A Summary of ASN.1 Types and their Usage

Abstract: The basics of how to design ASN.1 code is shown through a practical example.

By Jacob Palme (**http://www.palme.nu/jacob/**)

This document is also available in HTML format at URL

http://dsv.su.se/jpalme/internet-course/solving-asn-1-exercise.html

ASN.1 type	Usage
ANY	Data, whose format is to be specified in the future or by someone else. Example: **FutureData ::= SEQUENCE { type VisibleString, value ANY }**
BitString	A string of Boolean values. Example: **DaysOpen ::= BitString { monday (0), tuesday (2), wednesday (3), thursday (4), friday (5), saturday (6), sunday (7) }**
Boolean	A single Boolean (true/false, or 1/0) value. Example: **Gender ::= BOOLEAN -- Male=true, Female=false**
CharacterString: **NumericString** **PrintableString** **TeletexString** **VideotexString** **VisibleString** **IA5String** **GraphicString** **GeneralString** **UniversalString**	Character strings using different sets of allowed characters. Example: **Surname ::= VisibleString**
CHOICE	One of a list of different types, example: **CHOICE { car Motorcar, bike Bicycle, boat Boat }** or, tags needed since all elements must be of different type: **CHOICE { registrationnumber [1] VisibleString, name [2] VisibleString }**
ENUMERATED	Can have any of a limited set of enumerated values. Example: **Weekday ::= ENUMERATED { monday (0), tuesday (2), wednesday (3), thursday (4), friday (5), saturday (6), sunday (7) }**
INTEGER	An integer value, example: **Age ::= INTEGER (0 .. MAX)**
OCTET STRING	A string of octets, whose data is not specified in ASN.1. Example: **GIF-Picture ::= OCTET STRING**

REAL	A real value, example: **Windvelocity ::= REAL**
SEQUENCE	Several different types of data in sequence. Example: **Name ::= SEQUENCE { Givenname VisibleString, Surname VisibleString }**
SEQUENCE OF	A list of one or more items of the same type. Example: **Family ::= SEQUENCE OF Name**
SET	Same as **SEQUENCE**, but no assumed order.
SET OF	Same as **SEQUENCE OF**, but no assumed order.

Example of how you can Think when Solving an ASN.1 Exam Question

All page references are to pages in the book ASN.1 The Tutorial & Reference by Douglas Steedman.

Question 2 in the exam 1999-11-09

Below is a specification of a proposed addition to Internet e-mail. The specification is based on ABNF.

Write a specificaction which will convey the same information using ASN.1. You need only translate the syntax (down to "**Note:**") not the explanatory text which comes after "**Note:**".

Note: Your solution need only transfer the information, not the syntactical form.

Supersedes

Syntax

```
    Supersedes-field          = "Supersedes:" " " identifier
                                *(identifier)
                                optional-parameter-list
                                CRLF

    optional-parameter-list   = *( ";" " " parameter )

    parameter                 = parameter-name [ "="
                                parameter-value ]

    parameter-name            = "noshow" / "show" / "repost"
                                private-parameter /
                                future-parameter
```

Note: There is no comma between multiple values, and that each Message-ID value is to be surrounded by angle brackets.

Warning: Some software may not work correctly with comments in header fields, especially comments in other places than at the beginning and end of the field value.

Warning: This header MUST be spelled "Supersedes" and not "Supercedes".

Semantics

The Supersedes header identifies previous correspondence, which this message supersedes. Different messaging agents such as user agents, mailing list expanders and mailing list archives. A user agent is expected to handle this field in much the same way as the In-Reply-To and References header.

Note: The Message-ID of a superseding message MUST be different from the Message-ID of the superseded message. The Message-ID of the superseded message is used as value in the "Supersedes:" header, not in the Message-ID of the superseding message.

Parameters:

noshow	In the opinion of the sender, this message makes such a minor change to the superseded version, that a recipient, who has already seen the previous verson, will probably not want to see the new version, unless the user explicitly asks for it.
show	In the opinion of the sender, this message makes such a large change to the superseded version, that a recipient, who has already seen the previous version, will probably want to see the new version, too.
repost	This document is a document which is repeatedly, at regular or irregular intervals, reposted, such as FAQs or mailing list monthly information.

None of these parameters have values. The "noshow" and the "show" parameters are mutually exclusive, but both of them can occur together with the "repost" parameter.

How to solve this exam question

First analyse what information is sent from with this protocol element. The information sent is:

1. That this is a Supersedes header field.
2. A list of one or more identifiers of superseded messages.
3. A list of one or more optional parameters.
4. Each optional parameter can have a name, and an optional value.
5. The parameters noshow, show and repost are specified, additional private or future parameters can be added.

