L2/10-023

Technical Reports

Proposed Update Unicode Technical Standard #39

UNICODE SECURITY MECHANISMS

Version	2 (draft 3)
Authors	Mark Davis (markdavis@google.com), Michel Suignard (michel@suignard.com)
Date	2010-02-04
This Version	http://www.unicode.org/reports/tr39/tr39-3.html
Previous Version	http://www.unicode.org/reports/tr39/tr39-2.html
Latest Version	http://www.unicode.org/reports/tr39/
~~Latest Working Draft~~	~~http://www.unicode.org/draft/reports/tr39/tr39.html~~
Revision	3

Summary

Because Unicode contains such a large number of characters and incorporates the varied writing systems of the world, incorrect usage can expose programs or systems to possible security attacks. This document specifies mechanisms that can be used in detecting possible security problems.

[Review Note: The primary changes are to the data: see Modifications for more information, and how to submit suggestions. More identifer restrictions will be added over time, and the idnchars.txt file will need to be updated in accordance with UTS#46 Unicode IDNA Compatibility Processing.]

[Review Note: The tables will be renumbered consecutively from one in the final version.]

Status

*This is a **draft** document which may be updated, replaced, or superseded by other documents at any time. Publication does not imply endorsement by the Unicode Consortium. This is not a stable document; it is inappropriate to cite this document as other than a work in progress.*

> ***A Unicode Technical Standard (UTS)** is an independent specification. Conformance to the Unicode Standard does not imply conformance to any UTS.*

Please submit corrigenda and other comments with the online reporting form [Feedback]. Related information that is useful in understanding this document is found in References. For the latest version of the Unicode Standard see [Unicode]. For a list of current Unicode Technical Reports see [Reports]. For more information about versions of the Unicode Standard, see [Versions].

To allow access to the most recent work of the Unicode security subcommittee on this document, the "Latest Working Draft" link in the header points to the latest working-draft document under development.

Contents

---

1. Introduction

Unicode Technical Report #36: Unicode Security Considerations [UTR36] provides guidelines for detecting and avoiding security problems connected with the use of Unicode. This document specifies mechanisms that are used in that document, and can be used elsewhere. Readers should be familiar with [UTR36] before continuing.

2. Conformance

An implementation claiming conformance to this specification must do so in conformance to the following clauses.

C0. *An implementation claiming to implement the General Profile for Identifiers shall do so in accordance with the specifications in Section 3.1 General Security Profile for Identifiers.*

Alternatively, it shall declare that it uses a modification, and provide a precise list of characters that are added to or removed from the profile.

~~***C1.*** *An implementation claiming to implement the IDN Identifier Profiles shall do in accordance with the specifications in Section 3.2 IDN Security Profiles for Identifiers.*~~

~~*Alternatively, it shall declare that it uses a modification, and provide a precise list of characters that are added to or removed from the profile.*~~

C2. *An implementation claiming to implement any of the following confusable-detection functions must do so in accordance with the specifications in Section 4. Confusable Detection.*

1. *X and Y are single-script confusables*
2. *X and Y are mixed-script confusables*
3. *X and Y are whole-script confusables*
4. *X has any simple single-script confusables*
5. *X has any mixed-script confusable*
6. *X has any whole-script confusable*

Alternatively, it shall declare that it uses a modification, and provide a precise list of character mappings that are added to or removed from the provided ones.

C3. *An implementation claiming to detect mixed scripts must do so in accordance with the specifications in Section 5. Mixed Script Detection.*

Alternatively, it shall declare that it uses a modification, and provide a precise specification of the differences in behavior.

[Review Note: The conformance clauses will be renumbered.]

3. Identifier Characters

Identifiers are special-purpose strings used for identification — strings that are deliberately limited to particular repertoires for that purpose. Exclusion of characters from identifiers does not at all affect the general use of those characters, such as within documents. *UAX #31, Identifier and Pattern Syntax* [UAX31] provides a recommended method of determining which strings should qualify as identifiers. The UAX #31 specification extends the common practice of defining identifiers in terms of letters and numbers to the Unicode repertoire.

UAX #31 also permits other protocols to use that method as a base, and to define a *profile* that adds or removes characters. For example, identifiers for specific programming languages typically add some characters like '$', and remove others like '-' (because of the use as *minus*), while IDNA removes '_' (among others). For more information, see *UAX #31, Identifier and Pattern Syntax* [UAX31].

This document provides for additional identifier profiles for environments where security is at issue. These are profiles of the extended identifiers based on properties and specifications of the Unicode Standard [Unicode], including:

- The XID_Start and XID_Continue properties defined in the Unicode Character Database (see [DCore])
- The case folding operation defined in *Chapter 3. Conformance* of [Unicode]
- The NFKC and NFKD normalizations defined in [UAX15].

The data files used in defining these profiles follow the UCD File Format, which has a semicolon-delimited list of data fields associated with given characters, with each field referenced by number. For more details, see [UCDFormat].

