

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

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

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on August 23, 2015.

Copyright Notice

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

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

Table of Contents

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

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

1. Introduction

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2. Terminology

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

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

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

3. Preparation, Enforcement, and Comparison

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

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

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

4. String Classes

4.1. Overview

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

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

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

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

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

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

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

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

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

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

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

4.2. IdentifierClass

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

4.2.1. Valid

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

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

4.2.2. Contextual Rule Required

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

4.2.3. Disallowed

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

4.2.4. Unassigned

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

4.2.5. Examples

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

4.3. FreeformClass

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

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

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

4.3.1. Valid

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

4.3.2. Contextual Rule Required

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

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

4.3.3. Disallowed

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

4.3.4. Unassigned

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

4.3.5. Examples

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

5. Profiles

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

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

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

5.1. Profiles Must Not Be Multiplied Beyond Necessity

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

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

Nevertheless, profiles must not be multiplied beyond necessity.

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

5.2. Rules

5.2.1. Width Mapping Rule

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

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

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

5.2.2. Additional Mapping Rule

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

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

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

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

5.2.3. Case Mapping Rule

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

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

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

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

5.2.4. Normalization Rule

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

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

5.2.5. Directionality Rule

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

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

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

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

5.3. A Note about Spaces

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

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

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

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

6. Applications

6.1. How to Use PRECIS in Applications

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

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

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

6.2. Further Excluded Characters

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

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

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

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

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

6.3. Building Application-Layer Constructs

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

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

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

7. Order of Operations

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

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

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

8. Code Point Properties

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

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

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

PROTOCOL VALID Those code points that are allowed to be used in any PRECIS string class (currently, IdentifierClass and FreeformClass). The abbreviated term "PVALID" is used to refer to this value in the remainder of this document.

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

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

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

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

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

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

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

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

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

The mechanisms described here allow determination of the value of the property for future versions of Unicode (including characters added after Unicode 5.2 or 7.0 depending on the category, since some categories mentioned in this document are simply pointers to IDNA2008 and therefore were defined at the time of Unicode 5.2). Changes in Unicode properties that do not affect the outcome of this process therefore do not affect this framework. For example, a character can have its Unicode General_Category value (see Chapter 4 of the Unicode Standard [Unicode7.0]) change from So to Sm, or from Lo to Ll, without affecting the algorithm results. Moreover, even if such changes were to result, the BackwardCompatible list (Section 9.7) can be adjusted to ensure the stability of the results.

9. Category Definitions Used to Calculate Derived Property

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

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

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

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

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

9.1. LetterDigits (A)

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

9.2. Unstable (B)

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

9.3. IgnorableProperties (C)

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

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

9.4. IgnorableBlocks (D)

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

9.5. LDH (E)

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

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

9.6. Exceptions (F)

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

9.7. BackwardCompatible (G)

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

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

9.8. JoinControl (H)

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

9.9. OldHangulJamo (I)

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

9.10. Unassigned (J)

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

9.11. ASCII7 (K)

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

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

9.12. Controls (L)

This PRECIS-specific category consists of all control characters.

L: Control(cp) = True

9.13. PrecisIgnorableProperties (M)

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

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

The definition for Default_Ignorable_Code_Point can be found in the DerivedCoreProperties.txt [2] file.

9.14. Spaces (N)

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

N: General_Category(cp) is in {Zs}

9.15. Symbols (O)

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

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

9.16. Punctuation (P)

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

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

9.17. HasCompat (Q)

This PRECIS-specific category is used to group code points that have compatibility equivalents as explained in Chapter 2 and Chapter 3 of the Unicode Standard [Unicode7.0].

Q: toNFKC(cp) != cp

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

9.18. OtherLetterDigits (R)

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

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

10. Guidelines for Designated Experts

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

Therefore:

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

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

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

11. IANA Considerations

11.1. PRECIS Derived Property Value Registry

IANA is requested to create a PRECIS-specific registry with the Derived Properties for the versions of Unicode that are released after (and including) version 7.0. The derived property value is to be calculated in cooperation with a designated expert [RFC5226] according to the rules specified under Section 8 and Section 9.

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

11.2. PRECIS Base Classes Registry

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

The registration template is as follows:

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

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

Specification: [the RFC number]

The initial registrations are as follows:

Base Class: FreeformClass.

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

Specification: Section 4.3 of this document.

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

Base Class: IdentifierClass.

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

Specification: Section 4.2 of this document.

