Network Working Group
Request for Comments: 4790
Category: Standards Track

C. Newman
Sun Microsystems
M. Duerst
Aoyama Gakuin University
A. Gulbrandsen
Oryx

March 2007

Internet Application Protocol Collation Registry

Status of This Memo

Copyright Notice

Abstract

Many Internet application protocols include string-based lookup, searching, or sorting operations. However, the problem space for searching and sorting international strings is large, not fully explored, and is outside the area of expertise for the Internet Engineering Task Force (IETF). Rather than attempt to solve such a large problem, this specification creates an abstraction framework so that application protocols can precisely identify a comparison function, and the repertoire of comparison functions can be extended in the future.

Table of Contents

1. Introduction

The Application Configuration Access Protocol ACAP [11] specification introduced the concept of a comparator (which we call collation in this document), but failed to create an IANA registry. With the introduction of stringprep [6] and the Unicode Collation Algorithm [7], it is now time to create that registry and populate it with some initial values appropriate for an international community. This specification replaces and generalizes the definition of a comparator in ACAP, and creates a collation registry.

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as defined in "Key words for use in RFCs to Indicate Requirement Levels" [1].

The attribute syntax specifications use the Augmented Backus-Naur Form (ABNF) [2] notation, including the core rules defined in Appendix A. The ABNF production "Language-tag" is imported from Language Tags [5] and "reg-name" from URI: Generic Syntax [4].

2. Collation Definition and Purpose

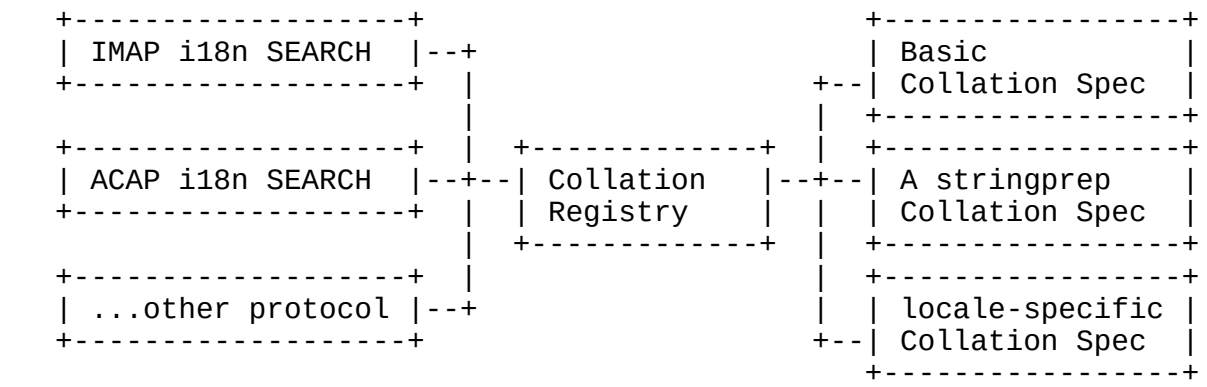
2.1. Definition

A collation is a named function which takes two arbitrary length strings as input and can be used to perform one or more of three basic comparison operations: equality test, substring match, and ordering test.

2.2. Purpose

Collations are an abstraction for comparison functions so that these comparison functions can be used in multiple protocols. The details of a particular comparison operation can be specified by someone with appropriate expertise, independent of the application protocols that use that collation. This is similar to the way a charset [13] separates the details of octet to character mapping from a protocol specification, such as MIME [9], or the way SASL [10] separates the details of an authentication mechanism from a protocol specification, such as ACAP [11].

Here is a small diagram to help illustrate the value of this abstraction:



Thus IMAP, ACAP, and future application protocols with international search capability simply specify how to interface to the collation registry instead of each protocol specification having to specify all the collations it supports.

2.3. Some Other Terms Used in this Document

The terms client, server, and protocol are used in somewhat unusual senses.

Client means a user, or a program acting directly on behalf of a user. This may be a mail reader acting as an IMAP client, or it may be an interactive shell, where the user can type protocol commands/requests directly, or it may be a script or program written by the user.

Server means a program that performs services requested by the client. This may be a traditional server such as an HTTP server, or it may be a Sieve [14] interpreter running a Sieve script written by a user. A server needs to use the operations provided by collations in order to fulfill the client’s requests.

