:\asn1ve_v300` (when bundled with ASN1VE) or `c:\asn1dec_v300` (when packaged standalone).

## ***Installing on a UNIX System***

To install ASN1DECODER on a UNIX system, simply unzip and untar the `.tar.gz` archive. The program may be unpacked in any directory in which the user has permissions. No installation program is available to install ASN1DECODER to `/usr/local` or other common installation paths, but it is not difficult to manually add links if needed.

## ***Command-line Options***

Invoking `asn1dec` without any options will show a usage message that contains the command-line options:

```
ASN1DECODER, version 3.0.0
```

```
ASN.1 to text decoder
```

```
Copyright (c) 2017-2021 Objective Systems, Inc. All Rights Reserved.
```

```
Usage: asn1dec <input files> options
```

```
<input files>          ASN.1 message file name (wildcards are okay)
```

```
Input encoding options:
```

```
-in <encode_rule>
```

```
ber                   Decode from BER (basic encoding rules)
per                   Decode from PER (aligned packed encoding rules)
uper                  Decode from U-PER (unaligned packed encoding rules)
```

```
Output encoding options:
```

```
-out <encode_rule>
```

```
xml                   Encode to XML type
xer                   Encode to XML XER type
json                  Encode to JSON type
```

```
Common options:
```

```
-schema <filename>    ASN.1 definition file name(s)
-config <filename>    Configuration file name
-I <directory>        Import ASN.1 files from <directory>
-pdu <typename>       Message PDU type name
```

-o <filename> Output filename (use "<base>.xml" for XML batch output)

-bcdhandling <default|none|bcd|tbcd>  
Define handling of OCTET STRINGS declared to be  
binary coded decimal (BCD) strings

-oidformat <oidDefValue|namedNumbers|numbersOnly|dottedNamedNumbers|  
dottedNumbers>  
Define format of Object Identifier display

-noopentype Disable automatic open type decoding

-paddingbyte <hexbyte> Additional padding byte

-bitsfmt <hex|bin> BIT STRING content output format

-inputFileType <binary|hextext|base64|pem|pcap|pcapng>  
Format of data in input file

-skip <num> Skip <num> bytes between messages

-skipend <num> Skip <num> bytes at end of message

-headerOffset <num> Skip the first <num> bytes in a data file

-q Turn off all output except errors

-named-number-ids Show named values for INTEGERS instead of numbers

XML options:

-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

-nowhitespace Remove all whitespace between elements

-noxmlcomments Do not include header comment or message count in output

-rootElement <element> Root Element Name

-skip-bad-records Skip messages which could not be decoded

-headers <ts-32-297|length4|eth|tcpip|tpkt>  
Headers to be processed before start of encoded  
data. Multiple header can be specified using +  
character (for example, eth+tcpip). Header  
types are:  
ts-32-397 - 3GPP TS 32.297 CDR records  
length4 - 4-byte length header  
eth - Ethernet  
tcpip - TCP/IP

tpkt - TPKT

License options:

-licfiles                   List discovered license file  
-licinstall <file>        Install license file  
-licdeact                  Deactivate existing license

The following sections summarize the command-line options.

## Input and Output Encoding Options

The following input and output encoding options describe the various transformations.

Option	Arguments	Description
-in	ber	Decode input from a BER file.
	per	Decode input from a PER file.
	uper	Decode input from a U-PER file.
-out	xml	Encode/write output to an XML file.
	xer	Encode/write output to an XER file.
	json	Encode/write output to a JSON file.

## Common Options

The following options are common to all transformations.

