Abstract

Application protocols using Unicode characters in protocol strings need to properly prepare such strings in order 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 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 March 6, 2015.

Copyright Notice

Copyright (c) 2014 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
2.  Terminology ......................................... 5
3.  String Classes ....................................... 6
   3.1.  Overview ...................................... 6
   3.2.  IdentifierClass ................................. 7
       3.2.1.  Valid ................................... 7
       3.2.2.  Contextual Rule Required ................. 8
       3.2.3.  Disallowed ............................... 8
       3.2.4.  Unassigned ............................... 9
       3.2.5.  Examples ................................ 9
   3.3.  FreeformClass ................................... 9
       3.3.1.  Valid ................................... 9
       3.3.2.  Contextual Rule Required ................. 10
       3.3.3.  Disallowed ............................... 10
       3.3.4.  Unassigned ............................... 10
       3.3.5.  Examples ................................ 10
4.  Profiles ............................................... 11
4.1.  Principles ......................................... 11
   4.1.1.  Width Mapping ................................ 11
   4.1.2.  Additional Mappings ........................ 12
   4.1.3.  Case Mapping ................................ 12
   4.1.4.  Normalization ............................... 12
   4.1.5.  Directionality ............................... 12
   4.1.6.  Exclusions ................................ 13
4.2.  Building Application-Layer Constructs .................. 13
4.3.  A Note about Spaces ................................ 14
5.  Order of Operations .................................. 14
6.  Code Point Properties ................................ 15
7.  Category Definitions Used to Calculate Derived Property ...
   7.1.  LetterDigits (A) ................................ 18
   7.2.  Unstable (B) .................................. 18
   7.3.  IgnorableProperties (C) ....................... 18
   7.4.  IgnorableBlocks (D) ........................... 19
   7.5.  LDH (E) ....................................... 19
   7.6.  Exceptions (F) ................................. 19
   7.7.  BackwardCompatible (G) ....................... 19
1. Introduction

As described in the problem statement for the preparation and comparison of internationalized strings ("PRECIS") [RFC6885], many IETF protocols have used the Stringprep framework [RFC3454] as the basis for preparing and comparing protocol strings that contain Unicode characters [Unicode7.0] 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 for the preparation and comparison of strings and identifiers. 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 has prompted other "customers" of Stringprep to consider new approaches to the preparation and comparison of internationalized strings, as described in [RFC6885].

This document defines a framework for a post-Stringprep approach to the preparation 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, addressing matters such as width mapping, case folding and other forms of character mapping, Unicode normalization, directionality, and further excluded code points or character categories.

Whereas the string classes define the "baseline" code points for a range of applications, profiling enables application protocols to further restrict the allowable code points beyond those specified for the relevant string class (e.g., characters with special or reserved meaning, such as "@" and "/" when used as separators within identifiers) and to apply the string classes in ways that are appropriate for constructs such as usernames and passwords [I-D.ietf-precis-saslprepbis], nicknames [I-D.ietf-precis-nickname], the localparts of instant messaging addresses [I-D.ietf-xmpp-6122bis], and free-form strings [I-D.ietf-xmpp-6122bis]. 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 can achieve the following objectives:

a. Determine if a given string conforms to the profile (e.g. to determine if it is allowed for use in the relevant "slot" specified by an application protocol).

b. Determine if any two given strings are equivalent (e.g., to make an access decision for purposes of authentication or authorization as further described in [RFC6943]).

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

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

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

The character categories and calculation rules defined under Section 7 and Section 6 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 are provided 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 6.3 [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. String Classes

3.1. 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 in chatrooms; 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 (e.g., 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 9.3 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 IdentiferClass 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 and character categories are treated as valid input to the string.

Contextual Rule Required: Defines which code points and character categories 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 and character categories 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 4, profiles of these string classes are responsible for defining the width mapping, additional mappings, case mapping, normalization, directionality, and exclusion rules.

3.2. 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.

3.2.1. Valid

- 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 7.1.

- Code points in the range U+0021 through U+007E, i.e., the (printable) ASCII7 ("K") rule defined under Section 7.11. 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.

3.2.2. Contextual Rule Required

- A number of characters from the Exceptions ("F") category defined under Section 7.6 (see Section 7.6 for a full list).
- Joining characters, i.e., the JoinControl ("H") category defined under Section 7.8.

3.2.3. Disallowed

- Old Hangul Jamo characters, i.e., the OldHangulJamo ("I") category defined under Section 7.9.
- Control characters, i.e., the Controls ("L") category defined under Section 7.12.
- Ignorable characters, i.e., the PrecisIgnorableProperties ("M") category defined under Section 7.13.
- Space characters, i.e., the Spaces ("N") category defined under Section 7.14.
- Symbol characters, i.e., the Symbols ("O") category defined under Section 7.15.
- Punctuation characters, i.e., the Punctuation ("P") category defined under Section 7.16.
- Any character that has a compatibility equivalent, i.e., the HasCompat ("Q") category defined under Section 7.17. These code points are disallowed even if they would otherwise be valid according to the property-based rules specified in the previous section.
- Letters and digits other than the "traditional" letters and digits allowed in IDNs, i.e., the OtherLetterDigits ("R") category defined under Section 7.18.
3.2.4. 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.

