Technical Reports

L2/25-192

Proposed Update Unicode® Technical Standard #46

UNICODE IDNA COMPATIBILITY PROCESSING

Version	17.0.0
Editors	Mark Davis (mark@unicode.org), Markus Scherer (markus.icu@gmail.com)
Date	2025-05-07
This Version	https://www.unicode.org/reports/tr46/tr46-34.html
Previous Version	https://www.unicode.org/reports/tr46/tr46-33.html
Latest Version	https://www.unicode.org/reports/tr46/
Latest Proposed Update	https://www.unicode.org/reports/tr46/proposed.html
Revision	34

Summary

Client software, such as browsers and emailers, faced a difficult transition from the version of international domain names approved in 2003 (IDNA2003), to the revision approved in 2010 (IDNA2008). The specification in this document has been providing a mechanism that minimizes the impact of this transition for client software, allowing client software to access domains that are valid under either system.

The specification provides two main features: One is a comprehensive mapping to support current user expectations for casing and other variants of domain names. Such a mapping is allowed by IDNA2008. The second is a compatibility mechanism that supports the existing domain names that were allowed under IDNA2003. This second feature was intended to improve client behavior during the transition period.

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*This is a **draft** document which may be updated, replaced, or superseded by other documents at any time.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 the 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].

Contents

---

1 Introduction

One of the great strengths of domain names is universality. The URL **https://Apple.com** goes to Apple's website from anywhere in the world, using any browser. The email address **mark@unicode.org** can be used to send email to an editor of this specification from anywhere in the world, using any emailer.

Initially, domain names were restricted to ASCII characters. This was a significant burden on people using other characters. Suppose, for example, that the domain name system had been invented by Greeks, and one could only use Greek characters in URLs. Rather than **apple.com**, one would have to write something like **αππλε.κομ**. An English speaker would not only have to be acquainted with Greek characters, but would also have to pick those Greek letters that would correspond to the desired English letters. One would have to guess at the spelling of particular words, because there are not exact matches between scripts.

Most of the world's population faced this situation until recently, because their languages use non-ASCII characters. A system was introduced in 2003 for internationalized domain names (IDN). This system is called *Internationalizing Domain Names for Applications*, or IDNA2003 for short. This mechanism supports IDNs by means of a client software transformation into a format known as

Punycode. A revision of IDNA was approved in 2010 (IDNA2008). This revision has a number of incompatibilities with IDNA2003.

The incompatibilities forced implementers of client software, such as browsers and emailers, to face difficult choices during the transition period as registries shifted from IDNA2003 to IDNA2008. This document specifies a mechanism that has minimized the impact of this transition for client software, allowing client software to access domains that are valid under either system.

The specification provides two main features. The first is a comprehensive mapping to support current user expectations for casing and other variants of domain names. Such a mapping is allowed by IDNA2008. The second feature is a compatibility mechanism that supports the existing domain names that were allowed under IDNA2003. This second feature was intended to improve client behavior during the transition period. Although the transition is complete and transitional processing is now deprecated, the mapping and processing defined in this specification, and the validation based on the latest version of Unicode, remain valuable and in widespread use.

This specification contains both normative and informative material. Only the conformance clauses and the text that they directly or indirectly reference are considered normative.

1.1 IDNA2003

The series of RFCs collectively known as IDNA2003 [IDNA2003] allows domain names to contain non-ASCII Unicode characters, which includes not only the characters needed for Latin-script languages other than English (such as Å, Ħ, or Þ), but also different scripts, such as Greek, Cyrillic, Tamil, or Korean. An internationalized domain name such as **Bücher.de** can then be used in an "internationalized" URL, called an IRI, such as **http://Bücher.de#titel**.

The IDNA mechanism for allowing non-ASCII Unicode characters in domain names involves applying the following steps to each label in the domain name that contains Unicode characters:

1. Transforming (mapping) a Unicode string to remove case and other variant differences.
2. Checking the resulting string for validity, according to certain rules.
3. Transforming the Unicode characters into a DNS-compatible ASCII string using a specialized encoding called *Punycode* [RFC3492].

For example, typing the IRI **http://Bücher.de** into the address bar of any modern browser goes to a corresponding site, even though the "ü" is not an ASCII character. This works because the IDN in that IRI resolves to the Punycode string which is actually stored by the DNS for that site. Similarly, when a browser interprets a web page containing a link such as <a href="http://Bücher.de">, the appropriate site is reached. (In this document, phrases such as "a browser interprets" refer to domain names parsed out of IRIs entered in an address bar *as well as* to those contained in links internal to HTML text.)

In the case of IDN **Bücher.de**, the Punycode value actually used for the domain names on the wire is **xn--bcher-kva.de**. The Punycode version is also typically transformed back into Unicode form for display. The resulting display string will be a string which has already been mapped according to the IDNA2003 rules. This example results in a display string for the IRI that has been casefolded to lowercase:

http://Bücher.de → **http://xn--bcher-kva.de** → **http://bücher.de**

A major limitation of IDNA2003 is its restriction to the repertoire of characters in Unicode 3.2, which means that some modern languages are excluded or not fully supported. Furthermore, within the constraints of IDNA2003, there is no simple way to extend the repertoire. IDNA2003 also does not make it clear to users of registries exactly which string they are registering for a domain name (between **Bücher.de** and **bücher.de**, for example).

1.2 IDNA2008

In early 2010, a new version of IDNA was approved. Like IDNA2003, this version consists of a collection of RFCs and is called IDNA2008 [IDNA2008]. IDNA2008 is intended to solve the major

problems in IDNA2003. It extends the valid repertoire of characters in domain names, and establishes an automatic process for updating to future versions of the Unicode Standard. Furthermore, it defines the concept of a valid domain name clearly, so that registrants understand exactly what domain name string is being registered.

Processing in IDNA2008 is identical to IDNA2003 for many common domain names. Both IDNA2003 and IDNA2008 transform a Unicode domain name in an IRI (like **http://öbb.at**) to the Punycode version (like **http://xn--bb-eka.at**). However, IDNA2008 does not maintain strict backward compatibility with IDNA2003. The main differences are:

- **Additions.** Some IDNs are invalid in IDNA2003, but valid in IDNA2008.
- **Subtractions.** Some IDNs are valid in IDNA2003, but invalid in IDNA2008.
- **Deviations.** Some IDNs are valid in both, but resolve to different destinations.

1.3 Transition Considerations

The differences between IDNA2008 and IDNA2003 may cause interoperability and security problems. They affect extremely common characters, such as all uppercase characters, all halfwidth or fullwidth characters (commonly used in Japan, China, and Korea), and certain other characters like the German *eszett* (U+00DF ß LATIN SMALL LETTER SHARP S) and Greek *final sigma* (U+03C2 ς GREEK SMALL LETTER FINAL SIGMA). Note that for the “deviation” characters like the sharp s and the sigma, the industry has fully transitioned to IDNA2008 behavior, and transitional processing has been deprecated.

1.3.1 Mapping

IDNA2003 requires a mapping phase, which maps **ÖBB.at** to **öbb.at**, for example. Mapping typically involves mapping uppercase characters to their lowercase pairs, but it also involves other types of mappings between equivalent characters, such as mapping halfwidth *katakana* characters to normal *katakana* characters in Japanese. The mapping phase in IDNA2003 was included to match the case insensitivity of ASCII domain names. Users are accustomed to having both **CNN.com** and **cnn.com** work identically. They expect domain names with accents to have the same casing behavior, so that **ÖBB.at** is the same as **öbb.at**. There are variations similar to case differences in other scripts. The IDNA2003 mapping is based on data specified in the Unicode Standard, Version 3.2; this mapping was later formalized as the Unicode property [NFKC_Casefold].