The protocol describes how the client tells the server what it wants done, and (if applicable) how the server tells the client about the results. IMAP is a protocol by this definition, and so is the Sieve language.

2.4. Sort Keys

One component of a collation is a transformation, which turns a string into a sort key, which is then used while sorting.

The transformation can range from an identity mapping (e.g., the i;octet collation Section 9.3) to a mapping that makes the string unreadable to a human.

This is an implementation detail of collations or servers. A protocol SHOULD NOT expose it to clients, since some collations leave the sort key's format up to the implementation, and current conformant implementations are known to use different formats.

3. Collation Identifier Syntax

3.1. Basic Syntax

The collation identifier itself is a single US-ASCII string. The identifier MUST NOT be longer than 254 characters, and obeys the following grammar:

```
collation-char  = ALPHA / DIGIT / "-" / ";" / "=" / "."

collation-id    = collation-prefix ";" collation-core-name
                  *collation-arg

collation-scope = Language-tag / "vnd-" reg-name

collation-core-name = ALPHA *( ALPHA / DIGIT / "-" )

collation-arg   = ";" ALPHA *( ALPHA / DIGIT ) "="
                  1*( ALPHA / DIGIT / "." )
```

Note: the ABNF production "Language-tag" is imported from Language Tags [5] and "reg-name" from URI: Generic Syntax [4].

There is a special identifier called "default". For protocols that have a default collation, "default" refers to that collation. For other protocols, the identifier "default" MUST match no collations, and servers SHOULD treat it in the same way as they treat nonexistent collations.

3.2. Wildcards

The string a client uses to select a collation MAY contain one or more wildcard ("*") characters that match zero or more collation-chars. Wildcard characters MUST NOT be adjacent. If the wildcard string matches multiple collations, the server SHOULD attempt to select a widely useful collation in preference to a narrowly useful one.

```
collation-wild  =  ("*" / (ALPHA ["*"])) *(collation-char ["*"])
                    ; MUST NOT exceed 254 characters total
```

3.3. Ordering Direction

When used as a protocol element for ordering, the collation identifier MAY be prefixed by either "+" or "-" to explicitly specify an ordering direction. "+" has no effect on the ordering operation, while "-" inverts the result of the ordering operation. In general, collation-order is used when a client requests a collation, and collation-selected is used when the server informs the client of the selected collation.

```
collation-selected =  ["+" / "-"] collation-id

collation-order =  ["+" / "-"] collation-wild
```

3.4. URIs

Some protocols are designed to use URIs [4] to refer to collations rather than simple tokens. A special section of the IANA URL space is reserved for such usage. The "collation-uri" form is used to refer to a specific named collation (the collation registration may not actually be present). The "collation-auri" form is an abstract name for an ordering, a collation pattern or a vendor private collator.

```
collation-uri   =  "http://www.iana.org/assignments/collation/"
                   collation-id ".xml"

collation-auri  =  ( "http://www.iana.org/assignments/collation/"
                   collation-order ".xml" ) / other-uri

other-uri       =  <absoluteURI>
                ;  excluding the IANA collation namespace.
```

3.5. Naming Guidelines

While this specification makes no absolute requirements on the structure of collation identifiers, naming consistency is important, so the following initial guidelines are provided.

Collation identifiers with an international audience typically begin with "i;". Collation identifiers intended for a particular language or locale typically begin with a language tag [5] followed by a ";". After the first ";" is normally the name of the general collation algorithm, followed by a series of algorithm modifications separated by the ";" delimiter. Parameterized modifications will use "=" to

delimit the parameter from the value. The version numbers of any lookup tables used by the algorithm SHOULD be present as parameterized modifications.

Collation identifiers of the form *;vnd-hostname;* are reserved for vendor-specific collations created by the owner of the hostname following the "vnd-" prefix (e.g., vnd-example.com for the vendor example.com). Registration of such collations (or the name space as a whole), with intended use of the "Vendor", is encouraged when a public specification or open-source implementation is available, but is not required.