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

11.3. PRECIS Profiles Registry

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

The registration template is as follows:

Name: [the name of the profile]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

12. Security Considerations

12.1. General Issues

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

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

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

12.2. Use of the IdentifierClass

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

12.3. Use of the FreeformClass

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

12.4. Local Character Set Issues

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

12.5. Visually Similar Characters

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

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

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

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

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

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

presentation of visually similar characters. The following recommendations are appropriate:

1. An application service SHOULD define a policy that specifies the scripts or blocks of characters that the service will allow to be registered (e.g., in an account name) or stored (e.g., in a file name). Such a policy SHOULD be informed by the languages and scripts that are used to write registered account names; in particular, to reduce confusion, the service SHOULD forbid registration or storage of strings that contain characters from more than one script and SHOULD restrict registrations to characters drawn from a very small number of scripts (e.g., scripts that are well-understood by the administrators of the service, to improve manageability).
2. User-oriented application software SHOULD define a policy that specifies how internationalized strings will be presented to a human user. Because every human user of such software has a preferred language or a small set of preferred languages, the software SHOULD gather that information either explicitly from the user or implicitly via the operating system of the user's device. Furthermore, because most languages are typically represented by a single script or a small set of scripts, and because most scripts are typically contained in one or more blocks of characters, the software SHOULD warn the user when presenting a string that mixes characters from more than one script or block, or that uses characters outside the normal range of the user's preferred language(s). (Such a recommendation is not intended to discourage communication across different communities of language users; instead, it recognizes the existence of such communities and encourages due caution when presenting unfamiliar scripts or characters to human users.)

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

12.6. Security of Passwords

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

passwords in a way that removes entire categories (e.g., by disallowing symbols or punctuation). Furthermore, it is NOT RECOMMENDED for application protocols to map uppercase and titlecase code points to their lowercase equivalents in such strings; instead, it is RECOMMENDED to preserve the case of all code points contained in such strings and to compare them in a case-sensitive manner.

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

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

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

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

13. Interoperability Considerations

13.1. Encoding

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

13.2. Character Sets

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

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

13.3. Unicode Versions

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

13.4. Potential Changes to Handling of Certain Unicode Code Points

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

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

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

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

14. References

14.1. Normative References

- [RFC20] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [Unicode7.0]
The Unicode Consortium, "The Unicode Standard, Version 7.0.0", 2014,
<<http://www.unicode.org/versions/Unicode7.0.0/>>.

14.2. Informative References

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

- [I-D.klensin-idna-5892upd-unicode70]
Klensin, J. and P. Faelststroem, "IDNA Update for Unicode 7.0.0", draft-klensin-idna-5892upd-unicode70-03 (work in progress), January 2015.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.
- [RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.
- [RFC3491] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.
- [RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and Recommendations for Internationalized Domain Names (IDNs)", RFC 4690, September 2006.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.
- [RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.
- [RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.
- [RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.
- [RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.
- [RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.
- [RFC6452] Faltstrom, P. and P. Hoffman, "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA) - Unicode 6.0", RFC 6452, November 2011.
- [RFC6885] Blanchet, M. and A. Sullivan, "Stringprep Revision and Problem Statement for the Preparation and Comparison of Internationalized Strings (PRECIS)", RFC 6885, March 2013.
- [RFC6943] Thaler, D., "Issues in Identifier Comparison for Security Purposes", RFC 6943, May 2013.
- [UAX9] The Unicode Consortium, "Unicode Standard Annex #9: Unicode Bidirectional Algorithm", September 2012, <<http://unicode.org/reports/tr9/>>.
- [UAX11] The Unicode Consortium, "Unicode Standard Annex #11: East Asian Width", September 2012, <<http://unicode.org/reports/tr11/>>.
- [UAX15] The Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", August 2012, <<http://unicode.org/reports/tr15/>>.

[UnicodeCurrent]