Note that case-folding generates a stable form of a string that erases functional case-differences. It is *not* the same as lowercasing. In particular, the lowercase Cherokee characters added in Unicode Version 8.0 are case-folded to their uppercase counterparts.

IDNA2008 does not require a mapping phase, but does *permit* one (called "Local Mapping" or "Custom Mapping"). For more information on the permitted mappings, see the *Protocol* document of [IDNA2008], *Section 4.2, Permitted Character and Label Validation* and *Section 5.2, Conversion to Unicode*.

The UTS #46 specification defines a mapping consistent with the normative requirements of the IDNA2008 protocol, and which is mostly compatible with IDNA2003. For client software, this provides behavior that is the most consistent with user expectations about the handling of domain names with existing data—namely, that domain names are case-insensitive.

1.3.2 Deviations

There are a few situations where the use of IDNA2008 without compatibility mapping will result in the resolution of IDNs to different IP addresses from in IDNA2003, unless the registry or registrant takes special action. This affects a very small number of characters, but because these characters are very common in particular languages, a significant number of domain names in those languages are affected. This set of characters is referred to as "Deviations" and is shown in *Table 1, Deviation Characters*, illustrated in the context of IRIs.

Table 1. Deviation Characters

Char	Example	IDNA2003 Result	IDNA2008 Result
ß 00DF	href="**http://faß.de"**	**http://fass.de** → **http://fass.de**	**http://faß.de** → **http://xn--fa-hia.de**
ς 03C2	href="**http://βόλος.com**"	**http://βόλοσ.com** → **http://xn--nxasmq6b.com**	**http://βόλος.com** → **http://xn--nxasmm1c.com**
ZWJ 200D	href="**http://ශ්‍රී.com**"	**http://ශ්‍රී.com** → **http://xn--10cl1a0b.com**	**http://ශ්‍රී.com** → **http://xn--10cl1a0b660p.com**
ZWNJ 200C	href="**http://نامه‌ای.com**"	**http://نامهای.com** → **http://xn--mgba3gch31f.com**	**http://نامه‌ای.com** → **http://xn--mgba3gch31f060k.com**

For more information on the rationale for the occurrence of these Deviations in IDNA2008, see the [IDN FAQ].

The differences in interpretation of Deviation characters result in potential for security exploits. Consider a scenario involving **http://www.sparkasse-gießen.de**, a German IRI containing an IDN for "Gießen Savings and Loan".

1. Alice's browser supports IDNA2003. Under those rules, **http://www.sparkasse-gießen.de** is mapped to **http://www.sparkasse-giessen.de**, which leads to a site with the IP address **01.23.45.67**.
2. She visits her friend Bob, and checks her bank statement on his browser. His browser supports IDNA2008. Under those rules, **http://www.sparkasse-gießen.de** is also valid, but converts to a different Punycode domain name in **http://www.xn--sparkasse-gieen-2ib.de**. This can lead to a different site with the IP address **101.123.145.167**, a spoof site.

Alice ends up at the phishing site, supplies her bank password, and her money is stolen. While the .DE registar (DENIC) might have a policy about bundling all of the variants of ß together (so that they all have the same owner) it is not required of registries. It is unlikely that all registries will have and enforce such a bundling policy in all such cases.

There are two Deviations of particular concern. IDNA2008 allows the joiner characters (ZWJ and ZWNJ) in labels. By contrast, these are removed by the mapping in IDNA2003. When used in the intended contexts in particular scripts, the joiner characters produce a noticeable change in displayed text. However, when used between any other characters in those scripts, or in any other scripts, they are invisible. For example, when used between the Latin characters "a" and "b" there is no visible different: the sequence "a<ZWJ>b" looks just like "ab".

Because of the visual confusability introduced by the joiner characters, IDNA2008 provides a special category for them called CONTEXTJ, and only permits CONTEXTJ characters in limited contexts: certain sequences of Arabic or Indic characters. However, applications that perform IDNA2008 lookup are not required to check for these contexts, so overall security is dependent on registries having correct implementations. Moreover, the IDNA2008 context restrictions do not catch most cases where distinct domain names have visually confusable appearances because of ZWJ and ZWNJ.

Note that for these “deviations”, the industry has fully transitioned to IDNA2008 behavior, and transitional processing has been deprecated.

2 Unicode IDNA Compatibility Processing

To satisfy user expectations for mapping, and (originally) provide compatibility with IDNA2003, this document specifies a mapping for use with IDNA2008. In addition, this document provides a Unicode algorithm for a standardized processing that allows conformant implementations to minimize the security and interoperability problems caused by the differences between IDNA2003 and IDNA2008. This Unicode IDNA Compatibility Processing is structured according to IDNA2003 principles, but extends those principles to Unicode 5.2 and later. It also incorporates the repertoire extensions provided by IDNA2008.

UTS #46 can be used purely as a preprocessing (local mapping) for IDNA2008 by claiming conformance specifically to *Conformance Clause C3*.

By using this Compatibility Processing, a domain name such as **ÖBB.at** will be mapped to the valid domain name **öbb.at**, thus matching user expectation for case behavior in domain names. For transitional use, the Compatibility Processing also allows domain names containing symbols and punctuation that were valid in IDNA2003, such as **√.com** (which has an associated web page). Such domain names containing symbols will gradually disappear as registries shift to IDNA2008.

Implementations may also restrict or flag (in a UI) domain names that include symbols and punctuation. For more information, see *Unicode Technical Report # 36, Unicode Security Considerations* [UTR36].

Using the Unicode IDNA Compatibility Processing to transform an IDN into a form suitable for DNS lookup is similar to the tactic of "try IDNA2008 then try IDNA2003". However, this approach avoids a potentially problematic dual lookup. It allows browsers and other clients, such as search engines, to have a single processing step, without the burden of maintaining two different implementations and multiple tables. It accounts for a number of edge cases that would cause problems, and provides a stable definition with predictable results.

The Unicode IDNA Compatibility Processing also provides alternate mappings for the Deviation characters. This facilitates the transition from IDNA2003 to IDNA2008. It is up to the registries to decide how to handle the transition, for example, by either bundling or blocking the Deviation characters that they support. **In practice, for the deviation characters, the transition is complete. All major implementations have switched to nontransitional processing of the four deviation characters.**

The term "registries" includes far more than top-level registries, such as for **.de** or **.com**. For example, **.blogspot.com** has more domain names registered than most top-level registries. There may be different policies in place for a registry and any of its subregistries. Thus millions of registries need to be considered in a transition strategy, not just hundreds.

In lookup software, transitions may be fine-grained: for example, it may be possible to transition to IDNA2008 rules regarding Deviations for **.subdomain.com** at a given point but not for **.com**, or vice versa. If **.tld** bundles or blocks the Deviation characters, then clients could transition Deviations for **.tld**, but not for (say) **.subdomain.tld**. Moreover, client software with a UI, such as the address bar in a browser, could provide more options for the transition. A full discussion of such transition strategies is outside of the scope of this document.

During the interim, authors of documents, such as HTML documents, can unambiguously refer to the IDNA2008 interpretation of characters by explicitly using the Punycode form of the domain name label.