4. Collation Specification Requirements

4.1. Collation/Server Interface

The collation itself defines what it operates on. Most collations are expected to operate on character strings. The i;octet (Section 9.3) collation operates on octet strings. The i;ascii-numeric (Section 9.1) operation operates on numbers.

This specification defines the collation interface in terms of octet strings. However, implementations may choose to use character strings instead. Such implementations may not be able to implement e.g., i;octet. Since i;octet is not currently mandatory to implement for any protocol, this should not be a problem.

4.2. Operations Supported

A collation specification MUST state which of the three basic operations are supported (equality, substring, ordering) and how to perform each of the supported operations on any two input character strings, including empty strings. Collations must be deterministic, i.e., given a collation with a specific identifier, and any two fixed input strings, the result MUST be the same for the same operation.

In general, collation operations should behave as their names suggest. While a collation may be new, the operations are not, so the new collation’s operations should be similar to those of older collations. For example, a date/time collation should not provide a "substring" operation that would morph IMAP substring SEARCH into e.g., a date-range search.

A non-obvious consequence of the rules for each collation operation is that, for any single collation, either none or all of the operations can return "undefined". For example, it is not possible to have an equality operation that never returns "undefined", and a substring operation that occasionally does.

4.2.1. Validity

The validity test takes one string as argument. It returns valid if its input string is a valid input to the collation's other operations, and invalid if not. (In other words, a string is valid if it is equal to itself according to the collation's equality operation.)

The validity test is provided by all collations. It MUST NOT be listed separately in the collation registration.

4.2.2. Equality

The equality test always returns "match" or "no-match" when it is supplied valid input, and MAY return "undefined" if one or both input strings are not valid.

The equality test MUST be reflexive and symmetric. For valid input, it MUST be transitive.

If a collation provides either a substring or an ordering test, it MUST also provide an equality test. The substring and/or ordering tests MUST be consistent with the equality test.

The return values of the equality test are called "match", "no-match" and "undefined" in this document.

4.2.3. Substring

The substring matching operation determines if the first string is a substring of the second string, i.e., if one or more substrings of the second string is equal to the first, as defined by the collation's equality operation.

A collation that supports substring matching will automatically support two special cases of substring matching: prefix and suffix matching, if those special cases are supported by the application protocol. It returns "match" or "no-match" when it is supplied valid input and returns "undefined" when supplied invalid input.

Application protocols MAY return position information for substring matches. If this is done, the position information SHOULD include both the starting offset and the ending offset for each match. This is important because more sophisticated collations can match strings of unequal length (for example, a pre-composed accented character can match a decomposed accented character). In general, overlapping matches SHOULD be reported (as when "ana" occurs twice within "banana"), although there are cases where a collation may decide not

to. For example, in a collation which treats all whitespace sequences as identical, the substring operation could be defined such that " 1 " (SP "1" SP) is reported just once within " 1 " (SP SP "1" SP SP), not four times (SP SP "1" SP, SP "1" SP, SP "1" SP SP and SP SP "1" SP SP), since the four matches are, in a sense, the same match.

A string is a substring of itself. The empty string is a substring of all strings.

Note that the substring operation of some collations can match strings of unequal length. For example, a pre-composed accented character can match a decomposed accented character. The Unicode Collation Algorithm [7] discusses this in more detail.

The return values of the substring operation are called "match", "no-match", and "undefined" in this document.

4.2.4. Ordering

The ordering operation determines how two strings are ordered. It MUST be reflexive. For valid input, it MUST be transitive and trichotomous.

Ordering returns "less" if the first string is listed before the second string, according to the collation; "greater", if the second string is listed before the first string; and "equal", if the two strings are equal, as defined by the collation’s equality operation. If one or both strings are invalid, the result of ordering is "undefined".

When the collation is used with a "+" prefix, the behavior is the same as when used with no prefix. When the collation is used with a "-" prefix, the result of the ordering operation of the collation MUST be reversed.

The return values of the ordering operation are called "less", "equal", "greater", and "undefined" in this document.

4.3. Sort Keys

A collation specification SHOULD describe the internal transformation algorithm to generate sort keys. This algorithm can be applied to individual strings, and the result can be stored to potentially optimize future comparison operations. A collation MAY specify that the sort key is generated by the identity function. The sort key may have no meaning to a human. The sort key may not be valid input to the collation.

