

Internet Engineering Task Force (IETF)
Request for Comments: 7564
Obsoletes: [3454](#)
Category: Standards Track
ISSN: 2070-1721

P. Saint-Andre
&yet
M. Blanchet
Viagenie
May 2015

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

Abstract

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

Status of This Memo

This 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 5741](#).

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc7564>.

Copyright Notice

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

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

Table of Contents

1.	Introduction	4
2.	Terminology	7
3.	Preparation, Enforcement, and Comparison	7
4.	String Classes	8
4.1.	Overview	8
4.2.	IdentifierClass	9
4.2.1.	Valid	9
4.2.2.	Contextual Rule Required	10
4.2.3.	Disallowed	10
4.2.4.	Unassigned	11
4.2.5.	Examples	11
4.3.	FreeformClass	11
4.3.1.	Valid	11
4.3.2.	Contextual Rule Required	12
4.3.3.	Disallowed	12
4.3.4.	Unassigned	12
4.3.5.	Examples	12
5.	Profiles	13
5.1.	Profiles Must Not Be Multiplied beyond Necessity	13
5.2.	Rules	14
5.2.1.	Width Mapping Rule	14
5.2.2.	Additional Mapping Rule	14
5.2.3.	Case Mapping Rule	14
5.2.4.	Normalization Rule	15
5.2.5.	Directionality Rule	15
5.3.	A Note about Spaces	16

6. Applications	17
6.1. How to Use PRECIS in Applications	17
6.2. Further Excluded Characters	18
6.3. Building Application-Layer Constructs	18
7. Order of Operations	19

8. Code Point Properties	20
9. Category Definitions Used to Calculate Derived Property	22
9.1. LetterDigits (A)	23
9.2. Unstable (B)	23
9.3. IgnorableProperties (C)	23
9.4. IgnorableBlocks (D)	23
9.5. LDH (E)	23
9.6. Exceptions (F)	23
9.7. BackwardCompatible (G)	23
9.8. JoinControl (H)	24
9.9. OldHangulJamo (I)	24
9.10. Unassigned (J)	24
9.11. ASCII7 (K)	24
9.12. Controls (L)	24
9.13. PrecisIgnorableProperties (M)	24
9.14. Spaces (N)	25
9.15. Symbols (O)	25
9.16. Punctuation (P)	25
9.17. HasCompat (Q)	25
9.18. OtherLetterDigits (R)	25
10. Guidelines for Designated Experts	26
11. IANA Considerations	27
11.1. PRECIS Derived Property Value Registry	27
11.2. PRECIS Base Classes Registry	27
11.3. PRECIS Profiles Registry	28
12. Security Considerations	29
12.1. General Issues	29
12.2. Use of the IdentifierClass	30
12.3. Use of the FreeformClass	30
12.4. Local Character Set Issues	31
12.5. Visually Similar Characters	31
12.6. Security of Passwords	33
13. Interoperability Considerations	34
13.1. Encoding	34
13.2. Character Sets	34
13.3. Unicode Versions	34

13.4. Potential Changes to Handling of Certain Unicode Code Points	34
14. References	35
14.1. Normative References	35
14.2. Informative References	36
Acknowledgements	40
Authors' Addresses	40

[1.](#) Introduction

Application protocols using Unicode characters [[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 characters and thus is agile with respect to versions of Unicode.

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

After the publication of [[RFC3454](#)] in 2002, several significant issues arose with the use of Stringprep in the IDN case, as documented in the IAB's recommendations regarding IDNs [[RFC4690](#)] (most significantly, Stringprep was tied to Unicode version 3.2).

Therefore, the newer IDNA specifications, here called "IDNA2008" ([[RFC5890](#)], [[RFC5891](#)], [[RFC5892](#)], [[RFC5893](#)], [[RFC5894](#)]), no longer use Stringprep and Nameprep. This migration away from Stringprep for IDNs prompted other "customers" of Stringprep to consider new approaches to the preparation, enforcement, and comparison of internationalized strings, as described in [[RFC6885](#)].

