```
-- Module:   GENERAL-TC
-- Editor:   Tom Hastings
-- File:     gentc.doc, .psr, .ps, .txt
-- Date:     October 24, 1995
-- Version:  0.1


--
-- General Textual Conventions for use with a number of MIBs, that use
-- the host resources MIB (see RFC 1514).
--
-- This module declares general Textual Conventions
-- and other items which are to be imported into other MIB
-- modules.  It is a companion to the General MIB module (file gen.txt).
--
```

Text Attributes

There are three different kinds of text strings used in MIBs:

1. Strings that are represented as Network Virtual Terminal (NVT) ASCII and are never localized. Such strings are specified in MIBs using the SMI textual convention: DisplayString. SMI specifies the size as a maximum of 255 octets as a comment, so all our MIBs must specify the size constraint as: (SIZE(0..255)) explicitly.

2. Strings that are localized by the agent according to one or more locales specified by the management station. A locale specificaion consists of a language, a territory, and a coded character set. Such strings are specified in MIBs using the host resources MIB (RFC 1514) textual convention: InternationalDisplayString. RFC 1514 does not constrain InternationalDisplayString, so all our MIBs must specify the size constraint as: (SIZE(0..255)) explicitly.

3. Strings that are represented in one or more coded character sets by the agent and that management stations can select which coded character set representation for the agent to return in a get by specifying the desired coded character set as an index. Such string are specified in MIBs using the (new) textual convention: CodeIndexedStringIndex that point to objects of type CodeIndexedString.

Objects of type InternationalDisplayString are used for description strings that need to be localized, so that the user of the management station can see the text in his/her locale (language, country, and coded character set). Because these InternationalDisplayString objects are for relatively static entities, the InternationDisplayString objects are typically stored in message catalog files, according to the various localizations supported. See RFC 1759 (Printer MIB) for a localization method for description strings that applies only to description strings in the Printer MIB.

On the other hand, CodeIndexedString objects are generated on the fly by submitting clients, so that there is no opportunity for a server or SNMP agent to use message catalogs. Examples of CodeIndexedString objects, include user-name, job-name, job-comment, document-name, etc. Also end

users, system operators, and system administrators don't expect to see such CodeIndexedString objects in their own language; rather these text strings remain in the language of the originator. The server and SNMP agent may perform transparent coded character set conversion on CodeIndexedString objects, preserving the characters and only changing the coding or providing meaningful fallback representation.

Localization Group

The Localization Group provides objects to control the localization of objects of the InternationalDisplayString type. The General MIB will provide a single 16-bit genCurrentLocalizationIndex object that management stations can set to the current localization that the agent is to use for subsequent gets. The value of the current localization is specified as an index value into the genLocalizationTable supplied by the agent. Thus the genLocalizationTable indicates the localizations that the agent supports.

Managment stations must include a get of the genCurrentLocalizationIndex whenever they get objects of type InternationalDisplayString, in order to detect when another management station might have set the index to a different value. If the value of genCurrentLocalizationIndex object was not as expected, the management station shall wait a random time period and try again by first re-setting the value of the genCurrentLocalizationIndex object to the desired genLocalizationTable index (similar to the Ethernet collision backoff algorithm).

Usually InternationalDisplayString objects are installed in the device or server by means outside SNMP and this MIB. For example, one such method is installing additional message catalog files into a special file system directory on the server. In such cases, management stations can not modify the values of InternationalDisplayString objects and the agent implements them as read-only. However, some agents may wish to allow management stations to alter the descriptions for supported localizations by implementing the InternationalDisplayString objects as read-write. However, such implementation shall allow the management station to set the genCurrentLocalizationIndex object to any of the supported localizations and shall keep distinct values for an InternationDisplayString object for each localization so written. If a management station writes a different value using the same localization, the agent shall replace the old value with the new value. If an agent is not able to keep distinct values for a particular InternationDisplayString object, one for each supported localization, the agent shall implement that InternationalDisplayString object as read-only.

A higher end agent may allow management stations to install new localizations in the agent by allowing the management station to add a row to the genLocalizationTable, using the RowStatus mechanism. In other words, the management station is able to effectively store a new message catalog for a new localization by adding a row to the genLocalizationTable and then writing each InternationalDisplayString object with a value appropriate to the new localization. Agents not capable of accepting new rows in the genLocalizationTable shall implement the genLocalizationRowStatus as read-only.