Option	Arguments	Description
<filename>		<filename> is the name of the input message to decode. This element is <i>required</i> . The use of wildcards ( <i>e.g.</i> * and ?) is supported.
-schema	<filename>	This option is <i>required</i> when decoding PER data. When converting BER data to XML or JSON, a schema is optional. Without a schema, ASN1DECODER will convert the data to a non-standard encoding, using tag names.
-config	<filename>	Allows an Obj-Sys configuration file to be used in translation. Configuration settings can be used to apply options to specific items within a schema.
-bcdhandling	<default   none   bcd   tbcd>	Define handling of OCTET STRINGs declared

Option	Arguments	Description
		to be binary coded decimal (BCD) strings. By default, types declared as BCD or TBCD strings will be translated as such. <none> forces translation to be performed as usual, while <bcd> and <tbcd> force their respective formatting on such OCTET STRINGS.
-bitsfmt	<hex   bin>	Specify how BIT STRING items are formatted. By default they are expressed as hexadecimal strings; use bin to express them as binary strings instead.
-inputFileType	<binary   hextext   base64   pem   pcap   pcapng>	Specify how the input data is formatted. By default ASN1DECODER will assume that the input data is binary, but it can also decode hexadecimal, base64, or PEM encoded data. Whitespace in the input is ignored when hextext is specified.
-noopentype		Disables the conversion of open types in the output.
-oidformat	<oidDefValue   namedNumbers   numbersOnly   dottedNamedNumbers   dottedNumbers>	Define format of Object Identifier display. By default (or using <oidDefValue>), the value is displayed as it was defined. Using <namedNumbers> or <numbersOnly>, the value is displayed as such (for example “iso(1)” or “1”, respectively) and includes as many arcs as possible.
-skip	<num>	Specify the number of bytes to skip at the start of each message in the data file.
-skipend	<num>	Specify the number of bytes to skip at the end of each message in the data file.
-headerOffset	<num>	Specify the number of bytes to skip at the start of the data file.
-I	<directory>	Specify directory to import additional ASN.1 files from.

-paddingbyte	<hexbyte>	<hexbyte> is the hexadecimal value of a padding byte that may appear in the input message. Call data records (CDRs) are commonly continuously dumped to files by telephony equipment. If no information is available, the records are often padded by 0x00 or 0xFF bytes. The default padding byte is 0x00. <hexbyte> may be formatted with or without a 0x prefix.
-pdu	<typename>	<typename> is the name of the PDU data type to be decoded. This option is necessary when the top-level data type is ambiguous. It is also required when converting PER data.
-o	<filename>	Specify the output XML or JSON <filename> instead of writing output to standard out. Set <filename> to “<base>.xml” to specify batch output when converting multiple files for XML.
-q		Operate in a “quiet” mode more suitable for batch processing. Informational messages are limited and only error output will be reported.
-named-number-ids		Show the text value of named INTEGERS rather than the integer itself.

## XML/JSON Options

The following options can be used when converting to XML or JSON.

Option	Arguments	Description
-ascii		If all bytes in a decoded value are within the ASCII character range, display as standard text. Otherwise display as formatted hexadecimal text. This option only has meaning if BER data is decoded without a schema file.
-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 ele-

Option	Arguments	Description
		ment in the schema that was declared to be optional.
-nowhitespace		Do not generate any whitespace (blanks and newline characters) between elements. This makes the generated document more compact at the expense of readability.
-rootElement	<name>	Specify the root element <name> 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 (a common feature of many Call Detail Record ASN.1 formats). Not used for JSON output.
-noxmlcomments		Remove XML header comment and message/block count from output. (XML only)
-skip-bad-records		This option enables more thorough detection of badly formed records and attempts to skip such records. This can occur if an unrecognized tag is encountered, for example. In some cases, it is impossible to continue translation after a bad record, such as when an incorrect length value was encoded.
-headers	<ts-32-297  length4   eth   tcpip   tpkt>	Specify a message file containing various types of headers, such as 3GPP TS 32.297 CDR records or Ethernet or TCP-IP headers.

## License File Options

The following options can be used to manage ASN1DECODER licenses, including installation, deactivation, and displaying license files.

Option	Arguments	Description
-licinstall	<file>	Install and activate the ASN1DECODER license file located at the path and file name specified by <file>.
-licdeact		Deactivate the currently installed license.
-licfiles		List the currently active license file.



## Configuration Files

ASN1DECODER provides the option of including a configuration file during the translation. This allows the user to set certain options for specific items in the schema.