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

1. Define a small set of string classes that specify the Unicode characters (i.e., specific "code points") appropriate for common application protocol constructs.
2. Define each PRECIS string class in terms of Unicode code points and their properties so that an algorithm can be used to determine whether each code point or character category is (a) valid, (b) allowed in certain contexts, (c) disallowed, or (d) unassigned.
3. Use an "inclusion model" such that a string class consists only of code points that are explicitly allowed, with the result that any code point not explicitly allowed is forbidden.
4. Enable application protocols to define profiles of the PRECIS string classes if necessary (addressing matters such as width mapping, case mapping, Unicode normalization, and directionality) but strongly discourage the multiplication of profiles beyond

necessity in order to avoid violations of the "Principle of Least Astonishment".

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

- o Application protocols will be agile with regard to Unicode versions.
- o Implementers will be able to share code point tables and software code across application protocols, most likely by means of software libraries.
- o End users will be able to acquire more accurate expectations about the characters that are acceptable in various contexts. Given this more uniform set of string classes, it is also expected that copy/paste operations between software implementing different application protocols will be more predictable and coherent.

Whereas the string classes define the "baseline" code points for a range of applications, profiling enables application protocols to apply the string classes in ways that are appropriate for common constructs such as usernames [[PRECIS-Users-Pwds](#)], opaque strings such as passwords [[PRECIS-Users-Pwds](#)], and nicknames [[PRECIS-Nickname](#)]. Profiles are responsible for defining the handling of right-to-left characters as well as various mapping operations of the kind also discussed for IDNs in [[RFC5895](#)], such as case preservation or

lowercasing, Unicode normalization, mapping of certain characters to other characters or to nothing, and mapping of fullwidth and halfwidth characters.

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

- a. Enforcing all the rules of a profile for a single output string (e.g., to determine if a string can be included in a protocol slot, communicated to another entity within a protocol, stored in a retrieval system, etc.).

- b. Comparing two output strings to determine if they are equivalent, typically through octet-for-octet matching to test for "bit-string identity" (e.g., to make an access decision for purposes of authentication or authorization as further described in [[RFC6943](#)]).

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

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

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

The character categories and calculation rules defined under 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.

[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](#)].

As of the date of writing, the version of Unicode published by the

Unicode Consortium is 7.0 [[Unicode7.0](#)]; however, PRECIS is not tied to a specific version of Unicode. The latest version of Unicode is always available [[Unicode](#)].

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

[3.](#) Preparation, Enforcement, and Comparison

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

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

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

client cannot necessarily be trusted to properly generate such

strings, especially for security-sensitive contexts such as authentication and authorization.

[4.](#) String Classes

[4.1.](#) Overview

Starting in 2010, various "customers" of Stringprep began to discuss the need to define a post-Stringprep approach to the preparation and comparison of internationalized strings other than IDNs. This community analyzed the existing Stringprep profiles and also weighed the costs and benefits of defining a relatively small set of Unicode characters that would minimize the potential for user confusion caused by visually similar characters (and thus be relatively "safe") vs. defining a much larger set of Unicode characters that would maximize the potential for user creativity (and thus be relatively "expressive"). As a result, the community concluded that most existing uses could be addressed by two string classes:

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

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

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

The following subsections discuss the IdentifierClass and FreeformClass in more detail, with reference to the dimensions described in [Section 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).

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

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

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

[4.2.](#) IdentifierClass

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

[4.2.1.](#) Valid

- o Code points traditionally used as letters and numbers in writing systems, i.e., the LetterDigits ("A") category first defined in [\[RFC5892\]](#) and listed here under [Section 9.1](#).
- o Code points in the range U+0021 through U+007E, i.e., the (printable) ASCII7 ("K") 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 characters allowed in the IdentifierClass is wider than the range of characters allowed in IDNA2008. The main reason is that IDNA2008 applies the Unstable category before the LetterDigits category, thus disallowing uppercase characters, whereas the IdentifierClass does not apply the Unstable category.