3.1. General Security Profile for Identifiers

The file [idmod] provides data for a profile of identifiers in environments where security is at issue. The file contains a set of characters recommended to be restricted from use. It also contains a small set of characters that are recommended as additions (to the list of characters defined by the XID_Start and XID_Continue properties), because they may be used in identifiers in a broader context than programming identifiers.

The restricted characters are characters not in common use, removed so as to further reduce the possibilities for visual confusion. Initially, the following are being excluded: characters not in modern use; characters only used in specialized fields, such as liturgical characters, mathematical letter-like symbols, and certain phonetic alphabetics~~; and ideographic characters that are not part of a set of core CJK ideographs consisting of the CJK Unified Ideographs block plus IICore (the set of characters defined by the IRG as the minimal set of required ideographs for East Asian use). A small number of such characters are allowed back in so that~~

~~the profile includes all the characters in the country-specific restricted IDN lists: see *Appendix F. Country-Specific IDN Restrictions*~~.

The principle has been to be more conservative initially, allowing for the set to be modified in the future as requirements for characters are refined. For information on handling that, see *Section 2.9.1 Backwards Compatibility* of [UTR36].

In the file [idmod], Field 1 is the character in question, Field 2 is an action (either *restricted* or *allowed*), and Field 3 (if present) is a reason. The reasons are:

Table 0. Identifier Modification Key

Action	Reason	Description
restricted	default-ignorable	Characters with the Unicode property Default_Ignorable_Code_Point
restricted	historic	Characters not in customary modern use; includes [UAX31] Table 4. Candidate Characters for Exclusion from Identifiers
restricted	limited-use	Characters whose status is uncertain, or that are used in limited environments, or those in [UAX31] Table 5. Recommended Scripts: Limited Usage
restricted	not-chars	Unassigned characters, private use characters, surrogates, most control characters
restricted	not-NFKC	Characters that are not NFKC.
restricted	not-xid	Other characters that don't qualify as Unicode identifiers
restricted	obsolete	Technical characters that are no longer in use; characters with the Unicode Property Deprecated

restricted	technical	Technical characters
allowed	**inclusion**	[UAX31] Table 3. Candidate Characters for Inclusion in Identifiers. See also the notes on MidLetter in [UAX29].
allowed	**recommended**	[UAX31] Table 5. Recommended Scripts (excluding *restricted*)

Restricted characters should be treated with caution in registration; disallowed unless there is good reason to allow them in the enviroment in question. In user interfaces for lookup of identifiers, warnings of some kind may be appropriate. For more information, see [UTR36].

Allowed characters may be further restricted by intersecting with the characters allowed in the particular identifier syntax in question, or where there is other information available in the environment in question. In particular, the candidate characters for inclusion are punctuation, and may fall outside of most identifer syntax.

The distinctions among the reasons is not strict; if there are multiple reasons for restricting a character only one is given. The important characteristic is the action: whether or not the character is restricted. *As more information is gathered about characters, this data may change in successive versions.* That can cause either the action or reason to change for a particular character. Thus users of this data should be prepared for changes in successive versions, such as by having a grandfathering policy in place for registrations.

[Review Note: The Reasons have been simplified in this version so that there are a small number of them.]

[Review Note: The terminology above is from the previous version. During the editorial pass, some of these may change. In particular: "Action" is not the best term, "identifier modification" could be improved; perhaps "Identifier Restriction" for both? "Reason" might also read better as "subcategory".]

This list is also used in deriving the IDN Identifiers list given below. It is, however, designed to be applied to other environments, and is not limited to Unicode 3.2 (as IDNA is currently), so that it can be applied to a future version of IDNA that includes the (large) repertoire of characters that have been added since Unicode 3.2.

3.2. IDN Security Profiles for Identifiers

The previous version of this document defined operations and data that apply to the version of IDNA defined in 2003, which has been superseded. The identifer modification data can be applied to whichever specification of IDNA is being used. For more information, see the [IDN FAQ].

~~The file [idn-chars] provides a recommended profile that that further restricts the characters allowed in for use in IDN, as described in the recommendations above.~~

~~The data for this profile is presented as a series of tables organized by the *type*, as given in Field 2 in the data file. The following table provides a description of this data.~~

~~**Table 1. IDN Identifier Profile Types**~~

~~**Type**~~	~~**Description**~~
~~output~~	~~This type marks characters that are retained in this profile in the output of IDN; that is, any characters outside of this set are not allowed by this profile. This list was formed by taking the characters allowed in IDNA [RFC3491], and intersecting that with the characters in *Section 3.1 General Security Profile for Identifiers*.~~
~~nonstarting~~	~~This type marks characters that are disallowed at the start of an identifier. (IDNA, unlike [UAX31] or most programming languages, does not place restrictions on which characters can start an identifier.)~~

~~Thus an IDN identifier that conforms to this profile is subject to all of the other conditions imposed by IDNA [RFC3491], *and* has the additional requirement that it have the following form:~~

```
~~<strict-profile-identifier> := <SP-start> <SP-continue>*~~
~~<SP-start> := [[:Field2=output:] - [:Field2=nonstarting:]]~~
~~<SP-continue> := [:Field2=output:]~~
```

