Network Working Group
Internet-Draft
Obsoletes: 3454 (if approved)
Intended status: Standards Track
Expires: February 20, 2012

M. Blanchet
Viagenie
P. Saint-Andre
Cisco
August 19, 2011

PRECIS Framework: Handling Internationalized Strings in Protocols
draft-blanchet-precis-framework-03

Abstract

Application protocols that make use of Unicode code points in protocol strings need to prepare such strings in order to perform comparison operations (e.g., for purposes of authentication or authorization). In general, this problem has been labeled the "preparation and comparison of internationalized strings" or "PRECIS". This document defines a framework that enables application protocols to prepare various classes of strings in a way that depends on the properties of Unicode code points. Because this framework does not depend on large tables of Unicode code points as in stringprep (RFC 3454), it is more agile with regard to changes in the underlying Unicode database and thus provides improved flexibility to application protocols. A specification that uses this framework either can directly use the base string classes defined in this document or can subclass the base string classes as needed. This framework uses an approach similar to that of the revised internationalized domain names in applications (IDNA) technology (RFC 5890, RFC 5891, RFC 5892, RFC 5893, RFC 5894) and thus adheres to the high-level design goals described in RFC 4690, albeit for application technologies other than the Domain Name System (DNS). This document obsoletes RFC 3454.

This Internet-Draft will expire on February 20, 2012.

Copyright Notice

Table of Contents

1. Introduction

A number of IETF application technologies use stringprep [RFC3454] as the basis for comparing protocol strings that contain Unicode characters or "code points" [UNICODE]. Since the publication of [RFC3454] in 2002, the Internet community has gained much more experience with internationalization, some of it reflected in [RFC4690]. In particular, the IETF's technology for internationalized domain names (IDNs) has changed significantly: IDNA2003 [RFC3490], which was based on stringprep, has been superseded by IDNA2008 ([RFC5890], [RFC5891], [RFC5892], [RFC5893], [RFC5894]), which does not use stringprep. This migration away from stringprep for internationalized domain names has prompted other "customers" of stringprep to consider new approaches to the preparation and comparison of internationalized strings ("PRECIS"), as described in [PROBLEM].

This document proposes a technical framework for a post-stringprep approach to the preparation and comparison of internationalized strings in application protocols. The framework is based on several principles:

1. Define a small set of base string classes appropriate for common application protocol constructs such as usernames, passwords, and free-form identifiers.

2. Define each base string class in terms of Unicode code points and their properties, specifying whether each code point or character category is valid, disallowed, or unassigned.

3. Enable application protocols to subclass the base string classes, mainly to disallow particular code points that are currently disallowed in the relevant application protocol (e.g., characters with special or reserved meaning, such as "@" and "/" when used as separators within identifiers).

4. Leave various mapping operations (e.g., case preservation or lowercasing, Unicode normalization, right-to-left characters) as the responsibility of application protocols, as was done for IDNA2008 via [RFC5895].

It is expected that this framework will yield the following benefits:

- Application protocols will be more version-agile with regard to the Unicode database.
- Implementers will be able to share code point tables and software code across application protocols, most likely by means of software libraries.

- o End users will be able to acquire more accurate expectations about the code points that are acceptable in various contexts. Given this more uniform set of string classes, it is also expected that copy/paste operations between software implementing different application protocols will be more predictable and coherent.

Although this framework is similar to IDNA2008 and borrows some of the character categories defined in [RFC5892], it defines additional string classes and character categories to meet the needs of common application protocols.

2. Terminology

Many important terms used in this document are defined in [PROBLEM], [I18N-TERMS], [RFC5890], and [UNICODE].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. String Classes

IDNA2008 essentially defines a base string class of internationalized domain name, although it does not use the term "string class". (This document does not define a string class for domain names, and application protocols are strongly encouraged to use IDNA2008 as the appropriate method to prepare domain names and hostnames.)

We propose the following additional base string classes for use in application protocols:

NameClass: a sequence of letters, numbers, and symbols that is used to identify or address a network entity such as a user, an account, a venue (e.g., a chatroom), an information source (e.g., a data feed), or a collection of data (e.g., a file).

SecretClass: a sequence of letters, numbers, and symbols that is used as a secret for access to some resource on a network (e.g., a password or passphrase).

FreeClass: a sequence of letters, numbers, symbols, spaces, and other code points that is used for more expressive purposes in an application protocol (e.g., a free-form identifier such as a human-friendly nickname in a chatroom).

The following subsections discuss these string classes in more detail, with reference to the dimensions described in Section 3 of [PROBLEM].

Each string class is defined by the following behavioral rules:

Valid: defines which code points and character categories are treated as valid input to preparation of the string.

Disallowed: defines which code points and character categories are treated as disallowed during preparation of the string.

Unassigned: defines application behavior in the presence of code points that are unassigned, i.e. unknown for the version of Unicode the application is built upon.

Directionality: defines application behavior in the presence of code points that have directionality, in particular right-to-left code points as defined in the Unicode database (see [UAX9]).

Casemapping: defines if case mapping is used for this class, and how the mapping is done.

Normalization: defines which Unicode normalization form (D, KD, C, or KC) is to be applied (see [UAX15]).

This document defines the valid, disallowed, and unassigned rules. Application protocols that use the PRECIS string classes MUST define the directionality, casemapping, and normalization rules, as further described under Section 9.2.

3.1. NameClass

Most application technologies need a special class of strings that can be used to refer to, include, or communicate things like usernames, chatroom names, file names, and data feed names. We group such things into a bucket called "NameClass" having the following features.

3.1.1. Valid

- Letters and numbers, i.e., the LetterDigits ("A") category first defined in [RFC5892] and listed here under Section 6.1.
- Code points in the range U+0021 through U+007E, i.e., the ASCII7 ("K") rule defined under Section 6.11. These code points are valid even if they would otherwise be disallowed according to the property-based rules specified in the next section.

3.1.2. Disallowed

- Control characters, i.e., the Controls ("L") category defined under Section 6.12.
- Space characters, i.e., the Spaces ("N") category defined under Section 6.14.
- Symbol characters, i.e., the Symbols ("O") category defined under Section 6.15.
- Punctuation characters, i.e., the Punctuation ("P") category defined under Section 6.16.
- Any character that has a compatibility equivalent, i.e., the HasCompat ("Q") category defined under Section 6.17. These code points are disallowed even if they would otherwise be valid according to the property-based rules specified in the previous section.

3.1.3. Unassigned

Any code points that are not yet assigned in the Unicode character set SHALL be considered Unassigned for purposes of the NameClass.

3.1.4. Directionality

The directionality rule MUST be specified by each application protocol that uses or subclasses the NameClass.

3.1.5. Case Mapping

The casemapping rule MUST be specified by each application protocol that uses or subclasses the NameClass.

3.1.6. Normalization

The normalization form MUST be specified by each application protocol that uses or subclasses the NameClass.

However, in accordance with [RFC5198], normalization form C (NFC) is RECOMMENDED.

3.2. SecretClass

Many application technologies need a special class of strings that can be used to communicate secrets of the kind that are typically used as passwords or passphrases. We group such things into a bucket called "SecretClass" having the following features.

NOTE: Consult Section 10.4 for relevant security considerations.

3.2.1. Valid

- Letters and numbers, i.e., the LetterDigits ("A") category first defined in [RFC5892] and listed here under Section 6.1.
- Code points in the range U+0021 through U+007E, i.e., the ASCII7 ("K") rule defined under Section 6.11. These code points are valid even if they would otherwise be disallowed according to the property-based rules specified in the next section.
- Any character that has a compatibility equivalent, i.e., the HasCompat ("Q") category defined under Section 6.17.
- Symbol characters, i.e., the Symbols ("O") category defined under Section 6.15.
- Punctuation characters, i.e., the Punctuation ("P") category defined under Section 6.16.

3.2.2. Disallowed

- Control characters, i.e., the Controls ("L") category defined under Section 6.12.
- Space characters, i.e., the Spaces ("N") category defined under Section 6.14.

3.2.3. Unassigned

Any code points that are not yet assigned in the Unicode character set SHALL be considered Unassigned for purposes of the SecretClass.

3.2.4. Directionality

The directionality rule MUST be specified by each application protocol that uses or subclasses the SecretClass.

3.2.5. Case Mapping

The casemapping rule MUST be specified by each application protocol that uses or subclasses the SecretClass.

However, in order to maximize the entropy of passwords and passphrases, it is NOT RECOMMENDED for application protocols to map uppercase and titlecase code points to their lowercase equivalents; instead, it is RECOMMENDED to preserve the case of all code points contained in string that conform to or subclass the SecretClass.

3.2.6. Normalization

The normalization form MUST be specified by each application protocol that uses or subclasses the SecretClass.

However, in accordance with [RFC5198], normalization form C (NFC) is RECOMMENDED.

3.3. FreeClass

Some application technologies need a special class of strings that can be used in a free-form way (e.g., a nickname in a chatroom). We group such things into a bucket called "FreeClass" having the following features.

3.3.1. Valid

- Letters and numbers, i.e., the LetterDigits ("A") category first defined in [RFC5892] and listed here under Section 6.1.
- Code points in the range U+0021 through U+007E, i.e., the ASCII7 ("K") rule defined under Section 6.11.
- Any character that has a compatibility equivalent, i.e., the HasCompat ("Q") category defined under Section 6.17.
- Space characters, i.e., the Spaces ("N") category defined under Section 6.14.
- Symbol characters, i.e., the Symbols ("O") category defined under Section 6.15.
- Punctuation characters, i.e., the Punctuation ("P") category defined under Section 6.16.

3.3.2. Disallowed

- Control characters, i.e., the Controls ("L") category defined under Section 6.12.

3.3.3. Unassigned

Any code points that are not yet assigned in the Unicode character set SHALL be considered Unassigned for purposes of the FreeClass.

3.3.4. Directionality

The directionality rule MUST be specified by each application protocol that uses or subclasses the FreeClass.

3.3.5. Case Mapping

The casemapping rule MUST be specified by each application protocol that uses or subclasses the FreeClass.

3.3.6. Normalization

The normalization form MUST be specified by each application protocol that uses or subclasses the FreeClass.