There are two slightly different compatibility mechanisms for domain names during a transition and afterward. UTS #46 therefore specifies two specific types of processing: Transitional Processing (*Conformance Clause C1*) and Nontransitional Processing (*Conformance Clause C2*). The only difference between them is the handling of the four Deviation characters.

Summarized briefly, UTS #46 builds upon IDNA2008 in three areas:

- **Mapping.** The UTS #46 mapping is used to maintain maximal compatibility and meet user expectations. It is conformant to IDNA2008, which allows for mapping input.
- **Symbols and Punctuation.** UTS #46 supports processing of symbols and punctuation. Registries which implement IDNA2008 will simply refuse the DNS lookups of IDNs with symbols.
- **Deviations (deprecated).** UTS #46 provides two ways of handling these to support a transition. Transitional Processing (deprecated) had been recommended to be used immediately before a DNS lookup in the circumstances where the registry does not guarantee a strategy of bundling or blocking. Nontransitional Processing, which is fully compatible with IDNA2008, should be used in all cases.

For a demonstration of differences between IDNA2003, IDNA2008, and the Unicode IDNA Compatibility Processing, see the [DemoIDN].

UTS #46 does not change any of the terms defined in IDNA2008, such as A-Label or U-Label.

Neither the Unicode IDNA Compatibility Processing nor IDNA2008 address security problems associated with confusables (the so-called "**paypal.com**" problem). IDNA2008 disallows certain symbols and punctuation characters that can be used for spoofing, such as spoofs of the slash character ("/"). However, these are an extremely small fraction of the confusable characters used for spoofing. Moreover, confusable characters themselves account for a small proportion of phishing problems: most are cases like "secure-wellsfargo.com". For more information, see [Bortzmeyer] and the [IDN FAQ]. It is strongly recommended that *Unicode Technical Report #36, Unicode Security Considerations* [UTR36] and *Unicode Technical Standard #39, Unicode Security Mechanisms* [UTS39] be consulted for information on dealing with confusables, both for client software and registries. In particular, [UTS39] provides information that can be used to drastically reduce the number of confusables when dealing with international domain names, much beyond what IDNA2008 does. See also the [DemoConf].

2.1 Display of Internationalized Domain Names

IDNA2003 applications customarily display the processed string to the user. This improves security by reducing the opportunity for visual confusability. Thus, for example, the URL **http://google.com** (with a capital I in place of the L) is revealed as **http://googie.com**.

2.2 Registries

This specification is primarily targeted at applications doing lookup of IDNs. There is, however, one strong recommendation for registries: *do not allow the registration of labels that are invalid according to Nontransitional Processing, and do use bundling or blocking for labels containing confusable characters*.

These tactics can be described as follows:

- **Bundling**: If two or more labels are different, but confusable, and more than one is registered, the registrant for each must be the same.
- **Blocking**: If two or more labels are different, but confusable, allow the registration of only one, and block the others. Registries that do not allow any Deviation characters at all count as **blocking**.

Note: Some implementations outside Unicode use different terminology for these strategies. In particular, in the ICANN Root Zone Label Generation Rules [RZLGR5], the term *allocatable variant* of X is used for labels that can be bundled with X, and the term *blocked variant* is used for a mutually exclusive label.

The label that is actually registered and inserted into a registry has always been processed. For example, **xn--bcher-kva** corresponds to **bücher**. However, it may be useful for a registry to also ask for "unprocessed" labels, such as **Bücher**, as part of the registration process, so that they are aware of the registrant's intent. However, such unprocessed labels must be handled carefully:

- Storing the unprocessed label as the sequence of characters that the registrant really wanted to apply for.
- Processing the unprocessed label, and displaying the processed label to the registrant for confirmation.
- Proceeding with the regular registration process using *only* the processed label.

2.3 Notation

Sets of code points are defined using properties and the syntax of *Unicode Technical Standard #18, Unicode Regular Expressions* [UTS18]. For example, the set of combining marks is represented by the syntax `\p{gc=M}` . Additionally, the "+" indicates the addition of elements to a set, for clarity.

In this document, a *label* is a substring of a domain name. That substring is bounded on both sides by either the start or the end of the string, or any of the following characters, called *label-separators*:

1. U+002E (.) FULL STOP
2. U+FF0E (．) FULLWIDTH FULL STOP
3. U+3002 (。) IDEOGRAPHIC FULL STOP
4. U+FF61 (｡) HALFWIDTH IDEOGRAPHIC FULL STOP

Many people use the terms "domain names" and "host names" interchangeably. This document follows [RFC3490] in use of the term "domain name".

A *Bidi domain name* is a domain name containing at least one character with Bidi_Class R, AL, or AN. See [IDNA2008] RFC 5893, Section 1.4.

3 Conformance

The requirements for conformance on implementations of the **Unicode IDNA Compatibility Processing** algorithm are stated in the following clauses. An implementation can claim conformance to any or all of these clauses independently.

C1 (deprecated). *Given a version of Unicode and a Unicode String, a conformant implementation of **Transitional Processing** shall replicate the results given by applying the Transitional Processing algorithm specified by Section 4, Processing*.

C2. *Given a version of Unicode and a Unicode String, a conformant implementation of **Nontransitional Processing** shall replicate the results given by applying the Nontransitional Processing algorithm specified by Section 4, Processing*.

C3. *Given a version of Unicode and a Unicode String, a conformant implementation of **Preprocessing for IDNA2008** shall replicate the results specified by Section 4.4, Preprocessing for IDNA2008*.

These specifications are *logical* ones, designed to be straightforward to describe. An actual implementation is free to use different methods as long the result is the same as that specified by the logical algorithm.

Any conformant implementation may also have *tighter* validity criteria than those imposed by *Section 4.1, Validity Criteria*. For example, an application could disallow or warn of domain name labels with certain characteristics, such as:

- labels with certain combinations of scripts (Safari)
- labels with characters outside of the user's specified languages (IE)
- labels with certain confusable characters (Firefox)
- labels that are detected by the Google Safe Browsing API [SafeBrowsing]
- labels that do not meet the validity requirements of IDNA2008
- labels produced by toUnicode that would not meet the label validity requirements if toASCII were performed.
- labels containing characters which are not contained in the General Security Profile for Identifiers from *Unicode Technical Standard #39, Unicode Security Mechanisms* [UTS39]
- labels that do not satisfy *Restriction Level 4, Moderately Restrictive* from *Unicode Technical Standard #39, Unicode Security Mechanisms* [UTS39]

For more information, see *Unicode Technical Report #36, Unicode Security Considerations* [UTR36] and *Unicode Technical Standard #39, Unicode Security Mechanisms* [UTS39].

3.1 STD3 Rules

IDNA2003 provides for a flag, **UseSTD3ASCIIRules**, that allows for implementations to choose whether or not to abide by the rules in [STD3]. These rules exclude ASCII characters outside the set consisting of A-Z, a-z, 0-9, and U+002D (-) HYPHEN-MINUS. For example, some browsers also allow characters such as U+005F (_) LOW LINE *(underbar)* in domain names, and thus use a custom set of valid ASCII characters when checking the *Validity Criteria*.

4 Processing

The input to Unicode IDNA Compatibility Processing is a prospective *domain_name* string expressed in Unicode, and a choice of Transitional or Nontransitional Processing. The domain name consists of a sequence of labels with dot separators, such as "Bücher.de". For more information about the composition of a URL, see Section 3.5 of [STD13].

