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 transcoded 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-FEOF) 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, 06DD, 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><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 [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.

<table>
<thead>
<tr>
<th>Code Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>0300-034F</td>
</tr>
<tr>
<td>0360-036F</td>
</tr>
<tr>
<td>0483-0486</td>
</tr>
<tr>
<td>0488-0489</td>
</tr>
<tr>
<td>0591-05A1</td>
</tr>
<tr>
<td>05A3-05B9</td>
</tr>
<tr>
<td>05BB-05BC</td>
</tr>
<tr>
<td>05BF 05C1-05C2</td>
</tr>
<tr>
<td>05C4 064B-0655</td>
</tr>
<tr>
<td>0670 06DE-06DC</td>
</tr>
<tr>
<td>06E7-06E8 06EA-06ED</td>
</tr>
<tr>
<td>0711 0730-074A</td>
</tr>
<tr>
<td>0751-0754 07A6-07B0</td>
</tr>
<tr>
<td>0901-0903 093C 0951-0954 0981-0983</td>
</tr>
<tr>
<td>0988-0989 09B3 09C7-09C8 09CB-09CD 09D7</td>
</tr>
<tr>
<td>09E2-09E3 0A02 0A3C 0A3E-0A42 0A47-0A48 0A4B-0A4D</td>
</tr>
<tr>
<td>0A70-0A71 0A81-0A83 0ABC 0ABC-0AC5 0AC7-0AC9 0ACB-0ACD</td>
</tr>
<tr>
<td>0BB1-0BB3 0B3C 0B3E-0B43 0B47-0B48 0B4B-0B4D 0B56-0B57</td>
</tr>
<tr>
<td>0B82 0B86-0B8C 0B8C-0B8E 0B8E-0B8F 0B8F-0B90 0B91-0B93 0B94-0B95</td>
</tr>
<tr>
<td>0B96-0B9C 0B9D-0B9F 0B9E-0B9F</td>
</tr>
<tr>
<td>0B9F-0BAC 0BBD-0BBF 0BC1-0BC4 0BC5-0BC7 0BC8-0BCD</td>
</tr>
<tr>
<td>0BCD-0BCF 0BCF-0C00 0C01-0C03 0C04-0C0C 0C0D-0C0F</td>
</tr>
<tr>
<td>0C0F-0C11 0C12-0C14 0C15-0C17 0C18-0C1A</td>
</tr>
<tr>
<td>0C1B-0C1D 0C1D-0C1F 0C20-0C22 0C22-0C24</td>
</tr>
<tr>
<td>0C24-0C26 0C26-0C28 0C28-0C2A 0C2A-0C2C 0C2C-0C2E</td>
</tr>
<tr>
<td>0C2E-0C30 0C30-0C32 0C32-0C34 0C34-0C36 0C36-0C38</td>
</tr>
<tr>
<td>0C38-0C3A 0C3A-0C3C 0C3C-0C3E 0C3E-0C40 0C40-0C42</td>
</tr>
<tr>
<td>0C42-0C44 0C44-0C46 0C46-0C48 0C48-0C4A 0C4A-0C4C</td>
</tr>
<tr>
<td>0C4C-0C4E 0C4E-0C50 0C50-0C52 0C52-0C54 0C54-0C56</td>
</tr>
<tr>
<td>0C56-0C58 0C58-0C5A 0C5A-0C5C 0C5C-0C5E 0C5E-0C60</td>
</tr>
<tr>
<td>0C60-0C62 0C62-0C64 0C64-0C66 0C66-0C68 0C68-0C6A</td>
</tr>
<tr>
<td>0C6A-0C6C 0C6C-0C6E 0C6E-0C70 0C70-0C72 0C72-0C74</td>
</tr>
<tr>
<td>0C74-0C76 0C76-0C78 0C78-0C7A 0C7A-0C7C 0C7C-0C7E</td>
</tr>
<tr>
<td>0C7E-0C80 0C80-0C82 0C82-0C84 0C84-0C86 0C86-0C88</td>
</tr>
<tr>
<td>0C88-0C8A 0C8A-0C8C 0C8C-0C8E 0C8E-0C90 0C90-0C92</td>
</tr>
<tr>
<td>0C92-0C94 0C94-0C96 0C96-0C98 0C98-0C9A 0C9A-0C9C</td>
</tr>
<tr>
<td>0C9C-0C9E 0C9E-0CA0 0CA0-0CA2 0CA2-0CA4 0CA4-0CA6</td>
</tr>
<tr>
<td>0CA6-0CA8 0CA8-0CAA 0CAA-0CAC 0CAC-0CAD 0CAD-0CCE</td>
</tr>
<tr>
<td>0CCE-0CCF 0CCF-0CD1 0CD1-0CD3 0CD3-0CD5 0CD5-0CD7</td>
</tr>
<tr>
<td>0CD7-0CD9 0CD9-0CDB 0CDB-0CDD 0CDD-0CDE 0CDE-0CF0</td>
</tr>
<tr>
<td>0CF0-0CF3 0CF3-0CF6 0CF6-0CF9 0CF9-0CFA 0CFA-0CFD</td>
</tr>
<tr>
<td>0CFD-0CFF 0D00-0D03 0D03-0D06 0D06-0D09 0D09-0D0C</td>
</tr>
<tr>
<td>0D0C-0D0F 0D0F-0D12 0D12-0D15 0D15-0D18 0D18-0D1B</td>
</tr>
<tr>
<td>0D1B-0D1E AD00-AD03 AD03-AD06 AD06-AD09 AD09-AD0C</td>
</tr>
<tr>
<td>AD0C-AD0F AD10-AD13 AD13-AD16 AD16-AD19 AD19-AD1C</td>
</tr>
<tr>
<td>AD1C-AD1F AD20-AD23 AD23-AD26 AD26-AD29 AD29-AD2C</td>
</tr>
<tr>
<td>AD2C-AD2F AD30-AD33 AD33-AD36 AD36-AD39 AD39-AD3C</td>
</tr>
<tr>
<td>AD3C-AD3F AD40-AD43 AD43-AD46 AD46-AD49 AD49-AD4C</td>
</tr>
<tr>
<td>AD4C-AD4F AD50-AD53 AD53-AD56 AD56-AD59 AD59-AD5C</td>
</tr>
<tr>
<td>AD5C-AD5F AD60-AD63 AD63-AD66 AD66-AD69 AD69-AD6C</td>
</tr>
<tr>
<td>AD6C-AD6F AD70-AD73 AD73-AD76 AD76-AD79 AD79-AD7C</td>
</tr>
<tr>
<td>AD7C-AD7F AD80-AD83 AD83-AD86 AD86-AD89 AD89-AD8C</td>
</tr>
<tr>
<td>AD8C-AD8F AD90-AD93 AD93-AD96 AD96-AD99 AD99-AD9C</td>
</tr>
<tr>
<td>AD9C-AD9F ADAC-ADAD ADAD-AE00 AE00-AE03 AE03-AE06</td>
</tr>
</tbody>
</table>

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*foo\*bar\*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><SPACE><SPACE>bar" and "foo<SPACE>bar" as these values only differ from "foo<SPACE><SPACE><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
certain 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

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).