XEP-0106: JID Escaping

Joe Hildebrand
mailto:jhildebr@cisco.com
xmpp:hildjj@jabber.org

Peter Saint-Andre
mailto:peter@andyet.net
xmpp:stpeter@stpeter.im
https://stpeter.im/

2007-06-18
Version 1.1

Status	Type	Short Name
Draft	Standards Track	jid\20escaping

This specification defines a mechanism that enables the display in Jabber Identifiers (JIDs) of characters disallowed by the Nodeprep profile of stringprep. Although these characters -- space, double quote, ampersand, single quote, forward slash, colon, less than, greater than, and at-sign -- cannot be included in XMPP node identifiers, JID Escaping provides a native XMPP escaping mechanism for these characters so that the displayed version of a Jabber Identifier can appear to include these characters. This mechanism can also be used to translate non-XMPP addreses into XMPP syntax, for example when gatewaying between XMPP and a non-XMPP communications technology such as email.

Legal

Copyright

Permissions

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <http://xmpp.org/about-xmpp/xsf/xsf-ipr-policy/> or obtained by writing to XMPP Standards Foundation, 1899 Wynkoop Street, Suite 600, Denver, CO 80202 USA).

Contents

1 Introduction

RFC 3920 [1] defines the Nodeprep profile of stringprep (RFC 3454 [2]), which specifies that the following nine Unicode code points are disallowed in the node identifier portion of a Jabber Identifier (JID):

- U+0020 (" ") [3]
- U+0022 (")
- U+0026 (&)
- U+0027 (')
- U+002F (/)
- U+003A (:)
- U+003C (<)
- U+003E (>)
- U+0040 (@)

This restriction is an inconvenience for users who have one or more of these "disallowed characters" in their desired usernames, particularly in the case of the ' character, which is common in names like O'Hara and D'Artagnan. The restriction is a positive hardship if existing email addresses are mapped to JIDs, since some of the disallowed characters are allowed in the username portion of an email address (specifically, the characters & ' / as described in Sections 3.2.4 and 3.2.5 of RFC 2822 [4]).

To overcome this restriction, we define a way to escape the disallowed characters in JIDs. An escaped JID contains none of the disallowed characters and therefore can be transported by native XMPP implementations without modification (e.g., existing XMPP servers do not require modification in order to handle escaped JIDs). The escaped JID is unescaped only for presentation to a human user (typically by an XMPP client) or for gatewaying to a non-XMPP system (such as an LDAP database or a messaging system that does not use XMPP).

[1] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.

[2] RFC 3454: Preparation of Internationalized Strings (stringprep) <http://tools.ietf.org/html/rfc3454>.

[3] In fact all ASCII and non-ASCII space characters are disallowed, since the Nodeprep profile of stringprep prohibits all the characters specified in Appendices C.1.1 and C.1.2 of RFC 3454; however, all of these characters reduce to U+0020, also called SP.

[4] RFC 2822: Internet Message Format <http://tools.ietf.org/html/rfc2822>.

2 Requirements

This document addresses the following requirements:

1. The escaping mechanism shall apply to the node identitier portion of a JID only, and MUST NOT be applied to domain identifiers or resource identifiers.

2. Escaped JIDs MUST conform to the definition of a Jabber ID as specified in RFC 3920, including the Nodeprep profile of stringprep. In particular this means that even after passing through Nodeprep, the JID MUST be valid, with the result that Unicode look-alikes like U+02BC (Modifier Letter Apostrophe) MUST NOT be used.

3. It MUST NOT be possible for clients to use this escaping mechanism to avoid the goal of stringprep; namely, that JIDs that look alike should have same character representation after being processed by stringprep. Therefore, this mechanism MUST NOT be applied to any characters other than the disallowed characters (with the exception that, in certain circumstances, the escaping character itself (”\”) might also be escaped).

4. Existing JIDs that include portions of the escaping mechanism MUST continue to be valid.

5. The escaping mechanism MUST NOT break commonly deployed Jabber/XMPP software implementations such as servers, components, gateways, and clients.

6. The escaping mechanism SHOULD NOT place undue strain upon server implementations; implementations or deployments that do not need to unescape SHOULD be able to ignore the escaping mechanism.

