NAMES: A..z, 0..9, – (hyphen), Case-sensitive.

AUTOMATIC TAGS: eliminate any potential issues with manual tag assignment.

MODULE: a collection of ASN.1 definition statements. Module body starts with BEGIN, ends with END.

IMPORTS: used when an ASN.1 module needs to use a definition from a different module.

TYPE NAMES: start with an upper case letter
<TypeName> ::= <TypeDefinition>

VALUE NAMES & FIELD NAMES: start with a lower case letter

OPTIONAL & DEFAULT: both indicate that the field is optional and need not be present in a SEQUENCE. The DEFAULT value is assumed when the field is absent.

VALUES: used mostly as examples and for testing. In real life data is assigned dynamically at runtime.

COMMENTS: start with -- (two hyphens) and end with -- or new line, or start with /* and end with */.

UNICODE: support for UTF8 strings.

XML: -- alternative way to assign values using XML notation.

ASN.1 is a notation used to describe data types. **ASN.1** also specifies the data encoding rules (serialization)

CONSTRAINTS: single value | value range | SIZE | permitted alphabet | PATTERN (regex) | duration.

EXTENSIBILITY: ensures new versions of the protocol will not be disruptive to existing implementations.

Version brackets (greyed out) can be optionally used to group extension items together.

ASN.1 & **XML**: ASN.1 can be used as your XML data schema definition.

INFORMATION OBJECTS & OPEN TYPES: allow complex restrictions for values to match entries in a reference set

ASN.1 provides an advanced way to define Item using Information Objects and Open Types as shown in the example here.

INFORMATION OBJECT CLASS: use of upper/lower case after & is semantically significant.

INFORMATION OBJECTS SET: each entry in Catalog becomes a "restriction" on what values Item can have.

Value of incorrectItem does not satisfy the constraints on its type defined in Catalog.

```
MyShop-Module1 { <oid> } -- oid - object identifier is optional
        DEFINITIONS
        AUTOMATIC TAGS ::=
BEGIN
        IMPORTS Item, Address FROM MyItems-Module2;

 PurchaseOrder ::= SEQUENCE {
   dateOfOrder       DATE,
   name              UTF8String (SIZE(3..50))  DEFAULT "N/A",
   address           Address,         -- imported
   phone             Phone OPTIONAL,  -- defined below
   items             ListOfItems      -- defined below via imported Item
 }

 -- types that are referenced by PurchaseOrder, but can also be used elsewhere
 ListOfItems   ::= SEQUENCE (SIZE (1..100)) OF Item -- an "array"
 Phone         ::= VisibleString (PATTERN "\d#3-\d#3-\d#4")

 -- examples of my values --
 myName UTF8String ::= "Ива́н Гро́зный"
 myPhone Phone     ::= "333-444-5555"
 myFavorite Item   ::= {id color:green, quantity crate, unitPrice 1.99}
 myAddr1 Address   ::= {street "1st Ave", city "Somerset", state "NJ", zip "08873"}
 myAddr2 ::= <Address> <street>2nd Ave</street> <city>Somerset</city>
                       <state>NJ</state> <zip>08873</zip>  </Address>
END

MyItems-Module2 DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
 -- By default, all defined types (Item and Address) are exported
 Item        ::= SEQUENCE {
   id              CHOICE {   -- id alternatives - code, url or color
                     code     INTEGER (0..MAX),
                     url      VisibleString,
                     color    ENUMERATED { black, blue, ... ,-- extended
                                           green, red}
                   } DEFAULT code:9999,
   quantity        INTEGER {single(1), dozen(12), crate(36)},
   options         BIT STRING DEFAULT '101100011'B,
   unitPrice       REAL ( 1.00 .. 9999.00 ),
                   ... , -- extension allowed below this line
   [[ isTaxable    BOOLEAN DEFAULT FALSE ]], -- added to Item in v.2
   [[ voltage      INTEGER (110 | 220) OPTIONAL ]]  -- added in v.3
 }

 Address     ::= SEQUENCE {
   street          VisibleString (SIZE (5 .. 50)),
   city            VisibleString (ALL EXCEPT "Springfield"),
   state           VisibleString (SIZE(2) ^ FROM ("A".."Z")),
   zip             NumericString (SIZE(5 | 9))
 }
END
```

```
PRODUCT ::= CLASS {          -- Information Object Class
    &id             INTEGER (1..MAX) UNIQUE,
 -- Open Type -- &Feature,  -- starts upper case, type varies per item
    &price          REAL
} WITH SYNTAX { &id, &Feature, &price }

Catalog PRODUCT ::= {                          -- Information Object Set
   --  id       Feature type       price       --
   {  101,   INTEGER (110 | 220), 20.00   } | -- Charger
   {  104,   NULL,                99.00   } | -- Glass Egg
   {  105,   Event,               9.99    }   -- $9.99 Basket
}
Event ::=   ENUMERATED {christmas, easter}

Item ::= SEQUENCE {
   ident        PRODUCT.&id ( {Catalog} ),
   feat         PRODUCT.&Feature ( {Catalog}{@ident} ),
   unitPrice    PRODUCT.&price ( {Catalog}{@ident} )
 }

correctItem   Item ::= {id 105, feat Event:easter,  unitPrice  9.99}
incorrectItem Item ::= {id 101, feat Event:easter,  unitPrice 99.00}
```