Option	Values	Description
displayFormat	ipv4   ipv6   tbcd   3gpp-timestamp   text	Formats an OCTET STRING (“789ABCDE”, for example) as an IPv4 address (“120.154.188.224”), IPv6 address (“78:9A:BC:DE”), or TBCD string (“87*9a#cb”), respectively. “imei” and “imsi” are aliases for “tbcd.” The “3gpp-timestamp” option converts the OCTET STRING to a 3GPP date/time (“YYMMDDThhmmssShhmm”). The “text” option displays the binary data as text.
format	hex	When applied to integer types, the format option allows users to output the contents in base 16 rather than base 10. Output is prefixed with “0x” to explicitly denote hexadecimal digits.

Configuration files are formatted as XML and use the top-level tag <asn1config>. Below the top level, <module>, <production>, and <element> tags can be nested, one within the other. At each level, the tag must include the “name” attribute.

So for example, given a schema like this:

```
MyData DEFINITIONS ::= BEGIN

  IPAddress ::= OCTET STRING (SIZE (4))

  TwoIPAddresses ::= SEQUENCE {
    ipAddress1  IPAddress,
    ipAddress2  IPAddress
  }

END
```

A configuration file might look like this:

```
<asn1config>
  <module name="MyData">
    <production name="TwoIPAddresses">
      <element name="ipAddress1">
        <displayFormat>ipv4</displayFormat>
```

```
        </element>
    </production>
</module>
</asn1config>
```

In this case, the configuration specifies `displayFormat` at the element level for the `ipAddress1` element. Note that the `<element>` tag must be nested within a `<production>` tag (which must also be nested within a `<module>` tag). Then, whenever a `TwoIPAddresses` is translated, it would be output like this (in XML, for example):

```
<message>
  <!-- message 1 -->
  <TwoIPAddresses>
    <ipAddress1>123.45.67.89</ipAddress1>
    <ipAddress2>7B2D4359</ipAddress2>
  </TwoIPAddresses>
</message>
```

Since the `displayFormat` option was set at the element level and only for `ipAddress1`, only that element is affected. In order to apply such formatting for all `IPAddress` types, the option can be set at the production level like so:

```
<asn1config>
  <module name="MyData">
    <production name="IPAddress">
      <displayFormat>ipv4</displayFormat>
    </production>
  </module>
</asn1config>
```

Note that in this case, the `<production>` tag's `name` attribute specifies the type which will be affected, rather than the type containing the affected element, as above.

## Licensing

The ASN1DECODER application must have access to a valid ASN1VE license in order for it to run properly. This license key is normally sent by e-mail after a license is purchased (permanent key) or when the user requests an evaluation version (evaluation key). If not already set up for ASN1VE, this `osyslic.txt` or RLM license file can be copied into one of several places:

1. To a directory on the system-wide `PATH`.
2. To the directory indicated by the `OSLICDIR` (for `osyslic.txt`) or `RLM_LICENSE` (for RLM file) environment variable.
3. To the directory from which the application is being executed.

## ASN.1 to XML Type Mappings

This chapter describes 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

Tag	XML Element Name
UNIVERSAL 8	EXTERNAL
UNIVERSAL 9	REAL
UNIVERSAL 10	ENUMERATED
UNIVERSAL 12	EMBEDDED_PDV
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, or 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. 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. By default, these mappings are not in the form defined in the ASN.1 XML Encoding Rules (XER) standard (ITU-T X.693).

When a schema is provided using the `-schema` option, the output may be adjusted to conform to XER if desired by using the `-xer` option. XER is more verbose and less validation-friendly than our native XML export. It is provided for those occasions when strict conformance is required. Differences between the two formats are provided along with the schemaless mappings below.

**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.

<b>b BOOLEAN ::= TRUE</b>	
Schemaless	<code>&lt;BOOLEAN&gt;TRUE&lt;/BOOLEAN&gt;</code>
XML Mode	<code>&lt;b&gt;true&lt;/b&gt;</code>
XER Mode	<code>&lt;b&gt;&lt;TRUE/&gt;&lt;/b&gt;</code>

**INTEGER.** An ASN.1 integer value is transformed into numeric text.

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.

