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

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

MODULE: starts with BEGIN, ends with END

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

TYPES: start with upper case letter
<TypeName> ::= <TypeDefinition>

FIELDS & VALUES: start with lower case letter
<valueName> <TypeName> ::= <value notation>

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

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

COMMENTS: -- delimited by pairs of hyphens --

UNICODE: support for UTF8 strings

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

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

CONSTRAINTS: value | size | range | PATTERN (regex) | MIN, MAX | ALL EXCEPT

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

Optional items and stylistic advice are grayed-out

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

An advanced way to define Item using Information Objects and Open Types (some fields of the above Item are left out for clarity).

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

INFORMATION OBJECTS SET: each entry in the set Catalog becomes a "restriction" for Items defined below.

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 ::= "Иван Грозный"
myPhone Phone     ::= "333-444-5555"
myFavorite Item   ::= {id color:green, quantity crate, unitPrice 1.99}
myAddr1 Address   ::= {street "1st Ave", city "Somerset", state "NY", 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
        EXPORTS Item, Address; -- exporting is optional
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
  [[2: isTaxable BOOLEAN DEFAULT FALSE ]], -- added to Item in v.2
  [[3: 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 Objects 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 Objects 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 {
   iid          PRODUCT.&id ( {Catalog} ),
   feat         PRODUCT.&Feature ( {Catalog}{@iid} ),
   unitPrice    PRODUCT.&price ( {Catalog}{@iid} )
 }

correctItem   Item ::= {iid 105, feat Event:easter,  unitPrice  9.99}
incorrectItem Item ::= {iid 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
```

{ 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]
OBJECT IDENTIFIER	[06/06]		
REAL	[09/09]	RELATIVE-OID	[13/0D]
ENUMERATED	[10/0A]	SET	[17/11]
SEQUENCE	[16/10]	SET OF	[17/11]
SEQUENCE OF	[16/10]		
CHOICE	----	UTCTime	[23/17]
		GeneralizedTime	[24/18]
UTF8String	[12/0C]		
NumericString	[18/12]	PrintableString	[19/13]
IA5String	[22/16]	T61String	[20/14]
VisibleString	[36/1A]	VideotexString	[21/15]
		GraphicString	[25/19]
DATE	[31/ *]	GeneralString	[27/1B]
TIME-OF-DAY	[32/ *]	UniversalString	[28/1C]
DATE-TIME	[33/ *]	CHARACTER STRING	[29/1D]
		BMPString	[30/1E]
NULL	[05/05]	ISO646String	[26/1A]
		TeletexString	[20/14]

*occupies two octets

OTHER SYNTAX

```
EXPORTS ALL

MODULE-NAME.TypeName  -- historical alternative to IMPORTS
ABSTRACT-SYNTAX       -- rarely used
IMPLICIT TAGS         -- historical, but often seen
EXPLICIT TAGS         -- historical, but rarely used

SEQUENCE {            -- an illustration of tags
   first  [0]  INTEGER OPTIONAL,
   second [1]  EXPLICIT INTEGER,
   last   [99] IMPLICIT UserData
}
[APPLICATION 29], [PRIVATE 6] -- More tag illustrations

EXTENSIBILITY IMPLIED         -- almost never used
selection < ChoiceType        -- occasionally used, not important

COMPONENTS OF SequenceType    -- used to insert common components
SEQUENCE { ......., ... !29 } -- An exception marker
INSTANCE OF                   -- Another mechanism for embedding material

-- Complex constraints being applied, and also a Value Set Assignment.
-- My-values INTEGER ::= { ..... } is equivalent to
-- My-values ::= INTEGER ( ..... ).
My-values INTEGER ::=
   { VSet1 INTERSECTION (VSet2 UNION VSet3) EXCEPT VSet4}

PrintableString (SIZE (NameSizes) ) -- NameSizes is defined below
NameSizes ::= INTEGER (0..64)


RELATIVE-OID    -- A type which carries the tail-end of an object
                -- identifier value, with the root statically determined.  Some
                -- Sometimes misused to provide an efficient encoding
                -- in BER of SEQUENCE OF INTEGER.

EMBEDDED PDV    -- Used to embed messages from other specifications,
                -- with both the message and the encoding identified
                -- at communication time.
EXTERNAL        -- Historical (earlier version of EMBEDDED PDV).
```

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
  -- LITERALS are optional, commas can be used as separators
   {  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 OBJECT-ID 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 – an INTEGER 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 },
  ... }
```

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

T=1 byte	L= 2..2+N bytes	V= 2+N..2+N+len bytes

7	6	5	4	3	2	1	0
00-universal 01-application 10-context 11-private		0-primitive 1-constructed	[01]..[1E] Type tag				

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.