However, in accordance with [RFC5198], normalization form C (NFC) is RECOMMENDED.

4. Use of PRECIS String Classes

4.1. Principles

This document defines the valid, disallowed, and unassigned rules. Application protocols that use the PRECIS string classes MUST define the directionality, casemapping, and normalization rules. Such definitions MUST at a minimum specify the following:

Directionality: Whether any instance of the class that contains a right-to-left code point is to be considered a right-to-left string, or whether some other rule is to be applied (e.g., the "Bidi Rule" from [RFC5893]).

Casemapping: Whether uppercase and titlecase code points are to be (a) preserved or (b) mapped to lowercase.

Normalization: Which Unicode normalization form (D, KD, C, or KC) is to be applied (see [UAX15] for background information); in accordance with [RFC5198], NFC is RECOMMENDED.

4.2. Subclassing

Application protocols are allowed to subclass the base string classes specified in this document. As the word "subclass" implies, a subclass MUST NOT add as valid any code points or character categories that are disallowed by the base string class. However, a subclass MAY do either of the following:

1. Exclude specific code points that are included in the base string class.
2. Exclude characters matching certain Unicode properties (e.g., math symbols) that are included in the base string class.

4.3. Registration

Application protocols that use the PRECIS string classes MUST register with the IANA as described under Section 9.2. This is especially important for protocols that subclass the PRECIS string

classes.

5. Code Point Properties

In order to implement the string classes described above, this document does the following:

1. Reviews and classifies the collections of code points in the Unicode character set by examining various code point properties.

2. Defines an algorithm for determining a derived property value, which can vary depending on the string class being used by the relevant application protocol.

This document is not intended to specify precisely how derived property values are to be applied in protocol strings. That information should be defined in the protocol specification that uses or subclasses a base string class from this document.

The value of the property is to be interpreted as follows.

PROTOCOL VALID Those code points that are allowed to be used in any PRECIS string class (NameClass, SecretClass, and FreeClass). Code points with this property value are permitted for general use in any string class. The abbreviated term PVALID is used to refer to this value in the remainder of this document.

SPECIFIC CLASS PROTOCOL VALID Those code points that are allowed to be used in specific string classes. Code points with this property value are permitted for use in specific string classes. In the remainder of this document, the abbreviated term *_PVALID is used, where * = (NAMECLASS | SECRETCLASS | FREECLASS).

CONTEXTUAL RULE REQUIRED Some characteristics of the character, such as its being invisible in certain contexts or problematic in others, require that it not be used in labels unless specific other characters or properties are present. The abbreviated term CONTEXT is used to refer to this value in the remainder of this document. There are two subdivisions of CONTEXTUAL RULE REQUIRED, the first for Join_controls (called CONTEXTJ) and the second for other characters (called CONTEXTO).

DISALLOWED Those code points that must not be included in any string class. Code points with this property value are not permitted in any string class.

SPECIFIC CLASS DISALLOWED Those code points that are not to be included in a specific string class. Code points with this property value are not permitted in one of the string classes but might be permitted in others. In the remainder of this document, the abbreviated term *_DISALLOWED is used, where * = (NAMECLASS | SECRETCLASS | FREECLASS).

UNASSIGNED Those code points that are not designated (i.e. are unassigned) in the Unicode Standard.

The mechanisms described here allow determination of the value of the property for future versions of Unicode (including characters added after Unicode 5.2 or 6.0 depending on the category, since some categories in this document are reused from IDNA2008). Changes in Unicode properties that do not affect the outcome of this process do not affect this framework. For example, a character can have its Unicode General_Category value [UNICODE] change from So to Sm, or from Lo to Ll, without affecting the algorithm results. Moreover, even if such changes were to result, the BackwardCompatible list (Section 6.7) can be adjusted to ensure the stability of the results.

Some code points need to be allowed in exceptional circumstances, but should be excluded in all other cases; these rules are also described in other documents. The most notable of these are the Join Control characters, U+200D ZERO WIDTH JOINER and U+200C ZERO WIDTH NON-JOINER. Both of them have the derived property value CONTEXTJ. A character with the derived property value CONTEXTJ or CONTEXTO (CONTEXTUAL RULE REQUIRED) is not to be used unless an appropriate rule has been established and the context of the character is consistent with that rule. It is invalid to generate a string containing these characters unless such a contextual rule is found and satisfied. PRECIS does not define its own contextual rules, but instead re-uses the contextual rules defined for IDNA2008; please see Appendix A of [RFC5892] for more information.

6. Category Definitions Used to Calculate Derived Property Value

The derived property obtains its value based on a two-step procedure:

1. Characters are placed in one or more character categories either (1) based on core properties defined by the Unicode Standard or (2) by treating the code point as an exception and addressing the code point as its code point value. These categories are not mutually exclusive.

2. Set operations are used with these categories to determine the values for a property that is specific to a given string class. These operations are specified under Section 7.

(NOTE: Unicode property names and property value names might have short abbreviations, such as "gc" for the General_Category property and "Ll" for the Lowercase_Letter property value of the gc property.)

In the following specification of character categories, the operation that returns the value of a particular Unicode character property for a code point is designated by using the formal name of that property (from the Unicode PropertyAliases.txt [1]) followed by '(cp)' for "code point". For example, the value of the General_Category property for a code point is indicated by General_Category(cp).

The first ten categories (A-J) shown below were previously defined for IDNA2008 and are copied directly from [RFC5892]. Some of these categories are reused in PRECIS and some of them are not; however, the lettering of categories is retained to prevent overlap and to ease implementation of both IDNA2008 and PRECIS in a single software application. The next seven categories (K-Q) are specific to PRECIS.

6.1. LetterDigits (A)

NOTE: This category is defined in [RFC5892] and copied here for use in PRECIS.

A: General_Category(cp) is in {Ll, Lu, Lo, Nd, Lm, Mn, Mc}

These rules identify characters commonly used in mnemonics and often informally described as "language characters".

For more information, see section 4.5 of [UNICODE].

The categories used in this rule are:
- Ll - Lowercase_Letter
- Lu - Uppercase_Letter
- Lo - Other_Letter
- Nd - Decimal_Number
- Lm - Modifier_Letter
- Mn - Nonspacing_Mark
- Mc - Spacing_Mark

6.2. Unstable (B)

NOTE: This category is defined in [RFC5892] but not used in PRECIS.

6.3. IgnorableProperties (C)

NOTE: This category is defined in [RFC5892] but not used in PRECIS. See the "PrecisIgnorableProperties (M)" category below for a more inclusive category used in PRECIS identifiers.

6.4. IgnorableBlocks (D)

NOTE: This category is defined in [RFC5892] but not used in PRECIS.

6.5. LDH (E)

NOTE: This category is defined in [RFC5892] but not used in PRECIS. See the "ASCII7 (K)" category below for a more inclusive category used in PRECIS identifiers.

6.6. Exceptions (F)

NOTE: This category is defined in [RFC5892] and might be used in a future version of this specification.

```
F: cp is in {00B7, 00DF, 0375, 03C2, 05F3, 05F4, 0640, 0660,
             0661, 0662, 0663, 0664, 0665, 0666, 0667, 0668,
             0669, 06F0, 06F1, 06F2, 06F3, 06F4, 06F5, 06F6,
             06F7, 06F8, 06F9, 06FD, 06FE, 07FA, 0F0B, 3007,
             302E, 302F, 3031, 3032, 3033, 3034, 3035, 303B,
             30FB}
```

This category explicitly lists code points for which the category cannot be assigned using only the core property values that exist in the Unicode standard. The values are according to the table below:

PVALID -- Would otherwise have been DISALLOWED

```
00DF; PVALID     # LATIN SMALL LETTER SHARP S
03C2; PVALID     # GREEK SMALL LETTER FINAL SIGMA
06FD; PVALID     # ARABIC SIGN SINDHI AMPERSAND
06FE; PVALID     # ARABIC SIGN SINDHI POSTPOSITION MEN
0F0B; PVALID     # TIBETAN MARK INTERSYLLABIC TSHEG
3007; PVALID     # IDEOGRAPHIC NUMBER ZERO
```

CONTEXTO -- Would otherwise have been DISALLOWED

```
00B7; CONTEXTO   # MIDDLE DOT
0375; CONTEXTO   # GREEK LOWER NUMERAL SIGN (KERAIA)
05F3; CONTEXTO   # HEBREW PUNCTUATION GERESH
05F4; CONTEXTO   # HEBREW PUNCTUATION GERSHAYIM
30FB; CONTEXTO   # KATAKANA MIDDLE DOT
```

CONTEXTO -- Would otherwise have been PVALID

```
0660; CONTEXTO   # ARABIC-INDIC DIGIT ZERO
0661; CONTEXTO   # ARABIC-INDIC DIGIT ONE
0662; CONTEXTO   # ARABIC-INDIC DIGIT TWO
0663; CONTEXTO   # ARABIC-INDIC DIGIT THREE
0664; CONTEXTO   # ARABIC-INDIC DIGIT FOUR
0665; CONTEXTO   # ARABIC-INDIC DIGIT FIVE
0666; CONTEXTO   # ARABIC-INDIC DIGIT SIX
0667; CONTEXTO   # ARABIC-INDIC DIGIT SEVEN
0668; CONTEXTO   # ARABIC-INDIC DIGIT EIGHT
0669; CONTEXTO   # ARABIC-INDIC DIGIT NINE
06F0; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT ZERO
06F1; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT ONE
06F2; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT TWO
06F3; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT THREE
06F4; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT FOUR
06F5; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT FIVE
06F6; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT SIX
06F7; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT SEVEN
06F8; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT EIGHT
06F9; CONTEXTO   # EXTENDED ARABIC-INDIC DIGIT NINE
```

DISALLOWED -- Would otherwise have been PVALID

```
0640; DISALLOWED # ARABIC TATWEEL
07FA; DISALLOWED # NKO LAJANYALAN
302E; DISALLOWED # HANGUL SINGLE DOT TONE MARK
302F; DISALLOWED # HANGUL DOUBLE DOT TONE MARK
3031; DISALLOWED # VERTICAL KANA REPEAT MARK
3032; DISALLOWED # VERTICAL KANA REPEAT WITH VOICED SOUND MARK
3033; DISALLOWED # VERTICAL KANA REPEAT MARK UPPER HALF
3034; DISALLOWED # VERTICAL KANA REPEAT WITH VOICED SOUND MARK
                   UPPER HA
3035; DISALLOWED # VERTICAL KANA REPEAT MARK LOWER HALF
303B; DISALLOWED # VERTICAL IDEOGRAPHIC ITERATION MARK
```

6.7. BackwardCompatible (G)

NOTE: This category is defined in [RFC5892] and copied here for use in PRECIS. Because of how the PRECIS string classes are defined, only changes that would result in code points being added to or removed from the LetterDigits ("A") category would result in backward-incompatible modifications to code point assignments. Therefore, management of this category is handled via the processes specified in [RFC5892].