4.4. Use of Lookup Tables

Some collations use customizable lookup tables, e.g., because the tables depend on locale, and may be modified after shipping the software. Collations that use more than one customizable lookup table in a documented format MUST assign numbers to the tables they use. This permits an application protocol command to access the tables used by a server collation, so that clients and servers use the same tables.

5. Application Protocol Requirements

This section describes the requirements and issues that an application protocol needs to consider if it offers searching, substring matching and/or sorting, and permits the use of characters outside the US-ASCII charset.

5.1. Character Encoding

The protocol specification has to make sure that it is clear on which characters (rather than just octets) the collations are used. This can be done by specifying the protocol itself in terms of characters (e.g., in the case of a query language), by specifying a single character encoding for the protocol (e.g., UTF-8 [3]), or by carefully describing the relevant issues of character encoding labeling and conversion. In the later case, details to consider include how to handle unknown charsets, any charsets that are mandatory-to-implement, any issues with byte-order that might apply, and any transfer encodings that need to be supported.

5.2. Operations

The protocol must specify which of the operations defined in this specification (equality matching, substring matching, and ordering) can be invoked in the protocol, and how they are invoked. There may be more than one way to invoke an operation.

The protocol MUST provide a mechanism for the client to select the collation to use with equality matching, substring matching, and ordering.

If a protocol needs a total ordering and the collation chosen does not provide it because the ordering operation returns "undefined" at least once, the recommended fallback is to sort all invalid strings after the valid ones, and use i;octet to order the invalid strings.

Although the collation’s substring function provides a list of matches, a protocol need not provide all that to the client. It may

provide only the first matching substring, or even just the information that the substring search matched. In this way, collations can be used with protocols that are defined such that "x is a substring of y" returns true-false.

If the protocol provides positional information for the results of a substring match, that positional information SHOULD fully specify the substring(s) in the result that matches, independent of the length of the search string. For example, returning both the starting and ending offset of the match would suffice, as would the starting offset and a length. Returning just the starting offset is not acceptable. This rule is necessary because advanced collations can treat strings of different lengths as equal (for example, pre-composed and decomposed accented characters).

5.3. Wildcards

The protocol MUST specify whether it allows the use of wildcards in collation identifiers. If the protocol allows wildcards, then:

The protocol MUST specify how comparisons behave in the absence of explicit collation negotiation, or when a collation of "default" is requested. The protocol MAY specify that the default collation used in such circumstances is sensitive to server configuration.

The protocol SHOULD provide a way to list available collations matching a given wildcard pattern, or patterns.

5.4. String Comparison

If a protocol compares strings in any nontrivial way, using a collation may be appropriate. As an example, many protocols use case-independent strings. In many cases, a simple ASCII mapping to upper/lower case works well. In other cases, it may be better to use a specifiable collation; for example, so that a server can treat "i" and "I" as equivalent in Italy, and different in Turkey (Turkish also has a dotted upper-case" I" and a dotless lower-case "i").

Protocol designers should consider, in each case, whether to use a specifiable collation. Keywords often have other needs than user variables, and search arguments may be different again.

5.5. Disconnected Clients

If the protocol supports disconnected clients, and a collation is used that can use configurable tables (e.g., to support locale-specific extensions), then the client may not be able to reproduce the server’s collation operations while offline.

A mechanism to download such tables has been discussed. Such a mechanism is not included in the present specification, since the problem is not yet well understood.

5.6. Error Codes

The protocol specification should consider assigning protocol error codes for the following circumstances:

- The client requests the use of a collation by identifier or pattern, but no implemented collation matches that pattern.
- The client attempts to use a collation for an operation that is not supported by that collation -- for example, attempting to use the "i;ascii-numeric" collation for substring matching.
- The client uses an equality or substring matching collation, and the result is an error. It may be appropriate to distinguish between the two input strings, particularly when one is supplied by the client and the other is stored by the server. It might also be appropriate to distinguish the specific case of an invalid UTF-8 string.

5.7. Octet Collation

The i;octet (Section 9.3) collation is only usable with protocols based on octet-strings. Clients and servers MUST NOT use i;octet with other protocols.

If the protocol permits the use of collations with data structures other than strings, the protocol MUST describe the default behavior for a collation with those data structures.