3 Transformations

3.1 Concepts

This document specifies that each disallowed character shall be escaped as \hexhex -- where ”hexhex” is the hexadecimal value of the Unicode code point in question, ignoring the leading ”00” in the code point (e.g., 27 for the ’ character, resulting in an escaping of \27).
If the & character had not been in the list of disallowed characters, then normal XML escaping conventions (as specified in XML 1.0 [5]) could have been used, with the result that D’Artagnan

[5] Extensible Markup Language (XML) 1.0 (Fourth Edition) <http://www.w3.org/TR/REC-xml/>.

(for example) could have been rendered as D&apos;artagnan [sic].
It might have been desirable to use percent-encoding (e.g., %27 for the ' character) as specified in Section 2.1 of RFC 3986 [6]. However, that approach was rejected since the % character is an often-used character in existing JIDs (e.g., to replace the @ character in gateway addresses) and the resulting ambiguity would have caused misdelivered or undeliverable messages.
To avoid the problems associated with using & or % as the escaping character, this document specifies a new escaping mechanism that uses the backslash character ("\") followed by "hex-hex" (the hexadecimal value of the Unicode code point in question). This escaping method is quite similar to that used for disallowed characters in LDAP distinguished names (see RFC 2253 [7]) but is used only for the characters that are disallowed in XMPP node identifiers (as well as the escaping character itself in certain special situations).
Here is an example of an escaped JID (this would be displayed but never natively transported as "d'artagnan@musketeers.lit"):

```
d\27artagnan@musketeers.lit
```

This document describes full escaping and unescaping transformations for all nine disallowed characters. In addition, escaping and unescaping transformations are shown for the \ character in case it also needs to be escaped when it occurs in a JID or non-XMPP address as part of a character sequence that corresponds to one of the escaped characters.
Note: All transformations are exactly as specified below. CASE IS SIGNIFICANT. Lowercase was selected since Nodeprep will case fold to lowercase for US-ASCII characters such as A, C, E, and F.

3.2 Escaping Transformations

The escaping transformations are defined in the following table, whereas the rules that define when to apply these transformations are specified in the Business Rules section of this specification. Typically, escaping is performed only by a client that is processing information provided by a human user in unescaped form, or by a gateway to some external system (e.g., email or LDAP) that needs to generate a JID.

Unescaped Character	Escaped Character
<space>	\20 *
"	\22
&	\26
'	\27
/	\2f
:	\3a

[6] RFC 3986: Uniform Resource Identifiers (URI): Generic Syntax <http://tools.ietf.org/html/rfc3986>.
[7] RFC 2253: Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names <http://tools.ietf.org/html/rfc2253>.

Unescaped Character	Escaped Character
<	\3c
>	\3e
@	\40
\	\5c

* Note: The character sequence \20 MUST NOT be the first or last character of an escaped node identifier. [8]

In the following example, Porthos starts a chat with D'Artagnan, typing into his client the string "d'artagnan@musketeers.lit" (which is escaped by his client to "d\27artagnan@musketeers.lit").

Listing 1: JID Escaping

```
<message
    from='porthos@musketeers.lit/gate'
    to='d\27artagnan@musketeers.lit'
    type='chat'>
  <body>And do you always forget your eyes when you run?</body>
</message>
```

3.3 Unescaping Transformations

The unescaping transformations are defined in the following table, whereas the rules that define when to apply these transformations are specified in the Business Rules section of this specification. Typically, unescaping is performed only by a client that wants to display JIDs containing escaped characters to a human user, or by a gateway to some external system (e.g., email or LDAP) that needs to generate identifiers for foreign systems.

Escaped Character	Unescaped Character
\20	<space>
\22	"
\26	&
\27	'
\2f	/
\3a	:
\3c	<
\3e	>
\40	@
\5c	\

[8] For a similar restriction, see Section 2.4 of RFC 2253.

In the following example, D'Artagnan the elder sends a message through an SMTP mail gateway (the JID is ”treville\40musketeers.lit@smtp.gascon.fr” and the destination email address is ”treville@musketeers.lit”).

Listing 2: JID Unescaping

```
<message
    from='d\27artagnan@gascon.fr/elder'
    to='tréville\40musketeers.lit@smtp.gascon.fr'>
  <body>I recommend my son to you.</body>
</message>
```

4 Business Rules

4.1 Native Processing

The following processing rules apply to native XMPP implementations:

1. A compliant client MUST render an escaped character as its unescaped equivalent when presenting it to a human user (e.g., present \27 as the ' character), but MAY provide a way for the user to view the escaped JID in its wire format (e.g., to compare two JIDs).