[4.2.2.](#) Contextual Rule Required

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

[4.2.3.](#) Disallowed

- o Old Hangul Jamo characters, i.e., the OldHangulJamo ("I") category defined under [Section 9.9](#).
- o Control characters, i.e., the Controls ("L") category defined under [Section 9.12](#).
- o Ignorable characters, i.e., the PrecisIgnorableProperties ("M") category defined under [Section 9.13](#).
- o Space characters, i.e., the Spaces ("N") category defined under [Section 9.14](#).
- o Symbol characters, i.e., the Symbols ("O") category defined under [Section 9.15](#).
- o Punctuation characters, i.e., the Punctuation ("P") category defined under [Section 9.16](#).
- o Any character that has a compatibility equivalent, i.e., the HasCompat ("Q") category defined under [Section 9.17](#). These code points are disallowed even if they would otherwise be valid

according to the property-based rules specified in the previous section.

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

[4.2.4](#). Unassigned

Any code points that are not yet designated in the Unicode character set are considered unassigned for purposes of the IdentifierClass, and such code points are to be treated as disallowed. See [Section 9.10](#).

[4.2.5](#). Examples

As described in the Introduction to this document, the string classes do not handle all issues related to string preparation and comparison (such as case mapping); instead, such issues are handled at the level of profiles. Examples for profiles of the IdentifierClass can be found in [[PRECIS-Users-Pwds](#)] (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 [[PRECIS-Users-Pwds](#)]) or a nickname in a chatroom (see [[PRECIS-Nickname](#)]). We group such things into a class called "FreeformClass" having the following features.

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

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

4.3.1. Valid

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

- o Space characters, i.e., the Spaces ("N") category defined under [Section 9.14](#).
- o Symbol characters, i.e., the Symbols ("O") category defined under [Section 9.15](#).
- o Punctuation characters, i.e., the Punctuation ("P") category defined under [Section 9.16](#).

4.3.2. Contextual Rule Required

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

4.3.3. Disallowed

- o Old Hangul Jamo characters, i.e., the OldHangulJamo ("I") category defined under [Section 9.9](#).
- o Control characters, i.e., the Controls ("L") category defined under [Section 9.12](#).

- o Ignorable characters, i.e., the `PrecisIgnorableProperties` ("M") category defined under [Section 9.13](#).

[4.3.4.](#) Unassigned

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

[4.3.5.](#) Examples

As described in the Introduction to this document, the string classes do not handle all issues related to string preparation and comparison (such as case mapping); instead, such issues are handled at the level of profiles. Examples for profiles of the `FreeformClass` can be found in [\[PRECIS-Users-Pwds\]](#) (the `OpaqueString` profile) and [\[PRECIS-Nickname\]](#) (the `Nickname` profile).

[5.](#) Profiles

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

Profiles of the PRECIS string classes are registered with the IANA as described under [Section 11.3](#). Profile names use the following convention: they are of the form "Profilename of BaseClass", where the "Profilename" string is a differentiator and "BaseClass" is the name of the PRECIS string class being profiled; for example, the profile of the `FreeformClass` used for opaque strings such as

passwords is the OpaqueString profile [[PRECIS-Users-Pwds](#)].

[5.1.](#) Profiles Must Not Be Multiplied beyond Necessity

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

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

Nevertheless, profiles must not be multiplied beyond necessity.

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

[5.2.](#) Rules

[5.2.1.](#) Width Mapping Rule

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

The normalization form specified by a profile (see below) has an impact on the need for width mapping. Because width mapping is performed as a part of compatibility decomposition, a profile employing either normalization form KD (NFKD) or normalization form KC (NFKC) does not need to specify width mapping. However, if Unicode normalization form C (NFC) is used (as is recommended) then the profile needs to specify whether to apply width mapping; in this case, width mapping is in general RECOMMENDED because allowing fullwidth and halfwidth characters to remain unmapped to their compatibility variants would violate the "Principle of Least Astonishment". For more information about the concept of width in East Asian scripts within Unicode, see Unicode Standard Annex #11 [[UAX11](#)].

[5.2.2.](#) Additional Mapping Rule

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

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

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

The PRECIS mappings document [[PRECIS-Mappings](#)] describes such mappings in more detail.

[5.2.3.](#) Case Mapping Rule

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

If case mapping is desired (instead of case preservation), it is RECOMMENDED to use Unicode Default Case Folding as defined in the Unicode Standard [[Unicode](#)] (at the time of this writing, the algorithm is specified in Chapter 3 of [[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 [[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 [Section 12.6](#) and in [[PRECIS-Users-Pwds](#)].

