

Open Source ASN.1 Compiler

asn1c quick start sheet

The ASN.1 compiler is a tool for creating data encoders and decoders out of formal ASN.1 specifications. An ASN.1 abstract syntax may look like this:

```
TestModule DEFINITIONS ::= BEGIN           -- Module parameters preamble
    Circle ::= SEQUENCE {                  -- Definition of Circle type
        position-x INTEGER,                -- Integer position
        position-y INTEGER,                -- Position along Y-axis
        radius      INTEGER (0..MAX)       -- Positive radius
    }                                       -- End of Circle type
END                                         -- End of TestModule
```

The following examples assume the above ASN.1 text exists in a file named `TestModule.asn1`.

How to...

Q: Test the module for syntactic correctness?

A: Issue the following command:

```
asn1c -EF TestModule.asn1
```

This will instruct the `asn1c` compiler to read the ASN.1 syntax and perform the semantics checking on the module. If the syntax is correct, the module contents will be printed according to compiler's understanding.

Q: Create a C/C++ source code for BER and XER encoder and decoder of the Circle type?

A: Issue the following command:

```
asn1c TestModule.asn1
```

This will instruct the ASN.1 compiler to generate a set of C files which will contain the `Circle_t` structure definition, as well as the set of encoding and decoding instructions for that structure.

Q: Create a source code for PER decoder and encoder for the Circle type?

A: PER codec generation requires a special command line option:

```
asn1c -gen-PER TestModule.asn1
```

This option will instruct the `asn1c` compiler to produce the PER codec support as well as BER and XER codecs produced by default.

Q: Generate a working Circle decoder with minimum hassle?

A: After `asn1c` finishes, compile all generated files with PDU set to Circle:

```
cc -DPDU=Circle
-I. -o CircleDecoder.exe *.c
```

This will produce an executable which will be able to convert the Circle type between BER/DER/XER/PER formats.

```
graph TD; TE[Textual encodings] --> XER[XER XML Encoding Rules]; XER --> BASIC_XER[BASIC-XER]; XER --> CXER[CXER Canonical XER]; BOE[Byte-oriented encodings] --> BER[BER Basic Encoding Rules]; BER --> DER[DER Distinguished Encoding Rules]; BER --> CER[CER Canonical Encoding Rules]; BOE --> PER[PER Packed Encoding Rules]; PER --> ABP[Aligned Basic PER]; PER --> APC[Aligned Canonical PER]; ABP --> UBP[Unaligned Basic PER]; APC --> UPC[Unaligned Canonical PER];
```

The diagram illustrates the relationship between encoding types and specific encoding rules. It is organized into three columns based on the type of encoding: Textual encodings, Byte-oriented encodings, and Bit-oriented encodings.

- Textual encodings** (Yellow box) lead to **XER (XML Encoding Rules)** (Blue box). XER further branches into **BASIC-XER** and **CXER (Canonical XER)** (Blue boxes).
- Byte-oriented encodings** (Orange box) lead to **BER (Basic Encoding Rules)** (Green box). BER further branches into **DER (Distinguished Encoding Rules)** and **CER (Canonical Encoding Rules)** (Green boxes).
- Bit-oriented encodings** (Pink box) lead to **PER (Packed Encoding Rules)** (Pink box). PER further branches into **Aligned Basic PER** and **Aligned Canonical PER** (Pink boxes). These then branch into **Unaligned Basic PER** and **Unaligned Canonical PER** (Pink boxes).

Encoding variant	Compactness	Interoperability
XER (BASIC or CXER)	Not compact	Human readable UTF-8 XML subset
BER (DER or CER)	Very good	BER decoder can read DER and CER encoded data. Universal debuggers and decoders exist (unber and enber are part of asn1c distribution).
Aligned PER	Nearly best	PER stream decoding can only be done using a corresponding ASN.1 syntax. Unaligned/Aligned variants are incompatible. Basic PER decoder can read Canonical PER encoded data.
Unaligned PER	Best	

```
Circle_t *c;
/* Allocate a new Circle */
c = calloc(1, sizeof *c);
assert(c); /* Infinite memory! */
/* Fill in the data */
c->position_x = 123;
c->position_y = 321;
c->radius = 33;
/* Print out the structure in XER
xer_fprint(stdout, &asn_DEF_Circle
```

```
<Circle>
  <position-x>123</position-x>
  <position-y>321</position-y>
  <radius>33</radius>
</Circle>
```