

PRECIS
Internet-Draft
Obsoletes: 3454 (if approved)
Intended status: Standards Track
Expires: August 23, 2015

P. Saint-Andre
&yet
M. Blanchet
Viagenie
February 19, 2015

PRECIS Framework: Preparation, Enforcement, and Comparison of
Internationalized Strings in Application Protocols
draft-ietf-precis-framework-23

Abstract

Application protocols using Unicode characters in protocol strings need to properly handle such strings in order to enforce internationalization rules for strings placed in various protocol slots (such as addresses and identifiers) and to perform valid comparison operations (e.g., for purposes of authentication or authorization). This document defines a framework enabling application protocols to perform the preparation, enforcement, and comparison of internationalized strings ("PRECIS") in a way that depends on the properties of Unicode characters and thus is agile with respect to versions of Unicode. As a result, this framework provides a more sustainable approach to the handling of internationalized strings than the previous framework, known as Stringprep (RFC 3454). This document obsoletes RFC 3454.

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 August 23, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
2. Terminology	6
3. Preparation, Enforcement, and Comparison	7
4. String Classes	7
4.1. Overview	7
4.2. IdentifierClass	9
4.2.1. Valid	9
4.2.2. Contextual Rule Required	9
4.2.3. Disallowed	10
4.2.4. Unassigned	10
4.2.5. Examples	10
4.3. FreeformClass	11
4.3.1. Valid	11
4.3.2. Contextual Rule Required	11
4.3.3. Disallowed	12
4.3.4. Unassigned	12
4.3.5. Examples	12
5. Profiles	12
5.1. Profiles Must Not Be Multiplied Beyond Necessity	13
5.2. Rules	13
5.2.1. Width Mapping Rule	13
5.2.2. Additional Mapping Rule	14
5.2.3. Case Mapping Rule	14
5.2.4. Normalization Rule	15
5.2.5. Directionality Rule	15
5.3. A Note about Spaces	16
6. Applications	17
6.1. How to Use PRECIS in Applications	17
6.2. Further Excluded Characters	17
6.3. Building Application-Layer Constructs	18
7. Order of Operations	19

8. Code Point Properties	19
9. Category Definitions Used to Calculate Derived Property . . .	22
9.1. LetterDigits (A)	22
9.2. Unstable (B)	22
9.3. IgnorableProperties (C)	23
9.4. IgnorableBlocks (D)	23
9.5. LDH (E)	23
9.6. Exceptions (F)	23
9.7. BackwardCompatible (G)	23
9.8. JoinControl (H)	23
9.9. OldHangulJamo (I)	23
9.10. Unassigned (J)	24
9.11. ASCII7 (K)	24
9.12. Controls (L)	24
9.13. PrecisIgnorableProperties (M)	24
9.14. Spaces (N)	24
9.15. Symbols (O)	24
9.16. Punctuation (P)	25
9.17. HasCompat (Q)	25
9.18. OtherLetterDigits (R)	25
10. Guidelines for Designated Experts	25
11. IANA Considerations	26
11.1. PRECIS Derived Property Value Registry	26
11.2. PRECIS Base Classes Registry	26
11.3. PRECIS Profiles Registry	27
12. Security Considerations	29
12.1. General Issues	29
12.2. Use of the IdentifierClass	30
12.3. Use of the FreeformClass	30
12.4. Local Character Set Issues	30
12.5. Visually Similar Characters	30
12.6. Security of Passwords	32
13. Interoperability Considerations	33
13.1. Encoding	33
13.2. Character Sets	33
13.3. Unicode Versions	34
13.4. Potential Changes to Handling of Certain Unicode Code Points	34
14. References	35
14.1. Normative References	35
14.2. Informative References	35
14.3. URIs	38
Appendix A. Acknowledgements	38
Authors' Addresses	39

1. Introduction

Application protocols using Unicode characters [Unicode7.0] in protocol strings need to properly handle such strings in order to enforce internationalization rules for strings placed in various protocol slots (such as addresses and identifiers) and to perform valid comparison operations (e.g., for purposes of authentication or authorization). This document defines a framework enabling application protocols to perform the preparation, enforcement, and comparison of internationalized strings ("PRECIS") in a way that depends on the properties of Unicode characters and thus is agile with respect to versions of Unicode.

As described in the PRECIS problem statement [RFC6885], many IETF protocols have used the Stringprep framework [RFC3454] as the basis for preparing, enforcing, and comparing protocol strings that contain Unicode characters, especially characters outside the ASCII range [RFC20]. The Stringprep framework was developed during work on the original technology for internationalized domain names (IDNs), here called "IDNA2003" [RFC3490], and Nameprep [RFC3491] was the Stringprep profile for IDNs. At the time, Stringprep was designed as a general framework so that other application protocols could define their own Stringprep profiles. Indeed, a number of application protocols defined such profiles.

After the publication of [RFC3454] in 2002, several significant issues arose with the use of Stringprep in the IDN case, as documented in the IAB's recommendations regarding IDNs [RFC4690] (most significantly, Stringprep was tied to Unicode version 3.2). Therefore, the newer IDNA specifications, here called "IDNA2008" ([RFC5890], [RFC5891], [RFC5892], [RFC5893], [RFC5894]), no longer use Stringprep and Nameprep. This migration away from Stringprep for IDNs prompted other "customers" of Stringprep to consider new approaches to the preparation, enforcement, and comparison of internationalized strings, as described in [RFC6885].

This document defines a framework for a post-Stringprep approach to the preparation, enforcement, and comparison of internationalized strings in application protocols, based on several principles:

1. Define a small set of string classes that specify the Unicode characters (i.e., specific "code points") appropriate for common application protocol constructs.
2. Define each PRECIS string class in terms of Unicode code points and their properties so that an algorithm can be used to determine whether each code point or character category is (a)

valid, (b) allowed in certain contexts, (c) disallowed, or (d) unassigned.

3. Use an "inclusion model" such that a string class consists only of code points that are explicitly allowed, with the result that any code point not explicitly allowed is forbidden.
4. Enable application protocols to define profiles of the PRECIS string classes if necessary (addressing matters such as width mapping, case mapping, Unicode normalization, and directionality) but strongly discourage the multiplication of profiles beyond necessity in order to avoid violations of the Principle of Least User Astonishment.

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

- o Application protocols will be agile with regard to Unicode versions.
- o 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 characters 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.

Whereas the string classes define the "baseline" code points for a range of applications, profiling enables application protocols to apply the string classes in ways that are appropriate for common constructs such as usernames [I-D.ietf-precis-saslprepbis], opaque strings such as passwords [I-D.ietf-precis-saslprepbis], and nicknames [I-D.ietf-precis-nickname]. Profiles are responsible for defining the handling of right-to-left characters as well as various mapping operations of the kind also discussed for IDNs in [RFC5895], such as case preservation or lowercasing, Unicode normalization, mapping of certain characters to other characters or to nothing, and mapping of full-width and half-width characters.

When an application applies a profile of a PRECIS string class, it transforms an input string (which might or might not be conforming) into an output string that definitively conforms to the profile. In particular, this document focuses on the resulting ability to achieve the following objectives:

- a. Enforcing all the the rules of a profile for a single output string (e.g., to determine if a string can be included in a protocol slot, communicated to another entity within a protocol, stored in a retrieval system, etc.).
- b. Comparing two output strings to determine if they are equivalent, typically through octet-for-octet matching to test for "bit-string identity" (e.g., to make an access decision for purposes of authentication or authorization as further described in [RFC6943]).

The opportunity to define profiles naturally introduces the possibility of a proliferation of profiles, thus potentially mitigating the benefits of common code and violating user expectations. See Section 5 for a discussion of this important topic.

In addition, it is extremely important for protocol designers and application developers to understand that the transformation of an input string to an output string is rarely reversible. As one relatively simple example, case mapping would transform an input string of "StPeter" to "stpeter", and information about the capitalization of the first and third characters would be lost. Similar considerations apply to other forms of mapping and normalization.

Although this framework is similar to IDNA2008 and includes by reference some of the character categories defined in [RFC5892], it defines additional character categories to meet the needs of common application protocols other than DNS.

The character categories and calculation rules defined under Section 8 and Section 9 are normative and apply to all Unicode code points. The code point table that results from applying the character categories and calculation rules to the latest version of Unicode can be found in an IANA registry.

2. Terminology

Many important terms used in this document are defined in [RFC5890], [RFC6365], [RFC6885], and [Unicode7.0]. The terms "left-to-right" (LTR) and "right-to-left" (RTL) are defined in Unicode Standard Annex #9 [UAX9].

As of the date of writing, the version of Unicode published by the Unicode Consortium is 7.0 [Unicode7.0]; however, PRECIS is not tied to a specific version of Unicode. The latest version of Unicode is always available [UnicodeCurrent].

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. Preparation, Enforcement, and Comparison

This document distinguishes between three different actions that an entity can take with regard to a string:

- o Enforcement entails applying all of the rules specified for a particular string class or profile thereof to an individual string, for the purpose of determining if the string can be used in a given protocol slot.
- o Comparison entails applying all of the rules specified for a particular string class or profile thereof to two separate strings, for the purpose of determining if the two strings are equivalent.
- o Preparation entails only ensuring that the characters in an individual string are allowed by the underlying PRECIS string class.

In most cases, authoritative entities such as servers are responsible for enforcement, whereas subsidiary entities such as clients are responsible only for preparation. The rationale for this distinction is that clients might not have the facilities (in terms of device memory and processing power) to enforce all the rules regarding internationalized strings (such as width mapping and Unicode normalization), although they can more easily limit the repertoire of characters they offer to an end user. By contrast, it is assumed that a server would have more capacity to enforce the rules, and in any case acts as an authority regarding allowable strings in protocol slots such as addresses and endpoint identifiers. In addition, a client cannot necessarily be trusted to properly generate such strings, especially for security-sensitive contexts such as authentication and authorization.

4. String Classes

4.1. Overview

Starting in 2010, various "customers" of Stringprep began to discuss the need to define a post-Stringprep approach to the preparation and comparison of internationalized strings other than IDNs. This community analyzed the existing Stringprep profiles and also weighed the costs and benefits of defining a relatively small set of Unicode

characters that would minimize the potential for user confusion caused by visually similar characters (and thus be relatively "safe") vs. defining a much larger set of Unicode characters that would maximize the potential for user creativity (and thus be relatively "expressive"). As a result, the community concluded that most existing uses could be addressed by two string classes:

IdentifierClass: a sequence of letters, numbers, and some symbols that is used to identify or address a network entity such as a user account, a venue (e.g., a chatroom), an information source (e.g., a data feed), or a collection of data (e.g., a file); the intent is that this class will minimize user confusion in a wide variety of application protocols, with the result that safety has been prioritized over expressiveness for this class.

FreeformClass: a sequence of letters, numbers, symbols, spaces, and other characters that is used for free-form strings, including passwords as well as display elements such as human-friendly nicknames for devices or for participants in a chatroom; the intent is that this class will allow nearly any Unicode character, with the result that expressiveness has been prioritized over safety for this class. Note well that protocol designers, application developers, service providers, and end users might not understand or be able to enter all of the characters that can be included in the FreeformClass - see Section 12.3 for details.

Future specifications might define additional PRECIS string classes, such as a class that falls somewhere between the IdentifierClass and the FreeformClass. At this time, it is not clear how useful such a class would be. In any case, because application developers are able to define profiles of PRECIS string classes, a protocol needing a construct between the IdentifierClass and the FreeformClass could define a restricted profile of the FreeformClass if needed.

The following subsections discuss the IdentifierClass and FreeformClass in more detail, with reference to the dimensions described in Section 3 of [RFC6885]. Each string class is defined by the following behavioral rules:

Valid: Defines which code points are treated as valid for the string.

Contextual Rule Required: Defines which code points are treated as allowed only if the requirements of a contextual rule are met (i.e., either CONTEXTJ or CONTEXTO).