[5.2.4.](#) Normalization Rule

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

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

[5.2.5.](#) Directionality Rule

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

Presenting bidirectional strings in different layout systems (e.g., a user interface that is configured to handle primarily an RTL script vs. an interface that is configured to handle primarily an LTR script) can yield display results that, while predictable to those who understand the display rules, are counter-intuitive to casual users. In particular, the same bidirectional string (in PRECIS

terms) might not be presented in the same way to users of those different layout systems, even though the presentation is consistent within any particular layout system. In some applications, these presentation differences might be considered problematic and thus the application designers might wish to restrict the use of bidirectional strings by specifying a directionality rule. In other applications, these presentation differences might not be considered problematic (this especially tends to be true of more "free-form" strings) and thus no directionality rule is needed.

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

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

[5.3](#). A Note about Spaces

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

- o Many Unicode characters are confusable with ASCII space.
- o Even if non-ASCII space characters are mapped to ASCII space (U+0020), space characters are often not rendered in user interfaces, leading to the possibility that a human user might consider a string containing spaces to be equivalent to the same string without spaces.

- o In some locales, some devices are known to generate a character other than ASCII space (such as ZERO WIDTH JOINER, U+200D) when a user performs an action like hitting the space bar on a keyboard.

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

However, the FreeformClass does allow spaces, which enables application protocols to define profiles of the FreeformClass that are more flexible than any profiles of the IdentifierClass. In addition, as explained in [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. Application developers still need to give some thought to how they will use the PRECIS string classes, or profiles thereof, in their applications. This section provides some guidelines to application developers (and to expert reviewers of application protocol specifications).

- o Don't define your own profile unless absolutely necessary (see [Section 5.1](#)). Existing profiles have been designed for wide reuse. It is highly likely that an existing profile will meet your needs, especially given the ability to specify further excluded characters ([Section 6.2](#)) and to build application-layer constructs (see [Section 6.3](#)).
- o Do specify:
 - * Exactly which entities are responsible for preparation,

enforcement, and comparison of internationalized strings (e.g., servers or clients).

- * Exactly when those entities need to complete their tasks (e.g., a server might need to enforce the rules of a profile before allowing a client to gain network access).

- * Exactly which protocol slots need to be checked against which profiles (e.g., checking the address of a message's intended recipient against the UsernameCaseMapped profile [[PRECIS-Users-Pwds](#)] of the IdentifierClass, or checking the password of a user against the OpaqueString profile [[PRECIS-Users-Pwds](#)] of the FreeformClass).

See [[PRECIS-Users-Pwds](#)] and [[XMPP-Addr-Format](#)] for definitions of these matters for several applications.

[6.2.](#) Further Excluded Characters

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

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

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

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

Typically, such exclusions are defined for the purpose of backward compatibility with legacy formats within an application protocol. These are defined for application protocols, not profiles, in order

to prevent multiplication of profiles beyond necessity (see [Section 5.1](#)).

[6.3](#). Building Application-Layer Constructs

Sometimes, an application-layer construct does not map in a straightforward manner to one of the base string classes or a profile thereof. Consider, for example, the "simple user name" construct in the Simple Authentication and Security Layer (SASL) [[RFC4422](#)]. Depending on the deployment, a simple user name might take the form of a user's full name (e.g., the user's personal name followed by a space and then the user's family name). Such a simple user name cannot be defined as an instance of the IdentifierClass or a profile thereof, since space characters are not allowed in the

IdentifierClass; however, it could be defined using a space-separated sequence of IdentifierClass instances, as in the following ABNF [[RFC5234](#)] from [[PRECIS-Users-Pwds](#)]:

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

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

[7](#). Order of Operations

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

1. Width Mapping Rule
2. Additional Mapping Rule
3. Case Mapping Rule

4. Normalization Rule
5. Directionality Rule
6. Behavioral rules for determining whether a code point is valid, allowed under a contextual rule, disallowed, or unassigned

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

[8.](#) Code Point Properties

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

1. Reviews and classifies the collections of code points in the Unicode character set by examining various code point properties.
2. Defines an algorithm for determining a derived property value, which can vary depending on the string class being used by the relevant application protocol.