~~The focus of this profile is on the characters allowed in the *output* of StringPrep, not on the input characters. Because of the additional restrictions on the output form, implementations should consider supplying additional input mappings to aid in keyboard entry. That is, in circumstances where the user is typing in a URL into an address bar, these additional mappings are recommended so as to allow people to type characters that they may not otherwise easily be able to type. However, this is not formally part of the identifier profile; simply a recommendation for GUIs, given the constraints of the identifier profile.~~

~~Table 2. Remapping Characters~~

~~0027 → 2019~~	~~' → ’~~	~~APOSTROPHE → MODIFIER LETTER APOSTROPHE~~
~~2018 → 02BB~~	~~‘ → ʻ~~	~~LEFT SINGLE QUOTATION MARK → MODIFIER LETTER TURNED COMMA~~
~~2019 → 02BC~~	~~’ → ʼ~~	~~RIGHT SINGLE QUOTATION MARK → MODIFIER LETTER APOSTROPHE~~
~~309B → 3099~~	~~゛ → ゙~~	~~KATAKANA-HIRAGANA VOICED SOUND MARK → COMBINING KATAKANA-HIRAGANA VOICED SOUND MARK~~
~~309C → 309A~~	~~゜ → ゚~~	~~KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK → COMBINING KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK~~

4. Confusable Detection

The tables in the data file [confusables] provide a mechanism for determining when two strings are visually confusable. The data in these files may be refined and extended over time. For information on handling that, see *Section 2.9.1 Backwards Compatibility* of [UTR36]. The data is organized into four different tables, depending on the desired parameters. Each table provides a mapping from source characters to target strings.

On the basis of this data, there are three main classes of confusable strings:

> X and Y are *single-script confusables* if they are confusable according to the Single-Script table, and each of them is a single script string according to *Section 5. Mixed Script Detection.* *Examples:* "søs" and "søs" in Latin.
>
> X and Y are *mixed-script confusables* if they are confusable according to the Mixed-Script table, and they are not single-script confusables. Example: "paypal" in Latin and "paypal" with the 'a' being in Cyrillic.
>
> X and Y are *whole-script confusables* if they are *mixed-script confusables,* and each of them is a single script string. Example: "scope" in Latin and "scope" in Cyrillic.

To see whether two strings X and Y are confusable according to a given table (abbreviated as X ≅ Y), an implementation uses a transform of X called a *skeleton(X)* defined by:

1. Converting X to ~~NFKD~~ NFD format, as described in [UAX15].
2. Successively mapping each source character in X to the target string according to the specified data table.
3. Reapplying ~~NFKD~~ NFD.

The resulting strings *skeleton*(X) and *skeleton*(Y) are then compared. If they are identical (codepoint-for-codepoint), then X ≅ Y according to the table.

> **Note**: the strings *skeleton*(X) and *skeleton*(Y) are ***not*** intended for display, storage or transmission. They should be thought of instead as an intermediate processing form, similar to a hashcode. The characters in *skeleton*(X) and *skeleton*(Y) are ***not*** guaranteed to be identifier characters.

Implementations do not have to recursively apply the mappings, because the transforms are idempotent. That is,

skeleton(skeleton(X)) = skeleton(X).

This mechanism does impose transitivity on the data, so if X ≅ Y and Y ≅ Z, then X ≅ Z. It would be possible to provide a more sophisticated confusable detection, by providing a metric between given characters, indicating their 'closeness'. However, that is computationally much more expensive, and requires more sophisticated data, so at this point in time the simpler mechanism has been chosen. That means that in some cases the test may be overly inclusive. However the frequency of such cases in real data should be small.

The data files are in the following format: for each line in the data file, Field 1 is the source, Field 2 is the target, and Field 3 is a type identifying the table.

Example:

309C ; 030A ; SL #* (゜ → ̊) KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK → COMBINING RING ABOVE # →゚→→゜→

The types are explained in the table below. The comments provide the character names. If the data was derived via transitivity, then there is an extra comment at the end. For instance, in the above example the derivation was:

- U+309A (゚) COMBINING KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK →
- U+FF9F (ﾟ) HALFWIDTH KATAKANA SEMI-VOICED SOUND MARK →
- U+309C (゜) KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK →
- U+030A (̊) COMBINING RING ABOVE

To reduce security risks, it is advised that identifiers use case-folded forms, thus eliminating uppercase variants where possible. Character with the script values COMMON or INHERITED are ignored when testing for differences in script.