The objects for controlling the localization of InternationalDisplayString objects are put into the optional General

Current Localization Group and General Localization Group in the General MIB so that they may be used with any MIB.

Note that this method for controlling localization in the agent is the same method as is used in the Printer MIB (RFC 1759), except that this method shall apply to all objects of type InternationDisplayString in any legacy or future MIB, whereas the localization method in the Printer MIB is only for certain specified objects in the printer MIB. Those objects are of type OCTET STRING, so there is no conflict or overlap between these two localization methods; they apply to disjoint sets of objects.

Code Indexed String Group

The Code Indexed String group provides objects to control the coded representation of objects of type: CodeIndexedStringIndex. The value of objects of type CodeIndexedStringIndex is the second index into a single genCodeIndexedStringTable for the device (the first index being hrDeviceIndex). The third index into this genCodeIndexedStringTable is the coded character set enum that has been registered with IANA (See RFC 1759 CodedCharSet textual-convention). Thus the management station can request any supported coded character set from the agent. The agent either has the string stored or performs on-the-fly code conversion to the character sets that the agent supports.

The management station must first request objects of type CodeIndexedStringIndex and then make a second get specifying the returned index and the coded character enum desired by the management station (from among the sets supported by the SNMP agent). If a management station requests a coded character set that the SNMP agent doesn't support, a V2 SNMP agent shall return a no such instance error. A V1 SNMP agent shall return nosuch [object].

Example: The job-name object is of type CodeIndexedStringIndex. Say it contains the index value 500. Lets also assume that the hrDeviceIndex value is 10, so that the printer in question is the 10th device in the host resources table. Finally, assume that the agent supports ASCII and ISO Latin1, which have registered IANA enum values of 3 and 4, respectively. The agent will appear to store job-name objects in the genCodedString table that the management station accesses by the following indexes:

```
{ ..., 10, 500, 3 }
{ ..., 10, 500, 4 }
```

The SNMP agent need not actually have all supported representations of a CodeIndexedStringIndex object stored, but may code convert to the requested coded character set on the fly in response to the Get operation, depending on which of the coded character sets the management station actually requested. Thus in the example above, the agent may only store the job-name object value as ISO Latin 1 {..., 10, 500, 4 } and convert to ASCII when a management station requests { ..., 10, 500, 3 }.

> NOTE TO MIB DESIGNERS: Since the CodeIndexedStringIndex type requires two gets, it should not be used except where there is no opportunity to use static message catalog files. Use InternationalDisplayString whenever possible, as long as the

contention problem between multiple management stations for different sets is not a problem. The job/document monitors that query the job monitoring MIB are more likely to have collisions if they had to set the genCurrentLocalizationIndex object, since many end users, not just system operators and system administrators, will have the job/document monitors running.

In order to help a management station discover the coded character sets supported for CodeIndexedString objects, the General MIB contains a genCodedCharSetTable. The genCodedCharSetTable contains the enums of the character sets registered with IANA that the SNMP agent supports, either directly or with code conversion, along with a name and description of each coded character set.

Code Conversions of CodeIndexedString objects

Each CodeIndexedStringIndex object shall reference a string of at least one of the following coded character sets:

- ASCII (X3.4-1968, NVT ASCII)
- ISO Latin 1 (ISO 8859-1)
- T61String (ITU/CCITT text communication which contains JIS 6226)
- ISO 10646 UCS-2 level 2 (Unicode is actually level 3 and has multiple representations for the same characters)
- Shift JIS
- EUC
- GB 2312 (PRC Chinese)

The first four coded character sets are required by ISO DPA, but we are relaxing that requirement for the MIB, since DPA implementors are having trouble meeting that requirement. Also DPA implementors want the freedom to support other national sets, especially in China and Japan where Unicode may not quite meet their needs.

Agents shall support ASCII. Since ISO Latin1 is the default coded character set of Windows, agents shall support ISO Latin1. Support of other ISO 8859 parts (5 other Latin sets, plus Latin-Cyrillic, Latin-Greek, Latin-Hebrew, and Latin-Arabic) may also be supported with this method.

