Abstract Syntax Notation One

Abstract Syntax Notation One (**ASN.1**) is a standard interface description language for defining data structures that can be serialized and deserialized in a cross-platform way. It is broadly used in telecommunications and computer networking, and especially in cryptography.

Protocol developers define data structures in ASN.1 modules, which are generally a section of a broader standards document written in the ASN.1 language. Because the language is both human-readable and machine-readable, modules can be automatically turned into libraries that process their data structures, using an ASN.1 compiler.

ASN.1 is a joint standard of the International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), and International Telecommunication Union Telecommunication Standardization Sector (ITU-T), originally defined in 1984 as part of CCITT X.409:1984.[1] In 1988, ASN.1 moved to its own standard, *X.208*, due to wide applicability. The substantially revised 1995 version is covered by the *X.680* series [2]. The latest revision of the X.680 series of recommendations is the 5.0 Edition, published in 2015.

Language support

ASN.1 is a data type declaration notation. It does not define how to manipulate a variable of such a type. This is actually defined in other languages such as SDL (Specification and Description Language) for executable modeling or TTCN-3 (Testing and Test Control Notation) for conformance testing. Both these languages natively support ASN.1 declarations. It is possible to import an ASN.1 module and declare variable of any of the ASN.1 types declared in the module.

Applications

ASN.1 is used in very diverse applications such as parcel tracking, power distribution and biomedicine. Its most extensive use continues to be in standard telecommunication protocols such as Intelligent networks, UMTS, Voice over IP, Interactive television and HiperAccess.

ASN.1 is used in X.509, which defines the format of certificates used in the HTTPS protocol for securely browsing the web, and in many other cryptographic systems.

It's also used in the PKCS group of cryptography standards, X.400 electronic mail, X.500 and Lightweight Directory Access Protocol (LDAP), H.323 (VoIP), Kerberos, BACnet, Simple Network Management Protocol (SNMP), and third- and fourth-generation wireless communications technologies (UMTS, LTE, and WiMAX 2).[3]

Encodings

ASN.1 is closely associated with a set of encoding rules that specify how to represent a data structure as a series of bytes. The standard ASN.1 encoding rules include:

- Basic Encoding Rules (BER) [4]
- Distinguished Encoding Rules (DER) [5]
- Canonical Encoding Rules (CER) [6]
- Packed Encoding Rules (PER, unaligned: UPER, canonical: CPER, canonical unaligned: CUPER) [7]
- Encoding Control Notation (ECN) [8]

- XML Encoding Rules (XER) [9]
- Canonical XML Encoding Rules (CXER) [10]
- Extended XML Encoding Rules (E-XER) [11]
- Octet Encoding Rules (OER, canonical: COER) [12]
- JSON Encoding Rules (JER) [13]
- Generic String Encoding Rules (GSER) [14]

The encoding rules are all platform-independent, and can be used across a variety of hardware and software.

The PEM format is often used to encapsulate DER-encoded ASN.1 certificates and keys in an ASCII-only format. The PEM version of a DER message consists of the base64 encoding of the DER message, preceded by "-----BEGIN FOO-----" and followed by "-----END FOO-----," where "FOO" may indicate "CERTIFICATE," "PUBLIC KEY," "PRIVATE KEY" or many other types of content.

Encoding Control Notation

ASN.1 recommendations provide a number of predefined encoding rules. If none of the existing encoding rules is satisfying the Encoding Control Notation provides a way for the user to define its own customized encoding rule.

Generic String Encoding Rules

Generic String Encoding Rules (GSER) are a set of ASN.1 encoding rules for producing a verbose, human-readable textual transfer syntax for data structures described in ASN.1. The purpose of GSER is to represent encoded data to the user or input data from the user, in a very straightforward format. GSER was originally designed for the Lightweight Directory Access Protocol (LDAP) and is rarely used outside of it. The use of GSER in actual protocols is discouraged since not all character string encodings supported by ASN.1 can be reproduced in it. The GSER encoding rules are specified in RFC 3641 and unlike other common types of encoding rules, are not standardised by ITU-T.

Packed Encoding Rules

Packed Encoding Rules (PER) are ASN.1 encoding rules for producing a compact transfer syntax for data structures described in ASN.1, defined in 1994.