6. Use by Existing Protocols

This section is informative.

Both ACAP [11] and Sieve [14] are standards track specifications that used collations prior to the creation of this specification and registry. Those standards do not meet all the application protocol requirements described in Section 5.

These protocols allow the use of the i;octet (Section 9.3) collation working directly on UTF-8 data, as used in these protocols.

In Sieve, all matches are either true or false. Accordingly, Sieve servers must treat "undefined" and "no-match" results of the equality and substring operations as false, and only "match" as true.

In ACAP and Sieve, there are no invalid strings. In this document's terms, invalid strings sort after valid strings.

IMAP [15] also collates, although that is explicit only when the COMPARATOR [17] extension is used. The built-in IMAP substring operation and the ordering provided by the SORT [16] extension may not meet the requirements made in this document.

Other protocols may be in a similar position.

In IMAP, the default collation is i;ascii-casemap, because its operations are understood to match IMAP's built-in operations.

7. Collation Registration

7.1. Collation Registration Procedure

The IETF will create a mailing list, collation@ietf.org, which can be used for public discussion of collation proposals prior to registration. Use of the mailing list is strongly encouraged. The IESG will appoint a designated expert who will monitor the collation@ietf.org mailing list and review registrations.

The registration procedure begins when a completed registration template is sent to iana@iana.org and collation@ietf.org. The designated expert is expected to tell IANA and the submitter of the registration within two weeks whether the registration is approved, approved with minor changes, or rejected with cause. When a registration is rejected with cause, it can be re-submitted if the concerns listed in the cause are addressed. Decisions made by the designated expert can be appealed to the IESG Applications Area Director, then to the IESG. They follow the normal appeals procedure for IESG decisions.

Collation registrations in a standards track, BCP, or IESG-approved experimental RFC are owned by the IETF, and changes to the registration follow normal procedures for updating such documents. Collation registrations in other RFCs are owned by the RFC author(s). Other collation registrations are owned by the individual(s) listed in the contact field of the registration, and IANA will preserve this information.

If the registration is a change of an existing collation, it MUST be approved by the owner. In the event the owner cannot be contacted

for a period of one month, and the designated expert deems the change necessary, the IESG MAY re-assign ownership to an appropriate party.

7.2. Collation Registration Format

Registration of a collation is done by sending a well-formed XML document to collation@ietf.org and iana@iana.org.

7.2.1. Registration Template

Here is a template for the registration:

```
<?xml version='1.0'?>
<!DOCTYPE collation SYSTEM 'collationreg.dtd'>
<collation rfc="YYYY" scope="global" intendedUse="common">
  <identifier>collation identifier</identifier>
  <title>technical title for collation</title>
  <operations>equality order substring</operations>
  <specification>specification reference</specification>
  <owner>email address of owner or IETF</owner>
  <submitter>email address of submitter</submitter>
  <version>1</version>
</collation>
```

7.2.2. The Collation Element

The root of the registration document MUST be a <collation> element. The collation element contains the other elements in the registration, which are described in the following sub-subsections, in the order given here.

The <collation> element MAY include an "rfc=" attribute if the specification is in an RFC. The "rfc=" attribute gives only the number of the RFC, without any prefix, such as "RFC", or suffix, such as ".txt".

The <collation> element MUST include a "scope=" attribute, which MUST have one of the values "global", "local", or "other".

The <collation> element MUST include an "intendedUse=" attribute, which must have one of the values "common", "limited", "vendor", or "deprecated". Collation specifications intended for "common" use are expected to reference standards from standards bodies with significant experience dealing with the details of international character sets.

Be aware that future revisions of this specification may add additional function types, as well as additional XML attributes,

values, and elements. Any system that automatically parses these XML documents MUST take this into account to preserve future compatibility.

7.2.3. The Identifier Element

The <identifier> element gives the precise identifier of the collation, e.g., i;ascii-casemap. The <identifier> element is mandatory.

7.2.4. The Title Element

The <title> element gives the title of the collation. The <title> element is mandatory.

7.2.5. The Operations Element

The <operations> element lists which of the three operations ("equality", "order" or "substring") the collation provides, separated by single spaces. The <operations> element is mandatory.