Disallowed: Defines which code points need to be excluded from the string.

Unassigned: Defines application behavior in the presence of code points that are unknown (i.e., not yet designated) for the version of Unicode used by the application.

This document defines the valid, contextual rule required, disallowed, and unassigned rules for the IdentifierClass and FreeformClass. As described under Section 5, profiles of these string classes are responsible for defining the width mapping, additional mappings, case mapping, normalization, and directionality rules.

4.2. IdentifierClass

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

4.2.1. Valid

- o Code points traditionally used as letters and numbers in writing systems, i.e., the LetterDigits ("A") category first defined in [RFC5892] and listed here under Section 9.1.
- o Code points in the range U+0021 through U+007E, i.e., the (printable) ASCII7 ("K") rule defined under Section 9.11. These code points are "grandfathered" into PRECIS and thus are valid even if they would otherwise be disallowed according to the property-based rules specified in the next section.

Note: Although the PRECIS IdentifierClass re-uses the LetterDigits category from IDNA2008, the range of characters allowed in the IdentifierClass is wider than the range of characters allowed in IDNA2008. The main reason is that IDNA2008 applies the Unstable category before the LetterDigits category, thus disallowing uppercase characters, whereas the IdentifierClass does not apply the Unstable category.

4.2.2. Contextual Rule Required

- o A number of characters from the Exceptions ("F") category defined under Section 9.6 (see Section 9.6 for a full list).
- o Joining characters, i.e., the JoinControl ("H") category defined under Section 9.8.

4.2.3. Disallowed

- o Old Hangul Jamo characters, i.e., the OldHangulJamo ("I") category defined under Section 9.9.
- o Control characters, i.e., the Controls ("L") category defined under Section 9.12.
- o Ignorable characters, i.e., the PrecisIgnorableProperties ("M") category defined under Section 9.13.
- o Space characters, i.e., the Spaces ("N") category defined under Section 9.14.
- o Symbol characters, i.e., the Symbols ("O") category defined under Section 9.15.
- o Punctuation characters, i.e., the Punctuation ("P") category defined under Section 9.16.
- o Any character that has a compatibility equivalent, i.e., the HasCompat ("Q") category defined under Section 9.17. These code points are disallowed even if they would otherwise be valid according to the property-based rules specified in the previous section.
- o Letters and digits other than the "traditional" letters and digits allowed in IDNs, i.e., the OtherLetterDigits ("R") category defined under Section 9.18.

4.2.4. Unassigned

Any code points that are not yet designated in the Unicode character set are considered Unassigned for purposes of the IdentifierClass, and such code points are to be treated as Disallowed. See Section 9.10.

4.2.5. Examples

As described in the Introduction to this document, the string classes do not handle all issues related to string preparation and comparison (such as case mapping); instead, such issues are handled at the level of profiles. Examples for two profiles of the IdentifierClass can be found in [I-D.ietf-precis-saslprepbis] (the UsernameIdentifierClass profile) and in [I-D.ietf-xmpp-6122bis] (the LocalpartIdentifierClass profile).

4.3. FreeformClass

Some application technologies need strings that can be used in a free-form way, e.g., as a password in an authentication exchange (see [I-D.ietf-precis-saslprepbis]) or a nickname in a chatroom (see [I-D.ietf-precis-nickname]). We group such things into a class called "FreeformClass" having the following features.

Security Warning: As mentioned, the FreeformClass prioritizes expressiveness over safety; Section 12.3 describes some of the security hazards involved with using or profiling the FreeformClass.

Security Warning: Consult Section 12.6 for relevant security considerations when strings conforming to the FreeformClass, or a profile thereof, are used as passwords.

4.3.1. Valid

- o Traditional letters and numbers, i.e., the LetterDigits ("A") category first defined in [RFC5892] and listed here under Section 9.1.
- o Letters and digits other than the "traditional" letters and digits allowed in IDNs, i.e., the OtherLetterDigits ("R") category defined under Section 9.18.
- o Code points in the range U+0021 through U+007E, i.e., the (printable) ASCII7 ("K") rule defined under Section 9.11.
- o Any character that has a compatibility equivalent, i.e., the HasCompat ("Q") category defined under Section 9.17.
- o Space characters, i.e., the Spaces ("N") category defined under Section 9.14.
- o Symbol characters, i.e., the Symbols ("O") category defined under Section 9.15.
- o Punctuation characters, i.e., the Punctuation ("P") category defined under Section 9.16.

4.3.2. Contextual Rule Required

- o A number of characters from the Exceptions ("F") category defined under Section 9.6 (see Section 9.6 for a full list).

- o Joining characters, i.e., the JoinControl ("H") category defined under Section 9.8.

4.3.3. Disallowed

- o Old Hangul Jamo characters, i.e., the OldHangulJamo ("I") category defined under Section 9.9.
- o Control characters, i.e., the Controls ("L") category defined under Section 9.12.
- o Ignorable characters, i.e., the PrecisIgnorableProperties ("M") category defined under Section 9.13.

4.3.4. Unassigned

Any code points that are not yet designated in the Unicode character set are considered Unassigned for purposes of the FreeformClass, and such code points are to be treated as Disallowed.

4.3.5. Examples

As described in the Introduction to this document, the string classes do not handle all issues related to string preparation and comparison (such as case mapping); instead, such issues are handled at the level of profiles. Examples for two profiles of the FreeformClass can be found in [I-D.ietf-precis-nickname] (the NicknameFreeformClass profile) and in [I-D.ietf-xmpp-6122bis] (the ResourcepartIdentifierClass profile).

5. Profiles

This framework document defines the valid, contextual-rule-required, disallowed, and unassigned rules for the IdentifierClass and the FreeformClass. A profile of a PRECIS string class MUST define the width mapping, additional mappings (if any), case mapping, normalization, and directionality rules. A profile MAY also restrict the allowable characters above and beyond the definition of the relevant PRECIS string class (but MUST NOT add as valid any code points that are disallowed by the relevant PRECIS string class). These matters are discussed in the following subsections.

Profiles of the PRECIS string classes are registered with the IANA as described under Section 11.3. Profile names use the following convention: they are of the form "Profilename of BaseClass", where the "Profilename" string is a differentiator and "BaseClass" is the name of the PRECIS string class being profiled; for example, the

profile of the Freeform used for opaque strings such as passwords is the "OpaqueString" profile [I-D.ietf-precis-saslprepbis].

5.1. Profiles Must Not Be Multiplied Beyond Necessity

The risk of profile proliferation is significant because having too many profiles will result in different behavior across various applications, thus violating what is known in user interface design as the Principle of Least Astonishment.

Indeed, we already have too many profiles. Ideally we would have at most two or three profiles. Unfortunately, numerous application protocols exist with their own quirks regarding protocol strings. Domain names, email addresses, instant messaging addresses, chatroom nicknames, filenames, authentication identifiers, passwords, and other strings are already out there in the wild and need to be supported in existing application protocols such as DNS, SMTP, XMPP, IRC, NFS, iSCSI, EAP, and SASL among others.

Nevertheless, profiles must not be multiplied beyond necessity.

To help prevent profile proliferation, this document recommends sensible defaults for the various options offered to profile creators (such as width mapping and Unicode normalization). In addition, the guidelines for designated experts provided under Section 10 are meant to encourage a high level of due diligence regarding new profiles.

5.2. Rules

5.2.1. Width Mapping Rule

The width mapping rule of a profile specifies whether width mapping is performed on the characters of a string, and how the mapping is done. Typically such mapping consists of mapping fullwidth and halfwidth characters, i.e., code points with a Decomposition Type of Wide or Narrow, to their decomposition mappings; as an example, FULLWIDTH DIGIT ZERO (U+FF10) would be mapped to DIGIT ZERO (U+0030).

The normalization form specified by a profile (see below) has an impact on the need for width mapping. Because width mapping is performed as a part of compatibility decomposition, a profile employing either normalization form KD (NFKD) or normalization form KC (NFKC) does not need to specify width mapping. However, if Unicode normalization form C (NFC) is used (as is recommended) then the profile needs to specify whether to apply width mapping; in this case, width mapping is in general RECOMMENDED because allowing fullwidth and halfwidth characters to remain unmapped to their compatibility variants would violate the Principle of Least

Astonishment. For more information about the concept of width in East Asian scripts within Unicode, see Unicode Standard Annex #11 [UAX11].

5.2.2. Additional Mapping Rule

The additional mapping rule of a profile specifies whether additional mappings is performed on the characters of a string, such as:

Mapping of delimiter characters (such as '@', ':', '/', '+', and '-')

Mapping of special characters (e.g., non-ASCII space characters to ASCII space or control characters to nothing).

The PRECIS mappings document [I-D.ietf-precis-mappings] describes such mappings in more detail.

5.2.3. Case Mapping Rule

The case mapping rule of a profile specifies whether case mapping (instead of case preservation) is performed on the characters of a string, and how the mapping is applied (e.g., mapping uppercase and titlecase characters to their lowercase equivalents).

If case mapping is desired (instead of case preservation), it is RECOMMENDED to use Unicode Default Case Folding as defined in Chapter 3 of the Unicode Standard [Unicode7.0].

Note: Unicode Default Case Folding is not designed to handle various localization issues (such as so-called "dotless i" in several Turkic languages). The PRECIS mappings document [I-D.ietf-precis-mappings] describes these issues in greater detail and defines a "local case mapping" method that handles some locale-dependent and context-dependent mappings.

In order to maximize entropy and minimize the potential for false positives, it is NOT RECOMMENDED for application protocols to map uppercase and titlecase code points to their lowercase equivalents when strings conforming to the FreeformClass, or a profile thereof, are used in passwords; instead, it is RECOMMENDED to preserve the case of all code points contained in such strings and then perform case-sensitive comparison. See also the related discussion in [I-D.ietf-precis-saslprep].

5.2.4. Normalization Rule

The normalization rule of a profile specifies which Unicode normalization form (D, KD, C, or KC) is to be applied (see Unicode Standard Annex #15 [UAX15] for background information).

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

5.2.5. Directionality Rule

The directionality rule of a profile specifies how to treat strings containing what are often called "right-to-left" (RTL) characters (see Unicode Standard Annex #9 [UAX9]). RTL characters come from scripts that are normally written from right to left and are considered by Unicode to, themselves, have right-to-left directionality. Some strings containing RTL characters also contain "left-to-right" (LTR) characters, such as numerals, as well as characters without directional properties. Consequently, such strings are known as "bidirectional strings".

Presenting bidirectional strings in different layout systems (e.g., a user interface that is configured to handle primarily an RTL script vs. an interface that is configured to handle primarily an LTR script) can yield display results that, while predictable to those who understand the display rules, are counter-intuitive to casual users. In particular, the same bidirectional string (in PRECIS terms) might not be presented in the same way to users of those different layout systems, even though the presentation is consistent within any particular layout system. In some applications, these presentation differences might be considered problematic and thus the application designers might wish to restrict the use of bidirectional strings by specifying a directionality rule. In other applications, these presentation differences might not be considered problematic (this especially tends to be true of more "free-form" strings) and thus no directionality rule is needed.

The PRECIS framework does not directly address how to deal with bidirectional strings across all string classes and profiles, and does not define any new directionality rules, since at present there is no widely accepted and implemented solution for the safe display of arbitrary bidirectional strings beyond the Unicode bidirectional algorithm [UAX9]. Although rules for management and display of bidirectional strings have been defined for domain name labels and similar identifiers through the "Bidi Rule" specified in the IDNA2008 specification on right-to-left scripts [RFC5893], those rules are quite restrictive and are not necessarily applicable to all bidirectional strings.

