Lightweight Directory Access Protocol (LDAP):
Internationalized String Preparation

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

The previous Lightweight Directory Access Protocol (LDAP) technical specifications did not precisely define how character string matching is to be performed. This led to a number of usability and interoperability problems. This document defines string preparation algorithms for character-based matching rules defined for use in LDAP.

1. Introduction

1.1. Background

A Lightweight Directory Access Protocol (LDAP) [RFC4510] matching rule [RFC4517] defines an algorithm for determining whether a presented value matches an attribute value in accordance with the criteria defined for the rule. The proposition may be evaluated to True, False, or Undefined.

True      - the attribute contains a matching value,
False     - the attribute contains no matching value,
Undefined - it cannot be determined whether the attribute contains a matching value.
For instance, the caseIgnoreMatch matching rule may be used to compare whether the commonName attribute contains a particular value without regard for case and insignificant spaces.

1.2. X.500 String Matching Rules

"X.520: Selected attribute types" [X.520] provides (among other things) value syntaxes and matching rules for comparing values commonly used in the directory [X.500]. These specifications are inadequate for strings composed of Unicode [Unicode] characters.

The caseIgnoreMatch matching rule [X.520], for example, is simply defined as being a case-insensitive comparison where insignificant spaces are ignored. For printableString, there is only one space character and case mapping is bijective, hence this definition is sufficient. However, for Unicode string types such as universalString, this is not sufficient. For example, a case-insensitive matching implementation that folded lowercase characters to uppercase would yield different results than an implementation that used uppercase to lowercase folding. Or one implementation may view space as referring to only SPACE (U+0020), a second implementation may view any character with the space separator (Zs) property as a space, and another implementation may view any character with the whitespace (WS) category as a space.

The lack of precise specification for character string matching has led to significant interoperability problems. When used in certificate chain validation, security vulnerabilities can arise. To address these problems, this document defines precise algorithms for preparing character strings for matching.

1.3. Relationship to "stringprep"

The character string preparation algorithms described in this document are based upon the "stringprep" approach [RFC3454]. In "stringprep", presented and stored values are first prepared for comparison so that a character-by-character comparison yields the "correct" result.

The approach used here is a refinement of the "stringprep" [RFC3454] approach. Each algorithm involves two additional preparation steps.

a) Prior to applying the Unicode string preparation steps outlined in "stringprep", the string is transcoded to Unicode.

b) After applying the Unicode string preparation steps outlined in "stringprep", the string is modified to appropriately handle characters insignificant to the matching rule.
Hence, preparation of character strings for X.500 [X.500] matching
[X.501] involves the following steps:

1) Transcode
2) Map
3) Normalize
4) Prohibit
5) Check Bidi (Bidirectional)
6) Insignificant Character Handling

These steps are described in Section 2.

It is noted that while various tables of Unicode characters included
or referenced by this specification are derived from Unicode
[Unicode] data, these tables are to be considered definitive for the
purpose of implementing this specification.

1.4. Relationship to the LDAP Technical Specification

This document is an integral part of the LDAP technical specification
[RFC4510], which obsoletes the previously defined LDAP technical
specification [RFC3377] in its entirety.

This document details new LDAP internationalized character string
preparation algorithms used by [RFC4517] and possible other technical
specifications defining LDAP syntaxes and/or matching rules.

1.5. Relationship to X.500

LDAP is defined [RFC4510] in X.500 terms as an X.500 access
mechanism. As such, there is a strong desire for alignment between
LDAP and X.500 syntax and semantics. The character string
preparation algorithms described in this document are based upon
"Internationalized String Matching Rules for X.500" [XMATCH] proposal
to ITU/ISO Joint Study Group 2.

1.6. Conventions and Terms

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in BCP 14 [RFC2119].

Character names in this document use the notation for code points
and names from the Unicode Standard [Unicode]. For example, the letter
"a" may be represented as either <U+0061> or <LATIN SMALL LETTER A>.
In the lists of mappings and the prohibited characters, the "U+" is
left off to make the lists easier to read. The comments for character ranges are shown in square brackets (such as "[CONTROL CHARACTERS]") and do not come from the standard.

Note: a glossary of terms used in Unicode can be found in [Glossary]. Information on the Unicode character encoding model can be found in [CharModel].

The term "combining mark", as used in this specification, refers to any Unicode [Unicode] code point that has a mark property (Mn, Mc, Me). Appendix A provides a definitive list of combining marks.

2. String Preparation

The following six-step process SHALL be applied to each presented and attribute value in preparation for character string matching rule evaluation.

