Internet Engineering Task Force (IETF)
Request for Comments: 8264
Obsoletes: 7564
Category: Standards Track
ISSN: 2070-1721

P. Saint-Andre
Jabber.org
M. Blanchet
Viagenie

October 2017

PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols

Abstract

Application protocols using Unicode code points 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 code points and thus is more 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 7564.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8264.

Table of Contents

1. Introduction

Application protocols using Unicode code points [Unicode] 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 code points and thus is more agile with respect to versions of Unicode. (Note: PRECIS is restricted to Unicode and does not support any other coded character set [RFC6365].)

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 code points, especially code points 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 code points appropriate for common application-protocol constructs (where possible, maintaining compatibility with IDNA2008 to help ensure a more consistent user experience).

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 Astonishment".

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

- Application protocols will be more agile with regard to Unicode versions (recognizing that complete agility cannot be realized in practice).

- 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 code points that are acceptable in various contexts. Given this more uniform set of string classes, it is also expected that copy/paste operations between software implementing different application protocols will be more predictable and coherent.

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 [RFC8265], opaque strings such as passwords [RFC8265], and nicknames [RFC8266]. Profiles are responsible for defining the handling of right-to-left code points 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 code points to other code points or to nothing, and mapping of fullwidth and halfwidth code points.

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 rules of a profile for a single output string to check whether the output string conforms to the rules of the profile and thus 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 an output string of "stpeter", thus leading to a loss of information about the capitalization of the first and third characters. 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 Sections 8 and 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 (see Section 11).

2. Terminology

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

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Preparation, Enforcement, and Comparison

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

- Enforcement entails applying all of the rules specified for a particular string class, or profile thereof, to a single input string, for the purpose of checking whether the string conforms to all of the rules and thus determining if the string can be used in a given protocol slot.

- Comparison entails applying all of the rules specified for a particular string class, or profile thereof, to two separate input strings, for the purpose of determining if the two strings are equivalent.

- Preparation primarily entails ensuring that the code points in a single input string are allowed by the underlying PRECIS string class, and sometimes also entails applying one or more of the rules specified for a particular string class or profile thereof. Preparation can be appropriate for constrained devices that can to some extent restrict the code points in a string to a limited repertoire of characters but that do not have the processing power or onboard memory to perform operations such as Unicode normalization. However, preparation does not ensure that an input string conforms to all of the rules for a string class or profile thereof.

  Note: The term "preparation" as used in this specification and related documents has a much more limited scope than it did in Stringprep; it essentially refers to a kind of preprocessing of an input string, not the actual operations that apply internationalization rules to produce an output string (here termed "enforcement") or to compare two output strings (here termed "comparison").

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 a server 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 code points that would minimize the potential for user confusion caused by visually similar code points (and thus be relatively "safe") vs. defining a much larger set of Unicode code points 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 chat room), 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 code points 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 chat room; the intent is that this class will allow nearly any Unicode code point, 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 code points 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 5 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 as originally defined in the IDNA2008 specifications).

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 mapping, 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, filenames, data feed identifiers, and chat room names. We group such strings into a class called "IdentifierClass" having the following features.

4.2.1. 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 9.1.

- Code points in the range U+0021 through U+007E, i.e., the (printable) ASCII7 ("K") category 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 reuses the LetterDigits category from IDNA2008, the range of code points allowed in the IdentifierClass is wider than the range of code points allowed in IDNA2008. The main reason is that IDNA2008 applies the Unstable ("B") category (Section 9.2) before the LetterDigits

category, thus disallowing uppercase code points, whereas the IdentifierClass does not apply the Unstable category.

4.2.2. Contextual Rule Required

- A number of code points from the Exceptions ("F") category defined under Section 9.6.
- Joining code points, i.e., the JoinControl ("H") category defined under Section 9.8.

4.2.3. Disallowed

- Old Hangul Jamo code points, i.e., the OldHangulJamo ("I") category defined under Section 9.9.
- Control code points, i.e., the Controls ("L") category defined under Section 9.12.
- Ignorable code points, i.e., the PrecisIgnorableProperties ("M") category defined under Section 9.13.
- Space code points, i.e., the Spaces ("N") category defined under Section 9.14.
- Symbol code points, i.e., the Symbols ("O") category defined under Section 9.15.
- Punctuation code points, i.e., the Punctuation ("P") category defined under Section 9.16.
- Any code point that is decomposed and recomposed into something other than itself under Unicode Normalization Form KC, 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.
- 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 coded 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 profiles of the IdentifierClass can be found in [RFC8265] (the UsernameCaseMapped and UsernameCasePreserved profiles).