G: cp is in {}

This category includes the code points for which property values in versions of Unicode after 5.2 have changed in such a way that the derived property value would no longer be PVALID or DISALLOWED. If changes are made to future versions of Unicode so that code points might change property value from PVALID or DISALLOWED, then this table can be updated and keep special exception values so that the property values for code points stay stable.

6.8. JoinControl (H)

NOTE: This category is defined in [RFC5892] and copied here for use in PRECIS.

H: Join_Control(cp) = True

This category consists of Join Control characters (i.e., they are not in LetterDigits (Section 6.1)) but are still required in strings under some circumstances.

6.9. OldHangulJamo (I)

NOTE: This category is defined in [RFC5892] and copied here for use in PRECIS.

I: Hangul_Syllable_Type(cp) is in {L, V, T}

This category consists of all conjoining Hangul Jamo (Leading Jamo, Vowel Jamo, and Trailing Jamo).

Elimination of conjoining Hangul Jamos from the set of PVALID characters results in restricting the set of Korean PVALID characters just to preformed, modern Hangul syllable characters. Old Hangul syllables, which must be spelled with sequences of conjoining Hangul Jamos, are not PVALID for string classes.

6.10. Unassigned (J)

NOTE: This category is defined in [RFC5892] and copied here for use in PRECIS.

J: General_Category(cp) is in {Cn} and
Noncharacter_Code_Point(cp) = False

This category consists of code points in the Unicode character set that are not (yet) assigned. It should be noted that Unicode distinguishes between 'unassigned code points' and 'unassigned characters'. The unassigned code points are all but (Cn - Noncharacters), while the unassigned *characters* are all but (Cn +

Cs).

6.11. ASCII7 (K)

This PRECIS-specific category exempts most characters in the ASCII-7 range from other rules that might be applied during PRECIS processing, on the assumption that these code points are in such wide use that disallowing them would be counter-productive.

K: cp is in {0021..007E}

6.12. Controls (L)

L: Control(cp) = True

6.13. PrecisIgnorableProperties (M)

This PRECIS-specific category is used to group code points that are not recommended for use in PRECIS string classes.

M: Default_Ignorable_Code_Point(cp) = True or
Noncharacter_Code_Point(cp) = True

The definition for Default_Ignorable_Code_Point can be found in the DerivedCoreProperties.txt [2] file, and at the time of Unicode 6.0 is as follows:

```
  Other_Default_Ignorable_Code_Point
+ Cf (Format characters)
+ Variation_Selector
- White_Space
- FFF9..FFFB (Annotation Characters)
- 0600..0603, 06DD, 070F (exceptional Cf characters
                          that should be visible)
```

6.14. Spaces (N)

This PRECIS-specific category is used to group code points that are space characters.

N: General_Category(cp) is in {Zs}

6.15. Symbols (O)

This PRECIS-specific category is used to group code points that are symbols.

O: General_Category(cp) is in {Sc}

6.16. Punctuation (P)

This PRECIS-specific category is used to group code points that are punctuation marks.

P: General_Category(cp) is in {Pi}

6.17. HasCompat (Q)

This PRECIS-specific category is used to group code points that have compatibility equivalents as explained in Chapter 2 and Chapter 3 of [UNICODE].

Q: toNFKC(cp) != cp

The toNFKC() operation returns the code point in normalization form KC. For more information, see Section 5 of [UAX15].

7. Calculation of the Derived Property

Possible values of the derived property are:

- PVALID
- NAMECLASS_VALID
- SECRETCLASS_VALID
- FREECLASS_VALID
- CONTEXTJ
- CONTEXTO
- DISALLOWED
- NAMECLASS_DISALLOWED
- SECRETCLASS_DISALLOWED
- FREECLASS_DISALLOWED
- UNASSIGNED

NOTE: In some instances, the value of the derived property calculated depends on the string class (e.g., if an identifier used in an application protocol is defined as using or subclassing the PRECIS NameClass, then a space character would be assigned to NAMECLASS_DISALLOWED).

The algorithm to calculate the value of the derived property is as follows. (NOTE: Use of the name of a rule (such as "Exception") implies the set of code points that the rule defines, whereas the same name as a function call (such as "Exception(cp)") implies the value that the code point has in the Exceptions table.)

```
If .cp. .in. Exceptions Then Exceptions(cp);
Else If .cp. .in. BackwardCompatible Then BackwardCompatible(cp);
Else If .cp. .in. Unassigned Then UNASSIGNED;
Else If .cp. .in. ASCII7 Then PVALID;
Else If .cp. .in. JoinControl Then CONTEXTJ;
Else If .cp. .in. PrecisIgnorableProperties Then DISALLOWED;
Else If .cp. .in. Controls Then DISALLOWED;
Else If .cp. .in. OldHangulJamo Then DISALLOWED;
Else If .cp. .in. LetterDigits Then PVALID;
Else If .cp. .in. Spaces Then NAMECLASS_DISALLOWED
                    or SECRETCLASS_DISALLOWED
                    or FREECLASS_VALID;
Else If .cp. .in. Symbols Then NAMECLASS_DISALLOWED
                    or SECRETCLASS_DISALLOWED
                    or FREECLASS_VALID;
Else If .cp. .in. Punctuation Then NAMECLASS_DISALLOWED
                    or SECRETCLASS_DISALLOWED
                    or FREECLASS_VALID;
Else If .cp. .in. HasCompat Then NAMECLASS_DISALLOWED
                    or SECRETCLASS_VALID
                    or FREECLASS_VALID;
Else DISALLOWED;
```

8. Code Points

The Categories and Rules defined in Section 6 and Section 7 apply to all Unicode code points. The table in Section 12 shows, for illustrative purposes, the consequences of the categories and classification rules, and the resulting property values.

The list of code points that can be found in Section 12 is non-normative. Instead, the rules defined by Section 6 and Section 7 are normative, and any tables are derived from the rules.

9. IANA Considerations

9.1. PRECIS Derived Property Value Registry

IANA is requested to create a PRECIS-specific registry with the Derived Properties for the versions of Unicode that are released after (and including) version 6.0. The derived property value is to be calculated in cooperation with a designated expert [RFC5226] according to the specifications in Section 6 and Section 7, and not by copying the non-normative table found in Section 12.

If during this process (creation of the table of derived property

values) followed by a designated expert review, either backward-incompatible changes to the table of derived properties are discovered, or otherwise problems during the creation of the table arises, that is to be flagged to the IESG. Changes to the rules (as specified in Section 6 and Section 7) require IETF Review, as described in [RFC5226].

9.2. PRECIS Usage Registry

IANA is requested to create a registry of application protocols that use the base string classes. The registry will include one entry for each use (e.g., if a protocol uses both the NameClass and the FreeClass then the specification for that protocol would submit two registrations). In accordance with [RFC5226], the registration policy is "First Come First Served".

The registration template is as follows:

Application Protocol: [the application protocol that is using or subclassing the PRECIS string class]
Base Class: [which base class is being used]
Subclassing: [whether the base class is being subclassed and, if so, where documentation of the subclassing can be found]
Directionality: [the behavioral rule for handling of right-to-left code points]
Casemapping: [the behavioral rule for handling of case]
Normalization: [which Unicode normalization form is applied]
Specification: [a pointer to relevant documentation, such as an RFC or Internet-Draft]

10. Security Considerations

10.1. General Issues

The security of applications that use this framework can depend in part on the proper preparation and comparison of internationalized strings. For example, such strings can be used to make authentication and authorization decisions, and the security of an application could be compromised if an entity providing a given string is connected to the wrong account or online resource based on different interpretations of the string.

Specifications of application protocols that use this framework are encouraged to describe how internationalized strings are used in the protocol, including the security implications of any false positives and false negatives that might result from various comparison operations. For some helpful guidelines, refer to [IDENTIFIER],

[RFC5890], [UTR36], and [UTR39].

10.2. Local Character Set Issues

When systems use local character sets other than ASCII and Unicode, these specifications leave the problem of converting between the local character set and Unicode up to the application or local system. If different applications (or different versions of one application) implement different rules for conversions among coded character sets, they could interpret the same name differently and contact different application servers or other network entities. This problem is not solved by security protocols, such as Transport Layer Security (TLS) [RFC5246] and the Simple Authentication and Security Layer (SASL) [RFC4422], that do not take local character sets into account.

10.3. Visually Similar Characters

Some characters are visually similar and thus can cause confusion among humans. Such characters are often called "confusable characters" or "confusables".

The problem of confusable characters is not necessarily caused by the use of Unicode code points outside the US-ASCII range. For example, in some presentations and to some individuals the string "ju1iet" (spelled with the Arabic numeral one as the third character) might appear to be the same as "juliet" (spelled with the lowercase version of the letter "L"), especially on casual visual inspection. This phenomenon is sometimes called "typejacking".

However, the problem is made more serious by introducing the full range of Unicode code points into protocol strings. For example, the characters U+13DA U+13A2 U+13B5 U+13AC U+13A2 U+13AC U+13D2 from the Cherokee block look similar to the US-ASCII characters "STPETER" as they might look when presented in a "creative" font.