Main Processing Steps

The following steps, performed in order, successively alter the input *domain_name* string and then output it as a converted Unicode string, plus a flag to indicate whether there was an error. Even if an error occurs, the conversion of the string is performed as much as is possible.

Input

- A prospective *domain_name* expressed as a sequence of Unicode code points
- A boolean flag: *UseSTD3ASCIIRules*
- A boolean flag: *CheckHyphens*
- A boolean flag: *CheckBidi*
- A boolean flag: *CheckJoiners*
- A boolean flag: *Transitional_Processing* (deprecated)
- A boolean flag: *IgnoreInvalidPunycode*

Processing

1. Map. For each code point in the *domain_name* string, look up the Status value in *Section 5, IDNA Mapping Table*, and take the following actions:
    - **disallowed**: Leave the code point unchanged in the string. Note: The Convert/Validate step below checks for disallowed characters, *after* mapping and normalization.
    - **ignored**: Remove the code point from the string. This is equivalent to mapping the code point to an empty string.
    - **mapped**: If *Transitional_Processing* (deprecated) and the code point is U+1E9E capital sharp s (ß), then replace the code point in the string by “ss”. Otherwise:
      Replace the code point in the string by the value for the mapping in *Section 5, IDNA Mapping Table*.
    - **deviation**:
        - If *Transitional_Processing* (deprecated), replace the code point in the string by the value for the mapping in *Section 5, IDNA Mapping Table*.
        - Otherwise, leave the code point unchanged in the string.
    - **valid**: Leave the code point unchanged in the string.
2. Normalize. Normalize the *domain_name* string to Unicode Normalization Form C.
3. Break. Break the string into labels at U+002E (.) FULL STOP.
4. Convert/Validate. For each label in the *domain_name* string:
    - If the label starts with “xn--”:
        1. If the label contains any non-ASCII code point (i.e., a code point greater than U+007F), record that there was an error, and continue with the next label.
        2. Attempt to convert the rest of the label to Unicode according to *Punycode* [RFC3492]. If that conversion fails **and** if not *IgnoreInvalidPunycode*, record that there was an error, and continue with the next label. Otherwise replace the original label in the string by the results of the conversion.
        3. If the label is empty, or if the label contains only ASCII code points, record that there was an error.
        4. Verify that the label meets the validity criteria in *Section 4.1, Validity Criteria* for Nontransitional Processing. If any of the validity criteria are not satisfied, record that there was an error.
    - If the label does not start with “xn--”:

- Verify that the label meets the validity criteria in *Section 4.1, Validity Criteria* for the input Processing choice (Transitional or Nontransitional). If any of the validity criteria are not satisfied, record that there was an error.

Any input *domain_name* string that does not record an error has been successfully processed according to this specification. Conversely, if an input *domain_name* string causes an error, then the processing of the input *domain_name* string fails. Determining what to do with error input is up to the caller, and not in the scope of this document. The processing is idempotent—reapplying the processing to the output will make no further changes. For examples, see *Table 2, Examples of Transitional Processing*.

Implementations may make further modifications to the resulting Unicode string when showing it to the user. For example, it is recommended that disallowed characters be replaced by a U+FFFD to make them visible to the user. Similarly, labels that fail processing during step 4 may be marked by the insertion of a U+FFFD or other visual device.

With either Transitional or Nontransitional Processing, sources already in Punycode are validated without mapping. In particular, Punycode containing Deviation characters, such as href="**xn--fu-hia.de**" (for fuß.de) is not remapped. This provides a mechanism allowing explicit use of Deviation characters even during a transition period.

4.1 Validity Criteria

Each of the following criteria must be satisfied for a non-empty label:

1. The label must be in Unicode Normalization Form NFC.
2. If *CheckHyphens*, the label must not contain a U+002D HYPHEN-MINUS character in both the third and fourth positions.
3. If *CheckHyphens*, the label must neither begin nor end with a U+002D HYPHEN-MINUS character.
4. If not *CheckHyphens*, the label must not begin with “xn--”.
5. The label must not contain a U+002E (.) FULL STOP.
6. The label must not begin with a combining mark, that is: General_Category=Mark.
7. Each code point in the label must only have certain Status values according to *Section 5, IDNA Mapping Table*:
    1. For Transitional Processing (deprecated), each value must be **valid**.
    2. For Nontransitional Processing, each value must be either **valid** or **deviation**.
    3. In addition, if **UseSTD3ASCIIRules=true** and the code point is an ASCII code point (U+0000..U+007F), then it must be a lowercase letter (a-z), a digit (0-9), or a hyphen-minus (U+002D). (Note: This excludes uppercase ASCII A-Z which are **mapped** in UTS #46 and **disallowed** in IDNA2008.)
8. If *CheckJoiners*, the label must satisify the **ContextJ rules** from *Appendix A,* in *The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)* [IDNA2008].
9. If *CheckBidi*, and if the domain name is a *Bidi domain name*, then the label must satisfy all six of the numbered conditions in [IDNA2008] RFC 5893, Section 2.

The first 6 criteria are from [IDNA2008], except for the fourth criterion. Criterion #2 in particular is meant to allow for future label extensions beyond just xn--, such as for future versions of IDNA. Some implementations appear to consider such extentions unlikely, and allow labels such as "r3---sn-apo3qvuoxuxbt-j5pe".

Any particular application *may* have tighter validity criteria, as discussed in *Section 3, Conformance*.

4.1.1 UseSTD3ASCIIRules

Starting with Unicode 16.0, **UseSTD3ASCIIRules=true** is handled only in the Validity Criteria. An implementation may choose to allow additional ASCII characters but should always consider ASCII lowercase letters, digits, and the hyphen-minus (`[\u002Da-z0-9]`) as **valid**.

Note: ASCII characters may have resulted from a mapping: for example, a U+005F (_) LOW LINE *(underbar)* may have originally been a U+FF3F (_) FULLWIDTH LOW LINE.

4.1.2 Right-to-Left Scripts

In addition, the label should meet the requirements for right-to-left characters specified in the Right-to-Left Scripts document of [IDNA2008], and for the CONTEXTJ requirements in the Protocol document of [IDNA2008]. It is strongly recommended that *Unicode Technical Report #36, Unicode Security Considerations* [UTR36] and *Unicode Technical Standard #39, Unicode Security Mechanisms* [UTS39] be consulted for information on dealing with confusables, and for characters that should be excluded from identifiers. Note that the recommended exclusions are a superset of those in [IDNA2008].

4.2 ToASCII

The operation corresponding to ToASCII of [RFC3490] is defined by the following steps:

Input

- A prospective *domain_name* expressed as a sequence of Unicode code points
- A boolean flag: *CheckHyphens*
- A boolean flag: *CheckBidi*
- A boolean flag: *CheckJoiners*
- A boolean flag: *UseSTD3ASCIIRules*
- A boolean flag: *Transitional_Processing* (deprecated)
- A boolean flag: *VerifyDnsLength*
- A boolean flag: *IgnoreInvalidPunycode*

Processing

1. To the input *domain_name*, apply the **Processing Steps** in *Section 4, Processing*, using the input boolean flags *Transitional_Processing*, *CheckHyphens*, *CheckBidi*, *CheckJoiners*, and *UseSTD3ASCIIRules*. This may record an error.
2. Break the result into labels at U+002E FULL STOP.
3. Convert each label with non-ASCII characters into Punycode [RFC3492], and prefix by “xn--”. This may record an error.
4. If the *VerifyDnsLength* flag is true, then verify DNS length restrictions. This may record an error. For more information, see [STD13] and [STD3].
    1. The length of the domain name, excluding the root label and its dot, is from 1 to 253.
    2. The length of each label is from 1 to 63.
        - Note: Technically, a complete domain name ends with an empty label for the DNS root (see [STD13] [RFC1034] section 3). This empty label, and the trailing dot, is almost always omitted.
        - When *VerifyDnsLength* is false, the empty root label is passed through.
        - When *VerifyDnsLength* is true, the empty root label is disallowed. This corresponds to the syntax in [RFC1034] section 3.5 Preferred name syntax which also defines the label length restrictions.
5. If an error was recorded in steps 1-4, then the operation has failed and a failure value is returned. No DNS lookup should be done.
6. Otherwise join the labels using U+002E FULL STOP as a separator, and return the result.

