Network Working Group
Request for Comments: 3641
Category: Standards Track

S. Legg
Adacel Technologies

October 2003

Generic String Encoding Rules (GSER) for ASN.1 Types

Status of this Memo

Copyright Notice

Abstract

This document defines a set of Abstract Syntax Notation One (ASN.1) encoding rules, called the Generic String Encoding Rules (GSER), that produce a human readable text encoding for values of any given ASN.1 data type.

Table of Contents

1. Introduction

This document defines a set of ASN.1 [8] encoding rules, called the Generic String Encoding Rules or GSER, that produce a human readable UTF-8 [6] character string encoding of ASN.1 values of any given arbitrary ASN.1 type.

Note that "ASN.1 value" does not mean a Basic Encoding Rules (BER) [12] encoded value. The ASN.1 value is an abstract concept that is independent of any particular encoding. BER is just one possible encoding of an ASN.1 value.

GSER is based on ASN.1 value notation [8], with changes to accommodate the notation's use as a transfer syntax, and to support well established ad-hoc string encodings for Lightweight Directory Access Protocol (LDAP) [14] directory data types.

Though primarily intended for defining the LDAP-specific encoding of new LDAP attribute syntaxes and assertion syntaxes, these encoding rules could also be used in other domains where human readable renderings of ASN.1 values would be useful.

Referencing GSER is sufficient to define a human readable text encoding for values of a specific ASN.1 type, however other specifications may wish to provide a customized Augmented Backus-Naur Form (ABNF) [3] description, independent of the ASN.1, as a convenience for the implementor (equivalent ABNF for the GSER encodings for ASN.1 types commonly occurring in LDAP syntaxes is provided in a separate document [15]). Such a specification SHOULD state that if there is a discrepancy between the customized ABNF and the GSER encoding defined by this document, that the GSER encoding takes precedence.

2. Conventions

Throughout this document, "type" shall be taken to mean an ASN.1 type, and "value" shall be taken to mean an ASN.1 value.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [1].

3. Generic String Encoding Rules

The GSER encoding of a value of any ASN.1 type is described by the following ABNF [3]:

```
Value = BitStringValue /
        BooleanValue /
        CharacterStringValue /
        ChoiceValue /
        EmbeddedPDVValue /
        EnumeratedValue /
        ExternalValue /
        GeneralizedTimeValue /
        IntegerValue /
        InstanceOfValue /
        NullValue /
        ObjectDescriptorValue /
        ObjectIdentifierValue /
        OctetStringValue /
        RealValue /
        RelativeOIDValue /
        SequenceOfValue /
        SequenceValue /
        SetOfValue /
        SetValue /
        StringValue /
        UTCTimeValue /
        VariantEncoding
```

The ABNF for each of the above rules is given in the following sections.

3.1 Type Referencing Notations

A value of a type with a defined type name is encoded according to the type definition on the right hand side of the type assignment for the type name.

A value of a type denoted by the use of a parameterized type with actual parameters is encoded according to the parameterized type with the DummyReferences [11] substituted with the actual parameters.

A value of a tagged or constrained type is encoded as a value of the type without the tag or constraint, respectively. Tags do not appear in the string encodings defined by this document. See X.680 [8] and X.682 [10] for the details of ASN.1 constraint notation.

A value of an open type denoted by an ObjectClassFieldType (Clause 14 of X.681 [9]) is encoded according to the specific type of the value.

A value of a fixed type denoted by an ObjectClassFieldType is encoded according to that fixed type.

A value of a selection type is encoded according to the type referenced by the selection type.

A value of a type described by TypeFromObject notation (Clause 15 of X.681 [9]) is encoded according to the denoted type.

A value of a type described by ValueSetFromObjects notation (Clause 15 of X.681 [9]) is encoded according to the governing type.

3.2. Restricted Character String Types

The contents of a string value are encoded as a UTF-8 character string between double quotes, regardless of the ASN.1 string type. Depending on the ASN.1 string type and an application's internal representation of that string type, a translation to or from the UTF-8 character encoding may be required. NumericString, PrintableString, IA5String, and VisibleString (ISO646String) are compatible with UTF-8 and do not require any translation. BMPString (UCS-2) and UniversalString (UCS-4) have a direct mapping to and from UTF-8 [6]. For the remaining string types see X.680 [8]. Any embedded double quotes in the resulting UTF-8 character string are escaped by repeating the double quote characters.