<b>i INTEGER ::= 35</b>	
Schemaless	<code>&lt;INTEGER&gt;35&lt;/INTEGER&gt;</code>
With schema	<code>&lt;i&gt;35&lt;/i&gt;</code>

**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.

<b>colors ENUMERATED {r, g, b} ::= g</b>	
Schemaless	<ENUMERATED>1</ENUMERATED>
XML Mode	<colors>g</colors>
XER Mode	<colors><g/></colors>

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

1. Binary Text (0's and 1's)
1. Hexadecimal text
2. 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 `asn1dec` 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.

<b>bs BIT STRING {z(0), a(1), b(2), c(3)} ::= '100100'B</b>	
Schemaless	<BIT_STRING>100100</BIT_STRING>
With Schema	<bs>100100</bs>

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

1. Hexadecimal text
2. 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 only printable ASCII characters. A special case of OCTET STRING handling is for binary-coded decimal (BCD) data types; these will be formatted as described below.

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.

**Binary-coded Decimal 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.

BCD strings usually pack two numeric digits into a single byte value by using a four-bit nibble to hold each digit. (Occasionally the nibbles are interpreted as control characters like like #, \*, and so on.) By convention, the nibbles are swapped in TBCD strings, but there are no official standards for this encoding.

The `-bcdhandling` command-line option can be used to force a certain type of conversion if an encoding does not follow the usual conventions. The default handling is to reverse digits in strings determined to be TBCD strings and not reverse digits in BCD strings. The `bcd` option instructs ASN1DECODER not to reverse digits for all BCD strings. The `tbcd` option instructs ASN1DECODER to reverse the digits for all BCD strings.



If no processing is desired, `-bcdhandling none` can be used to instruct ASN1DECODER to treat these strings as simple octet strings.

<b>os OCTET STRING ::= '3031'H</b>	
Schemaless	<OCTET_STRING>3031</OCTET_STRING>
With schema	<os>3031</os>
With -ascii	<os>01</os>

**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.

<b>n NULL ::= NULL</b>	
Schemaless	<NULL/>
XML Mode	<n/>
XER Mode	<n><NULL/></n>

**OBJECT IDENTIFIER and RELATIVE OID.** 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. The named-number format is a textual identifier followed by the corresponding numeric identifier in parentheses. It 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, an <OBJECT\_IDENTIFIER> tag is added.

<b>oid OBJECT IDENTIFIER ::=</b> <b>{ 1 2 840 113549 1 1 2 }</b>	
Schemaless	<OBJECT_IDENTIFIER>{1 2 840 113549 1 1 2} </OBJECT_IDENTIFIER>
With schema	<oid>{ 1 2 840 113549 1 1 2 }</oid>

The mapping for RELATIVE OID is the same as 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>.

<b>str UTF8String ::= "testing"</b>	
Schemaless	<UTF8String>testing</UTF8String>
With schema	<str>testing</str>

**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.

<b>r REAL ::= 137.035999074</b>	
Schemaless	<REAL>137.035999074</REAL>
With schema	<r>137.035999074</r>

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

<b>name ::= SEQUENCE {   first UTF8String,   middle UTF8String OPTIONAL,   last UTF8String }</b>	<b>name Name ::= {   first "Joe",   last "Jones" }</b>
Schemaless	<SEQUENCE> <CONTEXT_0> <UTF8String>Joe</UTF8String> </CONTEXT_0> <CONTEXT_2>

	<pre>&lt;UTF8String&gt;Jones&lt;/UTF8String&gt; &lt;/CONTEXT_2&gt; &lt;/SEQUENCE&gt;</pre>
With schema	<pre>&lt;name&gt;   &lt;first&gt;Joe&lt;/first&gt;   &lt;last&gt;Jones&lt;/last&gt; &lt;/name&gt;</pre>
With -emptyOptionals	<pre>&lt;name&gt;   &lt;first&gt;Joe&lt;/first&gt;   &lt;middle/&gt;   &lt;last&gt;Jones&lt;/last&gt; &lt;/name&gt;</pre>

When a SET is used instead, the outer SEQUENCE tag is replaced with SET. The mappings are otherwise identical.