Table 3. Confusable Data Table Types

Type	Name	Description
SL	Single-Script, Lowercase	This table is used to test cases of single-script confusables, where both the source character and the target string are case folded. For example: # (ø → ø̷) LATIN SMALL LETTER O WITH STROKE → LATIN SMALL LETTER O, COMBINING SHORT SOLIDUS OVERLAY
SA	Single-Script, Any-Case	This table is used to test cases of single-script confusables, where the output allows for mixed case (which may be later folded away). For example, this table contains the following entry not found in SL: # (O → 0) LATIN CAPITAL LETTER O → DIGIT ZERO
ML	Mixed-Script, Lowercase	This table is used to test cases of mixed-script and whole-script confusables, where both the source character and the target string are case folded. For example, this table contains the following entry not found in SL or SA: # (ν → v) GREEK SMALL LETTER NU → LATIN SMALL LETTER V
MA	Mixed-Script, Any-Case	This table is used to test cases of mixed-script and whole-script confusables, where the output allows for mixed case (which may be later folded away). For example, this table contains the following entry not found in SL, SA, or ML: # (Ι → l) GREEK CAPITAL LETTER IOTA → LATIN SMALL LETTER L

4.1 Whole-Script Confusables

Data is also provided for testing a string to see if a string X has any whole-script confusable, using the file [confusablesWS]. This file consists of a list of lines of the form:

```
<range>; <sourceScript>; <targetScript>; <type> #comment
```

The types are either L for lowercase-only, or A for any-case, where the any-case ranges are broader (including uppercase and lowercase characters). If the string is only lowercase, use the lowercase-only table. Otherwise, first test according to the any-case table, then case-fold the string and test according to the lowercase-only table.

In using the data, all of the lines having the same *sourceScript* and *targetScript* are collected together to form a set of Unicode characters. Logically, the file is thus a set of tuples of the form <*sourceScript, unicodeSet, targetScript*>. For example, the following lines are present for Latin to Cyrillic:

```
0061       ; Latn; Cyrl; L #     (a)    LATIN SMALL LETTER A
0063..0065 ; Latn; Cyrl; L # [3] (c..e) LATIN SMALL LETTER C..LATIN SMALL LETT
...
0292       ; Latn; Cyrl; L #     (ʒ)    LATIN SMALL LETTER EZH
```

They logically form a tuple <*Latin, [a c-e ... \u0292], Cyrillic*>, which indicates that a Latin string containing characters only from that Unicode set can have a whole-script confusable in Cyrillic (lowercase-only).

To test to see if a single-script string *givenString* has a whole-script confusable in *targetScript*, the following process is used.

1. Convert the givenString to ~~NFKD~~ NFD format, as specified in [UAX15]
2. Let *givenSet* be the set of all characters in givenString
3. Remove all [:script=common:] and [:script=inherited:] characters from *givenSet*
4. Let *givenScript* be the script of the characters in *givenSet*
   - (if there is more than one script, fail with error).
5. See if there is a tuple <*sourceScript, unicodeSet, targetScript*> where

  - *sourceScript* = *givenScript*
  - *unicodeSet* ⊇ *givenSet*
6. If so, then there is a whole-script confusable in *targetScript*

The test is actually slightly broader than simply a whole-script confusable; what it tests is whether the given string has a whole-script confusable string in another script, possibly with the addition or removal of common/inherited characters such as numbers and combining marks characters to both strings. In practice, however, this broadening has no significant impact.

Implementations would normally read the data into appropriate data structures in memory for processing. A quick additional optimization is to keep, for each script, a *fastReject* set, containing characters in the script contained in none of the *unicodeSet* values.

The following is a Java sample of how this code can work (using the Java version of [ICU]):

```
/*
 * For this routine, we don't care what the target scripts are,
 * just whether there is at least one whole-script confusable.
 */
boolean hasWholeScriptConfusable(String s) {
        int givenScript = getSingleScript(s);
        if (givenScript == UScript.INVALID_CODE) {
                throw new IllegalArgumentException("Not single script string")
        }
        UnicodeSet givenSet = new UnicodeSet()
                .addAll(s)
                .removeAll(commonAndInherited);
        if (fastReject[givenScript].containsSome(givenSet)) return false;
        UnicodeSet[] possibles = scriptToUnicodeSets[givenScript];
        for (int i = 0; i < possibles.length; ++i) {
                if (possibles[i].containsAll(givenSet)) return true;
        }
        return false;
}
```

The data in [confusablesWS] is built using the data in [confusables], and subject to the same caveat: The data in these files may be refined and extended over time. For information on handling that, see *Section 2.9.1 Backwards Compatibility* of [UTR36].

4.2 Mixed-Script Confusables

To test for mixed-script confusables, use the following process.

1. Convert the given string to ~~NFKD~~ NFD format, as specified in [UAX15].
2. For each script found in the given string, see if all the characters in the string outside of that script have whole-script confusables for that script (according to *Section 4.1 Whole-Script Confusables*).

Example 1: 'paypal', with Cyrillic 'a's.

There are two scripts, Latin and Cyrillic. The set of Cyrillic characters {a} has a whole-script confusable in Latin. Thus the string is a mixed-script confusable.

Example 2: 'toys-я-us', with one Cyrillic character 'я'.

The set of Cyrillic characters {я} does not have a whole-script confusable in Latin (there is no Latin character that looks like 'я', nor does the set of Latin characters {o s t u y} have a whole-script confusable in Cyrillic (there is no Cyrillic character that looks like 't' or 'u'). Thus this string is not a mixed-script confusable.

Example 3: '1ive', with a Greek 'v' and Cyrillic 'e'.