A value of the NumericString, PrintableString, TeletexString (T61String), VideotexString, IA5String, GraphicString, VisibleString (ISO646String), GeneralString, BMPString, UniversalString or UTF8String type is encoded according to the <StringValue> rule.

```
StringValue       = dquote *SafeUTF8Character dquote

dquote            = %x22 ; " (double quote)

SafeUTF8Character = %x00-21 / %x23-7F /   ; ASCII minus dquote
                    dquote dquote /       ; escaped double quote
                    %xC0-DF %x80-BF /     ; 2 byte UTF-8 character
                    %xE0-EF 2(%x80-BF) /  ; 3 byte UTF-8 character
                    %xF0-F7 3(%x80-BF)    ; 4 byte UTF-8 character
```

A value of the GeneralizedTime type, UTCTime type or ObjectDescriptor type is encoded as a string value. GeneralizedTime and UTCTime use the VisibleString character set so the conversion to UTF-8 is trivial. ObjectDescriptor uses the GraphicString type.

```
GeneralizedTimeValue  = StringValue
UTCTimeValue          = StringValue
ObjectDescriptorValue = StringValue
```

3.3. ChoiceOfStrings Types

It is not uncommon for ASN.1 specifications to define types that offer a CHOICE between two or more alternative ASN.1 string types, where the particular alternative chosen carries no semantic significance (DirectoryString [7] being a prime example). Such types are defined to avoid having to use a complicated character encoding for all values when most values could use a simpler string type, or to deal with evolving requirements that compel the use of a broader character set while still maintaining backward compatibility.

GSER encodes values of all the ASN.1 string types as UTF-8 character strings so the particular alternative that is chosen from a purely syntactic CHOICE of string types makes no material difference to the final encoding of the string value.

While there are certain ASN.1 constructs that betray the semantic significance of the alternatives within a CHOICE type, the absence of those constructs does not necessarily mean that a CHOICE type is purely syntactic. Therefore, it is necessary for specifications to declare the purely syntactic CHOICE types so that they may be more compactly encoded (see Section 3.12). These declared CHOICE types are referred to as ChoiceOfStrings types.

To be eligible to be declared a ChoiceOfStrings type, an ASN.1 type MUST satisfy the following conditions.

a) The type is a CHOICE type.

b) The component type of each alternative is one of the following ASN.1 restricted string types: NumericString, PrintableString, TeletexString (T61String), VideotexString, IA5String, GraphicString, VisibleString (ISO646String), GeneralString, BMPString, UniversalString or UTF8String.

c) All the alternatives are of different restricted string types, i.e., no two alternatives have the same ASN.1 restricted string type.

d) Either none of the alternatives has a constraint, or all of the alternatives have exactly the same constraint.

Tagging on the alternative types is ignored.

Consider the ASN.1 parameterized type definition of DirectoryString.

```
DirectoryString { INTEGER : maxSize } ::= CHOICE {
    teletexString     TeletexString (SIZE (1..maxSize)),
    printableString   PrintableString (SIZE (1..maxSize)),
    bmpString         BMPString (SIZE (1..maxSize)),
    universalString   UniversalString (SIZE (1..maxSize)),
    uTF8String        UTF8String (SIZE (1..maxSize)) }
```

Any use of the DirectoryString parameterized type with an actual parameter defines an ASN.1 type that satisfies the above conditions. Recognising that the alternative within a DirectoryString carries no semantic significance, this document declares (each and every use of) DirectoryString{} to be a ChoiceOfStrings type.

Other specifications MAY declare other types satisfying the above conditions to be ChoiceOfStrings types. The declaration SHOULD be made at the point where the ASN.1 type is defined, otherwise it SHOULD be made at the point where it is introduced as, or in, an LDAP attribute or assertion syntax.

3.4. Identifiers

An <identifier> conforms to the definition of an identifier in ASN.1 notation (Clause 11.3 of X.680 [8]). It begins with a lowercase letter and is followed by zero or more letters, digits, and hyphens. A hyphen is not permitted to be the last character, nor is it to be followed by another hyphen. The case of letters in an identifier is always significant.

```
identifier    = lowercase *alphanumeric *(hyphen 1*alphanumeric)
alphanumeric  = uppercase / lowercase / decimal-digit
uppercase     = %x41-5A  ; "A" to "Z"
lowercase     = %x61-7A  ; "a" to "z"
decimal-digit = %x30-39  ; "0" to "9"
hyphen        = "-"
```