**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.

<b>soi SEQUENCE OF INTEGER ::= {1, 2, 3}</b>	
Schemaless	<pre>&lt;SEQUENCE&gt;   &lt;INTEGER&gt;1&lt;/INTEGER&gt;   &lt;INTEGER&gt;2&lt;/INTEGER&gt;</pre>

	<pre>&lt;INTEGER&gt;3&lt;/INTEGER&gt; &lt;/SEQUENCE&gt;</pre>
With schema	<pre>&lt;soi&gt;   &lt;INTEGER&gt;1&lt;/INTEGER&gt;   &lt;INTEGER&gt;2&lt;/INTEGER&gt;   &lt;INTEGER&gt;3&lt;/INTEGER&gt; &lt;/soi&gt;</pre>

<pre>sos SEQUENCE OF UTF8String ::= {   "test 1",   "test 2" }</pre>	
Schemaless	<pre>&lt;SEQUENCE&gt;   &lt;UTF8STRING&gt;test 1&lt;/UTF8STRING&gt;   &lt;UTF8STRING&gt;test 2&lt;/UTF8STRING&gt; &lt;/SEQUENCE&gt;</pre>
With schema	<pre>&lt;sos&gt;   &lt;UTF8String&gt;test 1&lt;/UTF8String&gt;   &lt;UTF8String&gt;test 2&lt;/UTF8String&gt; &lt;/sos&gt;</pre>

<pre>Name ::= SEQUENCE {   first UTF8String,   middle UTF8String OPTIONAL,   last UTF8String }</pre>	<pre>son SEQUENCE OF Name ::= {   { first 'Joe',     last 'Jones' },   { first 'John',     middle 'P',     last 'Smith' } }</pre>
Schemaless	<pre>&lt;SEQUENCE&gt;   &lt;SEQUENCE&gt;     &lt;UTF8STRING&gt;Joe&lt;/UTF8STRING&gt;     &lt;UTF8STRING&gt;Jones&lt;/UTF8STRING&gt;   &lt;/SEQUENCE&gt;   &lt;SEQUENCE&gt;     &lt;UTF8STRING&gt;John&lt;/UTF8STRING&gt;     &lt;UTF8STRING&gt;P&lt;/UTF8STRING&gt;     &lt;UTF8STRING&gt;Smith&lt;/UTF8STRING&gt;</pre>

	</SEQUENCE>
With schema. This example shows the results with <code>-emptyOptionals</code> selected. If it were not, the first <code>&lt;middle/&gt;</code> element would be omitted.	<pre> &lt;son&gt;   &lt;Name&gt;     &lt;first&gt;Joe&lt;/first&gt;     &lt;middle/&gt;     &lt;last&gt;Jones&lt;/last&gt;   &lt;/Name&gt;   &lt;Name&gt;     &lt;first&gt;John&lt;/first&gt;     &lt;middle&gt;P&lt;/middle&gt;     &lt;last&gt;Smith&lt;/last&gt;   &lt;/Name&gt; &lt;/son&gt; </pre>

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

<pre> <b>PDU CHOICE ::= {</b>   <b>a INTEGER,</b>   <b>b OCTET STRING,</b>   <b>s UTF8String</b> <b>}</b> </pre>	<pre> <b>c PDU ::= { a : 42 }</b> </pre>
Schemaless	<INTEGER>42</INTEGER>
With schema	<pre> &lt;PDU&gt;   &lt;a&gt;42&lt;/a&gt; &lt;/PDU&gt; </pre>

**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. ASN1DECODER attempts to determine the actual type using the following methods (in this order):

1. Table constraints
2. Tag lookup in all defined schema types (BER/DER/CER only)
3. 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.

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>
```

## **ASN.1 to JSON Type Mappings**

This version of ASN1DECODER follows ITU-T X.697 JSON Encoding Rules (JER).

Earlier versions of ASN1DECODER followed a proprietary mapping. For information on that mapping and how it differs from X.697, refer to “Javascript Object Notation (JSON) Encoding Rules for ASN.1” at [www.obj-sys.com/docs/JSONEncodingRules.pdf](http://www.obj-sys.com/docs/JSONEncodingRules.pdf).