This document is not intended to specify precisely how derived property values are to be applied in protocol strings. That information is the responsibility of the protocol specification that uses or profiles a PRECIS string class from this document. The value of the property is to be interpreted as follows.

PROTOCOL VALID Those code points that are allowed to be used in any

PRECIS string class (currently, IdentifierClass and FreeformClass). The abbreviated term "PVALID" is used to refer to this value in the remainder of this document.

SPECIFIC CLASS PROTOCOL VALID Those code points that are allowed to be used in specific string classes. In the remainder of this document, the abbreviated term *_PVAL is used, where * = (ID | FREE), i.e., either "FREE_PVAL" or "ID_PVAL". In practice, the derived property ID_PVAL is not used in this specification, since every ID_PVAL code point is PVALID.

CONTEXTUAL RULE REQUIRED Some characteristics of the character, such as its being invisible in certain contexts or problematic in others, require that it not be used in labels unless specific other characters or properties are present. As in IDNA2008, there are two subdivisions of CONTEXTUAL RULE REQUIRED -- the first for Join_controls (called "CONTEXTJ") and the second for other characters (called "CONTEXT0"). A character with the derived property value CONTEXTJ or CONTEXT0 MUST NOT be used unless an appropriate rule has been established and the context of the character is consistent with that rule. The most notable of the CONTEXTUAL RULE REQUIRED characters are the Join Control characters U+200D ZERO WIDTH JOINER and U+200C ZERO WIDTH NON-JOINER, which have a derived property value of CONTEXTJ. See [Appendix A of \[RFC5892\]](#) for more information.

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

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

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

The algorithm to calculate the value of the derived property is as

follows (implementations MUST NOT modify the order of operations within this algorithm, since doing so would cause inconsistent results across implementations):

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

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

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

The mechanisms described here allow determination of the value of the property for future versions of Unicode (including characters added after Unicode 5.2 or 7.0 depending on the category, since some categories mentioned in this document are simply pointers to IDNA2008 and therefore were defined at the time of Unicode 5.2). Changes in

Unicode properties that do not affect the outcome of this process therefore do not affect this framework. For example, a character can have its Unicode General_Category value (at the time of this writing, see Chapter 4 of [[Unicode7.0](#)]) change from So to Sm, or from Lo to Ll, without affecting the algorithm results. Moreover, even if such changes were to result, the BackwardCompatible list ([Section 9.7](#)) can be adjusted to ensure the stability of the results.

[9.](#) Category Definitions Used to Calculate Derived Property

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

1. Characters are placed in one or more character categories either (1) based on core properties defined by the Unicode Standard or (2) by treating the code point as an exception and addressing the code point based on its code point value. These categories are not mutually exclusive.
2. Set operations are used with these categories to determine the values for a property specific to a given string class. These operations are specified under [Section 8](#).

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

In the following specification of character categories, the operation that returns the value of a particular Unicode character property for a code point is designated by using the formal name of that property (from the Unicode PropertyAliases.txt 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 characters. Some of these categories are reused in PRECIS, and some of them are not; however, the lettering of categories is retained to prevent overlap and to ease implementation of both IDNA2008 and PRECIS in a single software application. The next eight categories (K-R) are specific to PRECIS.

[9.1.](#) LetterDigits (A)

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

[9.2.](#) Unstable (B)

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

[9.3.](#) IgnorableProperties (C)

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

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

[9.4.](#) IgnorableBlocks (D)

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

[9.5.](#) LDH (E)

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

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

[9.6.](#) Exceptions (F)

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

[9.7.](#) BackwardCompatible (G)

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

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

[9.8.](#) JoinControl (H)

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

[9.9.](#) OldHangulJamo (I)

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

[9.10.](#) Unassigned (J)

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

[9.11.](#) ASCII7 (K)

This PRECIS-specific category consists of all printable, non-space characters from the 7-bit ASCII range. By applying this category, the algorithm specified under [Section 8](#) exempts these characters from other rules that might be applied during PRECIS processing, on the assumption that these code points are in such wide use that disallowing them would be counter-productive.

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

[9.12.](#) Controls (L)

This PRECIS-specific category consists of all control characters.