Implementations are advised to apply additional tests to these labels, such as those described in *Unicode Technical Report #36, Unicode Security Considerations* [UTR36] and *Unicode Technical Standard #39, Unicode Security Mechanisms* [UTS39], and take appropriate actions. For example, a label with mixed scripts or confusables may be called out in the UI. Note that the use of Punycode to signal problems may be counter-productive, as described in [UTR36].

4.3 ToUnicode

The operation corresponding to ToUnicode of [RFC3490] is defined by the following steps:

Input

- A prospective *domain_name* expressed as a sequence of Unicode code points
- A boolean flag: *CheckHyphens*
- A boolean flag: *CheckBidi*
- A boolean flag: *CheckJoiners*
- A boolean flag: *UseSTD3ASCIIRules*
- A boolean flag: *Transitional_Processing* (deprecated)
- A boolean flag: *IgnoreInvalidPunycode*

Processing

1. To the input *domain_name*, apply the **Processing Steps** in *Section 4, Processing*, using the input boolean flags *Transitional_Processing*, *CheckHyphens*, *CheckBidi*, *CheckJoiners*, and *UseSTD3ASCIIRules*. This may record an error.
2. Like [RFC3490], this will always produce a converted Unicode string. Unlike ToASCII of [RFC3490], this always signals whether or not there was an error.

Implementations are advised to apply additional tests to these labels, such as those described in *Unicode Technical Report #36, Unicode Security Considerations* [UTR36] and *Unicode Technical Standard #39, Unicode Security Mechanisms* [UTS39], and take appropriate actions. For example, a label with mixed scripts or confusables may be called out in the UI. Note that the use of Punycode to signal problems may be counter-productive, as described in [UTR36].

4.4 Preprocessing for IDNA2008

The table specified in *Section 5, IDNA Mapping Table* may also be used for a pure preprocessing step for IDNA2008, mapping a Unicode string for input directly to the algorithm specified in IDNA2008.

Preprocessing for IDNA2008 is specified as follows:

> Apply the *Section 4.3, ToUnicode* processing to the Unicode string.

Note that this preprocessing allows some characters that are invalid according to IDNA2008. However, the IDNA2008 processing will catch those characters. For example, a Unicode string containing a character listed as DISALLOWED in IDNA2008, such as U+2665 (♥) BLACK HEART SUIT, will pass the preprocessing step without an error, but subsequent application of the IDNA2008 processing will fail with an error, indicating that the string is not a valid IDN according to IDNA2008.

4.5 Implementation Notes

A number of optimizations can be applied to the Unicode IDNA Compatibility Processing. These optimizations can improve performance, reduce table size, make use of existing NFKC transform mechanisms, and so on. For example:

- There is an NFC check in *Section 4.1, Validity Criteria*. However, it only needs to be applied to labels that were converted from Punycode into Unicode in Step 3.
- A simple way to do much of the validity checking in *Section 4.1, Validity Criteria* is to reapply Steps 1 and 2, and verify that the result does not change.
- Because the four label separators are all mapped to U+002E (.) FULL STOP by Step 1, the parsing of labels in Steps 3 and 4 only need to detect U+002E (.) FULL STOP, and not the other label separators defined in IDNA [RFC3490].

Note that the input *domain_name* string for the Unicode IDNA Compatibility Processing must have had all escaped Unicode code points converted to Unicode code points. For example, `U+5341` (十) CJK

UNIFIED IDEOGRAPH-5341 could have been escaped as any of the following:

- 十 an HTML numeric character reference (NCR)
- \u5341 a Javascript escapes
- %E5%8D%81 a URI/IRI %-escape

Examples are shown in *Table 2, Examples of Processing:*

Table 2. Examples of Processing

Input	Map	Normalize	Convert	Validate	Comment
Bloß.de	bloss.de	=	*n/a*	***ok***	**Transitional (deprecated):** maps uppercase and sharp s
	bloß.de	=	*n/a*	***ok***	**Nontransitional:** maps uppercase
BLOß.de	bloß.de	=	*n/a*	***ok***	Maps uppercase
xn--blo-7ka.de	=	=	bloß.de	***ok***	Punycode is not mapped, so ß never changes (whether transitional or not).
u¨.com	=	ü.com	*n/a*	***ok***	Normalize changes *u* + *umlaut* to *ü*
xn--tda.com	=	=	ü.com	***ok***	Punycode **xn--tda** changes to *ü*
xn--u-ccb.com	=	=	u¨.com	***error***	Punycode is not mapped, but *is* validated. Because *u* + *umlaut* is not NFC, it fails.
a 1.com	***error***	***error***	***error***	***error***	The character " 1." is **disallowed**, because it would produce a dot when mapped.
xn--a-ecp.ru	xn--a-ecp.ru	=	a 1..ru	***error***	Punycode **xn--a-ecp** = a 1., which fails validation.
xn--0.pt	xn--0.pt	=	***error***	***error***	Punycode **xn--0** is invalid.
日本語。ＪＰ	日本語.jp	=	*n/a*	***ok***	Fullwidth characters are remapped, including 。
.us	=	=	*n/a*	***ok***	Post-Unicode 3.2 characters are allowed.

5 IDNA Mapping Table

For each code point in Unicode, the IDNA Mapping Table provides one of the following Status values:

- **valid**: the code point is valid, and not modified.
- **ignored**: the code point is removed: this is equivalent to mapping the code point to an empty string.
- **mapped**: the code point is replaced in the string by the value for the mapping.
- **deviation**: the code point is either mapped or valid, depending on whether the processing is transitional or not.
- **disallowed:** the code point is not allowed.

If this Status value is **mapped** or **deviation**, the table also supplies a mapping value for that code point.

A table is provided for each version of Unicode starting with Unicode 5.1~~, in versioned directories~~ under [IDNA-Table]. Each table for a version of the Unicode Standard will always be backward

compatible with previous versions of the table: only characters with the Status value **disallowed** may change in Status or Mapping value, with the following exception:

- As part of the deprecation of transitional processing, the following exceptional change has been made in Unicode 15.1:
  - Before Unicode 15.1, U+1E9E capital sharp s (ß) was unconditionally **mapped** to “ss”, consistent with transitional processing which maps U+00DF small sharp s (ß) also to “ss”.
  - Since Unicode 15.1, *when using nontransitional processing*, capital sharp s is **mapped** to small sharp s, which is treated as **valid** under nontransitional processing. This is the new Mapping value in the table.
    When using *transitional* processing (deprecated), U+1E9E capital sharp s (ß) continues to be **mapped** to “ss”, just like the **deviation** mapping for U+00DF small sharp s (ß). This is handled during processing.