7.2.6. The Specification Element

The <specification> element describes where to find the specification. The <specification> element is mandatory. It MAY have a URI attribute. There may be more than one <specification> element, in which case, they together form the specification.

If it is discovered that parts of a collation specification conflict, a new revision of the collation is necessary, and the collation@ietf.org mailing list should be notified.

7.2.7. The Submitter Element

The <submitter> element provides an RFC 2822 [12] email address for the person who submitted the registration. It is optional if the <owner> element contains an email address.

There may be more than one <submitter> element.

7.2.8. The Owner Element

The <owner> element contains either the four letters "IETF" or an email address of the owner of the registration. The <owner> element is mandatory. There may be more than one <owner> element. If so, all owners are equal. Each owner can speak for all.

7.2.9. The Version Element

The <version> element MUST be included when the registration is likely to be revised, or has been revised in such a way that the results change for one or more input strings. The <version> element is optional.

7.2.10. The Variable Element

The <variable> element specifies an optional variable to control the collation's behaviour, for example whether it is case sensitive. The <variable> element is optional. When <variable> is used, it must contain <name> and <default> elements, and it may contain one or more <value> elements.

7.2.10.1. The Name Element

The <name> element specifies the name value of a variable. The <name> element is mandatory.

7.2.10.2. The Default Element

The <default> element specifies the default value of a variable. The <default> element is mandatory.

7.2.10.3. The Value Element

The <value> element specifies a legal value of a variable. The <value> element is optional. If one or more <value> elements are present, only those values are legal. If none are, then the variable's legal values do not form an enumerated set, and the rules MUST be specified in an RFC accompanying the registration.

7.3. Structure of Collation Registry

Once the registration is approved, IANA will store each XML registration document in a URL of the form http://www.iana.org/assignments/collation/collation-id.xml, where collation-id is the content of the identifier element in the registration. Both the submitter and the designated expert are responsible for verifying that the XML is well-formed. The registration document should avoid using new elements. If any are necessary, it is important to be consistent with other registrations.

IANA will also maintain a text summary of the registry under the name http://www.iana.org/assignments/collation/collation-index.html. This summary is divided into four sections. The first section is for collations intended for common use. This section is intended for

collation registrations published in IESG-approved RFCs, or for locally scoped collations from the primary standards body for that locale. The designated expert is encouraged to reject collation registrations with an intended use of "common" if the expert believes it should be "limited", as it is desirable to keep the number of "common" registrations small and of high quality. The second section is reserved for limited-use collations. The third section is reserved for registered vendor-specific collations. The final section is reserved for deprecated collations.

7.4. Example Initial Registry Summary

The following is an example of how IANA might structure the initial registry summary.html file:

Collation	Functions	Scope	Reference
Common Use Collations:			
i;ascii-casemap	e, o, s	Local	[RFC 4790]
Limited Use Collations:			
i;octet	e, o, s	Other	[RFC 4790]
i;ascii-numeric	e, o	Other	[RFC 4790]
Vendor Collations:			
Deprecated Collations:			

References

[RFC 4790] Newman, C., Duerst, M., Gulbrandsen, A., "Internet Application Protocol Collation Registry", RFC 4790, Sun Microsystems, March 2007.

8. Guidelines for Expert Reviewer

The expert reviewer appointed by the IESG has fairly broad latitude for this registry. While a number of collations are expected (particularly customizations of the UCA for localized use), an explosion of collations (particularly common-use collations) is not desirable for widespread interoperability. However, it is important for the expert reviewer to provide cause when rejecting a registration, and, when possible, to describe corrective action to

permit the registration to proceed. The following table includes some example reasons to reject a registration with cause:

- The registration is not a well-formed XML document.
- The registration has an intended use of "common", but there is no evidence the collation will be widely deployed, so it should be listed as "limited".
- The registration has an intended use of "common", but it is redundant with the functionality of a previously registered "common" collation.
- The registration has an intended use of "common", but the specification is not detailed enough to allow interoperable implementations by others.
- The collation identifier fails to precisely identify the version numbers of relevant tables to use.
- The registration fails to meet one of the "MUST" requirements in Section 4.
- The collation identifier fails to meet the syntax in Section 3.
- The collation specification referenced in the registration is vague or has optional features without a clear behavior specified.
- The referenced specification does not adequately address security considerations specific to that collation.
- The registration’s operations are needlessly different from those of traditional operations.
- The registration’s XML is needlessly different from that of already registered collations.