2. A server or gateway MAY unescape an escaped character for communication with external systems (e.g. LDAP), but only *after* the Nodeprep profile of stringprep has been applied.

3. An entity MUST unescape only the specified sequences and MUST NOT unescape sequences that do not match the specified sequences.

4. An entity MUST NOT include the unescaped version of a disallowed character over the wire in any XML stanzas sent to another entity (since by definition the unescaped version of a disallowed character violates Nodeprep).

5. An entity MUST NOT use the unescaped version of a disallowed character when comparing two JIDs.

6. The character sequence \20 MUST NOT be the first or last character of an escaped node identifier.

7. If there are any instances of character sequences that correspond to escapings of the disallowed characters (e.g., the character sequence ”\27”) or the escaping character (i.e., the character sequence ”\5c”) in the unescaped address, the leading backslash character MUST be escaped to the character sequence ”\5c” (e.g., resulting in the character sequences ”\5c27” or ”\5c5c”). [9]

4.2 Address Transformation Algorithm

When transforming a non-XMPP (”source”) address into an escaped JID, an implementation MUST adhere to the following process:

1. If the source address is a URI, it MUST first be properly decoded according to the rules in RFC 3986 before it is transformed into a JID.

2. If the source address is a URI, the URI scheme component MUST be removed.

3. If there are any instances of character sequences that correspond to escapings of the disallowed characters (e.g., the character sequence ”\27”) or the escaping character (i.e., the character sequence ”\5c”) in the source address, the leading backslash character MUST be escaped to the character sequence ”\5c” (e.g., resulting in the character sequences ”\5c27” or ”\5c5c”).

4. All disallowed characters in the source address MUST be properly escaped in the resulting JID (as described above).

While the fourth step should be clear from the foregoing text and the second step is necessary since XMPP addresses are not URIs, the meaning of the first and third steps may not be obvious.

Regarding step one, many non-XMPP messaging systems use URIs to identify addresses (examples include the mailto:, sip:, sips:, im:, pres:, and wv: URI schemes) or follow some other encoding rules for an identifier (e.g., an LDAP distinguished name). Before transforming a non-XMPP address or identifier into a JID, the address or identifier MUST first be decoded according the rules specified for that type of address or identifier in order to ensure that the proper characters are transformed.

Regarding step three, it is possible for some non-XMPP addresses to contain character sequences that correspond to JID-escaped characters (e.g., ”\27”). Consider a Wireless

[9] It is possible that some existing JIDs already contain character sequences matching ”\5chexhex” (where ”hexhex” is the hexadecimal value of the Unicode code point for a disallowed character or the backslash character), which may result in confusion between escaped JIDs and their presentation in a client; however, a survey of one large XMPP deployment yielded no instances of such sequences or even of the character sequence ”\5c”.

Village address of <wv:\3and\2is\5cool@example.com> -- if that address were directly converted into a JID, the resulting XMPP address would be \3and\2is\5cool@example.com, which could be construed as :nd\2is\ool@example.com if JID escaping logic is applied. Therefore the leading \ character and the \ character before the character sequence 5c MUST be converted to the character sequence ”\5c” during the transformation, leading to a JID of \5c3and\2is\5c5cool@example.com (which would be presented to a human user as \3and\2is\5cool@example.com).

4.3 Exceptions

In order to maintain as much backward compatibility as possible, partial escape sequences and escape sequences corresponding to characters not on the list of disallowed characters MUST be ignored (with the exception of the escaping character ’\’ itself in the rare case when the source address includes the sequence ’\5c’).

Listing 3: Partial escape sequence

```
\textbf{\2plus\2is\4} is not modified by escaping or unescaping
   transformations.
```

Listing 4: Invalid escape sequence 1

```
\textbf{foo\bar} is not modified (to \textbf{fooºr}) by escaping or
   unescaping transformations.
```

Listing 5: Invalid escape sequence 2

```
\textbf{foob\41r} is not modified (to \textbf{foobAr}) by escaping or
   unescaping transformations.
```

However, \5c would be escaped if found in the source address (e.g., a source address of ”c:\5commas@example.com” would be escaped to ”c\3a\5c5commas@example.com”) and unescaped if contained in the JID-on-the-wire (e.g., a JID-on-the-wire of ”c\3a\5c5commas@example.com” would be unescaped back to ”c:\5commas@example.com”).