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 [RFC8265]) or a nickname in a chat room (see [RFC8266]). 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

- Traditional letters and numbers, i.e., the LetterDigits ("A") category first defined in [RFC5892] and listed here under Section 9.1.
- Code points in the range U+0021 through U+007E, i.e., the (printable) ASCII7 ("K") category defined under Section 9.11.
- Space code points, i.e., the Spaces ("N") category defined under Section 9.14.
- Symbol code points, i.e., the Symbols ("O") category defined under Section 9.15.
- Punctuation code points, i.e., the Punctuation ("P") category defined under Section 9.16.
- Any code point that is decomposed and recomposed into something other than itself under Unicode Normalization Form KC, i.e., the HasCompat ("Q") category defined under Section 9.17.

- Letters and digits other than the "traditional" letters and digits allowed in IDNs, i.e., the OtherLetterDigits ("R") category defined under Section 9.18.

4.3.2. Contextual Rule Required

- A number of code points from the Exceptions ("F") category defined under Section 9.6.

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

4.3.3. Disallowed

- Old Hangul Jamo code points, i.e., the OldHangulJamo ("I") category defined under Section 9.9.

- Control code points, i.e., the Controls ("L") category defined under Section 9.12.

- Ignorable code points, 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 coded 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 profiles of the FreeformClass can be found in [RFC8265] (the OpaqueString profile) and [RFC8266] (the Nickname profile).

4.4. Summary

The following table summarizes the differences between the IdentifierClass and the FreeformClass (i.e., the disposition of a code point as valid, contextual rule required, disallowed, or unassigned), depending on its PRECIS category.

CATEGORY	IDENTIFIERCLASS	FREEFORMCLASS
(A) LetterDigits	Valid	Valid
(B) Unstable	[N/A (unused)]	
(C) IgnorableProperties	[N/A (unused)]	
(D) IgnorableBlocks	[N/A (unused)]	
(E) LDH	[N/A (unused)]	
(F) Exceptions	Contextual Rule Required	Contextual Rule Required
(G) BackwardCompatible	[Handled by IDNA Rules]	
(H) JoinControl	Contextual Rule Required	Contextual Rule Required
(I) OldHangulJamo	Disallowed	Disallowed
(J) Unassigned	Unassigned	Unassigned
(K) ASCII7	Valid	Valid
(L) Controls	Disallowed	Disallowed
(M) PrecisIgnorableProperties	Disallowed	Disallowed
(N) Spaces	Disallowed	Valid
(O) Symbols	Disallowed	Valid
(P) Punctuation	Disallowed	Valid
(Q) HasCompat	Disallowed	Valid
(R) OtherLetterDigits	Disallowed	Valid

Table 1: Comparative Disposition of Code Points

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 mapping (if any), case mapping, normalization, and directionality rules. A profile MAY also restrict the allowable code points 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 used for opaque strings such as passwords is the OpaqueString profile of the FreeformClass [RFC8265].

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, chat room names, user nicknames or display names, filenames, authentication identifiers, passwords, and other strings already exist in the wild and need to be supported in existing application protocols such as DNS, SMTP, the Extensible Messaging and Presence Protocol (XMPP), Internet Relay Chat (IRC), NFS, the Internet Small Computer System Interface (iSCSI), the Extensible Authentication Protocol (EAP), and the Simple Authentication and Security Layer (SASL) [RFC4422], 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 a string and how the mapping is done. Typically, such mapping consists of mapping fullwidth and halfwidth code points, i.e., code points with a Decomposition Type of Wide or Narrow, to their decomposition mappings; as an example, "ï¼" (FULLWIDTH DIGIT ZERO, U+FF10) would be mapped to "0" (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 code points 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].

> Note: Because the East Asian width property is not guaranteed to be stable by the Unicode Standard (see <http://unicode.org/policies/stability_policy.html> for details), the results of applying a given width mapping rule might not be consistent across different versions of Unicode.