L: Control(cp) = True

[9.13.](#) PrecisIgnorableProperties (M)

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

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

The definition for Default_Ignorable_Code_Point can be found in the DerivedCoreProperties.txt file [[DerivedCoreProperties](#)].

[9.14.](#) Spaces (N)

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

N: `General_Category(cp)` is in {Zs}

[9.15.](#) Symbols (0)

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

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

[9.16.](#) Punctuation (P)

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

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

[9.17.](#) HasCompat (Q)

This PRECIS-specific category is used to group code points that have compatibility equivalents as explained in the Unicode Standard (at the time of this writing, see Chapters 2 and 3 of [[Unicode7.0](#)]).

Q: `toNFKC(cp) != cp`

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

[9.18.](#) OtherLetterDigits (R)

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

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

[10](#). Guidelines for Designated Experts

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

Therefore:

- o Protocol designers are strongly encouraged to question the assumption that they need to define new profiles, since existing profiles are designed for wide reuse (see [Section 5](#) for further discussion).
- o Those who persist in defining new profiles are strongly encouraged to clearly explain a strong justification for doing so, and to publish a stable specification that provides all of the information described under [Section 11.3](#).
- o The designated experts for profile registration requests ought to seek answers to all of the questions provided under [Section 11.3](#) and to encourage applicants to provide a stable specification documenting the profile (even though the registration policy for PRECIS profiles is Expert Review and a stable specification is not strictly required).
- o Developers of applications that use PRECIS are strongly encouraged

to apply the guidelines provided under [Section 6](#) and to seek out the advice of the designated experts or other knowledgeable individuals in doing so.

- o All parties are strongly encouraged to help prevent the multiplication of profiles beyond necessity, as described under [Section 5.1](#), and to use PRECIS in ways that will minimize user confusion and insecure application behavior.

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

[11.](#) IANA Considerations

[11.1.](#) PRECIS Derived Property Value Registry

IANA has created and now maintains the "PRECIS Derived Property Value" registry that records 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 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.

[11.2.](#) PRECIS Base Classes Registry

IANA has created the "PRECIS Base Classes" registry. In accordance with [[RFC5226](#)], the registration policy is "RFC Required".

The registration template is as follows:

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

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

Specification: [the RFC number]

The initial registrations are as follows:

Base Class: FreeformClass.

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

Specification: [Section 4.3 of RFC 7564](#).

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 7564](#).

[11.3](#). PRECIS Profiles Registry

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

The registration template is as follows:

Name: [the name of the profile]

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

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

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

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

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

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

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

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

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

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

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

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

- o Would an existing PRECIS string class or profile solve the problem? If not, why not? (See [Section 5.1](#) for related considerations.)

- o Is the problem being addressed by this profile well defined?
- o Does the specification define what kinds of applications are involved and the protocol elements to which this profile applies?
- o Is the profile clearly defined?
- o Is the profile based on an appropriate dividing line between user interface (culture, context, intent, locale, device limitations, etc.) and the use of conformant strings in protocol elements?
- o Are the width mapping, case mapping, additional mappings, normalization, and directionality rules appropriate for the intended use?
- o Does the profile explain which entities enforce the rules, and when such enforcement occurs during protocol operations?
- o Does the profile reduce the degree to which human users could be surprised or confused by application behavior (the "Principle of Least Astonishment")?
- o Does the profile introduce any new security concerns such as those described under [Section 12](#) of this document (e.g., false positives for authentication or authorization)?

[12.](#) Security Considerations

[12.1.](#) General Issues

If input strings that appear "the same" to users are programmatically considered to be distinct in different systems, or if input strings that appear distinct to users are programmatically considered to be "the same" in different systems, then users can be confused. Such confusion can have security implications, such as the false positives and false negatives discussed in [[RFC6943](#)]. One starting goal of work on the PRECIS framework was to limit the number of times that users are confused (consistent with the "Principle of Least Astonishment"). Unfortunately, this goal has been difficult to achieve given the large number of application protocols already in existence. Despite these difficulties, profiles should not be

multiplied beyond necessity (see [Section 5.1](#)). In particular, application protocol designers should think long and hard before defining a new profile instead of using one that has already been defined, and if they decide to define a new profile then they should clearly explain their reasons for doing so.