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

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

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

14.3. URIs

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

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

Appendix A. Acknowledgements

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

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

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

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

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

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

Authors' Addresses

Peter Saint-Andre
&yet

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

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

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

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

M. Blanchet
Viagenie
A. Sullivan
Dyn, Inc.
January 22, 2013

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

Abstract

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

Status of this Memo

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

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

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

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

Copyright Notice

Copyright (c) 2013 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. Keywords	5
3. Conventions	5
4. Stringprep Profiles Limitations	6
5. Major Topics for Consideration	7
5.1. Comparison	7
5.1.1. Types of Identifiers	7
5.1.2. Effect of comparison	8
5.2. Dealing with characters	8
5.2.1. Case folding, case sensitivity, and case preservation	8
5.2.2. Stringprep and NFKC	8
5.2.3. Character mapping	9
5.2.4. Prohibited characters	9
5.2.5. Internal structure, delimiters, and special characters	9
5.2.6. Restrictions because of glyph similarity	10
5.3. Where the data comes from and where it goes	10
5.3.1. User input and the source of protocol elements	10
5.3.2. User output	11
5.3.3. Operations	11
6. Considerations for Stringprep replacement	12
7. Security Considerations	13
8. IANA Considerations	13
9. Discussion home for this draft	13
10. Acknowledgements	13
11. Informative References	14
Appendix A. Classification of Stringprep Profiles	18
Appendix B. Evaluation of Stringprep Profiles	18
B.1. iSCSI Stringprep Profile: RFC3722 (and RFC3721, RFC3720)	18
B.2. SMTP/POP3/ManageSieve Stringprep Profiles: RFC4954,RFC5034,RFC 5804	20
B.3. IMAP Stringprep Profiles: RFC5738, RFC4314: Usernames	22
B.4. IMAP Stringprep Profiles: RFC5738: Passwords	23
B.5. Anonymous SASL Stringprep Profiles: RFC4505	24
B.6. XMPP Stringprep Profiles: RFC3920 Nodeprep	26
B.7. XMPP Stringprep Profiles: RFC3920 Resourceprep	27

B.8. EAP Stringprep Profiles: RFC3748	28
Authors' Addresses	28

1. Introduction

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

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

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

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

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

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

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

2. Keywords

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

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

Additionally, this document defines the following keyword:

- o PRECIS: Preparation and Comparison of Internationalized Strings

3. Conventions

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

4. Stringprep Profiles Limitations

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

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

Protocols that use Stringprep profiles use strings for different purposes:

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

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

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

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

5. Major Topics for Consideration

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

5.1. Comparison

5.1.1. Types of Identifiers

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

Absolute Identifiers Identifiers that can be compared byte-by-byte for equality.

Definite Identifiers Identifiers that have a well-defined comparison algorithm on which all parties agree.

Indefinite Identifiers Identifiers that have no single comparison algorithm on which all parties agree.

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

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

5.1.2. Effect of comparison

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

5.2. Dealing with characters

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

5.2.1. Case folding, case sensitivity, and case preservation

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

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

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

5.2.2. Stringprep and NFKC

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

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

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

5.2.3. Character mapping

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

5.2.4. Prohibited characters

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

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

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

5.2.5. Internal structure, delimiters, and special characters

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

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

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

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

5.2.6. Restrictions because of glyph similarity

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

5.3. Where the data comes from and where it goes

5.3.1. User input and the source of protocol elements

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

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

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

5.3.2. User output

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

5.3.3. Operations

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

5.3.3.1. String classes

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

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

5.3.3.2. Community Considerations

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

5.3.3.3. Unicode Incompatible Changes

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

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

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

6. Considerations for Stringprep replacement

The above suggests the following guidance:

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

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

7. Security Considerations

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

8. IANA Considerations

This document has no actions for IANA.

9. Discussion home for this draft

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

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

10. Acknowledgements

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

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

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

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

11. Informative References

- [I-D.iab-identifier-comparison]
Thaler, D., "Issues in Identifier Comparison for Security Purposes", draft-iab-identifier-comparison-07 (work in progress), August 2012.
- [NEWPREP] "Newprep BoF Meeting Minutes", March 2010.
- [RFC0952] Harrenstien, K., Stahl, M., and E. Feinler, "DoD Internet host table specification", RFC 952, October 1985.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.
- [RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.
- [RFC3491] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.
- [RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.
- [RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.
- [RFC3722] Bakke, M., "String Profile for Internet Small Computer Systems Interface (iSCSI) Names", RFC 3722, April 2004.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence

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

- [RFC3922] Saint-Andre, P., "Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)", RFC 3922, October 2004.
- [RFC4011] Waldbusser, S., Saperia, J., and T. Hongal, "Policy Based Management MIB", RFC 4011, March 2005.
- [RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, December 2005.
- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC4505] Zeilenga, K., "Anonymous Simple Authentication and Security Layer (SASL) Mechanism", RFC 4505, June 2006.
- [RFC4511] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.
- [RFC4513] Harrison, R., "Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms", RFC 4513, June 2006.
- [RFC4518] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Internationalized String Preparation", RFC 4518, June 2006.
- [RFC4616] Zeilenga, K., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, August 2006.
- [RFC4643] Vinocur, J. and K. Murchison, "Network News Transfer Protocol (NNTP) Extension for Authentication", RFC 4643, October 2006.
- [RFC4683] Park, J., Lee, J., Lee, H., Park, S., and T. Polk, "Internet X.509 Public Key Infrastructure Subject Identification Method (SIM)", RFC 4683, October 2006.
- [RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and