5.2.2. Additional Mapping Rule

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

- Mapping of delimiter code points (such as '@', ':', '/', '+', and '-').
- Mapping of special code points (e.g., non-ASCII space code points to SPACE (U+0020) or control code points to nothing).

The PRECIS mappings document [RFC7790] 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 a string and how the mapping is applied (e.g., mapping uppercase and titlecase code points to their lowercase equivalents).

If case mapping is desired (instead of case preservation), it is RECOMMENDED to use the Unicode toLowerCase() operation defined in the Unicode Standard [Unicode]. In contrast to the Unicode toCaseFold() operation, the toLowerCase() operation is less likely to violate the "Principle of Least Astonishment", especially when an application merely wishes to convert uppercase and titlecase code points to their lowercase equivalents while preserving lowercase code points. Although the toCaseFold() operation can be appropriate when an application needs to compare two strings (such as in search operations), in general few application developers and even fewer users understand its implications, so toLowerCase() is almost always the safer choice.

> Note: Neither toLowerCase() nor toCaseFold() is designed to handle various language-specific issues, such as the character "Ä±" (LATIN SMALL LETTER DOTLESS I, U+0131) in several Turkic languages. The reader is referred to the PRECIS mappings document [RFC7790], which describes these issues in greater detail.

In order to maximize entropy and minimize the potential for false accepts, 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 Section 12.6 of this document and in [RFC8265].

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.

Protocol designers and application developers need to understand that certain Unicode normalization forms, especially NFKC and NFKD, can result in significant loss of information in various circumstances and that these circumstances can depend on the language and script of

the strings to which the normalization forms are applied. Extreme care should be taken when specifying the use of these normalization forms.

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) code points (see Unicode Standard Annex #9 [UAX9]). RTL code points 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 code points also contain "left-to-right" (LTR) code points, such as ASCII numerals, as well as code points 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 counterintuitive 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 nor does it define any new directionality rules, because 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 should 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 the following:

- Many Unicode code points are confusable with SPACE (U+0020).
- Even if non-ASCII space code points are mapped to SPACE (U+0020), space code points 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 code point other than SPACE (U+0020), such as ZERO WIDTH JOINER (U+200D), when a user performs an action like pressing the space bar on a keyboard.

One consequence of disallowing space code points 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 code points (especially non-ASCII space code points) 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; this in turn enables application protocols to define profiles of the FreeformClass that are more flexible than any profiles of the IdentifierClass. In addition, as explained in Section 6.3, 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 through the use of PRECIS. Indeed, because it is extremely difficult for protocol

designers and application developers to do the right thing for all users when supporting internationalized strings, often the safest option is to support only the ASCII range [RFC20] in various protocol slots. This state of affairs is unfortunate but is the direct result of the complexities involved with human languages (e.g., the vast number of code points, scripts, user communities, and rules with their inevitable exceptions), which kinds of strings application developers and their users wish to support, the wide range of devices that users employ to access services enabled by various Internet protocols, and so on.

Despite these significant challenges, application and protocol developers sometimes persevere in attempting to support internationalized strings in their systems. These developers need to think carefully about 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).

- Don’t define your own profile unless absolutely necessary (see Section 5.1). Existing profiles have been designed for wide reuse. It is highly likely that an existing profile will meet your needs, especially given the ability to specify further excluded code points (Section 6.2) and to build application-layer constructs (see Section 6.3).

- 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 [RFC8265] of the IdentifierClass or checking the password of a user against the OpaqueString profile [RFC8265] of the FreeformClass).