In some examples of confusable characters, it is unlikely that the average human could tell the difference between the real string and the fake string. (Indeed, there is no programmatic way to distinguish with full certainty which is the fake string and which is the real string; in some contexts, the string formed of Cherokee characters might be the real string and the string formed of US-ASCII characters might be the fake string.) Because PRECIS-compliant strings can contain almost any properly encoded Unicode code point, it can be relatively easy to fake or mimic some strings in systems that use the PRECIS framework. The fact that some strings are easily confused introduces security vulnerabilities of the kind that have also plagued the World Wide Web, specifically the phenomenon known as

phishing.

Despite the fact that some specific suggestions about identification and handling of confusable characters appear in the Unicode Security Considerations [UTR36], it is also true (as noted in [RFC5890]) that "there are no comprehensive technical solutions to the problems of confusable characters". Because it is impossible to map visually similar characters without a great deal of context (such as knowing the fonts used), the PRECIS framework does nothing to map similar-looking characters together, nor does it prohibit some characters because they look like others.

However, specifications for application protocols that use this framework MUST describe how confusable characters can be used to compromise the security of systems that use the protocol in question, and any protocol-specific suggestions for overcoming those threats. In particular, software implementations and service deployments that use PRECIS-based technologies are strongly encouraged to define and implement consistent policies regarding the registration, storage, and presentation of visually similar characters. The following recommendations are appropriate:

1. An application service SHOULD define a policy that specifies the scripts or blocks of characters that the service will allow to be registered (e.g., in an account name) or stored (e.g., in a file name). Such a policy SHOULD be informed by the languages and scripts that are used to write registered account names; in particular, to reduce confusion, the service SHOULD forbid registration or storage of stings that contain characters from more than one script and to restrict registrations to characters drawn from a very small number of scripts (e.g., scripts that are well-understood by the administrators of the service, to improve manageability).

2. User-oriented application software SHOULD define a policy that specifies how internationalized strings will be presented to a human user. Because every human user of such software has a preferred language or a small set of preferred languages, the software SHOULD gather that information either explicitly from the user or implicitly via the operating system of the user’s device. Furthermore, because most languages are typically represented by a single script or a small set of scripts, and because and most scripts are typically contained in one or more blocks of characters, the software SHOULD warn the user when presenting a string that mixes characters from more than one script or block, or that uses characters outside the normal range of the user’s preferred language(s). (Such a recommendation is not intended to discourage communication across different

communities of language users; instead, it recognizes the existence of such communities and encourages due caution when presenting unfamiliar scripts or characters to human users.)

10.4. Security of the SecretClass

One goal of passwords and passphrases is to maximize the amount of entropy, for example by allowing a wide range of code points and by ensuring that secrets are not prepared in such a way that code points are compared aggressively. Therefore, it is NOT RECOMMENDED for application protocols to subclass the SecretClass in a way that removes entire categories (e.g., by disallowing symbols or punctuation). Furthermore, it is NOT RECOMMENDED for application protocols to map uppercase and titlecase code points to their lowercase equivalents; instead, it is RECOMMENDED to preserve the case of all code points contained in string that conform to or subclass the SecretClass.

That said, software implementers need to be aware that there exist tradeoffs between entropy and usability. For example, allowing a user to establish a password containing "uncommon" code points might make it difficult for the user to access an application when using an unfamiliar or constrained input device.

Some application protocols use passwords and passphrases directly, whereas others reuse technologies that themselves process passwords (one example is the Simple Authentication and Security Layer [RFC4422]). Moreover, passwords are often carried by a sequence of protocols with backends authentication systems or data storage systems such as RADIUS [RFC2865] and LDAP [RFC4510]. Developers of application protocols are encouraged to look into reusing these profiles instead of defining new ones, so that end-user expectations about passwords are consistent no matter which application protocol is used.

11. Acknowledgements

The authors would like to acknowledge the comments and contributions of the following individuals: David Black, Mark Davis, Alan DeKok, Martin Duerst, Patrik Faltstrom, Ted Hardie, Joe Hildebrand, Paul Hoffman, Jeffrey Hutzelman, Simon Josefsson, John Klensin, Alexey Melnikov, Pete Resnick, Andrew Sullivan, and Dave Thaler.

Some algorithms and textual descriptions have been borrowed from [RFC5892]. Some text regarding security has been borrowed from [RFC5890] and [XMPP-ADDR].

12. Codepoints 0x0000 - 0x10FFFF

To follow.

12.1. Codepoints in Unicode Character Database (UCD) format

To follow.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 6.0", 2010, <http://www.unicode.org/versions/Unicode6.0.0/>.

13.2. Informative References

[I18N-TERMS] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", draft-ietf-appsawg-rfc3536bis-06 (work in progress), July 2011.

[IDENTIFIER] Thaler, D., "Issues in Identifier Comparison for Security Purposes", draft-iab-identifier-comparison-00 (work in progress), July 2011.

[PROBLEM] Blanchet, M. and A. Sullivan, "Stringprep Revision Problem Statement", draft-ietf-precis-problem-statement-03 (work in progress), July 2011.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

[RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and Recommendations for Internationalized Domain Names (IDNs)", RFC 4690, September 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.

[RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.

[RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.

[RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.

[UAX15] The Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", September 2010, <http://unicode.org/reports/tr15/>.

[UAX9] The Unicode Consortium, "Unicode Standard Annex #9: Unicode Bidirectional Algorithm", September 2010, <http://unicode.org/reports/tr9/>.

[UTR36] The Unicode Consortium, "Unicode Technical Report #36: Unicode Security Considerations", August 2010, <http://unicode.org/reports/tr36/>.

[UTR39] The Unicode Consortium, "Unicode Technical Report #39: Unicode Security Mechanisms", August 2010, <http://unicode.org/reports/tr39/>.

[XMPP-ADDR] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Address Format", draft-saintandre-xmpp-6122bis-01 (work in progress), July 2011.

URIs

[1] <http://unicode.org/Public/UNIDATA/PropertyAliases.txt>

[2] <http://unicode.org/Public/UNIDATA/DerivedCoreProperties.txt>

Authors' Addresses

Marc Blanchet
Viagenie
2600 boul. Laurier, suite 625
Quebec, QC G1V 4W1
Canada

Email: Marc.Blanchet@viagenie.ca
URI: http://www.viagenie.ca/

Peter Saint-Andre
Cisco
1899 Wyknoop Street, Suite 600
Denver, CO 80202
USA

Phone: +1-303-308-3282
Email: psaintan@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 26, 2013

M. Blanchet
Viagenie
A. Sullivan
Dyn, Inc.

January 22, 2013

Stringprep Revision and PRECIS Problem Statement
draft-ietf-precis-problem-statement-09.txt

Abstract

If a protocol expects to compare two strings and is prepared only for those strings to be ASCII, then using Unicode codepoints in those strings requires they be prepared somehow. Internationalizing Domain Names in Applications (here called IDNA2003) defined and used Stringprep and Nameprep. Other protocols subsequently defined Stringprep profiles. A new approach different from Stringprep and Nameprep is used for a revision of IDNA2003 (called IDNA2008). Other Stringprep profiles need to be similarly updated or a replacement of Stringprep needs to be designed. This document outlines the issues to be faced by those designing a Stringprep replacement.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 26, 2013.

Table of Contents

1. Introduction

Internationalizing Domain Names in Applications (here called IDNA2003) [RFC3490], [RFC3491], [RFC3492], [RFC3454] describes a mechanism for encoding Unicode labels making up Internationalized Domain Names (IDNs) as standard DNS labels. The labels were processed using a method called Nameprep [RFC3491] and Punycode [RFC3492]. That method was specific to IDNA2003, but is generalized as Stringprep [RFC3454]. The general mechanism is used by other protocols with similar needs, but with different constraints than IDNA2003.

Stringprep defines a framework within which protocols define their Stringprep profiles. Some known IETF specifications using Stringprep are listed below:

- The Nameprep profile [RFC3490] for use in Internationalized Domain Names (IDNs);
- IAX using Nameprep [RFC5456];
- NFSv4 [RFC3530] and NFSv4.1 [RFC5661];
- The iSCSI profile [RFC3722] for use in Internet Small Computer Systems Interface (iSCSI) Names;
- EAP [RFC3748];
- The Nodeprep and Resourceprep profiles [RFC3920] for use in the Extensible Messaging and Presence Protocol (XMPP), and the XMPP to CPIM mapping [RFC3922] (the latter of these relies on the former);
- IRI and URI in XMPP [RFC5122];
- The Policy MIB profile [RFC4011] for use in the Simple Network Management Protocol (SNMP);
- TLS [RFC4279];
- The LDAP profile [RFC4518] for use with LDAP [RFC4511] and its authentication methods [RFC4513];
- PKIX subject identification using LDAPprep [RFC4683];
- PKIX CRL using LDAPprep [RFC5280];
- The SASLprep profile [RFC4013] for use in the Simple Authentication and Security Layer (SASL), and SASL itself [RFC4422];
- Plain SASL using SASLprep [RFC4616];
- SMTP Auth using SASLprep [RFC4954];
- POP3 Auth using SASLprep [RFC5034];
- TLS SRP using SASLprep [RFC5054];
- SASL SCRAM using SASLprep [RFC5802];
- Remote management of Sieve using SASLprep [RFC5804];
- NNTP using SASLprep [RFC4643];
- IMAP4 using SASLprep [RFC4314];
- The trace profile [RFC4505] for use with the SASL ANONYMOUS mechanism;

- Internet Application Protocol Collation Registry [RFC4790];
- The unicode-casemap Unicode Collation [RFC5051].

However, a review (see [ietf78precis]) of these protocol specifications found that they are very similar and can be grouped into a short number of classes. Moreover, many reuse the same Stringprep profile, such as the SASL one.

IDNA2003 was replaced because of some limitations described in [RFC4690]. The new IDN specification, called IDNA2008 [RFC5890], [RFC5891], [RFC5892], [RFC5893] was designed based on the considerations found in [RFC5894]. One of the effects of IDNA2008 is that Nameprep and Stringprep are not used at all. Instead, an algorithm based on Unicode properties of codepoints is defined. That algorithm generates a stable and complete table of the supported Unicode codepoints for each Unicode version. This algorithm uses an inclusion-based approach, instead of the exclusion-based approach of Stringprep/Nameprep. That is, IDNA2003 created an explicit list of excluded or mapped-away characters; anything in Unicode 3.2 that was not so listed could be assumed to be allowed under the protocol. IDNA2008 begins instead from the assumption that code points are disallowed, and then relies on Unicode properties to derive whether a given code point actually is allowed in the protocol.