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

- [RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, March 2007.
- [RFC4954] Siemborski, R. and A. Melnikov, "SMTP Service Extension for Authentication", RFC 4954, July 2007.
- [RFC5034] Siemborski, R. and A. Menon-Sen, "The Post Office Protocol (POP3) Simple Authentication and Security Layer (SASL) Authentication Mechanism", RFC 5034, July 2007.
- [RFC5051] Crispin, M., "i;unicode-casemap - Simple Unicode Collation Algorithm", RFC 5051, October 2007.
- [RFC5054] Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, "Using the Secure Remote Password (SRP) Protocol for TLS Authentication", RFC 5054, November 2007.
- [RFC5122] Saint-Andre, P., "Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP)", RFC 5122, February 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5456] Spencer, M., Capouch, B., Guy, E., Miller, F., and K. Shumard, "IAX: Inter-Asterisk eXchange Version 2", RFC 5456, February 2010.
- [RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.
- [RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, July 2010.
- [RFC5804] Melnikov, A. and T. Martin, "A Protocol for Remotely Managing Sieve Scripts", RFC 5804, July 2010.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework",

RFC 5890, August 2010.

- [RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.
- [RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.
- [RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.
- [RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.
- [RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.
- [RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.
- [RFC6452] Faltstrom, P. and P. Hoffman, "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA) - Unicode 6.0", RFC 6452, November 2011.
- [UAX15] "Unicode Standard Annex #15: Unicode Normalization Forms", UAX 15, September 2009.
- [Unicode61] The Unicode Consortium. The Unicode Standard, Version 6.1, defined by: "The Unicode Standard -- Version 6.1", (Mountain View, CA: The Unicode Consortium, 2012. ISBN 978-1-936213-02-3), September 2009, <<http://www.unicode.org/versions/Unicode6.1.0/>>.
- [ietf78precis] Blanchet, M., "PRECIS Framework", Proceedings of the Seventy-Eighth Internet Engineering Task Force <https://www.ietf.org/proceedings/78/>, July 2010, <<http://www.ietf.org/proceedings/78/slides/precis-2.pdf>>.

Appendix A. Classification of Stringprep Profiles

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

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

Table 1

Appendix B. Evaluation of Stringprep Profiles

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

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

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

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

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

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

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

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

Comparison Byte-by-byte

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

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

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

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

Normalization NFKC, as specified by RFC 3454.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Normalization: NFKC (as per RFC 4013)

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

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

String Classes: - simple username. See Section 2 of RFC 4013 for details on restrictions. Note that some implementations allow spaces in these. While implementations are not required to use a specific format, an authorization identity frequently has the same format as an email address (and EAI email address in the future), or as a left hand side of an email address. Note: whatever is recommended for SMTP/POP/ManageSieve authorization identity should also be used for IMAP authorization identities, as IMAP/POP3/SMTP/ManageSieve are frequently implemented together.

Internal Structure: None

User Output: Unlikely, but possible. For example, if it is the same as an email address.

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

How much tolerance for change from existing Stringprep approach? Not sure.

Background information: In RFC 5034, when describing the POP3 AUTH command: The authorization identity generated by the SASL exchange is a simple username, and SHOULD use the SASLprep profile (see RFC4013) of the StringPrep algorithm (see RFC3454) to prepare these names for matching. If preparation of the authorization identity fails or results in an empty string (unless it was transmitted as the empty string), the server MUST fail the authentication. In RFC 4954, when describing the SMTP AUTH command: The authorization identity generated by this SASL exchange is a "simple username" (in the sense defined in SASLprep), and both client and server SHOULD (*) use the SASLprep profile of the StringPrep algorithm to prepare these names for transmission or comparison. If preparation of the authorization identity fails or results in an empty string (unless it was transmitted as the empty string), the server MUST fail the authentication. (*) Note: Future revision of this specification may change this requirement to MUST. Currently, the SHOULD is used in order to avoid breaking the majority of existing implementations. In RFC 5804, when describing the ManageSieve AUTHENTICATE command: The authorization identity generated by this SASL exchange is a "simple username" (in the sense defined in SASLprep), and both client and server MUST use the SASLprep profile of the StringPrep algorithm to prepare these names for transmission or comparison. If preparation of the authorization

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

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

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

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

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

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

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

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

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

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

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

Normalization: NFKC (as per RFC 4013)

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

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

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

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

Internal Structure: None

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

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

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

B.4. IMAP Stringprep Profiles: RFC5738: Passwords

Description: "Password" parameter to the IMAP LOGIN command

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

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

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

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

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

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

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

Normalization: NFKC (as per RFC 4013)

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

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

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

Internal Structure: None

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

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

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

Background information: RFC 5738 (IMAP INTERNATIONALIZATION): 5.