Unlike the IDNA2008 table, this table is designed to be applied to the entire domain name, not just to individual labels. That design provides for the IDNA2003 handling of label separators. In particular, the table is constructed to forbid problematic characters such as U+2488 (1**.**) DIGIT ONE FULL STOP, whose decompositions contain a "dot".

The Unicode IDNA Compatibility Processing is based on the Unicode character mapping property [NFKC_Casefold]. *Section 6, Mapping Table Derivation* describes the derivation of these tables. Like derived properties in the Unicode Character Database, the description of the derivation is informative. Only the data in IDNA Mapping Table is normative for the application of this specification.

The files use a semicolon-delimited format similar to those in the Unicode Character Database [UAX44]. The field values are listed in *Table 2b, Data File Fields*:

Table 2b. Data File Fields

Num	Field	Description
0	Code point(s)	Hex value or range of values.
1	Status	**valid**, **ignored**, **mapped**, **deviation**, or **disallowed**
2	Mapping	Hex value(s). Only present if the Status is **ignored**, **mapped**, or **deviation**.
3	IDNA2008 Status	There are two values: **NV8** and **XV8**. **NV8** is only present if the Status is **valid** but the character is excluded by IDNA2008 from all domain names for all versions of Unicode. **XV8** is present when the character is excluded by IDNA2008 for the ***current*** version of Unicode. These are not normative values.

Example:

```
0000..002C    ; valid      ;      ; NV8    # 1.1  <control-0000>..COMMA
002D..002E    ; valid                     # 1.1  HYPHEN-MINUS..FULL STOP
002F          ; valid      ;      ; NV8    # 1.1  SOLIDUS
0030..0039    ; valid                     # 1.1  DIGIT ZERO..DIGIT NINE
003A..0040    ; valid      ;      ; NV8    # 1.1  COLON..COMMERCIAL AT
0041          ; mapped     ; 0061         # 1.1  LATIN CAPITAL LETTER A
...
0080..009F    ; disallowed                # 1.1  <control-0080>..<control-009F>
...
00A1..00A7    ; valid      ;      ; NV8    # 1.1  INVERTED EXCLAMATION MARK..SECTION SIGN
...
00AD          ; ignored                   # 1.1  SOFT HYPHEN
...
00DF          ; deviation  ; 0073 0073    # 1.1  LATIN SMALL LETTER SHARP S
...
19DA          ; valid      ;      ; XV8    # 5.2  NEW TAI LUE THAM DIGIT ONE
...
```

6 Mapping Table Derivation

The following describes the derivation of the mapping table. This description has nothing to do with the actual mapping of labels in *Section 4, Processing*. Instead, this section describes the derivation of the table in Section 5, IDNA Mapping Table. That table is then normatively used for mapping in *Section 4, Processing*.

The derivation is described as a series of steps. Step 1 defines a base mapping; Steps 2, 3, and 4 define three sets of characters. Step 5 will modify the base mapping or the sets of characters as needed to maintain backward compatiblity. The mapping and sets are all used in Step 6 to produce the mapping and Status values for the table. Step 7 removes characters whose mappings contain characters that are not valid. Each numbered step may have substeps: for example, Step 1 consists of Steps 1.1 through 1.2.

If a Unicode property changes in a future version in a way that would affect backward compatibility, a corresponding clause will be added to Step 5 to maintain compatibility. For more information on compatibility, see *Section 5, IDNA Mapping Table*.

Step 1: Define a base mapping

This step specifies a *base mapping*, which is a mapping from each Unicode code point to sequences of zero or more code points. The value resulting from mapping a particular code point C is called the *base mapping value of* C. The base mapping value for C may be identical to C.

1. Map the following exceptional characters:
    a. Map label separator characters to U+002E (.) FULL STOP:
        - U+FF0E (．) FULLWIDTH FULL STOP
        - U+3002 (。) IDEOGRAPHIC FULL STOP
        - U+FF61 (｡) HALFWIDTH IDEOGRAPHIC FULL STOP
    b. Map all Bidi_Control characters to themselves
    c. Map U+1E9E (ß) LATIN CAPITAL LETTER SHARP S to U+00DF (ß) LATIN SMALL LETTER SHARP S
2. Map each *other* character to its NFKC_Casefold value [NFKC_Casefold].

Unicode 6.3 adds Bidi_Control characters that were not present in Unicode 3.2. To preserve the intent of IDNA2003 in disallowing Bidi_Control characters rather than just ignoring them, Step 1.1.b was added. This step causes Step 6.3 to disallow all Bidi_Control characters.

Step 1.1.b only affects 5 new characters added in Unicode 6.3. It would also impact any new Bidi_Control characters in future versions of the standard.

Step 1.1.c (added in Unicode 15.1) maps the capital sharp s (ß) to the small sharp s (ß) rather than to ss because all major implementations have adopted nontransitional processing, which does not map ß to ss as in NFKC_Casefold.

Step 2: Specify the base valid set

The base valid set is defined by the sequential list of additions and subtractions in *Table 3, Base Valid Set*. This definition is based on the principles of IDNA2003. When applied to the repertoire of Unicode 3.2 characters, this produces a set which is closely aligned with IDNA2003.

Table 3. Base Valid Set

Formal Set Notation	Description
`\P{Changes_When_NFKC_Casefolded}`	Start with characters that are equal to their [NFKC_Casefold] value. This criterion excludes uppercase letters, for example, as well as characters that are unstable under NFKC normalization, and default ignorable code points.

	Note that according to Perl/Java syntax, \P means the inverse of \p, so these are the characters that *do not* change when individually mapped according to [NFKC_Casefold].
`+ \u00DF`	Add LATIN SMALL LETTER SHARP S (ß).
`- \p{c} - \p{z}`	Remove Unassigned, Controls, Private Use, Format, Surrogate, and Whitespace.

`- \p{IDS_Unary_Operator}`
`- \p{IDS_Binary_Operator}`
`- \p{IDS_Trinary_Operator}`
	Remove ideographic description characters.

`+ \p{ascii} - [\u002E]`	Add all ASCII except for "."

Step 3: Specify the base exclusion set

The base exclusion set consists of the following code points:

- U+FFFC OBJECT REPLACEMENT CHARACTER
- U+FFFD REPLACEMENT CHARACTER
- U+E0001..U+E007F Tag characters (includes some unassigned code points)

Step 4: Specify the deviation set

This is the set of characters that deviate between IDNA2003 and IDNA2008.

- U+200C ZERO WIDTH NON-JOINER
- U+200D ZERO WIDTH JOINER
- U+00DF (ß) LATIN SMALL LETTER SHARP S
- U+03C2 (ς) GREEK SMALL LETTER FINAL SIGMA

Step 5: Specify changes for backward compatibility

This set is currently empty. Adjustments to the above sets or base mapping will be made in this section if the steps would cause an already existing character to change Status or mapping under a future version of Unicode, so that backward compatibility is maintained.

Step 6: Produce the initial Status and Mapping values

For each code point:

1. If the code point is in the **deviation** set
    - the Status is **deviation** and the mapping value is the base mapping value for that code point.
2. Otherwise, if the code point is in the base exclusion set or is unassigned
    - the Status is **disallowed** and there is no mapping value in the table.
3. Otherwise, if the code point is not a label separator *and* some code point in its base mapping value is not in the base valid set
    - the Status is **disallowed** and there is no mapping value in the table.
4. Otherwise, if the base mapping value is an empty string
    - the Status is **ignored** and there is no mapping value in the table.
5. Otherwise, if the base mapping value is the same as the code point
    - the Status is **valid** and there is no mapping value in the table.
6. Otherwise,
    - the Status is **mapped** and the mapping value is the base mapping value for that code point.