There are three scripts, Latin, Greek, and Cyrillic. The set of Cyrillic characters {e} and the set of Greek characters {v} each have a whole-script confusable in Latin. Thus the string is a mixed-script confusable.

5. Mixed Script Detection

The Unicode Standard supplies information that can be used for determining the script of characters and detecting mixed-script text. The determination of script is according to the Unicode Standard [UAX24], using data from the Unicode Character Database [UCD].

In determining mixed script, *Common* and *Inherited* script characters are ignored. For example, "abc-def" counts as a single script: the script of "-" is ignored.

The following is a Java sample of how this process works (using the Java version of [ICU]):

```
/**
 * Returns the script of the input text. Script values of COMMON and INHERITED
 * @param source Input text.
 * @return Script value found in the text.
 * If more than one script values are found, then UScript.INVALID_CODE is retu
 * If no script value is found (other than COMMON or INHERITED), then UScript.
 */
public static int getSingleScript(String source) {
    if (source.length() == 0) return UScript.COMMON;
        int lastScript = UScript.COMMON; // temporary value
        int cp;
        for (int i = 0; i < source.length(); i += UTF16.getCharCount(cp)) {
        cp = UTF16.charAt(source, i);
            int script = UScript.getScript(cp);
            if (script == UScript.COMMON || script == UScript.INHERITED) {
                continue;
            }
        if (lastScript == UScript.COMMON) {
            lastScript = script;
        } else if (script != lastScript) {
            return UScript.INVALID_CODE;
        }
    }
    return lastScript;
}
```

Using the Unihan data in the Unicode Character Database [UCD] it is possible to extend this mechanism, to qualify strings as 'mixed script' where they contain both simplified-only and traditional-only Chinese characters.

6. Development Process

As discussed in [UTR36], confusability among characters cannot be an exact science. There are many factors that make confusability among character a matter of degree:

- Shapes of characters vary greatly among fonts used to represent them. The Unicode standard represents them in the chart section with representative glyphs, but font designers are free to create their own glyphs. Because fonts can easily be created representing any Unicode code position using an arbitrary glyph, character confusability with arbitrary fonts can never be avoided. For example, one could design a font where the ‘a’ looks like a ‘b’ , ‘c’ like a ‘d’, and so on.

- Writing systems using contextual shaping (such as Arabic, many south-Asian systems) introduce even more variation in text rendering. Characters don’t really have an abstract shape in isolation and are only rendered as part of cluster of characters making words, expressions, and sentences. It is in fact a fairly common occurrence to find the same visual text representation corresponding to very different logical words that can only be recognized by context, if at all.
- Font style variant may introduce a confusability which does not exist in another style (for example: normal versus italic). For example, in the Cyrillic script, the small letter TE (U+0442) looks like a small caps Latin ‘T’ in normal style while it looks like a small Latin ‘m’ in italic style.

The confusability tables were created by collecting a number of prospective confusables, examining those confusables according to a set of fonts, and processing the result for transitive closure.

The prospective confusables were gathered from a number of sources. Volunteers from within IBM and Microsoft, with native speakers for languages with different writing systems, gathered initial lists. The compatibility mappings were also used as a source~~, as were the mappings from the draft UTR #30 "Character Foldings" (since withdrawn)~~. Erik van der Poel also contributed a list derived from running a program over a large number of OpenType fonts to catch characters that shared identical glyphs within a font. More recently, engineers at Google examined font data on Windows and Macintosh to generate additional confusables.

The process of gathering visual confusables is ongoing: the Unicode Consortium welcomes submission of additional mappings. The complex scripts of South / South East Asia also need special attention.

~~*Please submit suggestions for additional confusables, or suggested corrections to the given ones, with the online reporting form [Feedback]. Additions must be listed in a plain-text file in the standard format, such as:*~~

```
~~#comment~~
~~2500 ; 4E00 # comment~~
~~002E ; 0702 # comment~~
~~...~~
```

[Review Note: For review of the data and suggesting changes:

- The most useful view of the confusables data is the confusablesSummary file.
  - This file groups all the confusables together. Note that the results may vary depending on the font used. Also, some "unnatural" confusables are added by transitivity (between characters, or between NFKC_Casefold equivalents).
- The most useful view of the identifier restrictions is the xidmodifications file.
- You can suggest changes with the form at security-mechanisms.]

The initial focus is on characters that can be in the recommended profile for identifiers, because they are of most concern. For mixed-script confusability, the initial focus is on confusable characters between the Latin script and other scripts, because this is currently perceived as the most important threat. Other combinations of scripts should be more extensively reviewed in the future.

In-script confusability is extremely user-dependent. For example, in the Latin script, characters with accents or appendices may look similar to the unadorned characters for some users, especially if they are not familiar with their meaning in a particular language. However, most users in position to trust identifiers will have at least a minimum understanding of the range of characters in their own script, and there are separate mechanisms available to deal with other scripts, as discussed in [UTR36].