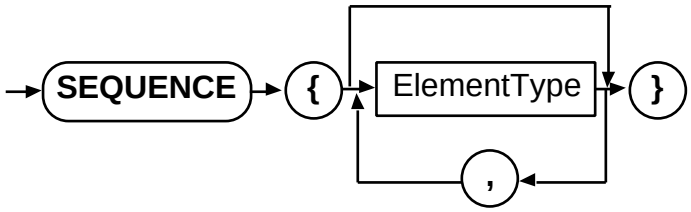
Note that the ":", ";", "=" are part of the ABNF syntactical structure, and should not be sent, since ASN.1 has its own, alternative methods of syntactically structuring the information sent.

How is this information formatted with ASN.1?

1. That this is a Supersedes header field

Presumably, there is a list of header fields, of which this is one. To create a list of elements of the same type, you use **SEQUENCE OF**.

Look at the syntax for **SEQUENCE OF** on page 141:



Using this syntax, you can construct it as follows:

```
HeaderFields ::= SEQUENCE OF Header
```

Each header contains information on which header it is, and the data for this header. Thus, we need two main parts of each header, name and value, so a **SEQUENCE** might be used. See the syntax for **SEQUENCE** on page 141:

```
Header ::= SEQUENCE {   headername,          -- Not complete yet
                        headerdata }
```

"**headername**" and "**headerdata**" have the syntax for ElementType, which you can find on page 795 of Compendium 1. As you can see there, it is a NamedType, possibly followed by **OPTIONAL** or **DEFAULT**.

And the syntax of NamedType can be found on page 139 as an identifer and a type:

```
Header ::= SEQUENCE {   headername Headernametype,
                        headerdata Headerdatatype }
```

Headernametype should tell which of a number of known headers this is. Since there are, presumably, a limited number of known headers, **ENUMERATED** is suitable. Thus, we could specify **Headernametype** as:

```
Headernametype ::= ENUMERATED {     From (0), To (1), Cc (2), Date (3),
                                    Supersedes (4) }
```

There are probably more values, but that is not part of this question.

The syntax for **ENUMERATED** can be found on page 135 of Compendium 1. It uses Namednumber, whose syntax can be found on page 139.

Another alternative would be to use a text string:

```
Headernametype ::= VisibleString
```

2. A list of one or more identifiers of superseded messages,

3. A list of one or more optional parameters

Headerdata consists of two main groups of data in sequence, so we use the **SEQUENCE** type:

```
Headerdatatype ::= SEQUENCE {   identifierlist Identifierlisttype,
                                parameterlist Parameterlisttype OPTIONAL }
```

The identifierlist consists of one or more identifiers, which are text strings:

```
Identifierlisttype ::= SEQUENCE OF VisibleString
```

4. A list of one or more optional parameters

5. The parameters noshow, show and repost are specified, additional private or future parameters can be added

Since this is a list of one or more parameters, we use **SEQUENCE OF**:

```
Parameterlisttype ::= SEQUENCE OF Parameter
```

Each Parameter is either a built in parameter, a private parameter or a future parameter. We use a **CHOICE**:

```
Parameter ::= CHOICE {   builtinparameter      Builtinparametetertype,
                         privateparameter      [1] Newparametertype,
                         futureparameter       [2] Newparametertype }
```

The tags are necessary, since no two elements in a **CHOICE** can have the same type.

The Builtinparametertype is an indication of one of three possible values, thus **ENUMERATED** is suitable:

```
Builtinparametertype ::= ENUMERATED { noshow (0), show (1), repost (3) }
```

The private and future parameters have a name and an optional value:

```
Newparametertype ::= SEQUENCE { name VisibleString,
                                value ANY OPTIONAL }
```

ANY is a placeholder where you can put any kind of data. Since this data is in text string format in ABNF, we might use for example **VisibleString** instead of **ANY** above.

We are ready

So now we are ready. Just collect the ASN.1 together:

```
HeaderFields ::= SEQUENCE OF Header

Header ::= SEQUENCE {   headername Headernametype,
                        headerdata Headerdatatype }

Headernametype ::= VisibleString

Headerdatatype ::= SEQUENCE {  identifierlist  Identifierlisttype,
                               parameterlistParameterlisttype OPTIONAL }

Identifierlisttype ::= SEQUENCE OF VisibleString

Parameterlisttype ::= SEQUENCE OF Parameter

Parameter ::= CHOICE {   builtinparameter      Builtinparametetertype,
                         privateparameter      [1] Newparametertype,
                         futureparameter       [2] Newparametertype }

Builtinparametertype ::= ENUMERATED { noshow (0), show(1), repost(3) }

Newparametertype ::= SEQUENCE { name VisibleString,
                                value ANY OPTIONAL }
```