  See [RFC8265] and [RFC7622] 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 code points 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 PRECIS string classes or a profile thereof. Consider, for example, the "simple username" construct in SASL [RFC4422]. Depending on the deployment, a simple username 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 username cannot be defined as an instance of the IdentifierClass or a profile thereof, because space code points 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 [RFC8265]:

```
username   = userpart *(1*SP userpart)
userpart   = 1*(idpoint)
             ;
             ; an "idpoint" is a Unicode code point that
             ; can be contained in a string conforming 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 [RFC7790]). In addition, this order is consistent with IDNA2008, and with both IDNA2003 and Stringprep before then, for the purpose of enabling code reuse and of ensuring as much continuity as possible with the Stringprep profiles that are obsoleted by several PRECIS profiles.

Because of the order of operations specified here, applying the rules for any given PRECIS profile is not necessarily an idempotent procedure (e.g., under certain circumstances, such as when Unicode Normalization Form KC is used, performing Unicode normalization after case mapping can still yield uppercase characters for certain code points). Therefore, an implementation SHOULD apply the rules repeatedly until the output string is stable; if the output string does not stabilize after reapplying the rules three (3) additional times after the first application, the implementation SHOULD terminate application of the rules and reject the input string as invalid.

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 coded character set by examining various code point properties.

2. Defines an algorithm for determining a derived property value, which can depend 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" for the FreeformClass or "ID_PVAL" for the IdentifierClass. In practice, the derived property ID_PVAL is not used in this specification, because every ID_PVAL code point is PVALID.

CONTEXTUAL RULE REQUIRED Some characteristics of the code point, such as its being invisible in certain contexts or problematic in others, require that it not be used in a string unless specific other code points or properties are present in the string. As in IDNA2008, there are two subdivisions of CONTEXTUAL RULE REQUIRED: the first for Join_controls (called "CONTEXTJ") and the second for other code points (called "CONTEXTO"). A string MUST NOT contain any characters whose validity is context-dependent, unless the validity is positively confirmed by a contextual rule. To check this, each code point identified as CONTEXTJ or CONTEXTO in the "PRECIS Derived Property Value" registry (Section 11.1) MUST have a non-null rule. If such a code point is missing a rule, the string is invalid. If the rule exists but the result of applying the rule is negative or inconclusive, the proposed string is invalid. The most notable of the CONTEXTUAL RULE REQUIRED code points are the Join Control code points ZERO WIDTH JOINER (U+200D) and ZERO WIDTH NON-JOINER (U+200C), 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" for the FreeformClass or "ID_DIS" for the IdentifierClass. In practice, the derived property FREE_DIS is not used in this specification, because 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, because 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 SPACE (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, because 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 code points added after Unicode 5.2 or 7.0, depending on the category, because 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 code point can have its Unicode General_Category value 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. Code points 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 code point property for a code point is designated by using the formal name of that property (from the Unicode PropertyAliases.txt file [PropertyAliases] 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 code points. 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.

Note: In particular, the code points ZERO WIDTH JOINER (U+200D) and ZERO WIDTH NON-JOINER (U+200C) are necessary to produce certain combinations of characters in certain scripts (e.g., Arabic, Persian, and Indic scripts), but if used in other contexts, they can have consequences that violate the "Principle of Least Astonishment". Therefore, these code points are allowed only in contexts where they are appropriate, specifically where the relevant rule (CONTEXTJ or CONTEXTO) has been defined. See [RFC5892] and [RFC5894] for further discussion.

9.9. OldHangulJamo (I)

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

Note: Exclusion of these code points results in disallowing certain archaic Korean syllables and in restricting supported Korean syllables to preformed, modern Hangul characters.

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 code points from the 7-bit ASCII range. By applying this category, the algorithm specified under Section 8 exempts these code points 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 counterproductive.

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

9.12. Controls (L)

This PRECIS-specific category consists of all control code points, such as LINE FEED (U+000A).

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 file [DerivedCoreProperties].