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

B.5. Anonymous SASL Stringprep Profiles: RFC4505

Description: RFC 4505 defines a "trace" field:

Comparison: this field is not intended for comparison (only used for logging)

Case folding; case sensitivity, preserve case: No case folding/case sensitive

Do users input the strings directly? Yes. Possibly entered in configuration UIs, or on a command line. Can also be stored in configuration files. The value can also be automatically generated by clients (e.g. a fixed string is used, or a user's email address).

How users input strings? Keyboard/voice, stylus (pick from a list). Copy-paste - possibly.

Normalization: None

Disallowed Characters Control characters are disallowed. (See Section 3 of RFC 4505)

Which other strings or identifiers are these most similar to? RFC 4505 says that the trace "should take one of two forms: an Internet email address, or an opaque string that does not contain the '@' (U+0040) character and that can be interpreted by the system administrator of the client's domain." In practice, this is a freeform text, so it belongs to a different class from "email address" or "username".

Are these strings or identifiers sometimes the same as strings or identifiers from other protocols (e.g., does an IM system sometimes use the same credentials database for authentication as an email system)? Yes: see above. However there is no strong need to keep them consistent in the future.

How are users exposed to these strings, how are they published? No. However, The value can be seen in server logs

Impacts of false positives and false negatives: False positive: a user can be confused with another user. False negative: two distinct users are treated as the same user. But note that the trace field is not authenticated, so it can be easily falsified.

Tolerance of changes in the community The community would be flexible.

Delimiters No internal structure, but see comments above about frequent use of email addresses.

Background information: The Anonymous Mechanism The mechanism consists of a single message from the client to the server. The client may include in this message trace information in the form of a string of UTF-8-encoded Unicode characters prepared in accordance with StringPrep and the "trace" Stringprep profile defined in Section 3 of this document. The trace information, which has no semantical value, should take one of two forms: an Internet email address, or an opaque string that does not contain the '@' (U+0040) character and that can be interpreted by the system administrator of the client's domain. For privacy reasons, an Internet email address or other information identifying the user should only be used with permission from the user. 3. The

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

B.6. XMPP Stringprep Profiles: RFC3920 Nodeprep

Description: Localpart of JabberID ("JID"), as in:

localpart@domainpart/resourcepart

How It's Used: - Usernames (e.g., stpeter@jabber.org) - Chatroom names (e.g., precis@jabber.ietf.org) - Publish-subscribe nodes - Bot names

Who Generates It: - Typically, end users via an XMPP client - Sometimes created in an automated fashion

User Input Methods: - Typed by user - Copy-and-paste - Perhaps voice input - Clicking a URI/IRI

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

Comparison Method: "Type 2" (common algorithm)

Case Folding, Sensitivity, Preservation: - Strings are always folded to lowercase - Case is not preserved

Impact of Comparison: False positives: - unable to authenticate at server (or authenticate to wrong account) - add wrong person to buddy list - join the wrong chatroom - improperly grant privileges (e.g., chatroom admin) - subscribe to wrong pubsub node - interact with wrong bot - allow communication with blocked entity False negatives: - unable to authenticate - unable to add someone to buddy list - unable to join desired chatroom - unable to use granted privileges (e.g., chatroom admin) - unable to subscribe to desired pubsub node - unable to interact with desired bot - disallow communication with unblocked entity

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

B.7. XMPP Stringprep Profiles: RFC3920 Resourceprep

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

B.8. EAP Stringprep Profiles: RFC3748

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

Comparison Methods: n/a (per-method)

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

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

User input: n/a (per-method)

Normalization: n/a (per-method)

Mapping: n/a (per-method)

Disallowed characters: n/a (per-method)

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

Internal structure: n/a (per-method)

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

Operations: n/a (per-method)

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

Authors' Addresses

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

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

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

Email: asullivan@dyn.com