3.5. BIT STRING

A value of the BIT STRING type is encoded according to the <BitStringValue> rule. If the definition of the BIT STRING type includes a named bit list, the <bit-list> form of <BitStringValue> MAY be used. If the number of bits in a BIT STRING value is a multiple of four, the <hstring> form of <BitStringValue> MAY be used. Otherwise, the <bstring> form of <BitStringValue> is used.

```
BitStringValue = bstring / hstring / bit-list
```

The <bit-list> rule encodes the one bits in the bit string value as a comma separated list of identifiers. Each <identifier> MUST be one of the identifiers in the named bit list, and MUST NOT appear more than once in the same <bit-list>. The <bstring> rule encodes each bit as the character "0" or "1" in order from the first bit to the last bit. The <hstring> rule encodes each group of four bits as a hexadecimal number where the first bit is the most significant. An odd number of hexadecimal digits is permitted.

```
bit-list          = "{" [ sp identifier
                       *( "," sp identifier ) ] sp "}"

hstring           = squote *hexadecimal-digit squote %x48 ; '...'H

hexadecimal-digit = %x30-39 /  ; "0" to "9"
                    %x41-46    ; "A" to "F"

bstring           = squote *binary-digit squote %x42  ; '...'B
binary-digit      = "0" / "1"

sp                = *%x20  ; zero, one or more space characters
squote            =  %x27  ; ' (single quote)
```

3.6. BOOLEAN

A value of the BOOLEAN type is encoded according to the <BooleanValue> rule.

```
BooleanValue = %x54.52.55.45 /   ; "TRUE"
               %x46.41.4C.53.45  ; "FALSE"
```

3.7. ENUMERATED

A value of the ENUMERATED type is encoded according to the <EnumeratedValue> rule. The <identifier> MUST be one of those in the list of enumerations in the definition of the ENUMERATED type.

```
EnumeratedValue = identifier
```

3.8. INTEGER

A value of the INTEGER type is encoded according to the <IntegerValue> rule. If the definition of the INTEGER type includes a named number list, the <identifier> form of <IntegerValue> MAY be used, in which case the <identifier> MUST be one of the identifiers in the named number list.

```
IntegerValue    = "0" /
                  positive-number /
                  ("-" positive-number) /
                  identifier

positive-number = non-zero-digit *decimal-digit
non-zero-digit  = %x31-39  ; "1" to "9"
```

3.9. NULL

A value of the NULL type is encoded according to the <NullValue> rule.

```
NullValue = %x4E.55.4C.4C  ; "NULL"
```

3.10. OBJECT IDENTIFIER and RELATIVE-OID

A value of the OBJECT IDENTIFIER type is encoded according to the <ObjectIdentifierValue> rule. The <ObjectIdentifierValue> rule allows either a dotted decimal representation of the OBJECT IDENTIFIER value or an object descriptor name, i.e., <descr>. The <descr> rule is described in RFC 2252 [4]. An object descriptor name is potentially ambiguous and should be used with care.

```
ObjectIdentifierValue = numeric-oid / descr
numeric-oid           = oid-component 1*( "." oid-component )
oid-component         = "0" / positive-number
```

A value of the RELATIVE-OID type is encoded according to the <RelativeOIDValue> rule.

```
RelativeOIDValue = oid-component *( "." oid-component )
```

3.11. OCTET STRING

A value of the OCTET STRING type is encoded according to the <OctetStringValue> rule. The octets are encoded in order from the first octet to the last octet. Each octet is encoded as a pair of hexadecimal digits where the first digit corresponds to the four most significant bits of the octet. If the hexadecimal string does not have an even number of digits, the four least significant bits in the last octet are assumed to be zero.

```
OctetStringValue = hstring
```

3.12. CHOICE

A value of a CHOICE type is encoded according to the <ChoiceValue> rule. The <ChoiceOfStringsValue> encoding MAY be used if the corresponding CHOICE type has been declared a ChoiceOfStrings type. This document declares DirectoryString to be a ChoiceOfStrings type (see Section 3.3). Otherwise, the <IdentifiedChoiceValue> form of <ChoiceValue> is used.

```
ChoiceValue           = IdentifiedChoiceValue /
                        ChoiceOfStringsValue
IdentifiedChoiceValue = identifier ":" Value
ChoiceOfStringsValue  = StringValue
```

For implementations that recognise the internal structure of the DirectoryString CHOICE type (e.g., X.500 directories [16]), if the character string between the quotes in a <StringValue> contains only characters that are permitted in a PrintableString, the DirectoryString is assumed to use the printableString alternative, otherwise it is assumed to use the uTF8String alternative. The <IdentifiedChoiceValue> rule MAY be used for a value of type DirectoryString to indicate an alternative other than the one that would be assumed from the string contents. No matter what alternative is chosen, the <Value> will still be a UTF-8 encoded character string. However, it is a syntax error if the characters in the UTF-8 string cannot be represented in the string type of the chosen alternative.