This document lists the shortcomings and issues found by protocols listed above that defined Stringprep profiles. It also lists the requirements for any potential replacement of Stringprep.

2. Keywords

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 [RFC2119].

This document uses various internationalization terms, which are defined and discussed in [RFC6365].

Additionally, this document defines the following keyword:

- PRECIS: Preparation and Comparison of Internationalized Strings

3. Conventions

A single Unicode code point in this memo is denoted by "U+" followed by four to six hexadecimal digits, as used in [Unicode61], Appendix A.

4. Stringprep Profiles Limitations

During IETF 77 (March 2010), a BOF discussed the current state of the protocols that have defined Stringprep profiles [NEWPREP]. The main conclusions from that discussion were as follows:

- Stringprep is bound to version 3.2 of Unicode. Stringprep has not been updated to new versions of Unicode. Therefore, the protocols using Stringprep are stuck at Unicode 3.2, and their specifications need to be updated to support new versions of Unicode.
- The protocols would like to not be bound to a specific version of Unicode, but rather have better Unicode version agility in the way of IDNA2008. This is important partly because it is usually impossible for an application to require Unicode 3.2; the application gets whatever version of Unicode is available on the host.
- The protocols require better bidirectional support (bidi) than currently offered by Stringprep.
- If the protocols are updated to use a new version of Stringprep or another framework, then backward compatibility is an important requirement. For example, Stringprep normalization is based on and profiles may use Unicode Normalization Form KC (NFKC) [UAX15], while IDNA2008 mostly uses Unicode Normalization Form C (NFC) [UAX15].
- Identifiers are passed between protocols. For example, the same username string of codepoints may be passed between SASL, XMPP, LDAP and EAP. Therefore, common set of rules or classes of strings are preferred over specific rules for each protocol. Without real planning in advance, many Stringprep profiles reuse other profiles, so this goal was accomplished by accident with Stringprep.

Protocols that use Stringprep profiles use strings for different purposes:

- XMPP uses a different Stringprep profile for each part of the XMPP address (JID): a localpart which is similar to a username and used for authentication, a domainpart which is a domain name, and a resource part which is less restrictive than the localpart.
- iSCSI uses a Stringprep profile for the names of protocol participants (called initiators and targets). The IQN format of iSCSI names contains a reversed DNS domain name.
- SASL and LDAP uses a Stringprep profile for usernames.
- LDAP uses a set of Stringprep profiles.

The apparent judgement of the BOF attendees [NEWPREP] was that it would be highly desirable to have a replacement of Stringprep, with similar characteristics to IDNA2008. That replacement should be defined so that the protocols could use internationalized strings

without a lot of specialized internationalization work, since internationalization expertise is not available in the respective protocols or working groups. Accordingly, the IESG formed the PRECIS working group to undertake the task.

Notwithstanding the desire evident in [NEWPREP] and the chartering of a working group, IDNA2008 may be a poor model for what other protocols ought to do, because it is designed to support an old protocol that is designed to operate on the scale of the entire Internet. Moreover, IDNA2008 is intended to be deployed without any change to the base DNS protocol. Other protocols may aim at deployment in more local environments, or may have protocol version negotiation built in.

5. Major Topics for Consideration

This section provides an overview of major topics that a Stringprep replacement needs to address. The headings correspond roughly with categories under which known Stringprep-using protocol RFCs have been evaluated. For the details of those evaluations, see Appendix A.

5.1. Comparison

5.1.1. Types of Identifiers

Following [I-D.iab-identifier-comparison], it is possible to organize identifiers into three classes in respect of how they may be compared with one another:

Absolute Identifiers Identifiers that can be compared byte-by-byte for equality.
Definite Identifiers Identifiers that have a well-defined comparison algorithm on which all parties agree.
Indefinite Identifiers Identifiers that have no single comparison algorithm on which all parties agree.

Definite Identifiers include cases like the comparison of Unicode code points in different encodings: they do not match byte for byte, but can all be converted to a single encoding which then does match byte for byte. Indefinite Identifiers are sometimes algorithmically comparable by well-specified subsets of parties. For more discussion of these categories, see [I-D.iab-identifier-comparison].

The section on treating the existing known cases, Appendix A uses the categories above.

5.1.2. Effect of comparison

The three classes of comparison style outlined in Section 5.1.1 may have different effects when applied. It is necessary to evaluate the effects if a comparison results in a false positive, and what the effects are if a comparison results in a false negative, especially in terms of the consequences to security and usability.

5.2. Dealing with characters

This section outlines a range of issues having to do with characters in the target protocols, and outlines the ways in which IDNA2008 might be a good analogy to other protocols, and ways in which it might be a poor one.

5.2.1. Case folding, case sensitivity, and case preservation

In IDNA2003, labels are always mapped to lower case before the Punycode transformation. In IDNA2008, there is no mapping at all: input is either a valid U-label or it is not. At the same time, upper-case characters are by definition not valid U-labels, because they fall into the Unstable category (category B) of [RFC5892].

If there are protocols that require upper and lower cases be preserved, then the analogy with IDNA2008 will break down. Accordingly, existing protocols are to be evaluated according to the following criteria:

1. Does the protocol use case folding? For all blocks of code points, or just for certain subsets?
2. Is the system or protocol case sensitive?
3. Does the system or protocol preserve case?

5.2.2. Stringprep and NFKC

Stringprep profiles may use normalization. If they do, they use NFKC [UAX15] (most profiles do). It is not clear that NFKC is the right normalization to use in all cases. In [UAX15], there is the following observation regarding Normalization Forms KC and KD: "It is best to think of these Normalization Forms as being like uppercase or lowercase mappings: useful in certain contexts for identifying core meanings, but also performing modifications to the text that may not always be appropriate." In general, it can be said that NFKC is more aggressive about finding matches between codepoints than NFC. For things like the spelling of users' names, then, NFKC may not be the best form to use. At the same time, one of the nice things about NFKC is that it deals with the width of characters that are otherwise similar, by canonicalizing half-width to full-width. This mapping

step can be crucial in practice. A replacement for Stringprep depends on analyzing the different use profiles and considering whether NFKC or NFC is a better normalization for each profile.

For the purposes of evaluating an existing example of Stringprep use, it is helpful to know whether it uses no normalization, NFKC, or NFC.

5.2.3. Character mapping

Along with the case mapping issues raised in Section 5.2.1, there is the question of whether some characters are mapped either to other characters or to nothing during Stringprep. [RFC3454], Section 3, outlines a number of characters that are mapped to nothing, and also permits Stringprep profiles to define their own mappings.

5.2.4. Prohibited characters

Along with case folding and other character mappings, many protocols have characters that are simply disallowed. For example, control characters and special characters such as "@" or "/" may be prohibited in a protocol.

One of the primary changes of IDNA2008 is in the way it approaches Unicode code points, using the new inclusion-based approach (see Section 1).

Because of the default assumption in IDNA2008 that a code point is not allowed by the protocol, it has more than one class of "allowed by the protocol"; this is unlike IDNA2003. While some code points are disallowed outright, some are allowed only in certain contexts. The reasons for the context-dependent rules have to do with the way some characters are used. For instance, the ZERO WIDTH JOINER and ZERO WIDTH NON-JOINER (ZWJ, U+200D and ZWNJ, U+200C) are allowed with contextual rules because they are required in some circumstances, yet are considered punctuation by Unicode and would therefore be DISALLOWED under the usual IDNA2008 derivation rules. The goal of IDNA2008 is to provide the widest repertoire of code points possible and consistent with the traditional DNS "LDH" (letters, digits, hyphen; see [RFC0952]) rule, trusting to the operators of individual zones to make sensible (and usually more restrictive) policies for their zones.

5.2.5. Internal structure, delimiters, and special characters

IDNA2008 has a special problem with delimiters, because the delimiter "character" in the DNS wire format is not really part of the data. In DNS, labels are not separated exactly; instead, a label carries with it an indicator that says how long the label is. When the label

is presented in presentation format as part of a fully qualified domain name, the label separator FULL STOP, U+002E (.) is used to break up the labels. But because that label separator does not travel with the wire format of the domain name, there is no way to encode a different, "internationalized" separator in IDNA2008.

Other protocols may include characters with similar special meaning within the protocol. Common characters for these purposes include FULL STOP, U+002E (.); COMMERCIAL AT, U+0040 (@); HYPHEN-MINUS, U+002D (-); SOLIDUS, U+002F (/); and LOW LINE, U+005F (_). The mere inclusion of such a character in the protocol is not enough for it to be considered similar to another protocol using the same character; instead, handling of the character must be taken into consideration as well.

An important issue to tackle here is whether it is valuable to map to or from these special characters as part of the Stringprep replacement. In some locales, the analogue to FULL STOP, U+002E is some other character, and users may expect to be able to substitute their normal stop for FULL STOP, U+002E. At the same time, there are predictability arguments in favour of treating identifiers with FULL STOP, U+002E in them just the way they are treated under IDNA2008.

5.2.6. Restrictions because of glyph similarity

Homoglyphs are similarly (or identically) rendered glyphs of different codepoints. For DNS names, homoglyphs may enable phishing. If a protocol requires some visual comparison by end-users, then the issue of homoglyphs is to be considered. In the DNS context, theses issues are documented in [RFC5894] and [RFC4690]. IDNA2008 does not, however, have a mechanism to deal with them, trusting to DNS zone operators to enact sensible policies for the subset of Unicode they wish to support, given their user community. A similar policy/ protocol split may not be desirable in every protocol.