The fonts used to assess the confusables included those used by the major operating systems in user interfaces. In addition, the representative glyphs used in the Unicode Standard were also considered. Fonts used for the user interface in operating systems are an important source, because they are the ones that will usually be seen by users in circumstances where confusability is important, such such as when using IRIS (Internationalized Resource Identifiers) and their sub-elements (e.g. domain names). These fonts have a number of other relevant characteristics. They rarely changed by OS and applications; changes brought by system upgrades tend to be gradual to avoid usability disruption. Because user interface elements need to be legible at low screen resolution (implying a small number of pixel per EM units), fonts used in these contexts tend to be designed in sans-serif style, which has the tendency to increase the possibility of confusables. (There are, however, some locales locales where a serif style is in common use, for example, Chinese.) Furthermore, strict bounding box requirements create even more constraints for scripts which use relatively large ascenders and descenders. This also limits space allocated for accent or tone marks, and can also create more opportunities for confusability.

Pairs of prospective confusables were removed if they were always visually distinct at common sizes, both within and across fonts.

This data was then closed under transitivity, so that if X≅Y and Y≅Z, then X≅Z. In addition, the data is closed under substring operations, so that if X≅Y then AXB≅AYB. It was then processed to produce the in-script and cross-script tables. This is so that a single table can be used to map an input string to a resulting *skeleton*.

~~The files contain some internal information in comments, indicating how the transitive closure was done. For example:~~

```
~~2500 ;  4E00 ;  MA       # ( ─ → 一) BOX DRAWINGS LIGHT HORIZONTAL → CJK UNIF~~
~~    # {source:1192} ─ {source:961} ─ {source:1785}~~
```

~~The second comment mark (#), here on a separate line, indicates intermediate steps in the transitive closure, with {..} indicating the reason (the original source mapping between the characters). In this case, the mappings are:~~

~~U+2500 (─) ⟷ U+2015 (―) ⟷ U+2014 (—) ⟷ U+4E00 (一)~~

A skeleton is intended *only* for internal use for testing confusability of strings; the resulting text is not at all suitable for display to users, since it will appear to be a hodgepodge of different scripts. In particular, the result of mapping an identifier will not necessary be an identifier. Thus the confusability mappings can be used to test whether two identifiers are confusable (if their skeletons are the same), but should definitely not be used as a "normalization" of identifiers.

As described elsewhere, there are cases where the data may be different than expected. Sometimes this is because two characters (or sequences) may only be confusable in some fonts. In other cases, it is because of transitivity. For example, the dotless and dotted I are considered equivalent (ı ↔ i), because they look the same when accents such as an *acute* are applied to each. However, for practical implementation usage, transitivity is sufficiently important that some oddities are accepted.

The data may be enhanced in future versions of this specification. For information on handling this, see *Section 2.9.1 Backwards Compatibility* of [UTR36].

Note allowing mixtures of upper and lowercase text would complicate the process, and produce a large number of false positives. For example, mixing cases in Latin and Greek may make the Latin letters pairs {Y, U} and {N, V} confusable. That is because *Y* is confusable with the Greek capital Upsilon, and the lowercase upsilon is confusable with the lowercase Latin *u*.

[Review Note: We will point to the security FAQ here.]

7 Data Files

The following files provide data used to implement the recommendations in this document. The data may be refined in future versions of this specification. For information on handling this, see *Section 2.9.1 Backwards Compatibility* of [UTR36].

[Review Note: the following revises the directory structure for the data to put the data in http://www.unicode.org/Public. Note that the headers in the data files will also be revised.]

The files are in http://www.unicode.org/Public/security/. The directories there contain data files associated with a given version, with names such as:

http://www.unicode.org/Public/security/revision-02

The data files for the latest approved version are also in the directory:

http://www.unicode.org/Public/security/latest

[Review Note: The following files are for the draft version, found in

http://www.unicode.org/Public/security/revision-03

There are some known glitches in the data that will be corrected, including improper expansions of characters like □/□.]

[data2.0]	uts39-data-xx.zip [Review Note: the zip file for revision 03 will only be created for the released version.]	**A zipped version of all the data files.**
~~[idnchars]~~	~~idnchars.txt~~	~~**IDN Characters:**~~ ~~Provides a profile of identifiers from *UAX #31, Identifier and Pattern Syntax* [UAX31] as a recommended restriction of IDN~~

		~~identifiers for security purposes.~~
[idmod]	xidmodifications.txt	**Identifier Modifications**: Provides the list of additions and restrictions recommended for building a profile of identifiers for environments where security is at issue.
[confusables]	confusables.txt	**Visually Confusable Characters**: Provides a mapping for visual confusables for use in further restricting identifiers for security. The usage of the file is described in *Section 4. Confusable Detection*.
[summary]	confusablesSummary.txt	**Summary of the confusables**: with transitive closure.
[confusablesWS]	confusablesWholeScript.txt	**Whole Script Confusables.** Data for testing for the possible existence of whole-script and mixed-script confusables. See *Appendix B. Confusable Detection*
[intentional]	intentional.txt	**Intentional Confusable Mappings.** The class of characters whose glyphs

		in any particular typeface would probably be designed to be identical in shape, by intention, at least when using a harmonized typeface design
[source]	source/	**Source Data Files.** These are the source data files used to build the above files.