Step 7: Produce the final Status and Mapping values

After processing all code points in previous steps:

1. Iterate through the set of characters with a Status of **mapped**. Any whose mapping values are not wholly in the union of the **valid** set and the **deviation** set, make **disallowed**.
2. Recursively apply these actions until there are no more Status changes.

For example, for Unicode 15.1, the set of characters set to disallowed in Step 7 consists of the following:

- U+FE12 (﹒) PRESENTATION FORM FOR VERTICAL IDEOGRAPHIC FULL STOP

Note: Characters such as U+2488 (⒈) DIGIT ONE FULL STOP are disallowed by Step 6.3.

7 IDNA Comparison

Until Unicode 15.1, this section provided a detailed comparison of the differences between IDNA2003, UTS #46, and IDNA2008. Due to the end of the transition period, starting with Unicode 16.0, the Mapping Table Derivation no longer takes IDNA2003 mappings into account; therefore that information is no longer applicable.

Unicode provides a derived property file matching IDNA2008. Compared with IDNA2008, UTS #46 mostly adds mappings and considers punctuation and symbols valid. For more information see *Section 2, Unicode IDNA Compatibility Processing* and consult the IDNA Mapping Table.

8 Conformance Testing

A conformance testing file (IdnaTestV2.txt) is provided for each version of Unicode starting with Unicode 6.0~~, in versioned directories~~ under [IDNA-Table]. It only provides test cases for **UseSTD3ASCIIRules=true**.

8.1 Format

The test file is UTF-8, with certain characters escaped using the \uXXXX or \x{XXXX} convention for readability. The details are in the header of the test file.

8.2 Testing Conformance

To test for conformance to UTS #46, an implementation will perform the toUnicode, toAsciiN, and toAsciiT operations on the source string, then verify the resulting strings and relevant Status values. The details are in the header of the test file.

Implementations may be more strict than the default settings for UTS46. In particular, an implementation conformant to IDNA2008 would disallow the input for lines marked with NV8. Implementations need only record that there is an error: they need not reproduce the precise Status codes (after removing any ignored Status values).

8.3 Migration

16.0

The test file for version 16.0 corrects some mistakes in the generation of status values and makes some improvements.

- Starting with Unicode 16.0, the test format uses `""` to mean the empty string. This is in contrast to a blank field value, which continues to have a different meaning. For example:

```
"";  ; [X4_2]; ; [A4_1, A4_2]; ;  #
\u200C; ; [C1]; xn--0ug; ; ""; [A4_1, A4_2] #
```

See the header of the test data file for details.

- One or more new source strings are ill-formed, containing an unpaired surrogate, so that status value A3 is covered by test cases.
- The status values V4-V6 have been renumbered to V5-V7, in order to match the insertion of validity criterion 4 in Unicode 15.1.
- Status value U1 is set instead of V7 for ASCII characters other than lowercase letters (a-z), digits (0-9), or hyphen-minus (U+002D), as had been suggested by the file header comments.
- The file header comments about several status values have been corrected or clarified.

11.0

The test format and file name changed in Version 11.0 so that it could express a variety of different combinations of input options that people needed. The new format allows the testing implementation to test for precisely the results of its combination of supported flags, by filtering out Status codes that correspond to an unsupported input flag. The value XV8 was also removed, since it was not very useful in practice.

The following illustrate the differences between the old and new format. The set of examples is not exhaustive, but shows how there is more information available for the same examples.

Sample lines in test data format prior to 11.0:

```
T;  Faß.de;     faß.de;     fass.de
N;  Faß.de;     faß.de;     xn--fa-hia.de
B;  Bücher.de;  bücher.de;  xn--bcher-kva.de
B;  à\u05D0;    [B5 B6];    [B5 B6]
B;  a。。b;      [A4_2];     [A4_2]
```

Sample lines in test data format since 11.0:

```
Faß.de;     faß.de;     [];       xn--fa-hia.de;     ;  fass.de;
Bücher.de;  bücher.de;  [];       xn--bcher-kva.de;  ;  ;
à\u05D0;    àא;         [B5 B6];  xn--0ca24w;        ;  ;
a。。b;      a..b;       [A4_2];   a..b;              ;  ;
```

9 IDNA Derived Property

To facilitate comparison between versions of the Unicode Character Database and to highlight the implications for the addition of new characters and changes of character properties, the Unicode Technical Committee has prepared a collection of IDNA Derived Property data files. Since Unicode 17.0, the version-specific Idna2008.txt data file is posted in the versioned [IDNA-Table] directory. Before Unicode 17.0, these data files ~~are permanently~~ were posted at [IDNA-Derived].

For each version of the Unicode Standard starting with Unicode 6.1.0, the value of the enumerated IDNA2008_Category property is calculated and listed explicitly in a separate data file. This property matches the "IDNA Derived Property" as defined in RFC 5892 (see [IDNA2008]). The explicit listing is provided as a convenience for implementers. It is the result of performing the exact calculations defined in RFC 5892 concurrent with the release of each version of the Unicode Character Database.

RFC 5892 gives a list of code points for which the derivation is overridden by exceptional values. All known exceptions are applied when a data file is created, but exceptions added in future updates of the IDNA protocol are not applied retroactively.

The format of these IDNA Derived Property data files is modeled closely on that specified in Appendix B.1 of RFC 5892, except that the comment section of each line is not truncated at column 72. For example, excerpted from RFC 5892:

```
007B..00B6  ; DISALLOWED  # LEFT CURLY BRACKET..PILCROW SIGN
00B7        ; CONTEXTO    # MIDDLE DOT
00B8..00DE  ; DISALLOWED  # CEDILLA..LATIN CAPITAL LETTER THORN
00DF..00F6  ; PVALID      # LATIN SMALL LETTER SHARP S..LATIN SMALL LETT
```

Compare the same ranges excerpted from the data files:

```
007B..00B6  ; DISALLOWED  # LEFT CURLY BRACKET..PILCROW SIGN
00B7        ; CONTEXTO    # MIDDLE DOT
00B8..00DE  ; DISALLOWED  # CEDILLA..LATIN CAPITAL LETTER THORN
00DF..00F6  ; PVALID      # LATIN SMALL LETTER SHARP S..LATIN SMALL LETTER O WITH DIAERESIS
```

This close match in format is designed to simplify scripted comparison between these IDNA Derived Property data files posted at unicode.org and other existing calculated listings based on RFC 5892 that have been posted at IANA or elsewhere.

Acknowledgments

Mark Davis and Michel Suignard authored the bulk of the original text of this document, under direction from the Unicode Technical Committee. For their contributions of ideas or text to this specification, the editors thank Julie Allen, Matitiahu Allouche, Peter Constable, Craig Cummings, Martin Dürst, Peter Edberg, Asmus Freytag, Deborah Goldsmith, Laurentiu Iancu, Gervase Markham, Simon Montagu, Lisa Moore, Eric Muller, Simon Sapin, Murray Sargent, Markus Scherer, Jungshik Shin, Henri Sivonen, Shawn Steele, Erik van der Poel, Chris Weber, and Ken Whistler. The specification builds upon [IDNA2008], developed in the IETF Idna-update working group, especially contributions from Matitiahu Allouche, Harald Alvestrand, Vint Cerf, Martin J. Dürst, Lisa Dusseault, Patrik Fältström, Paul Hoffman, Cary Karp, John Klensin, and Peter Resnick, and also upon [IDNA2003], authored by Marc Blanchet, Adam Costello, Patrik Fältström, and Paul Hoffman.