5.3. Where the data comes from and where it goes

5.3.1. User input and the source of protocol elements

Some protocol elements are provided by users, and others are not. Those that are not may presumably be subject to greater restrictions, whereas those that users provide likely need to permit the broadest range of code points. The following questions are helpful:

1. Do users input the strings directly?
2. If so, how? (keyboard, stylus, voice, copy-paste, etc.)

3. Where do we place the dividing line between user interface and protocol? (see [RFC5895])

5.3.2. User output

Just as only some protocol elements are expected to be entered directly by users, only some protocol elements are intended to be consumed directly by users. It is important to know how users are expected to be able to consume the protocol elements, because different environments present different challenges. An element that is only ever delivered as part of a vCard remains in machine-readable format, so the problem of visual confusion is not a great one. Is the protocol element published as part of a vCard, a web directory, on a business card, or on "the side of a bus"? Do users use the protocol element as an identifier (which means that they might enter it again in some other context)? (See also Section 5.2.6.)

5.3.3. Operations

Some strings are useful as part of the protocol but are not used as input to other operations (for instance, purely informative or descriptive text). Other strings are used directly as input to other operations (such as cryptographic hash functions), or are used together with other strings to (such as concatenating a string with some others to form a unique identifier).

5.3.3.1. String classes

Strings often have a similar function in different protocols. For instance, many different protocols contain user identifiers or passwords. A single profile for all such uses might be desirable.

Often, a string in a protocol is effectively a protocol element from another protocol. For instance, different systems might use the same credentials database for authentication.

5.3.3.2. Community Considerations

A Stringprep replacement that does anything more than just update Stringprep to the latest version of Unicode will probably entail some changes. It is important to identify the willingness of the protocol-using community to accept backwards-incompatible changes. By the same token, it is important to evaluate the desire of the community for features not available under Stringprep.

5.3.3.3. Unicode Incompatible Changes

IDNA2008 uses an algorithm to derive the validity of a Unicode code point for use under IDNA2008. It does this by using the properties of each code point to test its validity.

This approach depends crucially on the idea that code points, once valid for a protocol profile, will not later be made invalid. That is not a guarantee currently provided by Unicode. Properties of code points may change between versions of Unicode. Rarely, such a change could cause a given code point to become invalid under a protocol profile, even though the code point would be valid with an earlier version of Unicode. This is not merely a theoretical possibility, because it has occurred ([RFC6452]).

Accordingly, as in IDNA2008, a Stringprep replacement that intends to be Unicode version agnostic will need to work out a mechanism to address cases where incompatible changes occur because of new Unicode versions.

6. Considerations for Stringprep replacement

The above suggests the following guidance:

- A Stringprep replacement should be defined.
- The replacement should take an approach similar to IDNA2008, (e.g. by using codepoint properties instead of codepoint whitelisting) in that it enables better Unicode agility.
- Protocols share similar characteristics of strings. Therefore, defining internationalization preparation algorithms for the smallest set of string classes may be sufficient for most cases, providing coherence among a set of related protocols or protocols where identifiers are exchanged.
- The sets of string classes need to be evaluated according to the considerations that make up the headings in Section 5
- It is reasonable to limit scope to Unicode code points, and rule the mapping of data from other character encodings outside the scope of this effort.
- The replacement ought at least to provide guidance to applications using the replacement on how to handle protocol incompatibilities resulting from changes to Unicode. In an ideal world, the Stringprep replacement would handle the changes automatically, but it appears that such automatic handling would require magic and cannot be expected.
- Compatibility within each protocol between a technique that is Stringprep-based and the technique’s replacement has to be considered very carefully.

Existing deployments already depend on Stringprep profiles. Therefore, a replacement must consider the effects of any new strategy on existing deployments. By way of comparison, it is worth noting that some characters were acceptable in IDNA labels under IDNA2003, but are not protocol-valid under IDNA2008 (and conversely); disagreement about what to do during the transition has resulted in different approaches to mapping. Different implementers may make different decisions about what to do in such cases; this could have interoperability effects. It is necessary to trade better support for different linguistic environments against the potential side effects of backward incompatibility.

7. Security Considerations

This document merely states what problems are to be solved, and does not define a protocol. There are undoubtedly security implications of the particular results that will come from the work to be completed. Moreover, the Stringprep Security Considerations [RFC3454] Section applies. See also the analysis in the subsections of Appendix B, below.

8. IANA Considerations

This document has no actions for IANA.

9. Discussion home for this draft

Note: RFC-Editor, please remove this section before publication.

This document is intended to define the problem space discussed on the precis@ietf.org mailing list.

10. Acknowledgements

This document is the product of the PRECIS IETF Working Group, and participants in that Working Group were helpful in addressing issues with the text.

Specific contributions came from David Black, Alan DeKok, Simon Josefsson, Bill McQuillan, Alexey Melnikov, Peter Saint-Andre, Dave Thaler, and Yoshiro Yoneya.

Dave Thaler provided the "buckets" insight in Section 5.1.1, central to the organization of the problem.

Evaluations of Stringprep profiles that are included in Appendix B were done by: David Black, Alexey Melnikov, Peter Saint-Andre, Dave Thaler.

11. Informative References

[I-D.iab-identifier-comparison]
Thaler, D., "Issues in Identifier Comparison for Security Purposes", draft-iab-identifier-comparison-07 (work in progress), August 2012.

[NEWPREP] "Newprep BoF Meeting Minutes", March 2010.

[RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, October 1985.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC3491] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

[RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.

[RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.

[RFC3722] Bakke, M., "String Profile for Internet Small Computer Systems Interface (iSCSI) Names", RFC 3722, April 2004.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence

Protocol (XMPP): Core", RFC 3920, October 2004.

[RFC3922] Saint-Andre, P., "Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)", RFC 3922, October 2004.

[RFC4011] Waldbusser, S., Saperia, J., and T. Hongal, "Policy Based Management MIB", RFC 4011, March 2005.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, December 2005.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4505] Zeilenga, K., "Anonymous Simple Authentication and Security Layer (SASL) Mechanism", RFC 4505, June 2006.

[RFC4511] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[RFC4513] Harrison, R., "Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms", RFC 4513, June 2006.

[RFC4518] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Internationalized String Preparation", RFC 4518, June 2006.

[RFC4616] Zeilenga, K., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, August 2006.

[RFC4643] Vinocur, J. and K. Murchison, "Network News Transfer Protocol (NNTP) Extension for Authentication", RFC 4643, October 2006.

[RFC4683] Park, J., Lee, J., Lee, H., Park, S., and T. Polk, "Internet X.509 Public Key Infrastructure Subject Identification Method (SIM)", RFC 4683, October 2006.

[RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and

Recommendations for Internationalized Domain Names (IDNs)", RFC 4690, September 2006.

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, March 2007.

[RFC4954] Siemborski, R. and A. Melnikov, "SMTP Service Extension for Authentication", RFC 4954, July 2007.

[RFC5034] Siemborski, R. and A. Menon-Sen, "The Post Office Protocol (POP3) Simple Authentication and Security Layer (SASL) Authentication Mechanism", RFC 5034, July 2007.

[RFC5051] Crispin, M., "i;unicode-casemap - Simple Unicode Collation Algorithm", RFC 5051, October 2007.

[RFC5054] Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, "Using the Secure Remote Password (SRP) Protocol for TLS Authentication", RFC 5054, November 2007.

[RFC5122] Saint-Andre, P., "Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP)", RFC 5122, February 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5456] Spencer, M., Capouch, B., Guy, E., Miller, F., and K. Shumard, "IAX: Inter-Asterisk eXchange Version 2", RFC 5456, February 2010.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

[RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, July 2010.

[RFC5804] Melnikov, A. and T. Martin, "A Protocol for Remotely Managing Sieve Scripts", RFC 5804, July 2010.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework",

RFC 5890, August 2010.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.

[RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.

[RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.

[RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.

[RFC6452] Faltstrom, P. and P. Hoffman, "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA) - Unicode 6.0", RFC 6452, November 2011.

[UAX15] "Unicode Standard Annex #15: Unicode Normalization Forms", UAX 15, September 2009.

[Unicode61] The Unicode Consortium. The Unicode Standard, Version 6.1, defined by:, "The Unicode Standard -- Version 6.1", (Mountain View, CA: The Unicode Consortium, 2012. ISBN 978-1-936213-02-3), September 2009, <http://www.unicode.org/versions/Unicode6.1.0/>.

[ietf78precis] Blanchet, M., "PRECIS Framework", Proceedings of the Seventy-Eighth Internet Engineering Task Force https://www.ietf.org/proceedings/78/, July 2010, <http://www.ietf.org/proceedings/78/slides/precis-2.pdf>.

Appendix A. Classification of Stringprep Profiles

A number of the known cases of Stringprep use were evaluated during the preparation of this document. The known cases are here described in two ways. The types of identifiers the protocol uses is first called out in the ID type column (from Section 5.1.1), using the short forms "a" for Absolute, "d" for Definite, and "i" for Indefinite. Next, there is a column that contains an "i" if the protocol string comes from user input, an "o" if the protocol string becomes user-facing output, "b" if both are true, and "n" if neither is true.

RFC	IDtype	User?
3722	a	b
3748	-	-
3920	a,d	b
4505	a	i
4314	a,d	b
4954	a,d	b
5034	a,d	b
5804	a,d	b

Table 1

Appendix B. Evaluation of Stringprep Profiles

This section is a summary of evaluation of Stringprep profiles that was done to get a good understanding of the usage of Stringprep. This summary is by no means normative nor the actual evaluations themselves. A template was used for reviewers to get a coherent view of all evaluations.

B.1. iSCSI Stringprep Profile: RFC3722 (and RFC3721, RFC3720)

Description: An iSCSI session consists of an initiator (i.e., host or server that uses storage) communicating with a target (i.e., a storage array or other system that provides storage). Both the iSCSI initiator and target are named by iSCSI Names. The iSCSI Stringprep profile is used for iSCSI names.

How it is used: iSCSI initiators and targets (see above). They can also be used to identify SCSI ports (these are software entities in the iSCSI protocol, not hardware ports), and iSCSI logical units (storage volumes), although both are unusual in practice.

What entities create these identifiers? Generally a Human user (1) configures an Automated system (2) that generates the names. Advance configuration of the system is required due to the embedded use of external unique identifier (from the DNS or IEEE).