The security of applications that use this framework can depend in part on the proper preparation, enforcement, and comparison of internationalized strings. For example, such strings can be used to make authentication and authorization decisions, and the security of an application could be compromised if an entity providing a given string is connected to the wrong account or online resource based on different interpretations of the string (again, see [\[RFC6943\]](#)).

Specifications of application protocols that use this framework are strongly encouraged to describe how internationalized strings are used in the protocol, including the security implications of any false positives and false negatives that might result from various enforcement and comparison operations. For some helpful guidelines, refer to [\[RFC6943\]](#), [\[RFC5890\]](#), [\[UTR36\]](#), and [\[UTS39\]](#).

[12.2.](#) Use of the IdentifierClass

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

[12.3.](#) Use of the FreeformClass

Strings that conform to the FreeformClass and many profiles thereof can include virtually any Unicode character. This makes the FreeformClass quite expressive, but also problematic from the perspective of possible user confusion. Protocol designers are hereby warned that the FreeformClass contains code points they might not understand, and are encouraged to profile the IdentifierClass wherever feasible; however, if an application protocol requires more code points than are allowed by the IdentifierClass, protocol designers are encouraged to define a profile of the FreeformClass that restricts the allowable code points as tightly as possible.

(The PRECIS Working Group considered the option of allowing "superclasses" as well as profiles of PRECIS string classes, but decided against allowing superclasses to reduce the likelihood of security and interoperability problems.)

[12.4.](#) Local Character Set Issues

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

[12.5.](#) Visually Similar Characters

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

The problem of confusable characters is not necessarily caused by the use of Unicode code points outside the ASCII range. For example, in some presentations and to some individuals the string "juliet" (spelled with DIGIT ONE, U+0031, as the third character) might appear to be the same as "juliet" (spelled with LATIN SMALL LETTER L, U+006C), especially on casual visual inspection. This phenomenon is sometimes called "typejacking".

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

In some examples of confusable characters, it is unlikely that the average human could tell the difference between the real string and the fake string. (Indeed, there is no programmatic way to distinguish with full certainty which is the fake string and which is the real string; in some contexts, the string formed of Cherokee

characters might be the real string and the string formed of ASCII characters might be the fake string.) Because PRECIS-compliant strings can contain almost any properly encoded Unicode code point, it can be relatively easy to fake or mimic some strings in systems that use the PRECIS framework. The fact that some strings are easily

confused introduces security vulnerabilities of the kind that have also plagued the World Wide Web, specifically the phenomenon known as phishing.

Despite the fact that some specific suggestions about identification and handling of confusable characters appear in the Unicode Security Considerations [[UTR36](#)] and the Unicode Security Mechanisms [[UTS39](#)], it is also true (as noted in [[RFC5890](#)]) that "there are no comprehensive technical solutions to the problems of confusable characters." Because it is impossible to map visually similar characters without a great deal of context (such as knowing the font families used), the PRECIS framework does nothing to map similar-looking characters together, nor does it prohibit some characters because they look like others.

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

1. An application service SHOULD define a policy that specifies the scripts or blocks of characters that the service will allow to be registered (e.g., in an account name) or stored (e.g., in a 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 characters from more than one script and SHOULD restrict registrations to characters drawn from a very small number of scripts (e.g., scripts that are well understood by the administrators of the

service, to improve manageability).

2. User-oriented application software SHOULD define a policy that specifies how internationalized strings will be presented to a human user. Because every human user of such software has a preferred language or a small set of preferred languages, the software SHOULD gather that information either explicitly from the user or implicitly via the operating system of the user's device. Furthermore, because most languages are typically represented by a single script or a small set of scripts, and because most scripts are typically contained in one or more blocks of characters, the software SHOULD warn the user when

presenting a string that mixes characters from more than one script or block, or that uses characters outside the normal range of the user's preferred language(s). (Such a recommendation is not intended to discourage communication across different communities of language users; instead, it recognizes the existence of such communities and encourages due caution when presenting unfamiliar scripts or characters to human users.)