The authors of a PRECIS profile might believe that they need to define a new directionality rule of their own. Because of the complexity of the issues involved, such a belief is almost always misguided, even if the authors have done a great deal of careful research into the challenges of displaying bidirectional strings. This document strongly suggests that profile authors who are thinking about defining a new directionality rule think again, and instead consider using the "Bidi Rule" [RFC5893] (for profiles based on the IdentifierClass) or following the Unicode bidirectional algorithm [UAX9] (for profiles based on the FreeformClass or in situations where the IdentifierClass is not appropriate).

5.3. A Note about Spaces

With regard to the IdentifierClass, the consensus of the PRECIS Working Group was that spaces are problematic for many reasons, including:

- o Many Unicode characters are confusable with ASCII space.
- o Even if non-ASCII space characters are mapped to ASCII space (U+0020), space characters are often not rendered in user interfaces, leading to the possibility that a human user might consider a string containing spaces to be equivalent to the same string without spaces.
- o In some locales, some devices are known to generate a character other than ASCII space (such as ZERO WIDTH JOINER, U+200D) when a user performs an action like hitting the space bar on a keyboard.

One consequence of disallowing space characters in the IdentifierClass might be to effectively discourage their use within identifiers created in newer application protocols; given the challenges involved with properly handling space characters (especially non-ASCII space characters) in identifiers and other protocol strings, the PRECIS Working Group considered this to be a feature, not a bug.

However, the FreeformClass does allow spaces, which enables application protocols to define profiles of the FreeformClass that are more flexible than any profiles of the IdentifierClass. In addition, as explained in the previous section, application protocols can also define application-layer constructs containing spaces.

6. Applications

6.1. How to Use PRECIS in Applications

Although PRECIS has been designed with applications in mind, internationalization is not suddenly made easy though the use of PRECIS. Application developers still need to give some thought to how they will use the PRECIS string classes, or profiles thereof, in their applications. This section provides some guidelines to application developers (and to expert reviewers of application protocol specifications).

- o Don't define your own profile unless absolutely necessary (see Section 5.1). Existing profiles have been design for wide re-use. It is highly likely that an existing profile will meet your needs, especially given the ability to specify further excluded characters (Section 6.2) and to build application-layer constructs (see Section 6.3).
- o Do specify:
 - * Exactly which entities are responsible for preparation, enforcement, and comparison of internationalized strings (e.g., servers or clients).
 - * Exactly when those entities need to complete their tasks (e.g., a server might need to enforce the rules of a profile before allowing a client to gain network access).
 - * Exactly which protocol slots need to be checked against which profiles (e.g., checking the address of a message's intended recipient against the UsernameCaseMapped profile [I-D.ietf-precis-saslprepbis] of the IdentifierClass, or checking the password of a user against the OpaqueString profile [I-D.ietf-precis-saslprepbis] of the FreeformClass).

See [I-D.ietf-precis-saslprepbis] and [I-D.ietf-xmpp-6122bis] for definitions of these matters for several applications.

6.2. Further Excluded Characters

An application protocol that uses a profile MAY specify particular code points that are not allowed in relevant slots within that application protocol, above and beyond those excluded by the string class or profile.

That is, an application protocol MAY do either of the following:

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

As a result of such exclusions, code points that are defined as valid for the PRECIS string class or profile will be defined as disallowed for the relevant protocol slot.

Typically, such exclusions are defined for the purpose of backward-compatibility with legacy formats within an application protocol. These are defined for application protocols, not profiles, in order to prevent multiplication of profiles beyond necessity (see Section 5.1).

6.3. Building Application-Layer Constructs

Sometimes, an application-layer construct does not map in a straightforward manner to one of the base string classes or a profile thereof. Consider, for example, the "simple user name" construct in the Simple Authentication and Security Layer (SASL) [RFC4422]. Depending on the deployment, a simple user name might take the form of a user's full name (e.g., the user's personal name followed by a space and then the user's family name). Such a simple user name cannot be defined as an instance of the IdentifierClass or a profile thereof, since space characters are not allowed in the IdentifierClass; however, it could be defined using a space-separated sequence of IdentifierClass instances, as in the following ABNF [RFC5234] from [I-D.ietf-precis-saslprepbis]:

```
username    = userpart *(1*SP userpart)
userpart    = 1*(idbyte)
            ;
            ; an "idbyte" is a byte used to represent a
            ; UTF-8 encoded Unicode code point that can be
            ; contained in a string that conforms to the
            ; PRECIS "IdentifierClass"
            ;
```

Similar techniques could be used to define many application-layer constructs, say of the form "user@domain" or "/path/to/file".

7. Order of Operations

To ensure proper comparison, the rules specified for a particular string class or profile **MUST** be applied in the following order:

1. Width Mapping Rule
2. Additional Mapping Rule
3. Case Mapping Rule
4. Normalization Rule
5. Directionality Rule
6. Behavioral rules for determining whether a code point is valid, allowed under a contextual rule, disallowed, or unassigned

As already described, the width mapping, additional mapping, case mapping, normalization, and directionality rules are specified for each profile, whereas the behavioral rules are specified for each string class. Some of the logic behind this order is provided under Section 5.2.1 (see also the PRECIS mappings document [I-D.ietf-precis-mappings]).

8. 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 is the responsibility of the protocol specification that uses or profiles a PRECIS 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 (currently, IdentifierClass and FreeformClass). 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. In the remainder of this document, the abbreviated term *_PVAL is used, where * = (ID | FREE), i.e., either "FREE_PVAL" or "ID_PVAL". In practice, the derived property ID_PVAL is not used in this specification, since every ID_PVAL code point is PVALID.

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. As in IDNA2008, there are two subdivisions of CONTEXTUAL RULE REQUIRED, the first for Join_controls (called "CONTEXTJ") and the second for other characters (called "CONTEXT0"). A character with the derived property value CONTEXTJ or CONTEXT0 MUST NOT be used unless an appropriate rule has been established and the context of the character is consistent with that rule. The most notable of the CONTEXTUAL RULE REQUIRED characters are the Join Control characters U+200D ZERO WIDTH JOINER and U+200C ZERO WIDTH NON-JOINER, which have a derived property value of CONTEXTJ. See Appendix A of [RFC5892] for more information.

DISALLOWED Those code points that are not permitted in any PRECIS string class.

SPECIFIC CLASS DISALLOWED Those code points that are not to be included in one of the string classes but that might be permitted in others. In the remainder of this document, the abbreviated term *_DIS is used, where * = (ID | FREE), i.e., either "FREE_DIS" or "ID_DIS". In practice, the derived property FREE_DIS is not used in this specification, since every FREE_DIS code point is DISALLOWED.

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

The algorithm to calculate the value of the derived property is as follows (implementations MUST NOT modify the order of operations within this algorithm, since doing so would cause inconsistent results across implementations):

```
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. OldHangulJamo Then DISALLOWED;
Else If .cp. .in. PrecisIgnorableProperties Then DISALLOWED;
Else If .cp. .in. Controls Then DISALLOWED;
Else If .cp. .in. HasCompat Then ID_DIS or FREE_PVAL;
Else If .cp. .in. LetterDigits Then PVALID;
Else If .cp. .in. OtherLetterDigits Then ID_DIS or FREE_PVAL;
Else If .cp. .in. Spaces Then ID_DIS or FREE_PVAL;
Else If .cp. .in. Symbols Then ID_DIS or FREE_PVAL;
Else If .cp. .in. Punctuation Then ID_DIS or FREE_PVAL;
Else DISALLOWED;
```

The value of the derived property calculated can depend on the string class; for example, if an identifier used in an application protocol is defined as profiling the PRECIS IdentifierClass then a space character such as U+0020 would be assigned to ID_DIS, whereas if an identifier is defined as profiling the PRECIS FreeformClass then the character would be assigned to FREE_PVAL. For the sake of brevity, the designation "FREE_PVAL" is used herein, instead of the longer designation "ID_DIS or FREE_PVAL". In practice, the derived properties ID_PVAL and FREE_DIS are not used in this specification, since every ID_PVAL code point is PVALID and every FREE_DIS code point is DISALLOWED.

Use of the name of a rule (such as "Exceptions") implies the set of code points that the rule defines, whereas the same name as a function call (such as "Exceptions(cp)") implies the value that the code point has in the Exceptions table.

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 7.0 depending on the category, since some categories mentioned in this document are simply pointers to IDNA2008 and therefore were defined at the time of Unicode 5.2). Changes in Unicode properties that do not affect the outcome of this process therefore do not affect this framework. For example, a character can have its Unicode General_Category value (see Chapter 4 of the Unicode Standard [Unicode7.0]) 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 9.7) can be adjusted to ensure the stability of the results.

9. Category Definitions Used to Calculate Derived Property

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 based on 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 specific to a given string class. These operations are specified under Section 8.

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 referenced from [RFC5892] to ease the understanding of how PRECIS handles various characters. 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 eight categories (K-R) are specific to PRECIS.

9.1. LetterDigits (A)

This category is defined in Section 2.1 of [RFC5892] and is included by reference for use in PRECIS.

9.2. Unstable (B)

This category is defined in Section 2.2 of [RFC5892]. However, it is not used in PRECIS.

9.3. IgnorableProperties (C)

This category is defined in Section 2.3 of [RFC5892]. However, it is not used in PRECIS.

Note: See the "PrecisIgnorableProperties (M)" category below for a more inclusive category used in PRECIS identifiers.

9.4. IgnorableBlocks (D)

This category is defined in Section 2.4 of [RFC5892]. However, it is not used in PRECIS.

9.5. LDH (E)

This category is defined in Section 2.5 of [RFC5892]. However, it is not used in PRECIS.

Note: See the "ASCII7 (K)" category below for a more inclusive category used in PRECIS identifiers.

9.6. Exceptions (F)

This category is defined in Section 2.6 of [RFC5892] and is included by reference for use in PRECIS.

9.7. BackwardCompatible (G)

This category is defined in Section 2.7 of [RFC5892] and is included by reference for use in PRECIS.

Note: Management of this category is handled via the processes specified in [RFC5892]. At the time of this writing (and also at the time that RFC 5892 was published), this category consisted of the empty set; however, that is subject to change as described in RFC 5892.

9.8. JoinControl (H)

This category is defined in Section 2.8 of [RFC5892] and is included by reference for use in PRECIS.

9.9. OldHangulJamo (I)

This category is defined in Section 2.9 of [RFC5892] and is included by reference for use in PRECIS.

9.10. Unassigned (J)

This category is defined in Section 2.10 of [RFC5892] and is included by reference for use in PRECIS.

9.11. ASCII7 (K)

This PRECIS-specific category consists of all printable, non-space characters from the 7-bit ASCII range. By applying this category, the algorithm specified under Section 8 exempts these characters 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}

9.12. Controls (L)

This PRECIS-specific category consists of all control characters.

L: Control(cp) = True

9.13. PrecisIgnorableProperties (M)

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

9.14. Spaces (N)

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

N: General_Category(cp) is in {Zs}

9.15. Symbols (O)

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

O: General_Category(cp) is in {Sm, Sc, Sk, So}

9.16. Punctuation (P)

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

P: General_Category(cp) is in {Pc, Pd, Ps, Pe, Pi, Pf, Po}

9.17. 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 the Unicode Standard [Unicode7.0].

Q: toNFKC(cp) != cp

The toNFKC() operation returns the code point in normalization form KC. For more information, see Section 5 of Unicode Standard Annex #15 [UAX15].

9.18. OtherLetterDigits (R)

This PRECIS-specific category is used to group code points that are letters and digits other than the "traditional" letters and digits grouped under the LetterDigits (A) class (see Section 9.1).

R: General_Category(cp) is in {Lt, Nl, No, Me}

10. Guidelines for Designated Experts

Experience with internationalization in application protocols has shown that protocol designers and application developers usually do not understand the subtleties and tradeoffs involved with internationalization and that they need considerable guidance in making reasonable decisions with regard to the options before them.