This Recommendation or International Standard describes a set of encoding rules that can be applied to values of all ASN.1 types to achieve a much more compact representation than that achieved by the BER and its derivatives (described in ITU-T Rec. X.690 | ISO/IEC 8825-1).

It uses additional information, such as the lower and upper limits for numeric values, from the ASN.1 specification to represent the data units using the minimum number of bits. The compactness requires that the decoder knows the complete abstract syntax of the data structure to be decoded, however.

There are two variations of packed encoding rules: unaligned and aligned. With the unaligned encoding, the bits are packed with no regard for octet (byte) boundaries. With aligned encoding, certain types of data structures are aligned on octet boundaries, meaning there may be some number of wasted padding bits. Unaligned encoding uses the least number of bits, but presumably at some cost in processing time.

The packed encoding rules also define a restricted set of encoding rules, called **CANONICAL-PER**, which is intended to produce only a single possible encoding for any given data structure. CANONICAL-PER's role is therefore similar to the role of DER or CER.

Octet Encoding Rules

The *Octet Encoding Rules (OER)* were designed to be easy to implement and to produce encodings more compact than those produced by the Basic Encoding Rules (BER). In addition to reducing the effort of developing encoder/decoders, the use of OER can decrease bandwidth utilization (though not as much as the Packed Encoding Rules), save CPU cycles, and lower encoding/decoding latency.

JSON Encoding Rules

The ITU-T is standardizing[15] the new *JSON Encoding Rules (JER)*, which specify how to encode ASN.1 abstract values in JSON, so that the resulting encodings can be read by any JSON reader. When an existing ASN.1 schema is used with JER, a default JER encoding is produced, but the author of the ASN.1 schema will also be able to alter the structure of the JER encodings in specific ways by including *JER encoding instructions* in the schema. This will allow ASN.1 to be used as a schema language for JSON.

Example

This is an example ASN.1 module defining the messages (data structures) of a fictitious Foo Protocol:

```
FooProtocol DEFINITIONS ::= BEGIN

    FooQuestion ::= SEQUENCE {
        trackingNumber INTEGER,
        question       IA5String
    }

    FooAnswer ::= SEQUENCE {
        questionNumber INTEGER,
        answer         BOOLEAN
    }

END
```

This could be a specification published by creators of Foo Protocol. Conversation flows, transaction interchanges, and states are not defined in ASN.1, but are left to other notations and textual description of the protocol.

Assuming a message that complies with the Foo Protocol and that will be sent to the receiving party, this particular message (protocol data unit (PDU)) is:

```
myQuestion FooQuestion ::= SEQUENCE {
    trackingNumber     5,
    question           "Anybody there?"
}
```

ASN.1 supports constraints on values and sizes, and extensibility. The above specification can be changed to

```
FooProtocol DEFINITIONS ::= BEGIN

    FooQuestion ::= SEQUENCE {
        trackingNumber INTEGER(0..199),
        question       IA5String
    }

    FooAnswer ::= SEQUENCE {
        questionNumber INTEGER(10..20),
        answer         BOOLEAN
    }

    FooHistory ::= SEQUENCE {
        questions SEQUENCE(SIZE(0..10)) OF FooQuestion,
        answers   SEQUENCE(SIZE(1..10)) OF FooAnswer,
        anArray   SEQUENCE(SIZE(100))  OF INTEGER(0..1000),
        ...
    }
```

```
END
```

This change constrains trackingNumbers to have a value between 0 and 199 inclusive, and questionNumbers to have a value between 10 and 20 inclusive. The size of the questions array can be between 0 and 10 elements, with the answers array between 1 and 10 elements. The anArray field is a fixed length 100 element array of integers that must be in the range 0 to 1000. The '...' extensibility marker means that the FooHistory message specification may have additional fields in a future versions of the specification; systems compliant with one version should be able to receive and transmit transactions from a later version, though able to process only the fields specified in the earlier version. Good ASN.1 compilers will generate (in C, C++, Java, etc.) source code that will automatically check that transactions fall within these constraints. Transactions that violate the constraints should not be accepted from, or presented to, the application. Constraint management in this layer significantly simplifies protocol specification because the applications will be protected from constraint violations, reducing risk and cost.

To send the myQuestion message through the network, the message is serialized (encoded) as a series of bytes using one of the encoding rules. The Foo protocol specification should explicitly name one set of encoding rules to use, so that users of the Foo protocol know which one they should use and expect.