[Review Note: For review of the data and suggesting changes:

- The data can perhaps be most usefully viewed interactively, at
  - http://unicode.org/cldr/utility/list-unicodeset.jsp?a=\p{any}-\p{nfkdqc%3Dn}-\p{cn}-\p{cs}-\p{co}&g=sc+idr
  - This is a full listing; for a particular script, change the "any" to "sc=Latin" or other script name.
- The most useful static view of the confusables data is the confusablesSummary file.
  - This file groups all the confusables together. Note that the results may vary depending on the font used. Also, some "unnatural" confusables are added by transitivity (between characters, or between NFKC_Casefold equivalents).
- The most useful view of the identifier restrictions is the xidmodifications file.]

[Review Note: add a section for submitting suggested data for a future update; make the form a page on the Unicode site, roughly like security-mechanisms. Structure it like the [Feedback] link, and add to the references.]

Acknowledgements

Steven Loomis and other people on the ICU team were very helpful in developing the original proposal for this technical report. Thanks also to the following people for their feedback or contributions to this document or earlier versions of it: Douglas Davidson, Martin Dürst, Asmus Freytag, Deborah Goldsmith, Paul Hoffman, Peter Karlsson, Gervase Markham, Eric Muller, Erik van der Poel, Michael van Riper, Marcos Sanz, Alexander Savenkov, Dominikus Scherkl, and Kenneth Whistler. Thanks to Peter Peng for his assistance with font confusables.

References

Warning: all internet-drafts and news links have unstable links; you may have to adjust the URL to get to the latest document.

References not listed here may be found in http://www.unicode.org/reports/tr41/#UAX41.

~~[CharMod]~~	~~Character Model for the World Wide Web 1.0: Fundamentals~~ ~~http://www.w3.org/TR/charmod/~~
~~[Charts]~~	~~Unicode Charts (with Last Resort Glyphs)~~ ~~http://www.unicode.org/charts/lastresort.html~~ ~~See also:~~ ~~http://developer.apple.com/fonts/LastResortFont/~~ ~~http://developer.apple.com/fonts/LastResortFont/LastResort~~
[DCore]	Derived Core Properties http://www.unicode.org/Public/UNIDATA/DerivedCorePropert
~~[Display]~~	~~Display Problems?~~ ~~http://www.unicode.org/help/display_problems.html~~
[DemoConf]	http://unicode.org/cldr/utility/confusables.jsp
[DemoIDN]	http://unicode.org/cldr/utility/idna.jsp
[DemoIDNChars]	http://unicode.org/cldr/utility/list-unicodeset.jsp?a=\p{age% {cn}-\p{cs}-\p{co}&abb=on&g=uts46+idna+idna2008

~~[DNS-Case]~~	~~Donald E. Eastlake 3rd. "Domain Name System (DNS) Case Ins~~ ~~Clarification". Internet Draft, January 2005~~ ~~http://www.ietf.org/internet-drafts/draft-ietf-dnsext-insensi~~ ~~06.txt~~
~~[FAQSec]~~	~~Unicode FAQ on Security Issues~~ ~~http://www.unicode.org/faq/security.html~~
~~[ICANN]~~	~~Guidelines for the Implementation of Internationalized Domai~~ ~~http://www.icann.org/general/idn-guidelines-20jun03.htm~~
[ICU]	International Components for Unicode http://site.icu-project.org/
~~[IDNReg]~~	~~Registry for IDN Language Tables~~ ~~http://www.iana.org/assignments/idn/~~ ~~Tables are found at:~~ ~~http://www.iana.org/assignments/idn/registered.htm~~
~~[IDN-Demo]~~	~~ICU (International Components for Unicode) IDN Demo~~ ~~http://ibm.com/software/globalization/icu/demo/domain/~~
[IDN-FAQ]	http://www.unicode.org/faq/idn.html
[Feedback]	Reporting Errors and Requesting Information Online http://www.unicode.org/reporting.html
~~[Museum]~~	~~Internationalized Domain Names (IDN) in .museum – Supporte~~ ~~Languages~~ ~~http://about.museum/idn/language.html~~
[Reports]	Unicode Technical Reports http://www.unicode.org/reports/ *For information on the status and development process for te* *reports, and for a list of technical reports.*
~~[RFC1034]~~	~~P. Mockapetris. "DOMAIN NAMES – CONCEPTS AND FACILITIES~~ ~~1034, November 1987.~~ ~~http://ietf.org/rfc/rfc1034.txt~~
~~[RFC1035]~~	~~P. Mockapetris. "DOMAIN NAMES – IMPLEMENTATION AND~~ ~~SPECIFICATION", RFC 1034, November 1987.~~ ~~http://ietf.org/rfc/rfc1035.txt~~