Therefore:

- o Protocol designers are strongly encouraged to question the assumption that they need to define new profiles, since existing profiles are designed for wide re-use (see Section 5 for further discussion).
- o Those who persist in defining new profiles are strongly encouraged to clearly explain a strong justification for doing so, and to publish a stable specification that provides all of the information described under Section 11.3.

- o The designated experts for profile registration requests ought to seek answers to all of the questions provided under Section 11.3 and to encourage applicants to provide a stable specification documenting the profile (even though the registration policy for PRECIS profiles is Expert Review and a stable specification is not strictly required).
- o Developers of applications that use PRECIS are strongly encouraged to apply the guidelines provided under Section 6 and to seek out the advice of the designated experts or other knowledgeable individuals in doing so.
- o All parties are strongly encouraged to help prevent the multiplication of profiles beyond necessity, as described under Section 5.1, and to use PRECIS in ways that will minimize user confusion and insecure application behavior.

Internationalization can be difficult and contentious; designated experts, profile registrants, and application developers are strongly encouraged to work together in a spirit of good faith and mutual understanding to achieve rough consensus on profile registration requests and the use of PRECIS in particular applications. They are also encouraged to bring additional expertise into the discussion if that would be helpful in adding perspective or otherwise resolving issues.

11. IANA Considerations

11.1. 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 7.0. The derived property value is to be calculated in cooperation with a designated expert [RFC5226] according to the rules specified under Section 8 and Section 9.

The IESG is to be notified if backward-incompatible changes to the table of derived properties are discovered or if other problems arise during the process of creating the table of derived property values or during expert review. Changes to the rules defined under Section 8 and Section 9 require IETF Review.

11.2. PRECIS Base Classes Registry

IANA is requested to create a registry of PRECIS string classes. In accordance with [RFC5226], the registration policy is "RFC Required".

The registration template is as follows:

Base Class: [the name of the PRECIS string class]

Description: [a brief description of the PRECIS string class and its intended use, e.g., "A sequence of letters, numbers, and symbols that is used to identify or address a network entity."]

Specification: [the RFC number]

The initial registrations are as follows:

Base Class: FreeformClass.

Description: A sequence of letters, numbers, symbols, spaces, and other code points that is used for free-form strings.

Specification: Section 4.3 of this document.

[Note to RFC Editor: please change "this document" to the RFC number issued for this specification.]

Base Class: IdentifierClass.

Description: A sequence of letters, numbers, and symbols that is used to identify or address a network entity.

Specification: Section 4.2 of this document.

[Note to RFC Editor: please change "this document" to the RFC number issued for this specification.]

11.3. PRECIS Profiles Registry

IANA is requested to create a registry of profiles that use the PRECIS string classes. In accordance with [RFC5226], the registration policy is "Expert Review". This policy was chosen in order to ease the burden of registration while ensuring that "customers" of PRECIS receive appropriate guidance regarding the sometimes complex and subtle internationalization issues related to profiles of PRECIS string classes.

The registration template is as follows:

Name: [the name of the profile]

Base Class: [which PRECIS string class is being profiled]

Applicability: [the specific protocol elements to which this profile applies, e.g., "Localparts in XMPP addresses."]

Replaces: [the Stringprep profile that this PRECIS profile replaces, if any]

Width Mapping Rule: [the behavioral rule for handling of width, e.g., "Map fullwidth and halfwidth characters to their compatibility variants."]

Additional Mapping Rule: [any additional mappings are required or recommended, e.g., "Map non-ASCII space characters to ASCII space."]

Case Mapping Rule: [the behavioral rule for handling of case, e.g., "Unicode Default Case Folding"]

Normalization Rule: [which Unicode normalization form is applied, e.g., "NFC"]

Directionality Rule: [the behavioral rule for handling of right-to-left code points, e.g., "The 'Bidi Rule' defined in RFC 5893 applies."]

Enforcement: [which entities enforce the rules, and when that enforcement occurs during protocol operations]

Specification: [a pointer to relevant documentation, such as an RFC or Internet-Draft]

In order to request a review, the registrant shall send a completed template to the precis@ietf.org list or its designated successor.

Factors to focus on while defining profiles and reviewing profile registrations include the following:

- o Would an existing PRECIS string class or profile solve the problem? If not, why not? (See Section 5.1 for related considerations.)
- o Is the problem being addressed by this profile well-defined?
- o Does the specification define what kinds of applications are involved and the protocol elements to which this profile applies?
- o Is the profile clearly defined?
- o Is the profile based on an appropriate dividing line between user interface (culture, context, intent, locale, device limitations, etc.) and the use of conformant strings in protocol elements?
- o Are the width mapping, case mapping, additional mappings, normalization, and directionality rules appropriate for the intended use?

- o Does the profile explain which entities enforce the rules, and when such enforcement occurs during protocol operations?
- o Does the profile reduce the degree to which human users could be surprised or confused by application behavior (the "Principle of Least Astonishment")?
- o Does the profile introduce any new security concerns such as those described under Section 12 of this document (e.g., false positives for authentication or authorization)?

12. Security Considerations

12.1. General Issues

If input strings that appear "the same" to users are programmatically considered to be distinct in different systems, or if input strings that appear distinct to users are programmatically considered to be "the same" in different systems, then users can be confused. Such confusion can have security implications, such as the false positives and false negatives discussed in [RFC6943]. One starting goal of work on the PRECIS framework was to limit the number of times that users are confused (consistent with the "Principle of Least Astonishment"). Unfortunately, this goal has been difficult to achieve given the large number of application protocols already in existence. Despite these difficulties, profiles should not be multiplied beyond necessity (see Section 5.1. In particular, application protocol designers should think long and hard before defining a new profile instead of using one that has already been defined, and if they decide to define a new profile then they should clearly explain their reasons for doing so.

The security of applications that use this framework can depend in part on the proper preparation, enforcement, 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 (again, see [RFC6943]).

Specifications of application protocols that use this framework are strongly 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 enforcement and comparison operations. For some helpful guidelines, refer to [RFC6943], [RFC5890], [UTR36], and [UTS39].

12.2. Use of the IdentifierClass

Strings that conform to the IdentifierClass and any profile thereof are intended to be relatively safe for use in a broad range of applications, primarily because they include only letters, digits, and "grandfathered" non-space characters from the ASCII range; thus they exclude spaces, characters with compatibility equivalents, and almost all symbols and punctuation marks. However, because such strings can still include so-called confusable characters (see Section 12.5), protocol designers and implementers are encouraged to pay close attention to the security considerations described elsewhere in this document.

12.3. Use of the FreeformClass

Strings that conform to the FreeformClass and many profiles thereof can include virtually any Unicode character. This makes the FreeformClass quite expressive, but also problematic from the perspective of possible user confusion. Protocol designers are hereby warned that the FreeformClass contains codepoints they might not understand, and are encouraged to profile the IdentifierClass wherever feasible; however, if an application protocol requires more code points than are allowed by the IdentifierClass, protocol designers are encouraged to define a profile of the FreeformClass that restricts the allowable code points as tightly as possible. (The PRECIS Working Group considered the option of allowing "superclasses" as well as profiles of PRECIS string classes, but decided against allowing superclasses to reduce the likelihood of security and interoperability problems.)

12.4. Local Character Set Issues

When systems use local character sets other than ASCII and Unicode, this specification leaves 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.

12.5. 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 ASCII range. For example, in some presentations and to some individuals the string "juliet" (spelled with DIGIT ONE, U+0031, as the third character) might appear to be the same as "juliet" (spelled with LATIN SMALL LETTER L, U+006C), 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 ASCII characters "STPETER" as they might appear when presented using a "creative" font family.

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 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] and the Unicode Security Mechanisms [UTS39], 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 font families used), the PRECIS framework does nothing to map similar-looking characters together, nor does it prohibit some characters because they look like others.

Nevertheless, specifications for application protocols that use this framework are strongly encouraged to describe how confusable characters can be abused to compromise the security of systems that use the protocol in question, along with 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 strings that contain characters from more than one script and SHOULD 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 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.)

The challenges inherent in supporting the full range of Unicode code points have in the past led some to hope for a way to programmatically negotiate more restrictive ranges based on locale, script, or other relevant factors, to tag the locale associated with a particular string, etc. As a general-purpose internationalization technology, the PRECIS framework does not include such mechanisms.

12.6. Security of Passwords

Two goals of passwords are to maximize the amount of entropy and to minimize the potential for false positives. These goals can be achieved in part by allowing a wide range of code points and by ensuring that passwords are handled in such a way that code points are not compared aggressively. Therefore, it is NOT RECOMMENDED for application protocols to profile the FreeformClass for use in

passwords 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 in such strings; instead, it is RECOMMENDED to preserve the case of all code points contained in such strings and to compare them in a case-sensitive manner.

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 a service when using an unfamiliar or constrained input device.

Some application protocols use passwords directly, whereas others reuse technologies that themselves process passwords (one example of such a technology is the Simple Authentication and Security Layer [RFC4422]). Moreover, passwords are often carried by a sequence of protocols with backend 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.

In protocols that provide passwords as input to a cryptographic algorithm such as a hash function, the client will need to perform proper preparation of the password before applying the algorithm, since the password is not available to the server in plaintext form.

Further discussion of password handling can be found in [I-D.ietf-precis-saslprepbis].

13. Interoperability Considerations

13.1. Encoding

Although strings that are consumed in PRECIS-based application protocols are often encoded using UTF-8 [RFC3629], the exact encoding is a matter for the application protocol that uses PRECIS, not for the PRECIS framework.

13.2. Character Sets

It is known that some existing systems are unable to support the full Unicode character set, or even any characters outside the ASCII range. If two (or more) applications need to interoperate when exchanging data (e.g., for the purpose of authenticating a username or password), they will naturally need to have in common at least one

coded character set (as defined by [RFC6365])). Establishing such a baseline is a matter for the application protocol that uses PRECIS, not for the PRECIS framework.

13.3. Unicode Versions

Changes to the properties of Unicode code points can occur as the Unicode Standard is modified from time to time. For example, three code points underwent changes in their GeneralCategory between Unicode 5.2 (current at the time IDNA2008 was originally published) and Unicode 6.0, as described in [RFC6452]. Implementers might need to be aware that the treatment of these characters differs depending on which version of Unicode is available on the system that is using IDNA2008 or PRECIS. Other such differences might arise between the version of Unicode current at the time of this writing (7.0) and future versions.

13.4. Potential Changes to Handling of Certain Unicode Code Points

As part of the review of Unicode 7.0 for IDNA, a question was raised about a newly-added code point that led to a re-analysis of the Normalization Rules used by IDNA and inherited by this document (Section 5.2.4). Some of the general issues are described in [IAB-Statement] and pursued in more detail in [I-D.klensin-idna-5892upd-unicode70].

At the time of writing, these issues have yet to be settled. However, implementers need to be aware that this specification is likely to be updated in the future to address these issues. The potential changes include:

- o The range of characters in the LetterDigits category (Section 4.2.1 and Section 9.1) might be narrowed.
- o Some characters with special properties that are now allowed might be excluded.
- o More "Additional Mapping Rules" (Section 5.2.2) might be defined.
- o Alternative normalization methods might be added.

Nevertheless, implementations and deployments that are sensitive to the advice given in this specification are unlikely to run into significant problems as a consequence of these issues or potential changes - specifically the advice to use the more restrictive IdentifierClass whenever possible, or if using the FreeformClass to allow only a restricted set of characters, particularly avoiding characters whose implications they do not actually understand.

14. References

14.1. Normative References

- [RFC20] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.
- [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.
- [Unicode7.0]
The Unicode Consortium, "The Unicode Standard, Version 7.0.0", 2014,
<<http://www.unicode.org/versions/Unicode7.0.0/>>.