1) Transcode
2) Map
3) Normalize
4) Prohibit
5) Check bidi
6) Insignificant Character Handling

Failure in any step causes the assertion to evaluate to Undefined.

The character repertoire of this process is Unicode 3.2 [Unicode].

Note that this six-step process specification is intended to describe expected matching behavior. Implementations are free to use alternative processes so long as the matching rule evaluation behavior provided is consistent with the behavior described by this specification.

2.1. Transcode

Each non-Unicode string value is transcoded to Unicode.

PrintableString [X.680] values are transcoded directly to Unicode.

UniversalString, UTF8String, and bmpString [X.680] values need not be transcoded as they are Unicode-based strings (in the case of bmpString, a subset of Unicode).

TeletexString [X.680] values are transcoded to Unicode. As there is no standard for mapping TeletexString values to Unicode, the mapping is left a local matter.
For these and other reasons, use of TeletexString is NOT RECOMMENDED.

The output is the transcodded string.

2.2. Map

SOFT HYPHEN (U+00AD) and MONGOLIAN TODO SOFT HYPHEN (U+1806) code points are mapped to nothing. COMBINING GRAPHEME JOINER (U+034F) and VARIATION SELECTORS (U+180B-180D, FF00-FFEF) code points are also mapped to nothing. The OBJECT REPLACEMENT CHARACTER (U+FFFC) is mapped to nothing.

CHARACTER TABULATION (U+0009), LINE FEED (LF) (U+000A), LINE TABULATION (U+000B), FORM FEED (FF) (U+000C), CARRIAGE RETURN (CR) (U+000D), and NEXT LINE (NEL) (U+0085) are mapped to SPACE (U+0020).

All other control code (e.g., Cc) points or code points with a control function (e.g., Cf) are mapped to nothing. The following is a complete list of these code points: U+0000-0008, 000E-001F, 007F-0084, 0086-009F, 006D, 070F, 180E, 200C-200F, 202A-202E, 2060-2063, 206A-206F, FEFF, FFF9-FFFB, 1D173-1D17A, E0001, E0020-E007F.

ZERO WIDTH SPACE (U+200B) is mapped to nothing. All other code points with Separator (space, line, or paragraph) property (e.g., Zs, Zl, or Zp) are mapped to SPACE (U+0020). The following is a complete list of these code points: U+0020, 00A0, 1680, 2000-200A, 2028-2029, 202F, 205F, 3000.

For case ignore, numeric, and stored prefix string matching rules, characters are case folded per B.2 of [RFC3454].

The output is the mapped string.

2.3. Normalize

The input string is to be normalized to Unicode Form KC (compatibility composed) as described in [UAX15]. The output is the normalized string.

2.4. Prohibit

All Unassigned code points are prohibited. Unassigned code points are listed in Table A.1 of [RFC3454].

Characters that, per Section 5.8 of [RFC3454], change display properties or are deprecated are prohibited. These characters are listed in Table C.8 of [RFC3454].
Private Use code points are prohibited. These characters are listed in Table C.3 of [RFC3454].

All non-character code points are prohibited. These code points are listed in Table C.4 of [RFC3454].

Surrogate codes are prohibited. These characters are listed in Table C.5 of [RFC3454].

The REPLACEMENT CHARACTER (U+FFFD) code point is prohibited.

The step fails if the input string contains any prohibited code point. Otherwise, the output is the input string.

2.5. Check bidi

Bidirectional characters are ignored.

2.6. Insignificant Character Handling

In this step, the string is modified to ensure proper handling of characters insignificant to the matching rule. This modification differs from matching rule to matching rule.

Section 2.6.1 applies to case ignore and exact string matching.
Section 2.6.2 applies to numericString matching.
Section 2.6.3 applies to telephoneNumber matching.

2.6.1. Insignificant Space Handling

For the purposes of this section, a space is defined to be the SPACE (U+0020) code point followed by no combining marks.

NOTE - The previous steps ensure that the string cannot contain any code points in the separator class, other than SPACE (U+0020).

For input strings that are attribute values or non-substring assertion values: If the input string contains no non-space character, then the output is exactly two SPACES. Otherwise (the input string contains at least one non-space character), the string is modified such that the string starts with exactly one space character, ends with exactly one SPACE character, and any inner (non-empty) sequence of space characters is replaced with exactly two SPACE characters. For instance, the input strings "foo<SPACE>bar<SPACE><SPACE>", result in the output "<SPACE>foo<SPACE>bar<SPACE>".
For input strings that are substring assertion values: If the string being prepared contains no non-space characters, then the output string is exactly one SPACE. Otherwise, the following steps are taken:

- If the input string is an initial substring, it is modified to start with exactly one SPACE character;
- If the input string is an initial or an any substring that ends in one or more space characters, it is modified to end with exactly one SPACE character;
- If the input string is an any or a final substring that starts in one or more space characters, it is modified to start with exactly one SPACE character; and
- If the input string is a final substring, it is modified to end with exactly one SPACE character.