Note: In general, these code points are constructs such as so-called "soft hyphens", certain joining code points, various specialized code points for use within Unicode itself (e.g., language tags and variation selectors), and so on. Disallowing these code points in PRECIS reduces the potential for unexpected results in the use of internationalized strings.

9.14. Spaces (N)

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

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.

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 any code point that is decomposed and recomposed into something other than itself under Unicode Normalization Form KC.

Q: toNFKC(cp) != cp

Typically, this category is true of code points that are "compatibility decomposable characters" as defined in the Unicode Standard.

The toNFKC() operation returns the code point in Normalization Form KC. For more information, see 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") category (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 trade-offs involved with internationalization and that they need considerable guidance in making reasonable decisions with regard to the options before them.

Therefore:

- Protocol designers are strongly encouraged to question the assumption that they need to define new profiles, because existing profiles are designed for wide reuse (see Section 5 for further discussion).

- 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.

- The designated experts for profile registration requests ought to seek answers to all of the questions provided under Section 11.3 and ought 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).

- 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.

- 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 has created and now maintains the "PRECIS Derived Property Value" registry (<https://www.iana.org/assignments/precis-tables/>), which records the derived properties for each version of Unicode released starting from version 6.3. The derived property value is to be calculated in cooperation with a designated expert [RFC8126] according to the rules specified under Sections 8 and 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 Sections 8 and 9 require IETF Review.

Note: IANA is requested to not make further updates to this registry until it receives notice from the IESG that the issues described in [IAB-Statement] and Section 13.5 of this document have been settled.

11.2. PRECIS Base Classes Registry

IANA has created the "PRECIS Base Classes" registry (<https://www.iana.org/assignments/precis-parameters/>). In accordance with [RFC8126], 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."]

Reference: [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 RFC 8264

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 RFC 8264

11.3. PRECIS Profiles Registry

IANA has created the "PRECIS Profiles" registry (<https://www.iana.org/assignments/precis-parameters/>) to identify profiles that use the PRECIS string classes. In accordance with [RFC8126], 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., "Usernames in security and application protocols."]

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 code points to their compatibility variants."]

Additional Mapping Rule: [any additional mappings that are required or recommended, e.g., "Map non-ASCII space code points to SPACE (U+0020)."]

Case Mapping Rule: [the behavioral rule for handling of case, e.g., "Apply the Unicode toLowerCase() operation."]

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:

- Would an existing PRECIS string class or profile solve the problem? If not, why not? (See Section 5.1 for related considerations.)
- Is the problem being addressed by this profile well defined?
- Does the specification define what kinds of applications are involved and the protocol elements to which this profile applies?
- Is the profile clearly defined?
- 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?
- Are the width mapping, case mapping, additional mapping, normalization, and directionality rules appropriate for the intended use?
- Does the profile explain which entities enforce the rules and when such enforcement occurs during protocol operations?

- Does the profile reduce the degree to which human users could be surprised or confused by application behavior (the "Principle of Least Astonishment")?

- Does the profile introduce any new security concerns such as those described under Section 12 of this document (e.g., false accepts 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 accepts and false rejects discussed in [RFC6943] (the terms "false positives" and "false negatives" are used in that document). 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, designers of application protocols 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 accepts and false rejects 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 code points from the ASCII range; thus, they exclude spaces, code points with compatibility equivalents, and almost all symbols and punctuation marks. However, because such strings can still include so-called "confusable code points" (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 code point. 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 code points they might not understand, and they 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 SASL [RFC4422], that do not take local character sets into account.

12.5. Visually Similar Characters