9. Initial Collations

This section registers the three collations that were originally defined in [11], and are implemented in most [14] engines. Some of the behavior of these collations is perhaps not ideal, such as i;ascii-casemap accepting non-ASCII input. Compatibility with widely deployed code was judged more important than fixing the collations. Some of the aspects of these collations are necessary to maintain compatibility with widely deployed code.

9.1. ASCII Numeric Collation

9.1.1. ASCII Numeric Collation Description

The "i;ascii-numeric" collation is a simple collation intended for use with arbitrarily-sized, unsigned decimal integer numbers stored as octet strings. US-ASCII digits (0x30 to 0x39) represent digits of the numbers. Before converting from string to integer, the input string is truncated at the first non-digit character. All input is valid; strings that do not start with a digit represent positive infinity.

The collation supports equality and ordering, but does not support the substring operation.

The equality operation returns "match" if the two strings represent the same number (i.e., leading zeroes and trailing non-digits are disregarded), and "no-match" if the two strings represent different numbers.

The ordering operation returns "less" if the first string represents a smaller number than the second, "equal" if they represent the same number, and "greater" if the first string represents a larger number than the second.

Some examples: "0" is less than "1", and "1" is less than "4294967298". "4294967298", "04294967298", and "4294967298b" are all equal. "04294967298" is less than "". "", "x", and "y" are equal.

9.1.2. ASCII Numeric Collation Registration

```
<?xml version='1.0'?>
<!DOCTYPE collation SYSTEM 'collationreg.dtd'>
<collation rfc="4790" scope="other" intendedUse="limited">
  <identifier>i;ascii-numeric</identifier>
  <title>ASCII Numeric</title>
  <operations>equality order</operations>
  <specification>RFC 4790</specification>
  <owner>IETF</owner>
  <submitter>chris.newman@sun.com</submitter>
</collation>
```

9.2. ASCII Casemap Collation

9.2.1. ASCII Casemap Collation Description

The "i;ascii-casemap" collation is a simple collation that operates on octet strings and treats US-ASCII letters case-insensitively. It provides equality, substring, and ordering operations. All input is valid. Note that letters outside ASCII are not treated case-insensitively.

Its equality, ordering, and substring operations are as for i;octet, except that at first, the lower-case letters (octet values 97-122) in each input string are changed to upper case (octet values 65-90).

Care should be taken when using OS-supplied functions to implement this collation, as it is not locale sensitive. Functions, such as strcasecmp and toupper, are sometimes locale sensitive, and may inappropriately map lower-case letters other than a-z to upper case.

The i;ascii-casemap collation is well-suited for use with many Internet protocols and computer languages. Use with natural language is often inappropriate; even though the collation apparently supports languages such as Swahili and English, in real-world use, it tends to mis-sort a number of types of string:

- people and place names containing non-ASCII,
- words such as "naive" (if spelled with an accent, the accented character could push the word to the wrong spot in a sorted list),
- names such as "Lloyd" (which, in Welsh, sorts after "Lyon", unlike in English),
- strings containing euro and pound sterling symbols, quotation marks other than '"', dashes/hyphens, etc.

9.2.2. ASCII Casemap Collation Registration

```
<?xml version='1.0'?>
<!DOCTYPE collation SYSTEM 'collationreg.dtd'>
<collation rfc="4790" scope="local" intendedUse="common">
  <identifier>i;ascii-casemap</identifier>
  <title>ASCII Casemap</title>
  <operations>equality order substring</operations>
  <specification>RFC 4790</specification>
  <owner>IETF</owner>
  <submitter>chris.newman@sun.com</submitter>
</collation>
```

9.3. Octet Collation

9.3.1. Octet Collation Description

The "i;octet" collation is a simple and fast collation intended for use on binary octet strings rather than on character data. Protocols that want to make this collation available have to do so by explicitly allowing it. If not explicitly allowed, it MUST NOT be used. It never returns an "undefined" result. It provides equality, substring, and ordering operations.