Agents shall perform code conversions from a source coded character set to a destination coded character set when the destination coded character set contains the source coded character set as a subset. For example, servers shall support code conversions from the following sets, if they support both the source and destination coded character sets (as indicated in the genCodedCharSet table:

source coded character set	destination coded character set
ASCII (ANS X3.4)	ISO Latin1 (ISO 8859-1)
ASCII	ISO Latin-Greek (ISO 8859-5?)
ASCII	Unicode (ISO 10646, UCS-2, level 3)
ISO Latin1	Unicode
ISO Latin-Greek	Unicode
ASCII	JIS 6226 (Kanji which contains ASCII)
JIS 6226	Shift JIS
Shift JIS	JIS 6226

Code conversion between Shift JIS and EUC (and back) is fairly trivial,

since both embody the Japanese national coded character set standard for Kanji (and Latin and Cyrillic). Therefore, if an agent supports one, the agent shall support code conversion to the other.

Code conversion between the character repertoire of ISO Latin 1 between ISO Latin1 and Unicode representation is easy (add/remove a leading zero octet). Code conversion from ASCII or Latin1 to JIS or Shift JIS is straightforward. However, code conversion from Unicode to ASCII or Latin1 when there are characters outside the repertoire of the destination coded character set is harder. Alternatives for handling the case when source coded character data contains characters that are not representable in the destination coded character set:

1. Represent the characters that the agent cannot represent in the requested coded character set using a closely related character, such as the unaccented latin letter, or some error condition such as * or ? for those characters that don't have obvious closely related characters.

2. Don't return anything; make the management station request one of the other coded character representations (by providing a different CodedCharSet enum value as the third index into the genCodeIndexedString table.

3. Convert Unicode into the two character mnemonic representation contained in RFC 1345 which has a two character ASCII representation for all characters of Unicode. Do the same for conversion of ISO Latin1 into ASCII. For example, RFC 1345 represents LATIN SMALL LETTER A WITH ACUTE as a' and MICRO SIGN as My.

4. RFC 1345 was designed for use by software and implementors and, therefore, avoids the use of so-called national-use characters (ACCENT GRAVE (`), CIRCUMFLEX ACCENT (^), TILDE (~). This fourth alternative is to use better two-character approximations than those in RFC 1345 that would be recognized by end-users without special training and would use these obvious national-use characters in such approximations.

To help a management station that wants to avoid getting too many approximations or error characters, the management station can query the agent to determine the coded character set that was used to submit the job. The JobSubmittedCodedCharSet contains the character set enum used by the submitting client for attributes of type: CodeIndexedStringIndex. The management station can then request the data using that character set (if the management client supports that character set or perfers to perform its own coded character set conversion, rather than relying on the agent to perform the coded character set conversion).