How is the string input in the system? Keyboard and copy-paste are common. Copy-paste is common because iSCSI names are long enough to be problematic for humans to remember, causing use of email, sneaker-net, text files, etc. to avoid mistype mistakes.

Where do we place the dividing line between user interface and protocol? The iSCSI protocol requires that all internationalization string preparation occur in the user interface. The iSCSI protocol treats iSCSI names as opaque identifiers that are compared byte-by-byte for equality. iSCSI names are generally not checked for correct formatting by the protocol.

What entities enforce the rules? There are no iSCSI-specific enforcement entities, although the use of unique identifier information in the names relies on DNS registrars and the IEEE Registration Authority.

Comparison Byte-by-byte

Case Folding, Sensitivity, Preservation Case folding is required for the code blocks specified in RFC 3454, Table B.2. The overall iSCSI naming system (UI + protocol) is case-insensitive.

What is the impact if the comparison results in a false positive? Potential access to the wrong storage. - If the initiator has no access to the wrong storage, an authentication failure is the probable result. - If the initiator has access to the wrong storage, the resulting mis-identification could result in use of the wrong data and possible corruption of stored data.

What is the impact if the comparison results in a false negative? Denial of authorized storage access.

What are the security impacts? iSCSI names may be used as the authentication identities for storage systems. Comparison problems could result in authentication problems, although note that authentication failure ameliorates some of the false positive cases.

Normalization NFKC, as specified by RFC 3454.

Mapping Yes, as specified by table B.1 in RFC 3454

Disallowed Characters Only the following characters are allowed: - ASCII dash, dot, colon - ASCII lower case letters and digits - Unicode lower case characters as specified by RFC 3454 All other characters are disallowed.

Which other strings or identifiers are these most similar to? None - iSCSI names are unique to iSCSI.

Are these strings or identifiers sometimes the same as strings or identifiers from other protocols? No

Does the identifier have internal structure that needs to be respected? Yes - ASCII dot, dash and colon are used for internal name structure. These are not reserved characters in that they can occur in the name in locations other than those used for structuring purposes (e.g., only the first occurrence of a colon character is structural, others are not).

How are users exposed to these strings? How are they published? iSCSI names appear in server and storage system configuration interfaces. They also appear in system logs.

Is the string / identifier used as input to other operations? Effectively, no. The rarely used port and logical unit names involve concatenation, which effectively extends a unique iSCSI Name for a target to uniquely identify something within that target.

How much tolerance for change from existing Stringprep approach? Good tolerance; the community would prefer that internationalization experts solve internationalization problems.

How strong a desire for change (e.g., for Unicode agility)? Unicode agility is desired in principle as long as nothing significant breaks.

B.2. SMTP/POP3/ManageSieve Stringprep Profiles: RFC4954,RFC5034,RFC 5804

Description: Authorization identity (user identifier) exchanged during SASL authentication: AUTH (SMTP/POP3) or AUTHENTICATE (ManageSieve) command.

How It's Used: Used for proxy authorization, e.g. to [lawfully] impersonate a particular user after a privileged authentication

Who Generates It: Typically generated by email system administrators using some tools/conventions, sometimes from some backend database. - In some setups human users can register own usernames (e.g. webmail self registration)

User Input Methods: - Typed by user / selected from a list - Copy-and-paste - Perhaps voice input - Can also be specified in configuration files or on a command line

Enforcement: - Rules enforced by server / add-on service (e.g., gateway service) on registration of account

Comparison Method: "Type 1" (byte-for-byte) or "type 2" (compare by a common algorithm that everyone agrees on (e.g., normalize and then compare the result byte-by-byte))

Case Folding, Sensitivity, Preservation: Most likely case sensitive. Exact requirements on case-sensitivity/case-preservation depend on a specific implementation, e.g. an implementation might treat all user identifiers as case insensitive (or case insensitive for US-ASCII subset only).

Impact of Comparison: False positives: - an unauthorized user is allowed email service access (login) False negatives: - an authorized user is denied email service access
Normalization: NFKC (as per RFC 4013)
Mapping: (see Section 2 of RFC 4013 for the full list): Non ASCII spaces are mapped to space, etc.
Disallowed Characters: (see Section 2 of RFC 4013 for the full list): Unicode Control characters, etc.
String Classes: - simple username. See Section 2 of RFC 4013 for details on restrictions. Note that some implementations allow spaces in these. While implementations are not required to use a specific format, an authorization identity frequently has the same format as an email address (and EAI email address in the future), or as a left hand side of an email address. Note: whatever is recommended for SMTP/POP/ManageSieve authorization identity should also be used for IMAP authorization identities, as IMAP/POP3/SMTP/ ManageSieve are frequently implemented together.
Internal Structure: None
User Output: Unlikely, but possible. For example, if it is the same as an email address.
Operations: - Sometimes concatenated with other data and then used as input to a cryptographic hash function
How much tolerance for change from existing Stringprep approach? Not sure.
Background information: In RFC 5034, when describing the POP3 AUTH command: The authorization identity generated by the SASL exchange is a simple username, and SHOULD use the SASLprep profile (see RFC4013) of the StringPrep algorithm (see RFC3454) to prepare these names for matching. If preparation of the authorization identity fails or results in an empty string (unless it was transmitted as the empty string), the server MUST fail the authentication. In RFC 4954, when describing the SMTP AUTH command: The authorization identity generated by this SASL exchange is a "simple username" (in the sense defined in SASLprep), and both client and server SHOULD (*) use the SASLprep profile of the StringPrep algorithm to prepare these names for transmission or comparison. If preparation of the authorization identity fails or results in an empty string (unless it was transmitted as the empty string), the server MUST fail the authentication. (*) Note: Future revision of this specification may change this requirement to MUST. Currently, the SHOULD is used in order to avoid breaking the majority of existing implementations. In RFC 5804, when describing the ManageSieve AUTHENTICATE command: The authorization identity generated by this SASL exchange is a "simple username" (in the sense defined in SASLprep), and both client and server MUST use the SASLprep profile of the StringPrep algorithm to prepare these names for transmission or comparison. If preparation of the authorization

identity fails or results in an empty string (unless it was transmitted as the empty string), the server MUST fail the authentication.

B.3. IMAP Stringprep Profiles: RFC5738, RFC4314: Usernames

Evaluation Note These documents have 2 types of strings (usernames and passwords), so there are two separate templates.

Description: "username" parameter to the IMAP LOGIN command, identifiers in IMAP ACL commands. Note that any valid username is also an IMAP ACL identifier, but IMAP ACL identifiers can include other things like name of group of users.

How It's Used: Used for authentication (Usernames), or in IMAP Access Control Lists (Usernames or Group names)

Who Generates It: - Typically generated by email system administrators using some tools/conventions, sometimes from some backend database. - In some setups human users can register own usernames (e.g. webmail self registration)

User Input Methods: - Typed by user / selected from a list - Copy-and-paste - Perhaps voice input - Can also be specified in configuration files or on a command line

Enforcement: - Rules enforced by server / add-on service (e.g., gateway service) on registration of account

Comparison Method: Type 1" (byte-for-byte) or "type 2" (compare by a common algorithm that everyone agrees on (e.g., normalize and then compare the result byte-by-byte))

Case Folding, Sensitivity, Preservation: - Most likely case sensitive. Exact requirements on case-sensitivity/ case-preservation depend on a specific implementation, e.g. an implementation might treat all user identifiers as case insensitive (or case insensitive for US-ASCII subset only).

Impact of Comparison: False positives: - an unauthorized user is allowed IMAP access (login) - improperly grant privileges (e.g., access to a specific mailbox, ability to manage ACLs for a mailbox) False negatives: - an authorized user is denied IMAP access - unable to use granted privileges (e.g., access to a specific mailbox, ability to manage ACLs for a mailbox)

Normalization: NFKC (as per RFC 4013)

Mapping: (see Section 2 of RFC 4013 for the full list): non ASCII spaces are mapped to space

Disallowed Characters: (see Section 2 of RFC 4013 for the full list): Unicode Control characters, etc.

String Classes: - simple username. See Section 2 of RFC 4013 for details on restrictions. Note that some implementations allow spaces in these. While IMAP implementations are not required to use a specific format, an IMAP username frequently has the same format as an email address (and EAI email address in the future), or as a left hand side of an email address. Note: whatever is

recommended for IMAP username should also be used for ManageSieve, POP3 and SMTP authorization identities, as IMAP/POP3/SMTP/ ManageSieve are frequently implemented together.

Internal Structure: None

User Output: Unlikely, but possible. For example, if it is the same as an email address. - access control lists (e.g. in IMAP ACL extension), both when managing membership and listing membership of existing access control lists. - often show up as mailbox names (under Other Users IMAP namespace)

Operations: - Sometimes concatenated with other data and then used as input to a cryptographic hash function

How much tolerance for change from existing Stringprep approach? Not sure. Non-ASCII IMAP usernames are currently prohibited by IMAP (RFC 3501). However they are allowed when used in IMAP ACL extension.

B.4. IMAP Stringprep Profiles: RFC5738: Passwords

Description: "Password" parameter to the IMAP LOGIN command

How It's Used: Used for authentication (Passwords)

Who Generates It: Either generated by email system administrators using some tools/conventions, or specified by the human user.

User Input Methods: - Typed by user - Copy-and-paste - Perhaps voice input - Can also be specified in configuration files or on a command line

Enforcement: Rules enforced by server / add-on service (e.g., gateway service or backend databse) on registration of account

Comparison Method: "Type 1" (byte-for-byte)

Case Folding, Sensitivity, Preservation: Most likely case sensitive.

Impact of Comparison: False positives: - an unauthorized user is allowed IMAP access (login) False negatives: - an authorized user is denied IMAP access

Normalization: NFKC (as per RFC 4013)

Mapping: (see Section 2 of RFC 4013 for the full list): non ASCII spaces are mapped to space

Disallowed Characters: (see Section 2 of RFC 4013 for the full list): Unicode Control characters, etc.

String Classes: Currently defined as "simple username" (see Section 2 of RFC 4013 for details on restrictions.), however this is likely to be a different class from usernames. Note that some implementations allow spaces in these. Password in all email related protocols should be treated in the same way. Same passwords are frequently shared with web, IM, etc. applications.