Some code points 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 "ju1iet" (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. A well-known example is confusion between "Ð°" CYRILLIC SMALL LETTER A (U+0430) and "a" LATIN SMALL LETTER A (U+0061). As another example, the characters "áá¢áµáá¢áá" (U+13DA U+13A2 U+13B5 U+13AC U+13A2 U+13AC U+13D2) from the Cherokee block look similar to the ASCII code points representing "STPETER" as they might appear when presented using a "creative" font family. Confusion among such characters is perhaps not unexpected, given that the alphabetic writing systems involved all bear a family resemblance or historical lineage. Perhaps more surprising is confusion among characters from disparate writing systems, such as "O" (LATIN CAPITAL LETTER O, U+004F), "0" (DIGIT ZERO, U+0030), "à»" (LAO DIGIT ZERO, U+0ED0), "á" (ETHIOPIC SYLLABLE PHARYNGEAL A, U+12D0), and other graphemes that have the appearance of open circles. And the reader needs to be aware that the foregoing represent merely a small sample of characters that are confusable in Unicode.

In some instances 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 code points might be the real string and the string formed of ASCII code points 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 code points that the service will allow to be registered (e.g., in an account name) or stored (e.g., in a filename). 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 code points from more than one script and SHOULD restrict registrations to code points 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.

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 accepts. 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 trade-offs 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 SASL [RFC4422]). Moreover, passwords are often carried by a sequence of protocols with backend authentication systems or data storage systems such as RADIUS [RFC2865] and the Lightweight Directory Access Protocol (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, because the password is not available to the server in plaintext form.

Further discussion of password handling can be found in [RFC8265].

13. Interoperability Considerations

13.1. Coded Character Sets

It is known that some existing applications and systems do not support the full Unicode coded character set, or even any characters outside the ASCII repertoire [RFC20]. If two (or more) applications or systems need to interoperate when exchanging data (e.g., for the purpose of authenticating the combination of a username and password), naturally they will need to have in common at least one coded character set and the repertoire of characters being exchanged (see [RFC6365] for definitions of these terms). Establishing such a baseline is a matter for the application or system that uses PRECIS, not for the PRECIS framework.

13.2. Dependency on Unicode

The only coded character set supported by PRECIS is Unicode. If an application or system does not support Unicode or uses a different coded character set [RFC6365], then the PRECIS rules cannot be applied to that application or system.

13.3. 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 or for specifications that define PRECIS string classes or profiles thereof.

13.4. Unicode Versions

It is extremely important for protocol designers and application developers to understand that various changes can occur across versions of the Unicode Standard, and such changes can result in instability of PRECIS categories. The following are merely a few examples:

- As described in [RFC6452], between Unicode 5.2 (current at the time IDNA2008 was originally published) and Unicode 6.0, three code points underwent changes in their GeneralCategory, resulting in modified handling, depending on which version of Unicode is available on the underlying system.

- The HasCompat() categorization of a given input string could change if, for example, the string includes a precomposed character that was added in a recent version of Unicode.

- The East Asian width property, which is used in many PRECIS width mapping rules, is not guaranteed to be stable across Unicode versions.

13.5. 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 [IDNA-Unicode].

At the time of this 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 but might not be limited to the following:

- The range of code points in the LetterDigits category (Sections 4.2.1 and 9.1) might be narrowed.
- Some code points with special properties that are now allowed might be excluded.
- More additional mapping rules (Section 5.2.2) might be defined.
- Alternative normalization methods might be added.

As described in Section 11.1, until these issues are settled, it is reasonable for the IANA to apply the same precautionary principle described in [IAB-Statement] to the "PRECIS Derived Property Value" registry as is applied to the "IDNA Parameters" registry <https://www.iana.org/assignments/idna-tables/>: that is, to not make further updates to the registry.

Nevertheless, implementations and deployments are unlikely to encounter significant problems as a consequence of these issues or potential changes if they follow the advice given in this specification to use the more restrictive IdentifierClass whenever possible or, if using the FreeformClass, to allow only a restricted set of code points, particularly avoiding code points whose implications they do not understand.

14. References

14.1. Normative References

[RFC20] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/info/rfc5198>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <https://www.rfc-editor.org/info/rfc6365>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[Unicode] The Unicode Consortium, "The Unicode Standard", <http://www.unicode.org/versions/latest/>.

14.2. Informative References

[DerivedCoreProperties]
The Unicode Consortium, "DerivedCoreProperties-10.0.0.txt", Unicode Character Database, March 2017, <http://www.unicode.org/Public/UCD/latest/ucd/DerivedCoreProperties.txt>.