Obsolete, not commonly used or deprecated syntax is greyed out below

OBJECT IDENTIFIER VALUES

```
oid1 OBJECT IDENTIFIER ::= { iso standard 2345 modules (0) basic-types (1) }
oid2 OBJECT IDENTIFIER ::= { joint-iso-itu-t ds(5) }
oid3 OBJECT IDENTIFIER ::= { oid2 modules(0) }
oid4 OBJECT IDENTIFIER ::= { oid3 basic-types(1) }
oid5 OBJECT IDENTIFIER ::= { 2 5 0 1 } -- equals oid4
```

Object identifier value	Meaning
{ 1 2 }	ISO member bodies
{ 1 2 840 }	US (ANSI)
{ 1 2 840 113549 }	RSA Data Security, Inc.
{ 1 2 840 113549 1 }	RSA Data Security, Inc. PKCS
{ 2 5 }	directory services (X.500)
{ 2 5 8 }	directory services-algorithms

TYPES

Basic Types	Tag dec/hex	Other Types	Tag dec/hex
BOOLEAN	[01/01]	ObjectDescriptor	[07/07]
INTEGER	[02/02]		
BIT STRING	[03/03]	EXTERNAL	[08/08]
OCTET STRING	[04/04]	EMBEDDED PDV	[11/0B]
NULL	[05/05]		
OBJECT IDENTIFIER	[06/06]	OID-IRI	[35/ *]
RELATIVE-OID	[13/0D]	RELATIVE-OID-IRI	[36/ *]
REAL	[09/09]		
ENUMERATED	[10/0A]	SET	[17/11]
		SET OF	[17/11]
SEQUENCE	[16/10]		
SEQUENCE OF	[16/10]	UTCTime	[23/17]
CHOICE	----	GeneralizedTime	[24/18]
UTF8String	[12/0C]	PrintableString	[19/13]
NumericString	[18/12]	T61String	[20/14]
IA5String	[22/16]	VideotexString	[21/15]
VisibleString	[26/1A]	GraphicString	[25/19]
		GeneralString	[27/1B]
DATE	[31/ *]	UniversalString	[28/1C]
TIME-OF-DAY	[32/ *]	CHARACTER STRING	[29/1D]
DATE-TIME	[33/ *]	BMPString	[30/1E]
DURATION	[34/ *]	ISO646String	[26/1A]
		TeletexString	[20/14]

*occupies two octets

VALUES

Values are usually specified in ASN.1 modules only for indicating defaults, or ranges for constraining items (such as the maximum length of a name).

```
defaultOn BOOLEAN ::= TRUE
maxAge INTEGER ::= 120
bitmask BIT STRING ::= '7FFF'H
defaultBytes OCTET STRING ::= '010F'H
placeholder NULL ::= NULL
defaultID OBJECT IDENTIFIER ::= {joint-iso-itu-t country(16) us(840)}
defaultPrice REAL ::= 9.99

Item        ::= SEQUENCE {
   id             CHOICE {    -- id alternatives - code, url or color
                    code      INTEGER (0..MAX),
                    url       VisibleString,
                    color     ENUMERATED { black, blue, ... ,-- extended
                                           green, red}
                  } DEFAULT code:9999,
   quantity       INTEGER {single(1), dozen(12), crate(36)},
   options        BIT STRING DEFAULT '101100011'B,
   unitPrice      REAL ( 1.00 .. 9999.00 ),
                  ... , -- extension allowed below this line
   [[ isTaxable   BOOLEAN DEFAULT FALSE ]], -- added to Item in v.2
   [[ voltage     INTEGER (110 | 220) OPTIONAL ]]  -- added in v.3
 }
defaultItem Item ::= { -- This is a value for the type above
  id code : 1,
  quantity single,
  options '0'B
  unitPrice 1.99
}

ListOfNumbers ::= SEQUENCE OF INTEGER
firstPrimeNumbers ListOfNumbers ::= {1, 2, 3, 5, 7, 11, 13, 17}

name1 UTF8String ::= "Joe" -- can also hold international characters
phone NumericString ::= "8885551212"
text IA5String ::= "Arbitrary text - with punctuation, no problem."
name2 VisibleString ::= "Joe" -- US ASCII without control characters

myDay DATE ::= "2012-01-31"
noon TIME-OF-DAY ::= "12:00:00"
noonMyDay DATE-TIME ::= "2012-01-31T12:00:00"
lunchtime DURATION ::= "PT1H" -- one hour for lunch
```