~~[RFC1535]~~	~~E. Gavron. "A Security Problem and Proposed Correction With~~ ~~Deployed DNS Software", RFC 1535, October 1993~~ ~~http://ietf.org/rfc/rfc1535.txt~~
~~[RFC3454]~~	~~P. Hoffman, M. Blanchet. "Preparation of Internationalized Stri~~ ~~("stringprep")", RFC 3454, December 2002.~~ ~~http://ietf.org/rfc/rfc3454.txt~~
~~[RFC3490]~~	~~Faltstrom, P., Hoffman, P. and A. Costello, "Internationalizing~~ ~~Names in Applications (IDNA)", RFC 3490, March 2003.~~ ~~http://ietf.org/rfc/rfc3490.txt~~
[RFC3491]	Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile Internationalized Domain Names (IDN)", RFC 3491, March 200 http://ietf.org/rfc/rfc3491.txt
~~[RFC3492]~~	~~Costello, A., "Punycode: A Bootstring encoding of Unicode for~~ ~~Internationalized Domain Names in Applications (IDNA)", RFC~~ ~~March 2003.~~ ~~http://ietf.org/rfc/rfc3492.txt~~
~~[RFC3743]~~	~~Konishi, K., Huang, K., Qian, H. and Y. Ko, "Joint Engineering~~ ~~Guidelines for Internationalized Domain Names (IDN) Registra~~ ~~Administration for Chinese, Japanese, and Korean", RFC 3743,~~ ~~2004.~~ ~~http://ietf.org/rfc/rfc3743.txt~~
~~[RFC3986]~~	~~T. Berners-Lee, R. Fielding, L. Masinter. "Uniform Resource Ide~~ ~~(URI): Generic Syntax", RFC 3986, January 2005.~~ ~~http://ietf.org/rfc/rfc3986.txt~~
~~[RFC3987]~~	~~M. Duerst, M. Suignard. "Internationalized Resource Identifiers~~ ~~RFC 3987, January 2005.~~ ~~http://ietf.org/rfc/rfc3987.txt~~
[UCD]	Unicode Character Database. http://www.unicode.org/ucd/ *For an overview of the Unicode Character Database and a list associated files.*
[UCDFormat]	UCD File Format http://www.unicode.org/reports/tr44/#Format_Conventions

~~[UAX9]~~	~~UAX #9: The Bidirectional Algorithm~~ ~~http://www.unicode.org/reports/tr9/~~
~~[UAX15]~~	~~UAX #15: Unicode Normalization Forms~~ ~~http://www.unicode.org/reports/tr15/~~
~~[UAX24]~~	~~UAX #24: Script Names~~ ~~http://www.unicode.org/reports/tr24/~~
~~[UAX31]~~	~~UAX #31, Identifier and Pattern Syntax~~ ~~http://www.unicode.org/reports/tr31/~~
~~[UTR36]~~	~~UTR #36: Unicode Security Considerations~~ ~~http://www.unicode.org/reports/tr36/~~
~~[UTR30]~~	~~UTR #39: Character Foldings~~ ~~http://unicode.org/reports/tr30/~~
~~[UTS18]~~	~~UTS #18: Unicode Regular Expressions~~ ~~http://www.unicode.org/reports/tr18/~~
[Unicode]	The Unicode Consortium. The Unicode Standard, Version 5.2. by: The Unicode Standard, Version 5.2 (Mountain View, CA: Th Consortium, 2009. ISBN 978-1-936213-00-9)
[Versions]	Versions of the Unicode Standard http://www.unicode.org/standard/versions/ *For information on version numbering, and citing and referen Unicode Standard, the Unicode Character Database, and Unic Technical Reports.*

Modifications

The following summarizes modifications from the previous revision of this document.

Revision 3:

- **Draft 3**
- Made modifications resulting from UTC discussion.

- Section 3.2 IDN Security Profiles for Identifiers, conformance clause C1, and the idnchars.txt data file have been removed.
- The subsection Data Files is now Section 7.
- **Draft 2**
- Added Table 0. Identifier_Modification_Key and text following, explaining the identifer restrictions. Especially see the caveat about use of the data.
- Added pointer for interactive review.
- Added more review notes asking for feedback.
- Changed to NFD instead of NFKD, with relevant mappings moved into the data file.
- **Draft 1**
- Proposed update of the document.
- Revised the confusable data to add data extracted from a comparison of font data from windows and mac.
  - Data was generated for characters sharing the same outline in some font on that system.
  - Those were then reviewed to remove errors due to bad font mappings.
  - Additional mappings were also added, such as "rn"≅"m".
- The recommended characters in identifiers were updated based on UAX 31, with the following labels:
  - UAX31 Table 4 Candidate Exclusions
  - UAX31 Table 5 Limited Use
  - *Note: more work needs to be done to update the recommended characters.*
- The IICore information was removed, since it is not a good guide to usage.

Revision 2:

- Removed the "input" and "lenient" tables
- Minor editing and clarifications

Revision 1:

- Created from Appendix A, B, and D from [UTR36].
- Created *Section 6. Development Process* based on document L2/06-055.
- Removed DITTO Mark, added intentional mappings
- Added 5.0 scripts to *removals*: Balinese, Cuneiform, Phoenician, Phags_Pa
- Revised table formats
- Added the intentional mappings, plus a pointer to source data

---