14.2. Informative References

- [IAB-Statement]
Internet Architecture Board, "IAB Statement on Identifiers and Unicode 7.0.0", January 2015, <<https://www.iab.org/documents/correspondence-reports-documents/2015-2/iab-statement-on-identifiers-and-unicode-7-0-0/>>.
- [I-D.ietf-precis-mappings]
Yoneya, Y. and T. NEMOTO, "Mapping characters for PRECIS classes", draft-ietf-precis-mappings-08 (work in progress), June 2014.
- [I-D.ietf-precis-nickname]
Saint-Andre, P., "Preparation and Comparison of Nicknames", draft-ietf-precis-nickname-14 (work in progress), December 2014.
- [I-D.ietf-precis-saslprepbis]
Saint-Andre, P. and A. Melnikov, "Username and Password Preparation Algorithms", draft-ietf-precis-saslprepbis-13 (work in progress), December 2014.
- [I-D.ietf-xmpp-6122bis]
Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Address Format", draft-ietf-xmpp-6122bis-18 (work in progress), December 2014.

- [I-D.klensin-idna-5892upd-unicode70]
Klensin, J. and P. Faeltsstroem, "IDNA Update for Unicode 7.0.0", draft-klensin-idna-5892upd-unicode70-03 (work in progress), January 2015.
- [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.
- [RFC3491] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 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.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 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.
- [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.
- [RFC6885] Blanchet, M. and A. Sullivan, "Stringprep Revision and Problem Statement for the Preparation and Comparison of Internationalized Strings (PRECIS)", RFC 6885, March 2013.
- [RFC6943] Thaler, D., "Issues in Identifier Comparison for Security Purposes", RFC 6943, May 2013.
- [UAX9] The Unicode Consortium, "Unicode Standard Annex #9: Unicode Bidirectional Algorithm", September 2012, <<http://unicode.org/reports/tr9/>>.
- [UAX11] The Unicode Consortium, "Unicode Standard Annex #11: East Asian Width", September 2012, <<http://unicode.org/reports/tr11/>>.
- [UAX15] The Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", August 2012, <<http://unicode.org/reports/tr15/>>.

[UnicodeCurrent]

The Unicode Consortium, "The Unicode Standard",
2014-present, <<http://www.unicode.org/versions/latest/>>.

[UTR36] The Unicode Consortium, "Unicode Technical Report #36:
Unicode Security Considerations", July 2012,
<<http://unicode.org/reports/tr36/>>.

[UTS39] The Unicode Consortium, "Unicode Technical Standard #39:
Unicode Security Mechanisms", July 2012,
<<http://unicode.org/reports/tr39/>>.

14.3. URIs

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

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

Appendix A. Acknowledgements

The authors would like to acknowledge the comments and contributions of the following individuals during working group discussion: David Black, Edward Burns, Dan Chiba, Mark Davis, Alan DeKok, Martin Duerst, Patrik Faltstrom, Ted Hardie, Joe Hildebrand, Bjoern Hoehrmann, Paul Hoffman, Jeffrey Hutzelman, Simon Josefsson, John Klensin, Alexey Melnikov, Takahiro Nemoto, Yoav Nir, Mike Parker, Pete Resnick, Andrew Sullivan, Dave Thaler, Yoshiro Yoneya, and Florian Zeitz.

Special thanks are due to John Klensin and Patrik Faltstrom for their challenging feedback and detailed reviews.

Charlie Kaufman, Tom Taylor, and Tim Wicinski reviewed the document on behalf of the Security Directorate, the General Area Review Team, and the Operations and Management Directorate, respectively.

During IESG review, Alissa Cooper, Stephen Farrell, and Barry Leiba provided comments that led to further improvements.

Some algorithms and textual descriptions have been borrowed from [RFC5892]. Some text regarding security has been borrowed from [RFC5890], [I-D.ietf-precis-saslprep-bis], and [I-D.ietf-xmpp-6122bis].

Peter Saint-Andre wishes to acknowledge Cisco Systems, Inc., for employing him during his work on earlier versions of this document.

Authors' Addresses

Peter Saint-Andre
&yet

Email: peter@andyet.com
URI: <https://andyet.com/>

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

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

PRECIS
Internet-Draft
Intended status: Standards Track
Expires: March 6, 2016

P. Saint-Andre
&yet
September 3, 2015

Preparation, Enforcement, and Comparison of Internationalized Strings
Representing Nicknames
draft-ietf-precis-nickname-19

Abstract

This document describes methods for handling Unicode strings representing memorable, human-friendly names (variously called "nicknames", "display names", or "petnames") for people, devices, accounts, websites, and other entities.

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 March 6, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Overview	2
1.2. Terminology	3
2. Nickname Profile	3
2.1. Rules	3
2.2. Preparation	4
2.3. Enforcement	5
2.4. Comparison	5
3. Examples	5
4. Use in Application Protocols	6
5. IANA Considerations	7
6. Security Considerations	8
6.1. Reuse of PRECIS	8
6.2. Reuse of Unicode	8
6.3. Visually Similar Characters	8
7. References	8
7.1. Normative References	8
7.2. Informative References	9
7.3. URIs	10
Appendix A. Acknowledgements	10
Author's Address	10

1. Introduction

1.1. Overview

A number of technologies and applications provide the ability for a person to choose a memorable, human-friendly name in a communications context, or to set such a name for another entity entity such as a device, account, contact, or website. Such names are variously called "nicknames" (e.g., in chatroom applications), "display names" (e.g., in Internet mail), or "petnames" (see [1]); for consistency, these are all called "nicknames" in this document.

Nicknames are commonly supported in technologies for textual chatrooms, e.g., Internet Relay Chat [RFC2811] and multi-party chat technologies based on the Extensible Messaging and Presence Protocol (XMPP) [RFC6120] [XEP-0045], the Message Session Relay Protocol (MSRP) [RFC4975] [I-D.ietf-simple-chat], and Centralized Conferencing (XCON) [RFC5239] [I-D.boulton-xcon-session-chat]. Recent chatroom technologies also allow internationalized nicknames because they support characters from outside the ASCII range [RFC20], typically by means of the Unicode character set [Unicode]. Although such nicknames tend to be used primarily for display purposes, they are sometimes used for programmatic purposes as well (e.g., kicking users or avoiding nickname conflicts).

A similar usage enables a person to set their own preferred display name or to set a preferred display name for another user (e.g., the "display-name" construct in the Internet message format [RFC5322] and [XEP-0172] in XMPP).

Memorable, human-friendly names are also used in contexts other than personal messaging, such as names for devices (e.g., in a network visualization application), websites (e.g., for bookmarks in a web browser), accounts (e.g., in a web interface for a list of payees in a bank account), people (e.g., in a contact list application), and the like.

The rules specified in this document can be applied in all of the foregoing contexts.

To increase the likelihood that memorable, human-friendly names will work in ways that make sense for typical users throughout the world, this document defines rules for preparing, enforcing, and comparing internationalized nicknames.

1.2. Terminology

Many important terms used in this document are defined in [RFC7564], [RFC6365], 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 RFC 2119 [RFC2119].

2. Nickname Profile

2.1. Rules

The following rules apply within the Nickname profile of the PRECIS FreeformClass.

1. Width Mapping Rule: There is no width-mapping rule (such a rule is not necessary because width mapping is performed as part of normalization using NFKC as specified below).
2. Additional Mapping Rule: The additional mapping rule consists of the following sub-rules.
 1. Any instances of non-ASCII space MUST be mapped to ASCII space (U+0020); a non-ASCII space is any Unicode code point having a general category of "Zs", naturally with the exception of U+0020.

2. Any instances of the ASCII space character at the beginning or end of a nickname MUST be removed (e.g., "stpeter " is mapped to "stpeter").
3. Interior sequences of more than one ASCII space character MUST be mapped to a single ASCII space character (e.g., "St Peter" is mapped to "St Peter").
3. Case Mapping Rule: Uppercase and titlecase characters MUST be mapped to their lowercase equivalents using Unicode Default Case Folding as defined in the Unicode Standard [Unicode] (at the time of this writing, the algorithm is specified in Chapter 3 of [Unicode7.0]). In applications that prohibit conflicting nicknames, this rule helps to reduce the possibility of confusion by ensuring that nicknames differing only by case (e.g., "stpeter" vs. "StPeter") would not be presented to a human user at the same time.
4. Normalization Rule: The string MUST be normalized using Unicode Normalization Form KC (NFKC). Because NFKC is more "aggressive" in finding matches than other normalization forms (in the terminology of Unicode, it performs both canonical and compatibility decomposition before recomposing code points), this rule helps to reduce the possibility of confusion by increasing the number of characters that would match (e.g., U+2163 ROMAN NUMERAL FOUR would match the combination of U+0049 LATIN CAPITAL LETTER I and U+0056 LATIN CAPITAL LETTER V).
5. Directionality Rule: There is no directionality rule. The "Bidi Rule" (defined in [RFC5893]) and similar rules are unnecessary and inapplicable to nicknames, because it is perfectly acceptable for a given nickname to be presented differently in different layout systems (e.g., a user interface that is configured to handle primarily a right-to-left script vs. an interface that is configured to handle primarily a left-to-right script), as long as the presentation is consistent in any given layout system.

2.2. Preparation

An entity that prepares a string for subsequent enforcement according to this profile MUST ensure that the string consists only of Unicode code points that conform to the "FreeformClass" base string class defined in [RFC7564]. In addition, the entity MUST ensure that the string is encoded as UTF-8 [RFC3629].

2.3. Enforcement

An entity that performs enforcement according to this profile MUST prepare a string as described in Section 2.2 section and MUST also apply the rules specified in Section 2.1. The rules MUST be applied in the order shown.

After all of the foregoing rules have been enforced, the entity MUST ensure that the nickname is not zero bytes in length (this is done after enforcing the rules to prevent applications from mistakenly omitting a nickname entirely, because when internationalized characters are accepted a non-empty sequence of characters can result in a zero-length nickname after canonicalization).

2.4. Comparison

An entity that performs comparison of two strings according to this profile MUST prepare each string and enforce the rules as specified in Section 2.2 and Section 2.3. The two strings are to be considered equivalent if they are an exact octet-for-octet match (sometimes called "bit-string identity").

3. Examples

The following examples illustrate a small number of nicknames that are consistent with the format defined above, along with the output string resulting from application of the PRECIS rules (note that the characters < and > are used to delineate the actual nickname and are not part of the nickname strings).

Table 1: A sample of legal nicknames

#	Nickname	Output for Comparison
1	<Foo>	<foo>
2	<foo>	<foo>
3	<Foo Bar>	<foo bar>
4	<foo bar>	<foo bar>
5	<Σ>	GREEK SMALL LETTER SIGMA (U+03C3)
6	<σ>	GREEK SMALL LETTER SIGMA (U+03C3)
7	<ς>	GREEK SMALL LETTER FINAL SIGMA (U+03C2)
8	<♚>	BLACK CHESS KING (U+265A)
9	<Richard Ⅳ>	<richard iv>

Regarding examples 5, 6, and 7: applying Unicode Default Case Folding to GREEK CAPITAL LETTER SIGMA (U+03A3) results in GREEK SMALL LETTER SIGMA (U+03C3), and doing so during comparison would result in matching the nicknames in examples 5 and 6; however, because the PRECIS mapping rules do not account for the special status of GREEK SMALL LETTER FINAL SIGMA (U+03C2), the nicknames in examples 5 and 7 or examples 6 and 7 would not be matched. Regarding example 8: symbol characters such as BLACK CHESS KING (U+265A) are allowed by the PRECIS FreeformClass and thus can be used in nicknames. Regarding example 9: applying Unicode Default Case Folding to ROMAN NUMERAL FOUR (U+2163) results in SMALL ROMAN NUMERAL FOUR (U+2173), and applying NFKC to SMALL ROMAN NUMERAL FOUR (U+2173) results in LATIN SMALL LETTER I (U+0069) LATIN SMALL LETTER V (U+0086).