Implementations that do not care about the internal structure of a DirectoryString value MUST be able to parse the <IdentifiedChoiceValue> form for a DirectoryString value, though the particular identifier found will be of no interest.

3.13. SEQUENCE and SET

A value of a SEQUENCE type is encoded according to the <SequenceValue> rule. The <ComponentList> rule encodes a comma separated list of the particular component values present in the SEQUENCE value, where each component value is preceded by the corresponding identifier from the SEQUENCE type definition. The components are encoded in the order of their definition in the SEQUENCE type.

```
SequenceValue = ComponentList

ComponentList = "{" [ sp NamedValue *( "," sp NamedValue) ] sp "}"
NamedValue    = identifier msp Value
msp           = 1*%x20  ; one or more space characters
```

A value of a SET type is encoded according to the <SetValue> rule. The components are encoded in the order of their definition in the SET type (i.e., just like a SEQUENCE value). This is a deliberate departure from ASN.1 value notation where the components of a SET can be written in any order.

```
SetValue = ComponentList
```

SEQUENCE and SET type definitions are sometimes extended by the inclusion of additional component types, so an implementation SHOULD be capable of skipping over any <NamedValue> encoding with an identifier that is not recognised, on the assumption that the sender is using a more recent definition of the SEQUENCE or SET type.

3.14. SEQUENCE OF and SET OF

A value of a SEQUENCE OF type is encoded according to the <SequenceOfValue> rule, as a comma separated list of the instances in the value. Each instance is encoded according to the component type of the SEQUENCE OF type.

```
SequenceOfValue = "{" [ sp Value *( "," sp Value) ] sp "}"
```

A value of a SET OF type is encoded according to the <SetOfValue> rule, as a list of the instances in the value. Each instance is encoded according to the component type of the SET OF type.

```
SetOfValue      = "{" [ sp Value *( "," sp Value) ] sp "}"
```

3.15. CHARACTER STRING

A value of the unrestricted CHARACTER STRING type is encoded according to the corresponding SEQUENCE type defined in Clause 40.5 of X.680 [8] (see [15] for equivalent ABNF).

```
CharacterStringValue = SequenceValue
```

3.16. EMBEDDED PDV

A value of the EMBEDDED PDV type is encoded according to the corresponding SEQUENCE type defined in Clause 33.5 of X.680 [8] (see [15] for equivalent ABNF).

```
EmbeddedPDVValue = SequenceValue
```

3.17. EXTERNAL

A value of the EXTERNAL type is encoded according to the corresponding SEQUENCE type defined in Clause 8.18.1 of X.690 [12] (see [15] for equivalent ABNF).

```
ExternalValue = SequenceValue
```

3.18. INSTANCE OF

A value of the INSTANCE OF type is encoded according to the corresponding SEQUENCE type defined in Annex C of X.681 [9].

```
InstanceOfValue = SequenceValue
```

3.19. REAL

A value of the REAL type MUST be encoded as "0" if it is zero, otherwise it is encoded as the special value <PLUS-INFINITY>, the special value <MINUS-INFINITY>, an optionally signed <realnumber>, or as a value of the corresponding SEQUENCE type for REAL defined in Clause 20.5 of X.680 [8] (see [15] for equivalent ABNF).

```
RealValue  = "0"               ; zero REAL value
           / PLUS-INFINITY     ; positive infinity
           / MINUS-INFINITY    ; negative infinity
           / realnumber        ; positive base 10 REAL value
           / "-" realnumber    ; negative base 10 REAL value
           / SequenceValue     ; non-zero REAL value, base 2 or 10
```

```
realnumber = mantissa exponent
mantissa   = (positive-number [ "." *decimal-digit ])
             / ( "0." *("0") positive-number )
exponent   = "E" ( "0" / ([ "-" ] positive-number))

PLUS-INFINITY  = %x50.4C.55.53.2D.49.4E.46.49.4E.49.54.59
                    ; "PLUS-INFINITY"
MINUS-INFINITY = %x4D.49.4E.55.53.2D.49.4E.46.49.4E.49.54.59
                    ; "MINUS-INFINITY"
```