The ordering algorithm is as follows:

1. If both strings are the empty string, return the result "equal".

2. If the first string is empty and the second is not, return the result "less".

3. If the second string is empty and the first is not, return the result "greater".

4. If both strings begin with the same octet value, remove the first octet from both strings and repeat this algorithm from step 1.

5. If the unsigned value (0 to 255) of the first octet of the first string is less than the unsigned value of the first octet of the second string, then return "less".

6. If this step is reached, return "greater".

This algorithm is roughly equivalent to the C library function memcmp, with appropriate length checks added.

The matching operation returns "match" if the sorting algorithm would return "equal". Otherwise, the matching operation returns "no-match".

The substring operation returns "match" if the first string is the empty string, or if there exists a substring of the second string of length equal to the length of the first string, which would result in a "match" result from the equality function. Otherwise, the substring operation returns "no-match".

9.3.2. Octet Collation Registration

This collation is defined with intendedUse="limited" because it can only be used by protocols that explicitly allow it.

```
<?xml version='1.0'?>
<!DOCTYPE collation SYSTEM 'collationreg.dtd'>
<collation rfc="4790" scope="global" intendedUse="limited">
  <identifier>i;octet</identifier>
  <title>Octet</title>
  <operations>equality order substring</operations>
  <specification>RFC 4790</specification>
  <owner>IETF</owner>
  <submitter>chris.newman@sun.com</submitter>
</collation>
```

10. IANA Considerations

Section 7 defines how to register collations with IANA. Section 9 defines a list of predefined collations that have been registered with IANA.

11. Security Considerations

Collations will normally be used with UTF-8 strings. Thus, the security considerations for UTF-8 [3], stringprep [6], and Unicode TR-36 [8] also apply, and are normative to this specification.

12. Acknowledgements

The authors want to thank all who have contributed to this document, including Brian Carpenter, John Cowan, Dave Cridland, Mark Davis, Spencer Dawkins, Lisa Dusseault, Lars Eggert, Frank Ellermann, Philip Guenther, Tony Hansen, Ted Hardie, Sam Hartman, Kjetil Torgrim Homme, Michael Kay, John Klensin, Alexey Melnikov, Jim Melton, and Abhijit Menon-Sen.

13. References

13.1. Normative References

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

[2] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[3] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[4] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005.

[5] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 4646, September 2006.

[6] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[7] Davis, M. and K. Whistler, "Unicode Collation Algorithm version 14", May 2005, <http://www.unicode.org/reports/tr10/tr10-14.html>.

[8] Davis, M. and M. Suignard, "Unicode Security Considerations", February 2006, <http://www.unicode.org/reports/tr36/>.

13.2. Informative References

[9] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[10] Melnikov, A., "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[11] Newman, C. and J. Myers, "ACAP -- Application Configuration Access Protocol", RFC 2244, November 1997.

[12] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[13] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, October 2000.

[14] Showalter, T., "Sieve: A Mail Filtering Language", RFC 3028, January 2001.

[15] Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 3501, March 2003.

[16] Crispin, M. and K. Murchison, "Internet Message Access Protocol - Sort and Thread Extensions", Work in Progress, May 2004.

[17] Newman, C. and A. Gulbrandsen, "Internet Message Access Protocol Internationalization", Work in Progress, January 2006.

Authors' Addresses

Chris Newman
Sun Microsystems
1050 Lakes Drive
West Covina, CA 91790
USA

EMail: chris.newman@sun.com

Martin Duerst
Aoyama Gakuin University
5-10-1 Fuchinobe
Sagamihara, Kanagawa 229-8558
Japan

Phone: +81 42 759 6329
Fax: +81 42 759 6495
EMail: duerst@it.aoyama.ac.jp
URI: http://www.sw.it.aoyama.ac.jp/D%C3%BCrst/

Note: Please write "Duerst" with u-umlaut wherever possible, for example as "Dürst" in XML and HTML.

Arnt Gulbrandsen
Oryx Mail Systems GmbH
Schweppermannstr. 8
81671 Munich
Germany

Fax: +49 89 4502 9758
EMail: arnt@oryx.com
URI: http://www.oryx.com/arnt/

Full Copyright Statement

Intellectual Property

Acknowledgement

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