4. Use in Application Protocols

This specification defines only the PRECIS-based rules for handling of nickname strings. It is the responsibility of an application protocol (e.g., MSRP, XCON, or XMPP) or application definition to specify the protocol slots in which nickname strings can appear, the entities that are expected to enforce the rules governing nickname strings, and when in protocol processing or interface handling the

rules need to be enforced. See Section 6 of [RFC7564] for guidelines about using PRECIS profiles in applications.

Above and beyond the PRECIS-based rules specified here, application protocols can also define application-specific rules governing nickname strings (rules regarding the minimum or maximum length of nicknames, further restrictions on allowable characters or character ranges, safeguards to mitigate the effects of visually similar characters, etc.).

Naturally, application protocols can also specify rules governing the actual use of nicknames in applications (reserved nicknames, authorization requirements for using nicknames, whether certain nicknames can be prohibited, handling of duplicates, the relationship between nicknames and underlying identifiers such as SIP URIs or Jabber IDs, etc.).

Entities that enforce the rules specified in this document are encouraged to be liberal in what they accept by following this procedure:

1. Where possible, map characters (e.g, through width mapping, additional mapping, case mapping, or normalization) and accept the mapped string.
2. If mapping is not possible (e.g., because a character is disallowed in the FreeformClass), reject the string.

5. IANA Considerations

The IANA shall add the following entry to the PRECIS Profiles Registry:

Name: Nickname.

Base Class: FreeformClass.

Applicability: Nicknames in messaging and text conferencing technologies; petnames for devices, accounts, and people; and other uses of nicknames or petnames.

Replaces: None.

Width Mapping Rule: None (handled via NFKC).

Additional Mapping Rule: Map non-ASCII space characters to ASCII space, strip leading and trailing space characters, map interior sequences of multiple space characters to a single ASCII space.

Case Mapping Rule: Map uppercase and titlecase characters to lowercase using Unicode Default Case Folding.

Normalization Rule: NFKC.

Directionality Rule: None.

Enforcement: To be specified by applications.

Specification: RFC XXXX. [Note to RFC Editor: please change "XXXX" to the RFC number issued for this specification.]

6. Security Considerations

6.1. Reuse of PRECIS

The security considerations described in [RFC7564] apply to the "FreeformClass" string class used in this document for nicknames.

6.2. Reuse of Unicode

The security considerations described in [UTS39] apply to the use of Unicode characters in nicknames.

6.3. Visually Similar Characters

[RFC7564] describes some of the security considerations related to visually similar characters, also called "confusable characters" or "confusables".

Although the mapping rules defined under Section 2 of this document are designed in part to reduce the possibility of confusion about nicknames, this document does not provide more detailed recommendations regarding the handling of visually similar characters, such as those provided in [UTS39].

7. References

7.1. Normative References

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

- [RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.
- [RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.
- [RFC7564] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 7564, May 2015.
- [UTS39] The Unicode Consortium, "Unicode Technical Standard #39: Unicode Security Mechanisms", November 2013, <<http://unicode.org/reports/tr39/>>.
- [Unicode7.0] The Unicode Consortium, "The Unicode Standard, Version 7.0.0", 2014, <<http://www.unicode.org/versions/Unicode7.0.0/>>.
- [Unicode] The Unicode Consortium, "The Unicode Standard", 2015-present, <<http://www.unicode.org/versions/latest/>>.

7.2. Informative References

- [I-D.boulton-xcon-session-chat] Barnes, M., Boulton, C., and S. Loreto, "Chatrooms within a Centralized Conferencing (XCON) System", draft-boulton-xcon-session-chat-08 (work in progress), July 2011.
- [I-D.ietf-simple-chat] Niemi, A., Garcia, M., and G. Sandbakken, "Multi-party Chat Using the Message Session Relay Protocol (MSRP)", draft-ietf-simple-chat-18 (work in progress), January 2013.
- [RFC20] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.
- [RFC2811] Kalt, C., "Internet Relay Chat: Channel Management", RFC 2811, April 2000.
- [RFC4975] Campbell, B., Mahy, R., and C. Jennings, "The Message Session Relay Protocol (MSRP)", RFC 4975, September 2007.

- [RFC5239] Barnes, M., Boulton, C., and O. Levin, "A Framework for Centralized Conferencing", RFC 5239, June 2008.
- [RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [XEP-0045] Saint-Andre, P., "Multi-User Chat", XSF XEP 0045, February 2012.
- [XEP-0172] Saint-Andre, P. and V. Mercier, "User Nickname", XSF XEP 0172, March 2012.

7.3. URIs

- [1] <http://www.skyhunter.com/marcs/petnames/IntroPetNames.html>

Appendix A. Acknowledgements

Thanks to Kim Alvefur, Mary Barnes, Ben Campbell, Dave Cridland, Miguel Garcia, Salvatore Loreto, Enrico Marocco, Matt Miller, and Yoshiro YONEYA for their reviews and comments.

Paul Kyzivat and Melinda Shore reviewed the document for the General Area Review Team and Operations Directorate, respectively.

During IESG review, Ben Campbell and Kathleen Moriarty provided comments that led to further improvements.

Thanks to Matt Miller as document shepherd, Pete Resnick and Andrew Sullivan as IANA designated experts, Marc Blanchet and Alexey Melnikov as working group chairs, and Barry Leiba as area director.

The author wishes to acknowledge Cisco Systems, Inc., for employing him during his work on earlier draft versions of this document.

Author's Address

Peter Saint-Andre
&yet

Email: peter@andyet.com
URI: <https://andyet.com/>

Precis
Internet-Draft
Obsoletes: 4013 (if approved)
Intended status: Standards Track
Expires: March 27, 2013

P. Saint-Andre
Cisco Systems, Inc.
A. Melnikov
Isode Ltd
September 23, 2012

Preparation and Comparison of Internationalized Strings Representing
Simple User Names and Passwords
draft-melnikov-precis-saslprepbis-04

Abstract

This document describes how to handle Unicode strings representing simple user names and passwords, primarily for purposes of comparison. This profile is intended to be used by Simple Authentication and Security Layer (SASL) mechanisms (such as PLAIN and SCRAM-SHA-1), as well as other protocols that exchange simple user names or passwords. This document obsoletes RFC 4013.

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 March 27, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Overview	3
1.2. Terminology	3
2. Simple User Names	4
2.1. Definition	4
2.2. Preparation	4
3. Passwords	5
3.1. Definition	5
3.2. Preparation	5
4. Migration	6
4.1. User Names	6
4.2. Passwords	7
5. Security Considerations	8
5.1. Password/Passphrase Strength	8
5.2. Reuse of PRECIS	8
5.3. Reuse of Unicode	9
6. IANA Considerations	9
6.1. Use of NameClass	9
6.2. Use of FreeClass	9
7. Open Issues	9
8. References	10
8.1. Normative References	10
8.2. Informative References	10
Appendix A. Differences from RFC 4013	11
Appendix B. Acknowledgements	11
Authors' Addresses	12

1. Introduction

1.1. Overview

User names and passwords are used pervasively in authentication and authorization on the Internet. To increase the likelihood that the input and comparison of user names and passwords will work in ways that make sense for typical users throughout the world, this document defines rules for preparing and comparing internationalized strings that represent simple user names and passwords.

The algorithms defined in this document assume that all strings are comprised of characters from the Unicode character set [UNICODE].

The algorithms are designed for use in Simple Authentication and Security Layer (SASL) [RFC4422] mechanisms, such as PLAIN [RFC4616] and SCRAM-SHA-1 [RFC5802]. However, they might be applicable wherever simple user names or passwords are used. This profile is not intended for use in preparing strings that are not simple user names (e.g., email addresses, DNS domain names, LDAP distinguished names), nor in cases where identifiers or secrets are not strings (e.g., keys or certificates) or require different handling (e.g., case folding).

This document builds upon the PRECIS framework defined in [FRAMEWORK], which differs fundamentally from the stringprep technology [RFC3454] used in SASLprep [RFC4013]. The primary difference is that stringprep profiles allowed all characters except those which were explicitly disallowed, whereas PRECIS profiles disallow all characters except those which are explicitly allowed (this "inclusion model" was originally used for internationalized domain names in [RFC5891]; see [RFC5894] for further discussion). It is important to keep this distinction in mind when comparing the technology defined in this document to SASLprep [RFC4013].

This document obsoletes RFC 4013.

1.2. Terminology

Many important terms used in this document are defined in [FRAMEWORK], [RFC4422], [RFC5890], [RFC6365], and [UNICODE]. The term "non-ASCII space" refers to any Unicode code point with a general category of "Zs", with the exception of U+0020 (here called "ASCII space").

As used here, the term "password" is not literally limited to a word; i.e., a password could be a passphrase consisting of more than one word, perhaps separated by spaces or other such characters.

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].

2. Simple User Names

2.1. Definition

Some SASL mechanisms (e.g., CRAM-MD5, DIGEST-MD5, and SCRAM) specify that the authentication identity used in the context of such mechanisms is a "simple user name" (see Section 2 of [RFC4422] as well as [RFC4013]). However, the exact form of a simple user name in any particular mechanism or deployment thereof is a local matter, and a simple user name does not necessarily map to an application identifier such as the localpart of an email address.

For purposes of preparation and comparison of authentication identities, this document specifies that a simple user name is a string of Unicode code points [UNICODE], encoded using UTF-8 [RFC3629], and structured as an ordered sequence of "simpleparts" (where the complete simple user name can consist of a single simplepart or a space-separated sequence of simpleparts).

Therefore the syntax for a simple user name is defined as follows using the Augmented Backus-Naur Form (ABNF) as specified in [RFC5234].

```
simpleusername = simplepart [1*(1*SP simplepart)]
simplepart      = 1*(namepoint)
                ;
                ; a "namepoint" is a UTF-8 encoded
                ; Unicode code point that conforms to
                ; the "NameClass" string class defined
                ; in draft-ietf-precis-framework
                ;
```

Note well that all code points and blocks not explicitly allowed in the PRECIS NameClass are disallowed; this includes private use characters, surrogate code points, and the other code points and blocks defined as "Prohibited Output" in Section 2.3 of RFC 4013.

2.2. Preparation

A simple user name MUST NOT be zero bytes in length. This rule is to be enforced after any normalization and mapping of code points.

Each simplepart of a simple user name MUST conform to the definition of the PRECIS NameClass provided in [FRAMEWORK], where the normalization, casemapping, and directionality rules are as described below.

1. Unicode Normalization Form C (NFC) MUST be applied to all characters.
2. Uppercase and titlecase characters MUST be mapped to their lowercase equivalents.
3. Additional mappings MAY be applied, such as those defined in [I-D.yoneya-precis-mappings].

With regard to directionality, the "Bidi Rule" provided in [RFC5893] applies.

3. Passwords

3.1. Definition

For purposes of preparation and comparison of passwords, this document specifies that a password is a string of Unicode code points [UNICODE], encoded using UTF-8 [RFC3629], and conformant to the PRECIS FreeClass.

Therefore the syntax for a password is defined as follows using the Augmented Backus-Naur Form (ABNF) as specified in [RFC5234].

```
password      = 1*(freepoint)
                ;
                ; a "freepoint" is a UTF-8 encoded
                ; Unicode code point that conforms to
                ; the "FreeClass" string class defined
                ; in draft-ietf-precis-framework
                ;
```

Note well that all code points and blocks not explicitly allowed in the PRECIS FreeClass are disallowed; this includes private use characters, surrogate code points, and the other code points and blocks defined as "Prohibited Output" in Section 2.3 of RFC 4013.

3.2. Preparation

A password MUST NOT be zero bytes in length. This rule is to be enforced after any normalization and mapping of code points.

A password MUST be treated as follows, where the operations specified MUST be completed in the order shown:

1. Apply Unicode Normalization Form C (NFC) to all characters.
2. Map any instances of non-ASCII space to ASCII space (U+0020).
3. Ensure that the resulting string conforms to the definition of the PRECIS FreeClass.