While the ISO DPA standard provides for character coded attributes of a maximum of 4095 octets, the MIB objects shall support 255-octet coded character set data. This limit is because some transport mechanisms (such as Novell), cannot handle more than 576 bytes in a packet. Headers are about 30 bytes, leaving 546 bytes.

```
GENERAL-TC DEFINITIONS ::= BEGIN
```

```
301
302   IMPORTS
303           MODULE-IDENTITY, Counter64, experimental
304                   FROM SNMPv2-SMI       -- RFC 1442
305           TEXTUAL-CONVENTION
306                   FROM SNMPv2-TC;       -- RFC 1443
307
308
309     -- Upon publication as RFC, delete this comment and the line following
310     -- this comment and change the reference of { printmib 100 }
311     -- (below) to { mib-2 X }.
312     -- This will result in changing:
313     -- 1 3 6 1 3 54 generalTC(100)    to:
314     -- 1 3 6 1 2  1 generalTC(X)
315     -- This will make it easier to translate prototypes to
316     -- the standard namespace because the lengths of the OID's won't
317     -- change.
318      printmib OBJECT IDENTIFIER ::= { experimental 54 }
319
320   generalTC MODULE-IDENTITY
321           LAST-UPDATED "9510250000Z"
322           ORGANIZATION "IETF/DMTF Printer Working Group"
323           CONTACT-INFO
324           "       Thomas N. Hastings
325                   Xerox Corporation, MS ES-AE 242
326                   701 S. Aviation Blvd.
327                   El Segundo, CA 90245
328                   Phone:        1+ (310)333-6413
329                   FAX:          1+ (310)333-6342
330                   E-Mail:       hastings@cp10.es.xerox.com"
331           DESCRIPTION
332                   "File: gentc.doc, .psr, .ps, .txt
333                    Version: 0.1
334
335                   General Textual Conventions for use with other MIBs
336                   that use the host resources MIB (see RFC 1514)."
337           ::= { printmib 100 }
338
339   --
340   -- General Textual Conventions in alphabetical order.
341   --
342
343   Cardinal16 ::= TEXTUAL-CONVENTION
344       STATUS     current
345       DESCRIPTION
346           "The representation for non-negative integers.
347           Used for indexes in small tables where 0 means not specified.
348           It avoids use of the sign bit."
349       SYNTAX    INTEGER (0..32767)       -- biggest int = 2**15-1
350
351   Cardinal32 ::= TEXTUAL-CONVENTION
352       STATUS     current
353       DESCRIPTION
354           "The representation for non-negative integers.
355           Used for indexes in large tables where 0 means not specified.
356           Same size as ISO 10175 (avoids use of sign bit)."
357       SYNTAX    INTEGER (0..2147483647)  -- biggest int = 2**31-1
358
359   Cardinal64 ::= TEXTUAL-CONVENTION
360       STATUS     current
```

```
361         DESCRIPTION
362             "The representation for non-negative integers.
363             Used for indexes in very large tables where 0 means not
364     specified.
365             Same size as ISO 10175 (avoids use of sign bit)."
366         SYNTAX    Counter64
367                   -- Should be INTEGER (0..9223372036854775807)   2**63-1
368                   -- but some ASN.1 compilers reject such a large limit
369
370     CodedLanguage ::= TEXTUAL-CONVENTION
371         STATUS     current
372         DESCRIPTION
373             "A two character language code from ISO 639.
374             A blank string (two space characters) shall indicate that the
375             territory is not defined.
376             Examples EN, GB, CA, FR, DE."
377         SYNTAX     OCTET STRING (SIZE(2))
378
379     CodeIndexedStringIndex ::= TEXTUAL-CONVENTION
380         STATUS     current
381         DESCRIPTION
382             "The representation of string data which the agent
383             can provide in one or more character sets (but not further
384             localized).  Typically this representation is used because the
385             string data is relatively dynamic, changing too rapidly for full
386             localization; or because the data exists inherently in only one
387             or a limited number of character sets and cannot meaningfully be
388             further localized.
389
390             The value is an index into a single global string table,
391             genCodeIndexedStringTable.  A subsidiary index into the
392             genCodeIndexedStringTable is the IANA registered enum (see the
393             CodedCharSet textual-convention in RFC 1759) for the
394             coded character set desired by the management station (from
395             among the coded character sets supported by the SNMP agent).
396
397             A 0 index value shall indicate that there is no associated entry
398             in the string table.
399
400             32 bits are needed because Jobs can use up 10-12 code-indexed
401             strings per job."
402         SYNTAX     Cardinal32
403
404     CodedTerritory ::= TEXTUAL-CONVENTION
405         STATUS     current
406         DESCRIPTION
407             "A two character country code from ISO 3166.
408             A blank string (two space characters) shall indicate that the
409             territory is not defined.
410             Examples: US, FR, DE, ..."
411         SYNTAX     OCTET STRING (SIZE(2))
412
413     Gauge64 ::= TEXTUAL-CONVENTION
414         STATUS     current
415         DESCRIPTION
416             "The representation for a non-negative integer, which may
417             increase or decrease, but shall never exceed 2**63-1 in value.
418             Same size as ISO 10175 (avoids use of sign bit)."
419         SYNTAX    Counter64
420                   -- Should be INTEGER (0..9223372036854775807)   2**63-1
```

```
421                -- but some ASN.1 compilers reject such a large limit
422
423   Ordinal16 ::= TEXTUAL-CONVENTION
424       STATUS     current
425       DESCRIPTION
426           "The representation for positive integers.
427           Used for indexes in small tables where 0 is illegal.
428           It avoids use of the sign bit.."
429       SYNTAX    INTEGER (1..32767)  -- biggest int = 2**15-1
430
431   Ordinal32 ::= TEXTUAL-CONVENTION
432       STATUS     current
433       DESCRIPTION
434           "The representation for positive integers.
435           Same size as ISO 10175 (avoids use of sign bit)."
436       SYNTAX    INTEGER (1..2147483647)  -- biggest int = 2**31-1
437
438   Ordinal64 ::= TEXTUAL-CONVENTION
439       STATUS     current
440       DESCRIPTION
441           "The representation for positive integers.
442           Same size as ISO 10175 (avoids use of sign bit)."
443       SYNTAX    Counter64
444                 -- Should be INTEGER (1..9223372036854775807)   2**63-1
445                 -- but some ASN.1 compilers reject such a large limit
446   END
447
```