For instance, for the input string "foo<SPACE>bar<SPACE><SPACE>" as an initial substring, the output would be "<SPACE>foo<SPACE>bar<SPACE>". As an any or final substring, the same input would result in "foo<SPACE>bar<SPACE>".

Appendix B discusses the rationale for the behavior.

2.6.2. numericString Insignificant Character Handling

For the purposes of this section, a space is defined to be the SPACE (U+0020) code point followed by no combining marks.

All spaces are regarded as insignificant and are to be removed.

For example, removal of spaces from the Form KC string:
"<SPACE><SPACE>123<SPACE><SPACE>456<SPACE><SPACE>"
would result in the output string:
"123456"
and the Form KC string:
"<SPACE><SPACE><SPACE>"
would result in the output string:
"" (an empty string).

2.6.3. telephoneNumber Insignificant Character Handling

For the purposes of this section, a hyphen is defined to be a HYPHEN-MINUS (U+002D), ARmenian Hyphen (U+058A), Hyphen (U+2010), Non-Breaking Hyphen (U+2011), Minus Sign (U+2212), Small Hyphen-Minus (U+FE63), or Fullwidth Hyphen-Minus (U+FF0D) code point followed by
no combining marks and a space is defined to be the SPACE (U+0020) code point followed by no combining marks.

All hyphens and spaces are considered insignificant and are to be removed.

For example, removal of hyphens and spaces from the Form KC string: "<SPACE><HYPHEN>123<SPACE><SPACE>456<SPACE><HYPHEN>" would result in the output string: "123456"

and the Form KC string: "<HYPHEN><HYPHEN><HYPHEN>" would result in the (empty) output string: "".

3. Security Considerations

"Preparation of Internationalized Strings ("stringprep")" [RFC3454] security considerations generally apply to the algorithms described here.

4. Acknowledgements

The approach used in this document is based upon design principles and algorithms described in "Preparation of Internationalized Strings ("stringprep")" [RFC3454] by Paul Hoffman and Marc Blanchet. Some additional guidance was drawn from Unicode Technical Standards, Technical Reports, and Notes.

This document is a product of the IETF LDAP Revision (LDAPBIS) Working Group.

5. References

5.1. Normative References


5.2. Informative References


Appendix A. Combining Marks

This appendix is normative.

This table was derived from Unicode data files; it lists all code points with the Mn, Mc, or Me properties. This table is to be considered definitive for the purposes of implementation of this specification.

0300-034F 0360-036F 0483-0486 0488-0489 0591-05A1
05A3-05B9 05BB-05BC 05BF 05C1-05C2 05C4 064B-0655 0670
0666-066D 06E7-06E8 06EA-06EE 0711 0730-074A
07A6-07BD 0901-0903 093C 093E-094F 0951-0954 0962-0963
0A02 0A3C 0A3D-0A42 0A47-0A48 0A4B-0A4D
0A70-0A71 0A81-0A83 0ABC 0ABE-0AC5 0AC7-0AC9 0ACB-0ACD
0B01-0B03 0B3C 0B3E-0B43 0B47-0B48 0B4B-0B4D 0B56-0B57
0B82 0B8B-0B8D 0B8E-0B8F 0B92-0B94 0B97-0BA0 0B9C-0BA3
0C82-0C83 0CBE-0CCF 0CC6-0CC8 0CCA-0CCD 0CDD-0CDE 0D01-0D03
0D05-0D09 0D0B-0D10 0D12-0D13 0D15-0D16 0D18-0D19
0D1B-0D1C 0D1E-0D1F 0D21-0D25 0D27-0D28 0D2A-0D2B
0D30-0D31 0D33-0D34 0D36-0D37 0D39-0D3A 0D3C-0D3D
0D3F-0D40 0D42-0D43 0D46-0D47 0D49-0D4A 0D4C-0D4D
0D50-0D51 0D53-0D54 0D56-0D57 0D59-0D5A 0D5C-0D5D
0D60-0D61 0D63-0D64 0D66-0D67 0D69-0D6A 0D6C-0D6D
0D70-0D71 0D73-0D74 0D76-0D77 0D79-0D7A 0D7C-0D7D
0D80-0D81 0D83-0D84 0D86-0D87 0D89-0D8A 0D8C-0D8D
0D90-0D91 0D93-0D94 0D96-0D97 0D99-0D9A 0D9C-0D9D
0DA0-0DA1 0DA3-0DA4 0DA6-0DA7 0DA9-0DAA 0DAB-0DAD
0DB0-0DB1 0DB3-0DB4 0DBC-0DBD 0DBF-0DC0 0DCC2-0DCC3
0DC5-0DC6 0DC8-0DC9 0DCA-0DCB 0DD0-0DD1 0DD3-0DD4
0DD6-0DD7 0DD9-0DDE 0DE0-0DE1 0DE3-0DE4 0DE6-0DE7
0DE9-0DEA 0DEC-0DEE 0DF0-0DF1 0DF3-0DF4 0DF6-0DF7
0DF9-0DFB 0FC6 102C-1032 1036-1039 1056-1059 1712-1714
1732-1734 1752-1753 1772-1773 17B4-17D3 180B-180D 18A9
20D0-20EA 302A-302F 3099-309A FB1E FE00-FE0F FE20-FE23
1D65-1D69 1D6D-1D72 1D7B-1D82 1D85-1D8B 1D1A8-1D1B3
1D1AA-1D1AD