With regard to directionality, the "Bidi Rule" (defined in [RFC5893]) and similar rules are unnecessary and inapplicable to passwords, since they can reduce the range of characters that are allowed in a string and therefore reduce the amount of entropy that is possible in a password. Furthermore, such rules are intended to minimize the possibility that the same string will be displayed differently on a system set for right-to-left display and a system set for left-to-right display; however, passwords are typically not displayed at all and are rarely meant to be interoperable across different systems in the way that non-secret strings like domain names and user names are.

4. Migration

The rules defined in this specification differ slightly from those defined by the SASLprep specification [RFC4013]. The following sections describe these differences, along with their implications for migration, in more detail.

4.1. User Names

Deployments that currently use SASLprep for handling user names might need to scrub existing data when migrating to use of the rules defined in this specification. In particular:

- o SASLprep specified the use of Unicode Normalization Form KC (NFKC), whereas this usage of the PRECIS NameClass employs Unicode Normalization Form C (NFC). In practice this change is unlikely to cause significant problems, because NFKC provides methods for mapping Unicode code points with compatibility equivalents to those equivalents, whereas the PRECIS NameClass entirely disallows Unicode code points with compatibility equivalents (i.e., during comparison NFKC is more "aggressive" about finding matches than is NFC). A few examples might suffice to indicate the nature of the problem: (1) U+017F LATIN SMALL LETTER LONG S is compatibility equivalent to U+0073 LATIN SMALL LETTER S (2) U+2163 ROMAN NUMERAL FOUR is compatibility equivalent to U+0049 LATIN CAPITAL LETTER I and U+0056 LATIN CAPITAL LETTER V (3) U+FB01 LATIN SMALL LIGATURE

FI is compatibility equivalent to U+0066 LATIN SMALL LETTER F and U+0069 LATIN SMALL LETTER I. Under SASLprep, the use of NFKC also handled the mapping of fullwidth and halfwidth code points to their decomposition equivalents (see [I-D.yoneya-precis-mappings]). Although it is expected that code points with compatibility equivalents are rare in existing user names, for migration purposes deployments might want to search their database of user names for Unicode code points with compatibility equivalents and map those code points to their compatibility equivalents.

- o SASLprep mapped non-ASCII spaces to ASCII space (U+0020), whereas the PRECIS NameClass entirely disallows non-ASCII spaces. The non-ASCII space characters are U+00A0 NO-BREAK SPACE, U+1680 OGHAM SPACE MARK, U+180E MONGOLIAN VOWEL SEPARATOR, U+2000 EN QUAD through U+200A HAIR SPACE, U+202F NARROW NO-BREAK SPACE, U+205F MEDIUM MATHEMATICAL SPACE, and U+3000 IDEOGRAPHIC SPACE. For migration purposes, deployments might want to convert non-ASCII space characters to ASCII space in simple user names.
- o SASLprep mapped the "characters commonly mapped to nothing" from Appendix B.1 of [RFC3454]) to nothing, whereas the PRECIS NameClass entirely disallows most of these characters, which correspond to the code points from the "M" category defined under Section 6.13 of [FRAMEWORK] (with the exception of U+1806 MONGOLIAN TODO SOFT HYPHEN, which was "commonly mapped to nothing" in Unicode 3.2 but at the time of this writing does not have a derived property of `Default_Ignorable_Code_Point` in Unicode 6.1). For migration purposes, deployments might want to remove code points contained in the PRECIS "M" category from simple user names.
- o SASLprep allowed uppercase and titlecase characters, whereas this usage of the PRECIS NameClass maps uppercase and titlecase characters to their lowercase equivalents. For migration purposes, deployments can either convert uppercase and titlecase characters to their lowercase equivalents in simple user names (thus losing the case information) or preserve uppercase and titlecase characters and ignore the case difference when comparing simple user names.

4.2. Passwords

Depending on local service policy, migration from RFC 4013 to this specification might not involve any scrubbing of data (since passwords might not be stored in the clear anyway); however, service providers need to be aware of possible issues that might arise during migration. In particular:

- o SASLprep specified the use of Unicode Normalization Form KC (NFKC), whereas this usage of the PRECIS FreeClass employs Unicode Normalization Form C (NFC). Because NFKC is more aggressive about finding matches than NFC, in practice this change is unlikely to cause significant problems and indeed has the security benefit of probably resulting in fewer false positives when comparing passwords. A few examples might suffice to indicate the nature of the problem: (1) U+017F LATIN SMALL LETTER LONG S is compatibility equivalent to U+0073 LATIN SMALL LETTER S (2) U+2163 ROMAN NUMERAL FOUR is compatibility equivalent to U+0049 LATIN CAPITAL LETTER I and U+0056 LATIN CAPITAL LETTER V (3) U+FB01 LATIN SMALL LIGATURE FI is compatibility equivalent to U+0066 LATIN SMALL LETTER F and U+0069 LATIN SMALL LETTER I. Under SASLprep, the use of NFKC also handled the mapping of fullwidth and halfwidth code points to their decomposition equivalents (see [I-D.yoneya-precis-mappings]). Although it is expected that code points with compatibility equivalents are rare in existing passwords, some passwords that matched when SASLprep was used might no longer work when the rules in this specification are applied.
- o SASLprep mapped the "characters commonly mapped to nothing" from Appendix B.1 of [RFC3454]) to nothing, whereas the PRECIS FreeClass entirely disallows such characters, which correspond to the code points from the "M" category defined under Section 6.13 of [FRAMEWORK] (with the exception of U+1806 MONGOLIAN TODO SOFT HYPHEN, which was commonly mapped to nothing in Unicode 3.2 but at the time of this writing is allowed by Unicode 6.1). In practice, this change will probably have no effect on comparison, but user-oriented software might reject such code points instead of ignoring them during password preparation.

5. Security Considerations

5.1. Password/Passphrase Strength

The ability to include a wide range of characters in passwords and passphrases can increase the potential for creating a strong password with high entropy. However, in practice, the ability to include such characters ought to be weighed against the possible need to reproduce them on various devices using various input methods.

5.2. Reuse of PRECIS

The security considerations described in [FRAMEWORK] apply to the "NameClass" and "FreeClass" base string classes used in this document for simple user names and passwords, respectively.

5.3. Reuse of Unicode

The security considerations described in [UTR39] apply to the use of Unicode characters in user names and passwords.

6. IANA Considerations

6.1. Use of NameClass

The IANA shall add an entry to the PRECIS Usage Registry for reuse of the PRECIS NameClass in SASL, as follows:

Applicability: Usernames in SASL and Kerberos.
Base Class: NameClass.
Subclass: No.
Replaces: The SASLprep profile of Stringprep.
Normalization: NFC.
Casemapping: Map uppercase and titlecase characters to lowercase.
Additional Mappings: None.
Directionality: The "Bidi Rule" defined in RFC 5893 applies.
Specification: RFC XXXX. [Note to RFC Editor: please change XXXX to the number issued for this specification.]

6.2. Use of FreeClass

The IANA shall add an entry to the PRECIS Usage Registry for reuse of the PRECIS FreeClass in SASL, as follows:

Applicability: Passwords in SASL and Kerberos.
Base Class: FreeClass
Subclass: No.
Replaces: The SASLprep profile of Stringprep.
Normalization: NFC.
Casemapping: None.
Additional Mappings: Map non-ASCII space characters to ASCII space.
Directionality: None.
Specification: RFC XXXX. [Note to RFC Editor: please change XXXX to the number issued for this specification.]

7. Open Issues

We need to compare the output obtained when applying the new rules with Unicode 3.2 and Unicode 6.1 data to the output obtained when applying the SASLprep rules with Unicode 3.2 data, then make sure that the PRECIS Working Group and KITTEN Working Group are comfortable with any changes to the Unicode characters that are

allowed and disallowed. (See also the migration issues described under Section 4.)

8. References

8.1. Normative References

[FRAMEWORK]

Saint-Andre, P. and M. Blanchet, "Precis Framework: Handling Internationalized Strings in Protocols", draft-ietf-precis-framework-05 (work in progress), August 2012.

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 6.1", 2012,
<<http://www.unicode.org/versions/Unicode6.1.0/>>.

8.2. Informative References

[I-D.yoneya-precis-mappings]

YONEYA, Y. and T. NEMOTO, "Mapping characters for PRECIS classes", draft-yoneya-precis-mappings-02 (work in progress), July 2012.

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

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

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

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

- [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.
- [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.
- [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.
- [RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.
- [UTR39] The Unicode Consortium, "Unicode Technical Report #39: Unicode Security Mechanisms", August 2010, <<http://unicode.org/reports/tr39/>>.

Appendix A. Differences from RFC 4013

The following substantive modifications were made from RFC 4013.

- o A single SASLprep algorithm was replaced by two separate algorithms: one for simple user names and another for passwords.
- o The new preparation algorithms use PRECIS instead of a stringprep profile. The new algorithms work independently of Unicode versions.
- o As recommended in the PRECIS framework, changed the Unicode normalization form from NFKC to NFC.
- o Some Unicode code points that were mapped to nothing in RFC 4013 are simply disallowed by PRECIS.

Appendix B. Acknowledgements

Thanks to Yoshiro YONEYA and Takahiro NEMOTO for implementation feedback. Thanks also to Marc Blanchet, Joe Hildebrand, Alan DeKok, Simon Josefsson, Jonathan Lennox, Matt Miller, Pete Resnick, and

Andrew Sullivan for their input regarding the text.

This document borrows some text from RFC 4013 and RFC 6120.

Authors' Addresses

Peter Saint-Andre
Cisco Systems, Inc.
1899 Wynkoop Street, Suite 600
Denver, CO 80202
USA

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

Alexey Melnikov
Isode Ltd
5 Castle Business Village
36 Station Road
Hampton, Middlesex TW12 2BX
UK

Email: Alexey.Melnikov@isode.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 6, 2013

Y. YONEYA
JPRS
T. NEMOTO
Keio University
October 3, 2012

Mapping characters for precis classes
draft-yoneya-precis-mappings-03

Abstract

Preparation and comparison of internationalized strings ("precis") framework [I-D.ietf-precis-framework] is defining several classes of strings for preparation and comparison. In the document, case mapping is defined because many of protocols handle case sensitive or case insensitive string comparison and therefore preparation of string is mandatory. As described in IDNA mapping [RFC5895] and precis problem statement [I-D.ietf-precis-problem-statement], mappings in internationalized strings are not limited to case, but also width, delimiters and/or other specials are taken into consideration. This document is a guideline for authors of protocol profiles of precis framework and describes the mappings that must be considered between receiving user input and passing permitted code points to internationalized protocols.

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 April 6, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

In many cases, user input of internationalized strings is generated by input method editor ("IME") or copy-and-paste from free text. Usually users do not care case and/or width of input characters because they are identical for users' eyes. Further, users rarely switch IME state to input special characters such as protocol elements. For Internationalized Domain Names ("IDNs"), IDNA Mapping [RFC5895] describes methods to treat these issues. For precis strings, case mapping is defined as a process in precis framework [I-D.ietf-precis-framework], but width mapping, delimiter mapping and/or special mapping are not defined. Handling of mappings other than case is also important to increase chance of strings match as users expect. This document is a guideline for authors of protocol profiles of precis framework and describes the mappings that must be considered between receiving user input and passing permitted code points to internationalized protocols.

2. Types of mapping

This document defines two types of mapping. One is protocol independent mapping that doesn't depend on protocol rules and the other is protocol dependent mapping that depend on protocol rules. This document defines some mappings in these mapping types. Authors of protocol profiles of precis framework should need to give careful consideration to choice of mappings.

Each mapping type is described in following sections.

3. Protocol independent mapping

Protocol independent mapping is a mapping that doesn't depend on protocol rules.

3.1. Width mapping

Fullwidth and halfwidth characters (those defined with Decomposition Types <wide> and <narrow>) are mapped to their decomposition mappings as shown in the Unicode character database [Unicode].