Internal Structure: None

User Output: - text of email messages (e.g. in "you forgot your password" email messages) - web page / directory - side of the bus / in ads -- possible

Operations: Sometimes concatenated with other data and then used as input to a cryptographic hash function. Frequently stored as is, or hashed.

How much tolerance for change from existing Stringprep approach? Not sure. Non-ASCII IMAP passwords are currently prohibited by IMAP (RFC 3501), however they are likely to be in widespread use.

Background information: RFC 5738 (IMAP INTERNATIONALIZATION): 5. UTF8=USER Capability If the "UTF8=USER" capability is advertised, that indicates the server accepts UTF-8 user names and passwords and applies SASLprep RFC4013 to both arguments of the LOGIN command. The server MUST reject UTF-8 that fails to comply with the formal syntax in RFC 3629 RFC3629 or if it encounters Unicode characters listed in Section 2.3 of SASLprep RFC 4013 RFC4013. RFC 4314 (IMAP4 Access Control List (ACL) Extension): 3. Access control management commands and responses Servers, when processing a command that has an identifier as a parameter (i.e., any of SETACL, DELETEACL, and LISTRIGHTS commands), SHOULD first prepare the received identifier using "SASLprep" profile SASLprep of the "Stringprep" algorithm Stringprep. If the preparation of the identifier fails or results in an empty string, the server MUST refuse to perform the command with a BAD response. Note that Section 6 recommends additional identifier’s verification steps. and in Section 6: This document relies on SASLprep to describe steps required to perform identifier canonicalization (preparation). The preparation algorithm in SASLprep was specifically designed such that its output is canonical, and it is well-formed. However, due to an anomaly PR29 in the specification of Unicode normalization, canonical equivalence is not guaranteed for a select few character sequences. Identifiers prepared with SASLprep can be stored and returned by an ACL server. The anomaly affects ACL manipulation and evaluation of identifiers containing the selected character sequences. These sequences, however, do not appear in well-formed text. In order to address this problem, an ACL server MAY reject identifiers containing sequences described in PR29 by sending the tagged BAD response. This is in addition to the requirement to reject identifiers that fail SASLprep preparation as described in Section 3.

B.5. Anonymous SASL Stringprep Profiles: RFC4505

Description: RFC 4505 defines a "trace" field:

Comparison: this field is not intended for comparison (only used for logging)
Case folding; case sensitivity, preserve case: No case folding/case sensitive
Do users input the strings directly? Yes. Possibly entered in configuration UIs, or on a command line. Can also be stored in configuration files. The value can also be automatically generated by clients (e.g. a fixed string is used, or a user's email address).
How users input strings? Keyboard/voice, stylus (pick from a list). Copy-paste - possibly.
Normalization: None
Disallowed Characters Control characters are disallowed. (See Section 3 of RFC 4505)
Which other strings or identifiers are these most similar to? RFC 4505 says that the trace "should take one of two forms: an Internet email address, or an opaque string that does not contain the '@' U+0040) character and that can be interpreted by the system administrator of the client's domain." In practice, this is a freeform text, so it belongs to a different class from "email address" or "username".
Are these strings or identifiers sometimes the same as strings or identifiers from other protocols (e.g., does an IM system sometimes use the same credentials database for authentication as an email system)? Yes: see above. However there is no strong need to keep them consistent in the future.
How are users exposed to these strings, how are they published? No. However, The value can be seen in server logs
Impacts of false positives and false negatives: False positive: a user can be confused with another user. False negative: two distinct users are treated as the same user. But note that the trace field is not authenticated, so it can be easily falsified.
Tolerance of changes in the community The community would be flexible.
Delimiters No internal structure, but see comments above about frequent use of email addresses.
Background information: The Anonymous Mechanism The mechanism consists of a single message from the client to the server. The client may include in this message trace information in the form of a string of UTF-8-encoded Unicode characters prepared in accordance with StringPrep and the "trace" Stringprep profile defined in Section 3 of this document. The trace information, which has no semantical value, should take one of two forms: an Internet email address, or an opaque string that does not contain the '@' (U+0040) character and that can be interpreted by the system administrator of the client's domain. For privacy reasons, an Internet email address or other information identifying the user should only be used with permission from the user. 3. The

"trace" Profile of "Stringprep" This section defines the "trace" profile of StringPrep. This profile is designed for use with the SASL ANONYMOUS Mechanism. Specifically, the client is to prepare the message production in accordance with this profile. The character repertoire of this profile is Unicode 3.2. No mapping is required by this profile. No Unicode normalization is required by this profile. The list of unassigned code points for this profile is that provided in Appendix A of StringPrep. Unassigned code points are not prohibited. Characters from the following tables of StringPrep are prohibited: - C.2.1 (ASCII control characters) - C.2.2 (Non-ASCII control characters) - C.3 (Private use characters) - C.4 (Non-character code points) - C.5 (Surrogate codes) - C.6 (Inappropriate for plain text) - C.8 (Change display properties are deprecated) - C.9 (Tagging characters) No additional characters are prohibited. This profile requires bidirectional character checking per Section 6 of StringPrep.

B.6. XMPP Stringprep Profiles: RFC3920 Nodeprep

Description: Localpart of JabberID ("JID"), as in: localpart@domainpart/resourcepart
How It's Used: - Usernames (e.g., stpeter@jabber.org) - Chatroom names (e.g., precis@jabber.ietf.org) - Publish-subscribe nodes - Bot names
Who Generates It: - Typically, end users via an XMPP client - Sometimes created in an automated fashion
User Input Methods: - Typed by user - Copy-and-paste - Perhaps voice input - Clicking a URI/IRI
Enforcement: - Rules enforced by server / add-on service (e.g., chatroom service) on registration of account, creation of room, etc.
Comparison Method: "Type 2" (common algorithm)
Case Folding, Sensitivity, Preservation: - Strings are always folded to lowercase - Case is not preserved
Impact of Comparison: False positives: - unable to authenticate at server (or authenticate to wrong account) - add wrong person to buddy list - join the wrong chatroom - improperly grant privileges (e.g., chatroom admin) - subscribe to wrong pubsub node - interact with wrong bot - allow communication with blocked entity False negatives: - unable to authenticate - unable to add someone to buddy list - unable to join desired chatroom - unable to use granted privileges (e.g., chatroom admin) - unable to subscribe to desired pubsub node - unable to interact with desired bot - disallow communication with unblocked entity

Normalization: NFKC
Mapping: Spaces are mapped to nothing
Disallowed Characters: ",&,',/,:,<,>,@
String Classes: - Often similar to generic username - Often similar to localpart of email address - Sometimes same as localpart of email address
Internal Structure: None
User Output: - vCard - email signature - web page / directory - text of message (e.g., in a chatroom)
Operations: - Sometimes concatenated with other data and then used as input to a cryptographic hash function

B.7. XMPP Stringprep Profiles: RFC3920 Resourceprep

Description: - Resourcepart of JabberID ("JID"), as in: localpart@domainpart/resourcepart - Typically free-form text
How It's Used: - Device / session names (e.g., stpeter@jabber.org/Home) - Nicknames (e.g., precis@jabber.ietf.org/StPeter)
Who Generates It: - Often human users via an XMPP client - Often generated in an automated fashion by client or server
User Input Methods: - Typed by user - Copy-and-paste - Perhaps voice input - Clicking a URI/IRI
Enforcement: - Rules enforced by server / add-on service (e.g., chatroom service) on account login, joining a chatroom, etc.
Comparison Method: "Type 2" (byte-for-byte)
Case Folding, Sensitivity, Preservation: - Strings are never folded - Case is preserved
Impact of Comparison: False positives: - interact with wrong device (e.g., for file transfer or voice call) - interact with wrong chatroom participant - improperly grant privileges (e.g., chatroom moderator) - allow communication with blocked entity False negatives: - unable to choose desired chatroom nick - unable to use granted privileges (e.g., chatroom moderator) - disallow communication with unblocked entity
Normalization: NFKC
Mapping: Spaces are mapped to nothing
Disallowed Characters: None
String Classes: Basically a free-form identifier
Internal Structure: None
User Output: - text of message (e.g., in a chatroom) - device names often not exposed to human users
Operations: Sometimes concatenated with other data and then used as input to a cryptographic hash function

B.8. EAP Stringprep Profiles: RFC3748

Description: RFC 3748 section 5 references Stringprep, but the WG did not agree with the text (was added by IESG) and there are no known implementations that use Stringprep. The main problem with that text is that the use of strings is a per-method concept, not a generic EAP concept and so RFC 3748 itself does not really use Stringprep, but individual EAP methods could. As such, the answers to the template questions are mostly not applicable, but a few answers are universal across methods. The list of IANA registered EAP methods is at http://www.iana.org/assignments/eap-numbers/eap-numbers.xml#eap-numbers-3

Comparison Methods: n/a (per-method)

Case Folding, Case Sensitivity, Case Preservation: n/a (per-method)

Impact of comparison: A false positive results in unauthorized network access (and possibly theft of service if some else is billed). A false negative results in lack of authorized network access (no connectivity).

User input: n/a (per-method)

Normalization: n/a (per-method)

Mapping: n/a (per-method)

Disallowed characters: n/a (per-method)

String classes: Although some EAP methods may use a syntax similar to other types of identifiers, EAP mandates that the actual values must not be assumed to be identifiers usable with anything else.

Internal structure: n/a (per-method)

User output: Identifiers are never human displayed except perhaps as they’re typed by a human.

Operations: n/a (per-method)

Community considerations: There is no resistance to change for the base EAP protocol (as noted, the WG didn’t want the existing text). However actual use of Stringprep, if any, within specific EAP methods may have resistance. It is currently unknown whether any EAP methods use Stringprep.

Authors’ Addresses

Marc Blanchet
Viagenie
246 Aberdeen
Quebec, QC G1R 2E1
Canada

Email: Marc.Blanchet@viagenie.ca
URI: http://viagenie.ca

Andrew Sullivan
Dyn, Inc.
150 Dow St
Manchester, NH 03101
U.S.A.

Email: asullivan@dyn.com