3.2.5. 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 JIDlocalIdentifierClass profile).

3.3. 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 9.3 describes some of the security hazards involved with using or profiling the FreeformClass.

Security Warning: Consult Section 9.6 for relevant security considerations when strings conforming to the FreeformClass, or a profile thereof, are used as passwords.

3.3.1. Valid

- Traditional letters and numbers, i.e., the LetterDigits ("A") category first defined in [RFC5892] and listed here under Section 7.1.

- Letters and digits other than the "traditional" letters and digits allowed in IDNs, i.e., the OtherLetterDigits ("R") category defined under Section 7.18.

- Code points in the range U+0021 through U+007E, i.e., the (printable) ASCII7 ("K") rule defined under Section 7.11.

- Any character that has a compatibility equivalent, i.e., the HasCompat ("Q") category defined under Section 7.17.
3.3.2. Contextual Rule Required

- A number of characters from the Exceptions ("F") category defined under Section 7.6 (see Section 7.6 for a full list).
- Joining characters, i.e., the JoinControl ("H") category defined under Section 7.8.

3.3.3. Disallowed

- Old Hangul Jamo characters, i.e., the OldHangulJamo ("I") category defined under Section 7.9.
- Control characters, i.e., the Controls ("L") category defined under Section 7.12.
- Ignorable characters, i.e., the PrecisIgnorableProperties ("M") category defined under Section 7.13.

3.3.4. 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.

3.3.5. 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 JIDresourceIdentifierClass profile).
4. Profiles

4.1. Principles

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, directionality, and exclusion 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 or character categories 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 8.3. Profile names use the following convention: they are of the form "ProfilenameBaseClass", 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 IdentifierClass used for localparts of Jabber Identifiers (JIDs) in the Extensible Messaging and Presence Protocol (XMPP) is named "JIDlocalIdentifierClass" [I-D.ietf-xmpp-6122bis].

4.1.1. Width Mapping

The width mapping rule of a profile specifies whether width mapping is performed on fullwidth and halfwidth characters, 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 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 user surprise. For more information about the concept of width in East Asian scripts within Unicode, see Unicode Standard Annex #11 [UAX11].
4.1.2. Additional Mappings

The additional mappings rule of a profile specifies whether additional mappings are to be applied, such as mapping of delimiter characters and mapping of special characters (e.g., non-ASCII space characters to ASCII space or certain characters to nothing).

4.1.3. Case Mapping

The case mapping rule of a profile specifies whether case mapping is performed (instead of case preservation) on uppercase and titlecase characters, and how the mapping is done (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-saslprepbis].

4.1.4. Normalization

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.

4.1.5. Directionality

The directionality rule of a profile specifies how to treat strings containing left-to-right (LTR) and right-to-left (RTL) characters (see Unicode Standard Annex #9 [UAX9]). A profile usually specifies
a directionality rule that restricts strings to be entirely LTR strings or entirely RTL strings and defines the allowable sequences of characters in LTR and RTL strings. Possible rules include, but are not limited to, (a) considering any string that contains a right-to-left code point to be a right-to-left string, or (b) applying the "Bidi Rule" from [RFC5893].

Mixed-direction strings are not directly supported by the PRECIS framework itself, since there is currently no widely accepted and implemented solution for the safe display of mixed-direction strings. An application protocol that uses the PRECIS framework (or an extension to the framework) could define better ways to present mixed-direction strings; however, that work is outside the scope of this framework and would likely require a great deal of careful research into the problems of displaying bidirectional text.

4.1.6. Exclusions

The exclusions rule of a profile specifies whether the profile excludes additional code points or character categories above and beyond those excluded by the string class being profiled. That is, a profile 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 being profiled will be defined as disallowed for the profile.

4.2. 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 pseudo-ABNF [RFC5234]:

```plaintext
```

Saint-Andre & Blanchet Expires March 6, 2015 [Page 13]
fullname = namepart *(1*SP namepart)
namepart = 1*idpoint

; an "idpoint" is a UTF-8 encoded Unicode code point
; 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".

4.3. 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:

- Many Unicode characters are confusable with ASCII space.
- 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.
- 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 hit 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 in properly handling space characters (especially non-ASCII space characters) in identifiers and other protocol strings, the 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.

5. Order of Operations

To ensure proper comparison, the following order of operations is REQUIRED:

1. Width mapping
2. Optionally, additional mappings such as mapping of delimiters (e.g., characters such as '@', ':', '/', '+', and '-') and special handling of certain characters or classes of characters (e.g., mapping of non-ASCII spaces to ASCII space or mapping of control characters to nothing); the PRECIS mappings document [I-D.ietf-precis-mappings] describes such mappings in more detail.