Example encoded in DER

Below is the data structure shown above encoded in DER format (all numbers are in hexadecimal):

```
30 13 02 01 05 16 0e 41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f
```

DER is a type-length-value encoding, so the sequence above can be interpreted, with reference to the standard SEQUENCE, INTEGER, and IA5String types, as follows:

```
30 – type tag indicating SEQUENCE
13 – length in octets of value that follows
  02 – type tag indicating INTEGER
  01 – length in octets of value that follows
    05 – value (5)
  16 – type tag indicating IA5String
     (IA5 means the full 7-bit ISO 646 set, including variants,
      but is generally US-ASCII)
  0e – length in octets of value that follows
    41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f – value ("Anybody there?")
```

Example encoded in XER

Alternatively, it is possible to encode the same ASN.1 data structure with XML Encoding Rules (XER) to achieve greater human readability "over the wire". It would then appear as the following 108 octets, (space count includes the spaces used for indentation):

```
<FooQuestion>
    <trackingNumber>5</trackingNumber>
    <question>Anybody there?</question>
</FooQuestion>
```

Example encoded in PER (unaligned)

Alternatively, if Packed Encoding Rules are employed, the following 122 bits (16 octets amount to 128 bits, but here only 122 bits carry information and the last 6 bits are merely padding) will be produced:

```
01 05 0e 83 bb ce 2d f9 3c a0 e9 a3 2f 2c af c0
```

In this format, type tags for the required elements are not encoded, so it cannot be parsed without knowing the expected schemas used to encode. Additionally, the bytes for the value of the IA5String are packed using 7-bit units instead of 8-bit units, because the encoder knows that encoding an IA5String byte value requires only 7 bits. However the length bytes are still encoded here, even for the first integer tag 01 (but a PER packer could also omit it if it knows that the allowed value range fits on 8 bits, and it could even compact the single value byte 05 with less than 8 bits, if it knows that allowed values can only fit in a smaller range).

Note also that the last 6 bits in the encoded PER are padded with null bits in the 6 least significant bits of the last byte c0 : these extra bits may not be transmitted or used for encoding something else if this sequence is inserted as a part of a longer unaligned PER sequence.

This means that unaligned PER data is essentially an ordered stream of bits, and not an ordered stream of bytes like with aligned PER, and that it will be a bit more complex to decode by software on usual processors because it will require additional contextual bit-shifting and masking and not direct byte addressing (but the same remark would be true with modern processors and memory/storage units whose minimum addressable unit is larger than 1 octet). However modern processors and signal processors include hardware support for fast internal decoding of bit streams with automatic handling of computing units that are crossing the boundaries of addressable storage units (this is needed for efficient processing in data codecs for compression/decompression or with some encryption/decryption algorithms).

If alignment on octet boundaries was required, an aligned PER encoder would produce:

```
01 05 0e 41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f
```

(in this case, each octet is padded individually with null bits on their unused most significant bits).

Tools

Most of the tools supporting ASN.1 do the following:

- parse the ASN.1 files,
- generates the equivalent declaration in a programming language (like C or C++),
- generates the encoding and decoding functions based on the previous declarations.

A list of tools supporting ASN.1 can be found on the ITU-T Tool web page (https://www.itu.int/en/ITU-T/asn1/Pages/Tools.aspx).

Comparison to similar schemes

ASN.1 is similar in purpose and use to protocol buffers and Apache Thrift, which are also interface description languages for cross-platform data serialization. Like those languages, it has a schema (in ASN.1, called a "module"), and a set of encodings, typically type-length-value encodings. However, ASN.1, defined in 1984, predates them by many years. It also includes a wider variety of basic data types, some of which are obsolete, and has more options for extensibility. A single ASN.1 message can include data from multiple modules defined in multiple standards, even standards defined years apart. ASN.1 also includes built-in support for constraints on values. For instance, a module can specify an integer field that must be in the range 0 to 100.

ASN.1 is visually similar to Augmented Backus-Naur form (ABNF), which is used to define many Internet protocols like HTTP and SMTP. However, in practice they are quite different: ASN.1 defines a data structure, which can be encoded in various ways (e.g. JSON, XML, binary). ABNF, on the other hand, defines the encoding ("syntax") at the same time it defines the data structure ("semantics"). ABNF tends to be used more frequently for defining textual, human-readable protocols, and generally is not used to define type-length-value encodings.