References

[Bortzmeyer] http://www.bortzmeyer.org/idn-et-phishing.html

The most interesting studies cited there (originally from Mike Beltzner of **Mozilla**) are:

- *Decision Strategies and Susceptibility to Phishing* by Downs, Holbrook & Cranor
- *Why Phishing Works* by Dhamija, Tygar & Hearst
- *Do Security Toolbars Actually Prevent Phishing Attacks* by Wu, Miller & Garfinkel
- *Phishing Tips and Techniques* by Gutmann.

[DemoConf] https://util.unicode.org/UnicodeJsps/confusables.jsp

[DemoIDN] https://util.unicode.org/UnicodeJsps/idna.jsp

[DemoIDNChars] https://util.unicode.org/UnicodeJsps/list-unicodeset.jsp?a=\p{age%3D3.2}-\p{cn}-\p{cs}-\p{co}&abb=on&g=uts46+idna+idna2008

[IDNA2003] The IDNA2003 specification is defined by a cluster of IETF RFCs:

- IDNA [RFC3490]
- Nameprep [RFC3491]
- Punycode [RFC3492]
- Stringprep [RFC3454].

[IDNA2008] The IDNA2008 specification is defined by a cluster of IETF RFCs:

- Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework
  https://www.rfc-editor.org/info/rfc5890
- Internationalized Domain Names in Applications (IDNA) Protocol
  https://www.rfc-editor.org/info/rfc5891

- The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)
  https://www.rfc-editor.org/info/rfc5892
- Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)
  https://www.rfc-editor.org/info/rfc5893

There is also an informative document:

- Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale
  https://www.rfc-editor.org/info/rfc5894

[IDNA-Derived] https://www.unicode.org/Public/idna2008derived

[IDNA-Table] ~~https://www.unicode.org/Public/idna~~ https://www.unicode.org/Public/17.0.0/idna (Before Unicode 17.0: https://www.unicode.org/Public/idna)

[IDN-FAQ] https://www.unicode.org/faq/idn.html

[NFKC_Casefold] The Unicode property specified in [UAX44], and defined by the data in DerivedNormalizationProps.txt (search for "NFKC_Casefold").

[RFC1034] P. Mockapetris "Domain names - concepts and facilities", RFC 1034, November 1987.
https://www.rfc-editor.org/info/rfc1034

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

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

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

[RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)", RFC 3492, March 2003.
https://www.rfc-editor.org/info/rfc3492

[RZLGR5] Integration Panel, "Root Zone Label Generation Rules — LGR-5", 22 May 2022.
https://www.icann.org/sites/default/files/lgr/rz-lgr-5-overview-26may22-en.pdf

[SafeBrowsing] http://code.google.com/apis/safebrowsing/

[Stability] Unicode Consortium Stability Policies
https://www.unicode.org/policies/stability_policy.html

[STD3] Braden, R., "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, and "Requirements for Internet Hosts -- Application and Support", STD 3, RFC 1123, October 1989.
https://www.rfc-editor.org/info/std3

[STD13] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034 and "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
https://www.rfc-editor.org/info/std13

[UAX44] UAX #44:*Unicode Character Database*
https://www.unicode.org/reports/tr44/

[Unicode] The Unicode Standard
For the latest version, see:
https://www.unicode.org/versions/latest/

[UTR36] UTR #36: *Unicode Security Considerations*
https://www.unicode.org/reports/tr36/

[UTS18] UTS #18: *Unicode Regular Expressions*
https://www.unicode.org/reports/tr18/

[UTS39] UTS #39: *Unicode Security Mechanisms*
https://www.unicode.org/reports/tr39/

Modifications

The following summarizes modifications from the previous published version of this document.

Revision 34

- **Proposed Update** for Unicode 17.0.0.
- Updated data file references to point to new locations for Version 17.0. ([182-A11])

~~Revision 33~~

- ~~**Reissued** for Unicode 16.0.0.~~
- ~~The handling of *UseSTD3ASCIIRules* has been simplified. Conditional data involving disallowed_STD3_* Status values has been replaced with simple checking for a subset of ASCII characters in the Validity Criteria. This simplifies the data format and data lookup, makes standard **UseSTD3ASCIIRules=true** handling consistent with custom UseSTD3ASCIIRules, and avoids unnecessarily disallowing certain labels that contain disallowed_STD3_mapped characters but which do not contain non-LDH ASCII characters when the mappings are applied.~~
  ~~*Behavior for **UseSTD3ASCIIRules=false** is unchanged.*~~
  ~~Examples for **UseSTD3ASCIIRules=true** behavior changes:~~
  - ~~Example for a label which continues to fail the Validity Criteria despite the change in Processing:~~
    ~~In Unicode 15.1, input label "⑷" was unchanged in Processing and failed the Validity Criteria. (U+2477 disallowed_STD3_mapped was resolved to disallowed, and its mapping was not applied.)~~
    ~~In Unicode 16.0, "⑷" is Mapped to "(4)", which still fails the Validity Criteria, except if a custom set of valid ASCII characters is used that includes the parentheses.~~
  - ~~Example for a label which newly passes the Validity Criteria due to the change in Processing:~~
    ~~In Unicode 15.1, input label "\uFF1D\u0338" (fullwidth equals + combining solidus overlay) was unchanged in Processing and failed the Validity Criteria. (U+FF1D disallowed_STD3_mapped was resolved to disallowed, and its mapping was not applied.)~~
    ~~In Unicode 16.0, "\uFF1D\u0338" is Mapped to "\u003D\u0338" and Normalized to "\u2260" (not equal to), which is valid.~~
- ~~In *Section 4, Processing*, if the label starts with "xn--", and the conversion from Punycode yields either an empty label or an all-ASCII label, then an error is now recorded, consistent with IDNA2008.~~
- ~~Changed *Section 6 Mapping Table Derivation, Table 3. Base Valid Set*, replacing \p{Block=Ideographic_Description_Characters} and \u31EF with the equivalent [\p{IDS_Unary_Operator}\p{IDS_Binary_Operator}\p{IDS_Trinary_Operator}].~~
- ~~Changed *Section 6 Mapping Table Derivation, Step 3: Specify the base exclusion set*, to define a small, fixed base exclusion set. Previously, the base exclusion set had been derived from differences between IDNA2003 data and UTS #46 principles.~~
  ~~Changes in the Unicode 15.1 version led to unexpected edge cases in processing. At the same time, transitional processing was deprecated.~~
  ~~The UTC concluded that it was no longer necessary to disallow characters on the basis of differences from IDNA2003, and decided to simplify the definition of the base exclusion set. As a result, a number of characters that were disallowed before are now ignored, mapped, or (in the one case of U+1806 MONGOLIAN TODO SOFT HYPHEN) valid. In xn-- Punycode labels, characters with Status ignored and mapped are still not valid. The recent edge cases and~~

~~processing complications are no longer present.~~
~~For details see the proposal in document L2/24-064 item 6.2.~~

- ~~Removed the content of *Section 7, IDNA Comparison*, which is no longer applicable.~~
- ~~Noted in *Section 8.3, Migration* new syntax in the test file: "" means an empty string. There are also other test data corrections and improvements.~~

Modifications for previous versions are listed in those respective versions.