3.20. Variant Encodings

The values of some named complex ASN.1 types have special string encodings. These special encodings are always used instead of the encoding that would otherwise apply based on the ASN.1 type definition.

```
VariantEncoding = RDNSequenceValue /
                  RelativeDistinguishedNameValue /
                  ORAddressValue
```

A value of the RDNSequence type, i.e., a distinguished name, is encoded according to the <RDNSequenceValue> rule, as a quoted LDAPDN character string. The character string is first derived according to the <distinguishedName> rule in Section 3 of RFC 2253 [5], and then encoded as if it were a UTF8String value, i.e., between double quotes with any embedded double quotes escaped by being repeated.

```
RDNSequenceValue = StringValue
```

A RelativeDistinguishedName value that is not part of an RDNSequence value is encoded according to the <RelativeDistinguishedNameValue> rule as a quoted character string. The character string is first derived according to the <name-component> rule in Section 3 of RFC 2253 [5], and then encoded as if it were a UTF8String value.

```
RelativeDistinguishedNameValue = StringValue
```

A value of the ORAddress type is encoded according to the <ORAddressValue> rule as a quoted character string. The character string is first derived according to the textual representation of MTS.ORAddress from RFC 2156 [2], and then encoded as if it were an IA5String value.

```
ORAddressValue = StringValue
```

4. GSER Transfer Syntax

The following OBJECT IDENTIFIER has been assigned by Adacel Technologies, under an arc assigned to Adacel by Standards Australia, to identify the Generic String Encoding Rules:

```
{ 1 2 36 79672281 0 0 }
```

This OBJECT IDENTIFIER would be used, for example, to describe the transfer syntax for a GSER encoded data-value in an EMBEDDED PDV value.

5. Security Considerations

The Generic String Encoding Rules do not define a canonical encoding. That is, a transformation from a GSER encoding into some other encoding (e.g., BER) and back into GSER will not necessarily reproduce the original GSER octet encoding. Therefore, GSER MUST NOT be used where a canonical encoding is needed.

Furthermore, GSER does not necessarily enable the exact octet encoding of values of the TeletexString, VideotexString, GraphicString or GeneralString types to be reconstructed, so a transformation from a Distinguished Encoding Rules (DER) [12] encoding to GSER and back to DER may not reproduce the original DER encoding. Therefore, GSER MUST NOT be used to re-encode, whether for storage or transmission, ASN.1 abstract values whose original binary encoding must be recoverable. Such recovery is needed for the verification of digital signatures. In such cases, protocols ought to use DER or a DER-reversible encoding.

When interpreting security-sensitive fields, and in particular fields used to grant or deny access, implementations MUST ensure that any comparisons are done on the underlying abstract value, regardless of the particular encoding used.

6. References

6.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Kille, S., "MIXER (Mime Internet X.400 Enhanced Relay): Mapping between X.400 and RFC 822/MIME", RFC 2156, January 1998.

[3] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[4] Wahl, M., Coulbeck, A., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997.

[5] Wahl, M., Kille S. and T. Howes. "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

[6] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[7] ITU-T Recommendation X.520 (1993) | ISO/IEC 9594-6:1994, Information Technology - Open Systems Interconnection - The Directory: Selected attribute types

[8] ITU-T Recommendation X.680 (07/02) | ISO/IEC 8824-1:2002 Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation

[9] ITU-T Recommendation X.681 (07/02) | ISO/IEC 8824-2:2002 Information technology - Abstract Syntax Notation One (ASN.1): Information object specification

[10] ITU-T Recommendation X.682 (07/02) | ISO/IEC 8824-3:2002 Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification

[11] ITU-T Recommendation X.683 (07/02) | ISO/IEC 8824-4:2002 Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications

[12] ITU-T Recommendation X.690 (07/02) | ISO/IEC 8825-1:2002 Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

6.2. Informative References

[13] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[14] Hodges, J. and R. Morgan, "Lightweight Directory Access Protocol (v3): Technical Specification", RFC 3377, September 2002.

[15] Legg, S., "Common Elements of Generic String Encoding Rules (GSER) Encodings", RFC 3642, October 2003.

[16] ITU-T Recommendation X.500 (1993) | ISO/IEC 9594-1:1994, Information Technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services

7. Intellectual Property Notice

8. Author's Address

Steven Legg
Adacel Technologies Ltd.
250 Bay Street
Brighton, Victoria 3186
AUSTRALIA

Phone: +61 3 8530 7710
Fax: +61 3 8530 7888
EMail: steven.legg@adacel.com.au

9. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.