Appendix B. Substrings Matching

This appendix is non-normative.

In the absence of substrings matching, the insignificant space handling for case ignore/exact matching could be simplified. Specifically, the handling could be to require that all sequences of one or more spaces be replaced with one space and, if the string contains non-space characters, removal of all leading spaces and trailing spaces.

In the presence of substrings matching, this simplified space handling would lead to unexpected and undesirable matching behavior. For instance:

1) (CN=foo\20*\20bar) would match the CN value "foobar";
2) (CN=*\20foobar\20*) would match "foobar", but
(CN=*\20*foobar*\20*) would not.

Note to readers not familiar with LDAP substrings matching: the LDAP
filter [RFC4515] assertion (CN=A*B*C) says to "match any value (of
the attribute CN) that begins with A, contains B after A, ends with C
where C is also after B."

The first case illustrates that this simplified space handling would
cause leading and trailing spaces in substrings of the string to be
regarded as insignificant. However, only leading and trailing (as
well as multiple consecutive spaces) of the string (as a whole) are
insignificant.

The second case illustrates that this simplified space handling would
cause sub-partitioning failures. That is, if a prepared any
substring matches a partition of the attribute value, then an
assertion constructed by subdividing that substring into multiple
substrings should also match.

In designing an appropriate approach for space handling for
substrings matching, one must study key aspects of X.500 case
exact/ignore matching. X.520 [X.520] says:

The [substrings] rule returns TRUE if there is a partitioning of
the attribute value (into portions) such that:

- the specified substrings (initial, any, final) match
different portions of the value in the order of the strings
sequence;
- initial, if present, matches the first portion of the value;
- final, if present, matches the last portion of the value;
- any, if present, matches some arbitrary portion of the
value.

That is, the substrings assertion (CN=foo\20*\20bar) matches the
attribute value "foo<SPACE><SPACE>bar" as the value can be
partitioned into the portions "foo<SPACE>" and "<SPACE>bar" meeting
the above requirements.
X.520 also says:

[T]he following spaces are regarded as not significant:

- leading spaces (i.e., those preceding the first character that is not a space);
- trailing spaces (i.e., those following the last character that is not a space);
- multiple consecutive spaces (these are taken as equivalent to a single space character).

This statement applies to the assertion values and attribute values as whole strings, and not individually to substrings of an assertion value. In particular, the statements should be taken to mean that if an assertion value and attribute value match without any consideration to insignificant characters, then that assertion value should also match any attribute value that differs only by inclusion nor removal of insignificant characters.

Hence the assertion (CN=foo\20*\20bar) matches "foo<SPACE><SPACE>bar" and "foo<SPACE>bar" as these values only differ from "foo<SPACE><SPACE>bar" by the inclusion or removal of insignificant spaces.

Astute readers of this text will also note that there are special cases where the specified space handling does not ignore spaces that could be considered insignificant. For instance, the assertion (CN=\20*\20*\20) does not match "<SPACE><SPACE><SPACE>" (insignificant spaces present in value) or " " (insignificant spaces not present in value). However, as these cases have no practical application that cannot be met by simple assertions, e.g., (cn=\20), and this minor anomaly can only be fully addressed by a preparation algorithm to be used in conjunction with character-by-character partitioning and matching, the anomaly is considered acceptable.

Author’s Address

Kurt D. Zeilenga
OpenLDAP Foundation

EMail: Kurt@OpenLDAP.org
Full Copyright Statement

Copyright (C) The Internet Society (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).