[Err4568] RFC Errata, Erratum ID 4568, RFC 7564, <https://www.rfc-editor.org/errata/eid4568>.

[IAB-Statement]
Internet Architecture Board, "IAB Statement on Identifiers and Unicode 7.0.0", February 2015, <https://www.iab.org/documents/correspondence-reports-documents/2015-2/iab-statement-on-identifiers-and-unicode-7-0-0/>.

[IDNA-Unicode]
Klensin, J. and P. Faltstrom, "IDNA Update for Unicode 7.0.0", Work in Progress, draft-klensin-idna-5892upd-unicode70-04, March 2015.

[PropertyAliases]
The Unicode Consortium, "PropertyAliases-10.0.0.txt", Unicode Character Database, February 2017, <http://www.unicode.org/Public/UCD/latest/ucd/PropertyAliases.txt>.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <https://www.rfc-editor.org/info/rfc2865>.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, DOI 10.17487/RFC3454, December 2002, <https://www.rfc-editor.org/info/rfc3454>.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, DOI 10.17487/RFC3490, March 2003, <https://www.rfc-editor.org/info/rfc3490>.

[RFC3491] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, DOI 10.17487/RFC3491, March 2003, <https://www.rfc-editor.org/info/rfc3491>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <https://www.rfc-editor.org/info/rfc4422>.

[RFC4510] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, DOI 10.17487/RFC4510, June 2006, <https://www.rfc-editor.org/info/rfc4510>.

[RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and Recommendations for Internationalized Domain Names (IDNs)", RFC 4690, DOI 10.17487/RFC4690, September 2006, <https://www.rfc-editor.org/info/rfc4690>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://www.rfc-editor.org/info/rfc5891>.

[RFC5892] Faltstrom, P., Ed., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, DOI 10.17487/RFC5892, August 2010, <https://www.rfc-editor.org/info/rfc5892>.

[RFC5893] Alvestrand, H., Ed. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, DOI 10.17487/RFC5893, August 2010, <https://www.rfc-editor.org/info/rfc5893>.

[RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, DOI 10.17487/RFC5894, August 2010, <https://www.rfc-editor.org/info/rfc5894>.

[RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, DOI 10.17487/RFC5895, September 2010, <https://www.rfc-editor.org/info/rfc5895>.

[RFC6452] Faltstrom, P., Ed. and P. Hoffman, Ed., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA) - Unicode 6.0", RFC 6452, DOI 10.17487/RFC6452, November 2011, <https://www.rfc-editor.org/info/rfc6452>.

[RFC6885] Blanchet, M. and A. Sullivan, "Stringprep Revision and Problem Statement for the Preparation and Comparison of Internationalized Strings (PRECIS)", RFC 6885, DOI 10.17487/RFC6885, March 2013, <https://www.rfc-editor.org/info/rfc6885>.

[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <https://www.rfc-editor.org/info/rfc6943>.

[RFC7564] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 7564, DOI 10.17487/RFC7564, May 2015, <https://www.rfc-editor.org/info/rfc7564>.

[RFC7622] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Address Format", RFC 7622, DOI 10.17487/RFC7622, September 2015, <https://www.rfc-editor.org/info/rfc7622>.

[RFC7790] Yoneya, Y. and T. Nemoto, "Mapping Characters for Classes of the Preparation, Enforcement, and Comparison of Internationalized Strings (PRECIS)", RFC 7790, DOI 10.17487/RFC7790, February 2016, <https://www.rfc-editor.org/info/rfc7790>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8265] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 8265, DOI 10.17487/RFC8265, October 2017, <https://www.rfc-editor.org/info/rfc8265>.

[RFC8266] Saint-Andre, P., "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Nicknames", RFC 8266, DOI 10.17487/RFC8266, October 2017, <https://www.rfc-editor.org/info/rfc8266>.

[UAX11] Unicode Standard Annex #11, "East Asian Width", edited by Ken Lunde. An integral part of The Unicode Standard, <http://unicode.org/reports/tr11/>.

[UAX15] Unicode Standard Annex #15, "Unicode Normalization Forms", edited by Mark Davis and Ken Whistler. An integral part of The Unicode Standard, <http://unicode.org/reports/tr15/>.

[UAX9] Unicode Standard Annex #9, "Unicode Bidirectional Algorithm", edited by Mark Davis, Aharon Lanin, and Andrew Glass. An integral part of The Unicode Standard, <http://unicode.org/reports/tr9/>.

[UTR36] Unicode Technical Report #36, "Unicode Security Considerations", edited by Mark Davis and Michel Suignard, <http://unicode.org/reports/tr36/>.

[UTS39] Unicode Technical Standard #39, "Unicode Security Mechanisms", edited by Mark Davis and Michel Suignard, <http://unicode.org/reports/tr39/>.

Appendix A. Changes from RFC 7564

The following changes were made from [RFC7564].

- Recommended the Unicode toLowerCase() operation over the Unicode toCaseFold() operation in most PRECIS applications.
- Clarified the meaning of "preparation", and described the motivation for including it in PRECIS.
- Updated references.

See [RFC7564] for a description of the differences from [RFC3454].

Acknowledgements

Thanks to Martin Duerst, William Fisher, John Klensin, Christian Schudt, and Sam Whited for their feedback. Thanks to Sam Whited also for submitting [Err4568].

See [RFC7564] for acknowledgements related to the specification that this document supersedes.

Some algorithms and textual descriptions have been borrowed from [RFC5892]. Some text regarding security has been borrowed from [RFC5890], [RFC8265], and [RFC7622].

Authors' Addresses

Peter Saint-Andre
Jabber.org
P.O. Box 787
Parker, CO 80134
United States of America

Phone: +1 720 256 6756
Email: stpeter@jabber.org
URI: https://www.jabber.org/

Marc Blanchet
Viagenie
246 Aberdeen
QuÃ©bec, QC G1R 2E1
Canada

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