Width mapping will increase backward compatibility with Stringprep [RFC3454] and precis framework [I-D.ietf-precis-framework]. Because in a Stringprep profile which specifies Unicode normalization form KC (NFKC) for normalization method, fullwidth/halfwidth characters are mapped into its compatible form. If a precis framework profile specified NFKC (which is not recommended), width mapping might not be useful.

4. Protocol dependent mapping

Protocol dependent mapping is a mapping that depend on protocol rules.

4.1. Delimiter mapping

Definitions of delimiters in certain protocols are differ from each other. Therefore, delimiter mapping table should be based on well defined mapping table for each protocol.

One of the most useful case of delimiter mapping is when FULL STOP character (U+002E) is a delimiter as well as domain name. Some of IME generates FULL STOP compatible characters such as IDEOGRAPHIC FULL STOP (U+3002) when users type FULL STOP on the keyboard.

4.2. Special mapping

Certain protocols have characters which need to map different character from precis framework defined mapping rule other than delimiter characters. In this document, these mappings are named special mapping. They are differ from each protocol. Therefore, special mapping table should be based on well defined mapping table for each protocol. Examples of special mapping are following;

- o White spaces are mapped to SPACE (U+0020)
- o Some characters such as control characters are mapped to nothing (Deletion)

LDAPprep[RFC4518] defines the rule that some codepoints(Appendix B.4) are mapped to SPACE (U+0020).

4.3. Local case mapping

Local case mapping is case folding that depend on language context. For example, given there is upper case I in a user ID strings, you should care what's language context that this user ID depend on when this character is mapped into lower case character. And if this depends on Turkish, the character should be mapped into LATIN SMALL LETTER DOTLESS I (U+0131) as this character's lower case.

This document defines characters that need local case mapping based on the Specialcasing.txt [Specialcasing] in section 3.13 of The Unicode Standard [Unicode] to solve such a problem. Local case mapping targets only characters that get two different results to perform just casefolding that is defined in the Casefolding.txt [Casefolding] and perform special casefolding that is defined in the

Specialcasing.txt then casefolding, because precis framework have casefolding.

There are two types casefoldings defined as Unconditional Mappings and Conditional Mappings in the Specialcasing.txt. Conditional mappings have Language-Insensitive Mappings that targets characters whose full case mappings do not depend on language, but do depend on context and Language-Sensitive Mappings that these are characters whose full case mappings depend on language and perhaps also context.

Of these mappings, characters that Unconditional Mappings and Language-Insensitive Mappings in Conditional Mappings target are mapped into same codepoint(s) with just casefolding and special casefolding then casefolding. But characters that Language-Sensitive Mappings in Conditional Mappings targets are mapped into different codepoint with them. Therefore this document defined characters that are a part of characters of Lithuanian(lt), Turkish(tr) and Azerbaijanian(az) that Language-Sensitive Mappings targets as targets for local case mapping.

A list of characters that need Local case mapping are as follows.

Format:

<Language>; <Codepoint>; <Lowercase>; <Comments>

```
lt; 0049; 0069 0307; LATIN CAPITAL LETTER I
lt; 004A; 006A 0307; LATIN CAPITAL LETTER J
lt; 012E; 012F 0307; LATIN CAPITAL LETTER I WITH OGONEK
lt; 00CC; 0069 0307 0300; LATIN CAPITAL LETTER I WITH GRAVE
lt; 00CD; 0069 0307 0301; LATIN CAPITAL LETTER I WITH ACUTE
lt; 0128; 0069 0307 0303; LATIN CAPITAL LETTER I WITH TILDE
tr; 0130; 0069; LATIN CAPITAL LETTER I WITH DOT ABOVE
tr; 0049; 0131; LATIN CAPITAL LETTER I
az; 0130; 0069; LATIN CAPITAL LETTER I WITH DOT ABOVE
az; 0049; 0131; LATIN CAPITAL LETTER I
```

Section 6 "IANA Considerations" contains a template to registry these characters to IANA as precis local case mapping registry.

5. Applying order of mapping

Basically, applying order of mapping that this document describes aren't sensitive. This section defines applying order of mapping to minimize effect of codepoint change by mappings. This mapping order is very general and was designed to be acceptable to the widest user community.

1. width mapping
2. delimiter mapping
3. special mapping
4. local case mapping
5. precis framework

Mappings that this document describes should be performed before precis framework.

6. IANA Considerations

6.1. precis local case mapping registry

IANA is requested to create a registry of precis local case mapping. In accordance with [RFC5226], the registration policy is "RFC Required".

6.2. Template for precis local case mapping registry

The following information is to be given when a new precis local case mapping rule is created. The registration template is as follows:

Language: language name

Codepoint: Local case mapping that can be applied when this code point exists in the strings

Local lowercase: The lowercase codepoint after performing local case mapping

Comment: Character name of the code point

Appendix C contains further discussion and a table from which that registry can be initialized.

7. Security Considerations

TBD.

8. Acknowledgment

Martin Duerst suggested a need for the case folding about the mapping(map final sigma to sigma, German sz to ss,.).

Pete Resnick et al. gave important suggestion for this document during at WG meeting.

9. References

- [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.
- [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.
- [RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.
- [RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, December 2005.
- [RFC4518] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Internationalized String Preparation", RFC 4518, June 2006.
- [RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.
- [RFC6122] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Address Format", RFC 6122, March 2011.
- [I-D.ietf-precis-framework]
Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation and Comparison of Internationalized Strings in Application Protocols", draft-ietf-precis-framework-03 (work in progress), May 2012.
- [I-D.ietf-precis-problem-statement]
Blanchet, M. and A. Sullivan, "Stringprep Revision and PRECIS Problem Statement", draft-ietf-precis-problem-statement-06 (work in progress),

July 2012.

[Unicode] The Unicode Consortium, "The Unicode Standard, Version 6.1.0", <<http://www.unicode.org/versions/Unicode6.1.0/>>, 2012.

[Casefolding] "CaseFolding-6.1.0.txt", Unicode Character Database, July 2011, <<http://www.unicode.org/Public/6.1.0/ucd/CaseFolding.txt>>.

[Specialcasing] "SpecialCasing-6.1.0.txt", Unicode Character Database, July 2011, <<http://www.unicode.org/Public/6.1.0/ucd/SpecialCasing.txt>>.

Appendix A. Mapping type list each protocol

A.1. Mapping type list for each protocol

This table is the mapping type list for each protocol. Values marked "o" indicate that the protocol use the type of mapping. Values marked "-" indicate that the protocol doesn't use the type of mapping.

\ Type of mapping RFC \	Width (NFKC)	Delimiter	Case	Special
3490	-	o	-	-
3491	o	-	o	-
3722	o	-	o	-
3748	o	-	-	o
4013	o	-	-	o
4314	o	-	-	o
4518	o	-	o	o
6120	-	-	o	-

Appendix B. Codepoints which need special mapping

B.1. RFC3748

Non-ASCII space characters [StringPrep, C.1.2] that can be mapped to SPACE (U+0020).

B.2. RFC4013

Non-ASCII space characters [StringPrep, C.1.2] that can be mapped to SPACE (U+0020).

B.3. RFC4314

Non-ASCII space characters [StringPrep, C.1.2] that can be mapped to SPACE (U+0020).

B.4. RFC4518

Codepoints mapped to SPACE (U+0020) are following;

U+0009 (CHARACTER TABULATION)
U+000A (LINE FEED (LF))
U+000B (LINE TABULATION)
U+000C (FORM FEED (FF))
U+000D (CARRIAGE RETURN (CR))
U+0085 (NEXT LINE (NEL))
U+0020 (SPACE)
U+00A0 (NO-BREAK SPACE)
U+1680 (OGHAM SPACE MARK)
U+2000 (EN QUAD)
U+2001 (EM QUAD)
U+2002 (EN SPACE)
U+2003 (EM SPACE)
U+2004 (THREE-PER-EM SPACE)
U+2005 (FOUR-PER-EM SPACE)
U+2006 (SIX-PER-EM SPACE)
U+2007 (FIGURE SPACE)
U+2008 (PUNCTUATION SPACE)
U+2009 (THIN SPACE)
U+200A (HAIR SPACE)
U+2028 (Line Separator)
U+2029 (Paragraph Separator)
U+202F (NARROW NO-BREAK SPACE)
U+205F (MEDIUM MATHEMATICAL SPACE)
U+3000 (IDEOGRAPHIC SPACE)

All other control code (e.g., Cc) points or code points with a

control function (e.g., Cf) are mapped to nothing. Codepoints mapped to nothing that aren't specified by Stringprep are following;

U+0000-0008
U+000E-001F
U+007F-0084
U+0086-009F
U+06DD
U+070F
U+180E
U+200E-200F
U+202A-202E
U+2061-2063
U+206A-206F
U+FFF9-FFFB
U+1D173-1D17A
U+E0001
U+E0020-E007F

Appendix C. The initial precis local case mapping registrations

C.1. Lithuanian

language: Lithuanian

Codepoint: U+0049
Local lowercase: U+0069 U+0307
Comment: LATIN CAPITAL LETTER I

Codepoint: U+004A
Local lowercase: U+006A U+0307
Comment: LATIN CAPITAL LETTER J

Codepoint: U+012E
Local lowercase: U+012F U+0307
Comment: LATIN CAPITAL LETTER I WITH OGONEK

Codepoint: U+00CC
Local lowercase: U+0069 U+0307 U+0300
Comment: LATIN CAPITAL LETTER I WITH GRAVE

Codepoint: U+00CD
Local lowercase: U+0069 U+0307 U+0301
Comment: LATIN CAPITAL LETTER I WITH ACUTE

Codepoint: U+0128
Local lowercase: U+0069 U+0307 U+0303
Comment: LATIN CAPITAL LETTER I WITH TILDE

C.2. Turkish

language: Turkish

Codepoint: U+0130
Local lowercase: U+0069
Comment: LATIN CAPITAL LETTER I WITH DOT ABOVE

Codepoint: U+0049
Local lowercase: U+0131
Comment: LATIN CAPITAL LETTER I

C.3. Azerbaijani

language: Azerbaijani

Codepoint: U+0130
Local lowercase: U+0069

Comment: LATIN CAPITAL LETTER I WITH DOT ABOVE

Codepoint: U+0049

Local lowercase: U+0131

Comment: LATIN CAPITAL LETTER I

Appendix D. Change Log

D.1. Changes since -00

- o Add the Section 2.3 "Special mapping" in Section 2 Type of mappings.
- o Add the topic about the special mapping and additional case mapping in Section 3 "Discussion".
- o Add Appendices;
Appendix A "Mapping type list each protocols"
Appendix B "Code point list is need special mapping"
Appendix D "Change Log"
- o Add the Section 8 "Acknowledgment".

D.2. Changes since -01

- o Modify document structure as a guideline for authors of protocol profiles of precis framework.
- o Group mappings that this document defines into two types.
- o Add the Section 5 "Applying order of mapping".
- o Delete the section 3 "Discussion".

D.3. Changes since -02

- o Modify the Section 4.3 "Local case mapping" for defining characters that local case mapping targets.
- o Request creating registry of precis local case mapping to IANA and define a template for registry of precis local case mapping in the Section 6 "IANA Considerations".
- o Add the Appendix C "The initial precis local case mapping registrations".

Authors' Addresses

Yoshiro YONEYA
JPRS
Chiyoda First Bldg. East 13F
3-8-1 Nishi-Kanda
Chiyoda-ku, Tokyo 101-0065
Japan

Phone: +81 3 5215 8451
Email: yoshiro.yoneya@jprs.co.jp

Takahiro NEMOTO
Keio University
Graduate School of Media Design
4-1-1 Hiyoshi, Kohoku-ku
Yokohama, Kanagawa 223-8526
Japan

Phone: +81 45 564 2517
Email: t.nemo10@kmd.keio.ac.jp