3. Case mapping as described under Section 4.1.3 of this document

4. Normalization

5. 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 mappings, case mapping, and normalization operations 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 4.1.1 (see also the PRECIS mappings document [I-D.ietf-precis-mappings]).

6. 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). Code points with this property value are permitted for general use in any string class. The abbreviated term "PVALID" is used to refer to this value in the remainder of this document.

**SPECIFIC CLASS PROTOCOL VALID** Those code points that are allowed to be used in specific string classes. Code points with this property value are permitted for use in specific string classes.
In the remainder of this document, the abbreviated term *_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 "CONTEXTO"). A character with the derived property value CONTEXTJ or CONTEXTO 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 NONJOINER, 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 a specific string class. Code points with this property value are not permitted in one of the string classes but might be permitted in others. In the remainder of this document, the abbreviated term *_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.

To summarize, the assigned values of the derived property are:

- PVALID
- FREE_PVAL
- CONTEXTJ
- CONTEXTO
- DISALLOWED
- UNASSIGNED
The algorithm to calculate the value of the derived property is as follows:

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 in the code point tables, 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 in this document are reused from 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 7.7) can be adjusted to ensure the stability of the results.
7. 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 6.

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

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

The first ten categories (A–J) shown below were previously defined
for IDNA2008 and are copied directly from [RFC5892] 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.

7.1. LetterDigits (A)

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

7.2. Unstable (B)

This category is defined in Secton 2.2 of [RFC5892] but not used in
PRECIS.
7.3. IgnorableProperties (C)

This category is defined in Section 2.3 of [RFC5892] but not used in PRECIS.

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

7.4. IgnorableBlocks (D)

This category is defined in Section 2.4 of [RFC5892] but not used in PRECIS.

7.5. LDH (E)

This category is defined in Section 2.5 of [RFC5892] but not used in PRECIS.

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

7.6. Exceptions (F)

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

7.7. BackwardCompatible (G)

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

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

7.8. JoinControl (H)

This category is defined in Section 2.8 of [RFC5892] and is included by reference for use in PRECIS.
7.9. OldHangulJamo (I)

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

7.10. Unassigned (J)

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

7.11. ASCII? (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 6 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}

7.12. Controls (L)

L: Control(cp) = True

7.13. 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, and at the time of Unicode 7.0 is as follows:

Other_Default_Ignorable_Code_Point
+ Cf (Format characters)
+ Variation_Selector
- White_Space
- FFF9..FFFF (Annotation Characters)
- 0600..0604, 06DD, 070F, 110BD (exceptional Cf characters that should be visible)
7.14. Spaces (N)

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

N: General_Category(cp) is in {Zs}

7.15. 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}

7.16. 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}

7.17. 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].

7.18. 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 7.1).

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

8. IANA Considerations

8.1. 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 7 and Section 6.

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 7 and Section 6 require IETF Review.

8.2. 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 3.3 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 3.2 of this document.
[Note to RFC Editor: please change "this document" to the RFC number issued for this specification.]

8.3. 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]

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

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

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

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

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

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

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

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

Exclusions:  [a brief description of the specific code points or characters categories are excluded, e.g., "Eight legacy characters in the ASCII range" or "Any character that has a compatibility equivalent, i.e., the HasCompat category"]

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 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 Would an existing PRECIS string class or profile solve the problem?

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, exclusion, 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 user surprise")?

o Does the profile introduce any new security concerns such as those described under Section 9 of this document (e.g., false positives for authentication or authorization)?

9. Security Considerations

9.1. 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, each with its own conventions regarding allowable characters (see for example [I-D.saintandre-username-interop] with regard to various username constructs). Despite these difficulties, profiles should not be multiplied beyond necessity. In particular, application protocol designers should think long and hard before

Saint-Andre & Blanchet Expires March 6, 2015 [Page 24]
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 and comparison of internationalized
strings. For example, such strings can be used to make
authentication and authorization decisions, and the security of an
application could be compromised if an entity providing a given
string is connected to the wrong account or online resource based on
different interpretations of the string.

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

9.2. 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 9.5), protocol designers and implementers are encouraged to
pay close attention to the security considerations described
elsewhere in this document.

9.3. 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.)
9.4. 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.

9.5. 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
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.


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].

10. Interoperability Considerations

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.

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.

Three Unicode 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, and that other such differences are possible between the version of Unicode current at the time of this writing (7.0) and future versions.

11. References

11.1. Normative References


11.2. Informative References


Internet-Draft              PRECIS Framework              September 2014


Saint-Andre & Blanchet    Expires March 6, 2015    [Page 31]


11.3. URIs


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.
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-saslprepbis], 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
P.O. Box 787
Parker, CO  80134
USA

Email: peter@andyet.net

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

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