Here is a common use for value notation for limiting a string size, especially if the same value will be used in multiple places:

```
upperSize INTEGER ::= 64
VisibleString (SIZE (0..upperSize) )
ItemList ::= SEQUENCE (SIZE(0..upperSize)) OF Item
```

INFORMATION OBJECTS

Use of upper/lower case after '&' is semantically significant.

```
MY-SIMPLE-CLASS ::= TYPE-IDENTIFIER


MY-CLASS ::= CLASS {
  &id                 OBJECT IDENTIFIER UNIQUE,
  &simple-value       ENUMERATED {high, low} DEFAULT low,
  &Set-of-values      INTEGER OPTIONAL,
  &Any-type,
  &an-inform-object   SOME-CLASS,
  &A-set-of-objects   SOME-OTHER-CLASS
} WITH SYNTAX
   {  KEY             &id
    [ URGENCY         &simple-value  ] -- Optional
    [ VALUE-RANGE     &Set-of-values ]
      PARAMETERS      &Any-type
      SYNTAX          &an-inform-object
      MATCHING-RULES &A-set-of-objects
   }

my-object  MY-CLASS ::= {
  KEY             { ..... }
  URGENCY         high
  VALUE-RANGE     { 1..10 | 20..30 }
  PARAMETERS      My-type
  SYNTAX          defined-syntax
  MATCHING-RULES { at-start | at-end | exact }
}

My-object-set   MY-CLASS ::= {
          object1 | object2 | object3,
          ...,
          version2-object
}

Message ::= SEQUENCE {
  -- Has to be an OBJECT IDENTIFIER (KEY) from the set:
  key  MY-CLASS.&id ({My-object-set}),
  -- Has to be the PARAMETERS for the object with KEY:
  parms MY-CLASS.&Any-type ( {My-object-set} {@key} )
}
```

Variable type value fields and value set fields are out of the scope of this reference card

PARAMETERIZATION

All assignments defining reference names (type, value, class definitions, object definitions, object set) can be given a dummy parameter list. Here we have two dummy parameters – normal-priority and Parameter.

```
Invoke-message {INTEGER:normal-priority, Parameter} ::=
  SEQUENCE {
    component1 INTEGER DEFAULT normal-priority,
    component2 Parameter }
```

Now we define our messages as a choice of two possibilities that differ only in the default priority and the Type that is to be used:

```
Messages ::= CHOICE {
  first  Invoke-message { low-priority,  Type1 },
  second Invoke-message { high-priority, Type2 },
  ... }

Messages ::= CHOICE { -- This is what the above expands to
  first  SEQUENCE {
    component1 INTEGER DEFAULT low-priority,
    component2 Type1 },
  second SEQUENCE {
    component1 INTEGER DEFAULT high-priority,
    component2 Type2 },
  ... }
```

ENCODINGS

Bit-wide	**PER**: A compact binary encoding transferring the minimum information needed to identify a value.
Byte-wide	**BER**: A type-length-value (TLV) style of encoding
	DER: An encoding with only one way to encode a given value, used in security work.
	CER: Another security-related encoding, rarely used.
XML	**XER**: Encoding ASN.1 values as XML syntax.
	There are also Encoding Instructions that can vary XER and other encodings, for example, to determine which components of a sequence are to be encoded as XML attributes.
ECN	An encoding control notation (**ECN**) is available to completely determine the encoding of ASN.1 values.