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

[12.6.](#) Security of Passwords

Two goals of passwords are to maximize the amount of entropy and to minimize the potential for false positives. These goals can be achieved in part by allowing a wide range of code points and by ensuring that passwords are handled in such a way that code points are not compared aggressively. Therefore, it is NOT RECOMMENDED for application protocols to profile the FreeformClass for use in passwords in a way that removes entire categories (e.g., by disallowing symbols or punctuation). Furthermore, it is NOT RECOMMENDED for application protocols to map uppercase and titlecase code points to their lowercase equivalents in such strings; instead, it is RECOMMENDED to preserve the case of all code points contained in such strings and to compare them in a case-sensitive manner.

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

Some application protocols use passwords directly, whereas others reuse technologies that themselves process passwords (one example of such a technology is the Simple Authentication and Security Layer [[RFC4422](#)]). Moreover, passwords are often carried by a sequence of protocols with backend authentication systems or data storage systems such as RADIUS [[RFC2865](#)] and 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, since the password is not available to the server in plaintext form.

Further discussion of password handling can be found in [[PRECIS-Users-Pwds](#)].

[13.](#) Interoperability Considerations

[13.1.](#) Encoding

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

[13.2.](#) Character Sets

It is known that some existing systems are unable to support the full Unicode character set, or even any characters outside the ASCII range. If two (or more) applications need to interoperate when

exchanging data (e.g., for the purpose of authenticating a username or password), they will naturally need to have in common at least one coded character set (as defined by [[RFC6365](#)]). Establishing such a baseline is a matter for the application protocol that uses PRECIS, not for the PRECIS framework.

[13.3.](#) Unicode Versions

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

[13.4.](#) Potential Changes to Handling of Certain Unicode Code Points

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

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

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

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

14. References

14.1. Normative References

- [RFC20] Cerf, V., "ASCII format for network interchange", STD 80, [RFC 20](#), DOI 10.17487/RFC0020, October 1969, <<http://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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc6365>>.

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

[Unicode7.0]

The Unicode Consortium, "The Unicode Standard, Version 7.0.0", (Mountain View, CA: The Unicode Consortium, 2014 ISBN 978-1-936213-09-2), <<http://www.unicode.org/versions/Unicode7.0.0/>>.

14.2. Informative References

[DerivedCoreProperties]

The Unicode Consortium, "DerivedCoreProperties-7.0.0.txt", Unicode Character Database, February 2014, <<http://www.unicode.org/Public/UCD/latest/ucd/DerivedCoreProperties.txt>>.

[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.

[PRECIS-Mappings]

Yoneya, Y. and T. Nemoto, "Mapping characters for PRECIS classes", Work in Progress, [draft-ietf-precis-mappings-10](#), May 2015.

[PRECIS-Nickname]

Saint-Andre, P., "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Nicknames", Work in Progress, [draft-ietf-precis-nickname-17](#), April 2015.

[PRECIS-Users-Pwds]

Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", Work in Progress, [draft-ietf-precis-saslprepbis-17](#), May 2015.

[PropertyAliases]

The Unicode Consortium, "PropertyAliases-7.0.0.txt", Unicode Character Database, November 2013, <<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, <<http://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, <<http://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, <<http://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, <<http://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, <<http://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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc4690>>.

- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, [RFC 5234](#), DOI 10.17487/RFC5234, January 2008, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc5890>>.
- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", [RFC 5891](#), DOI 10.17487/RFC5891, August 2010, <<http://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, <<http://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, <<http://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, <<http://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, <<http://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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc6943>>.
- [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", 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/>>.
- [XMPP-Addr-Format] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Address Format", Work in Progress,

Acknowledgements

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

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

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

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

Some algorithms and textual descriptions have been borrowed from [\[RFC5892\]](#). Some text regarding security has been borrowed from [\[RFC5890\]](#), [\[PRECIS-Users-Pwds\]](#), and [\[XMPP-Addr-Format\]](#).

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

Authors' Addresses

Peter Saint-Andre

&yet

E-Mail: peter@andyet.com

URI: <https://andyet.com/>

Marc Blanchet

Viagenie

246 Aberdeen

Quebec, QC G1R 2E1

Canada

E-Mail: Marc.Blanchet@viagenie.ca

URI: <http://www.viagenie.ca/>