Many programming languages define language-specific serialization formats. For instance, Python's "pickle" module and Ruby's "Marshal" module. These formats are generally language specific. They also don't require a schema, which makes them easier to use in ad-hoc storage scenarios, but inappropriate for communications protocols.

JSON and XML similarly do not require a schema, making them easy to use. However, they are both cross-platform standards, and are broadly popular for communications protocols, particularly when combined with an XML schema or JSON schema.

For more detail, see Comparison of data serialization formats.

References

1. "ITU-T Recommendation database" (https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=6947). *ITU*. Retrieved 2017-03-06.
2. ITU-T X.680 - Specification of basic notation (https://www.itu.int/rec/T-REC-X.680-201508-I/en)
3. ITU-T website - Uses of ASN.1 (http://www.itu.int/ITU-T/asn1/uses/index.htm)
4. ITU-T X.690 - Basic Encoding Rules (BER) (https://www.itu.int/rec/T-REC-X.690-201508-I/en)
5. ITU-T X.690 - Distinguished Encoding Rules (DER) (https://www.itu.int/rec/T-REC-X.690-201508-I/en)
6. ITU-T X.690 - Canonical Encoding Rules (CER) (https://www.itu.int/rec/T-REC-X.690-201508-I/en)
7. ITU-T X.691 - Packed Encoding Rules (PER) (https://www.itu.int/rec/T-REC-X.691-201508-I/en)
8. ITU-T X.692 - Encoding Control Notation (ECN) (https://www.itu.int/rec/T-REC-X.692-201508-I/en)
9. ITU-T X.693 - XML Encoding Rules (XER) (https://www.itu.int/rec/T-REC-X.693/en)
10. ITU-T X.693 - XML Encoding Rules (CXER) (https://www.itu.int/rec/T-REC-X.693/en)
11. ITU-T X.693 - Extended XML Encoding Rules (E-XER) (https://www.itu.int/rec/T-REC-X.693/en)
12. ITU-T X.696 - Octet Encoding Rules (OER) (https://www.itu.int/rec/T-REC-X.696-201508-I/en)
13. ITU-T X.697 - JavaScript Object Notation Encoding Rules (JER) (https://www.itu.int/rec/T-REC-X.697/en)
14. IETF RFC 3641 - Generic String Encoding Rules (GSER) (https://tools.ietf.org/html/rfc3641)
15. "ITU-T work programme" (http://www.itu.int/itu-t/workprog/wp_item.aspx?isn=13581). Retrieved 2017-09-01.

External links

- A Layman's Guide to a Subset of ASN.1, BER, and DER (http://luca.ntop.org/Teaching/Appunti/asn1.html) A good introduction for beginners
- ITU-T website - Introduction to ASN.1 (http://www.itu.int/ITU-T/asn1/introduction/)
- A video introduction to ASN.1 (http://www.pragmadev.com/presentations.html)
- ASN.1 Tutorial (http://www.obj-sys.com/asn1tutorial/asn1only.html) Tutorial on basic ASN.1 concepts
- ASN.1 Tutorial (http://www.oss.com/asn1/resources/asn1-made-simple/introduction.html) Tutorial on ASN.1
- ASN.1 decoder (http://www.marben-products.com/asn.1/services/decoder-asn1.html) Allows decoding ASN.1 encoded messages into XML output.
- ASN.1 syntax checker and encoder/decoder (http://asn1-playground.oss.com/) Checks the syntax of an ASN.1 schema and encodes/decodes messages.
- ASN.1 encoder/decoder of 3GPP messages (http://3gpp-message-analyser.com/) Encodes/decodes ASN.1 3GPP messages and allows easy editing of these messages.
- Free books about ASN.1 (http://www.oss.com/asn1/resources/books-whitepapers-pubs/asn1-books.html)
- List of ASN.1 tools at IvmaiAsn project (http://ivmaiasn.sourceforge.net/asn1lnk.html)
- Overview of the Octet Encoding Rules (OER) (http://www.oss.com/asn1/resources/books-whitepapers-pubs/Overview%20of%20OER.pdf)
- Overview of the JSON Encoding Rules (JER) (http://www.oss.com/asn1/resources/asn1-papers/Overview%20of%20JER.pdf)