4.4 JID Escaping vs. Older Methods

When a client attempts to communicate with another entity through a gateway, it needs to know which escaping mechanism to use. A client MUST assume that the gateway does not support the JID escaping mechanism unless it explicitly discovers support for the **jid\20escaping** [sic] feature as described under Determining Support. If there are any errors in the service discovery exchange or if support for JID escaping is not discovered, the client SHOULD proceed as follows:

1. If the gateway supports the ’jabber:iq:gateway’ protocol (as specified in Gateway Interaction (XEP-0100) [10]), use that protocol.
2. If the gateway does not support the ’jabber:iq:gateway’ protocol, use customary escaping mechanisms (such as transformation of the @ character to the % character).

5 Examples

In order to assist developers, this section shows a large number of examples for XMPP-native JIDs as well as mappings between JIDs and addresses or identifiers used in the following standardized protocols:

- Mailboxes and the mailto: URI scheme as used in email.
- The sip: and sips: URI schemes as used in SIP/SIMPLE.
- The im: and pres: URI schemes.
- The wv: URI scheme as used in Wireless Village (IMPS).
- LDAP distinguished names.

5.1 Jabber Identifiers

The following table shows user input, the escaped JID for sending over the wire, and client display (same as user input) for node identifiers that might possibly be used in native JIDs. The examples are numbered for easy reference. Naturally, a client that does not perform JID escaping would display the JIDs in their escaped form (e.g., ”space\20cadet” instead of ”space cadet”).

#	User Input	Escaped JID	Client Display
1	space cadet@example.com	space\20cadet@example.com	space cadet@example.com
2	call me ”ishmael”@example.com	call\20me\20\22ishmael\22@example.com	call me ”ishmael”@example.com
3	at&t guy@example.com	at\26t\20guy@example.com	at&t guy@example.com
4	d’artagnan@example.com	d\27artagnan@example.com	d’artagnan@example.com
5	/.fanboy@example.com	\2f.fanboy@example.com	/.fanboy@example.com
6	::foo::@example.com	\3a\3afoo\3a\3a@example.com	::foo::@example.com
7	<foo>@example.com	\3cfoo\3e@example.com	<foo>@example.com
8	user@host@example.com	user\40host@example.com	user@host@example.com
9	c:\net@example.com	c\3a\net@example.com	c:\net@example.com
10	c:\\net@example.com	c\3a\\net@example.com	c:\\net@example.com

[10] XEP-0100: Gateway Interaction <http://xmpp.org/extensions/xep-0100.html>.

#	User Input	Escaped JID	Client Display
11	c:\cool stuff@example.com	c\3a\cool\20stuff@example.com	c:\cool stuff@example.com
12	c:\5commas@example.com	c\3a\5c5commas@example.com	c:\5commas@example.com

5.2 Email Addresses

The address format for an Internet mailbox is specified in RFC 2822. The identifier of interest in this context is the "addr-spec" address and more particularly the "dot-atom-text" rule specified in Section 3.2.4, i.e., the email address shorn of angle brackets, display names, comments, quoted strings, and the like. Because some deployments of XMPP messaging systems may want to re-use existing email addresses as JIDs, it is helpful to define how to transform an email address into a JID.

In general, it is straightforward to transform an email address (i.e., a "dot-atom-text") into a JID, since traditional email addresses allow US-ASCII characters only rather than the nearly full range of Unicode code points allowed in a JID. [11] However, there are three characters allowed in the local-part of an email address that are not allowed in the node identifier portion of a JID: namely, the characters & ' / as described in Sections 3.2.4 and 3.2.5 of RFC 2822. In order to transform these characters, a compliant implementation MUST use the methods specified herein.

Listing 6: An Email Address Containing JID-Disallowed Characters

```
here's_a_wild_&_/cr%zy/_address@example.com
```

Listing 7: The Transformed JID

```
here\27s_a_wild_\26_\2fcr%zy\2f_address@example.com
```

Listing 8: The JID as Presented to a User

```
here's_a_wild_&_/cr%zy/_address@example.com
```

(Note: Because the backslash character is forbidden in the "dot-atom-text" construction, an email address should not contain a character sequence that corresponds to one of the escaped characters specified in the Transformations section of this document; therefore, no such examples are shown.)

An email address may also exist in the form of a mailto: URI as specified in RFC 2368 [12]. Before transforming a mailto: URI into a JID, it MUST be URL-decoded and all headers MUST be removed, leaving a mailbox identifier, as shown in the following example.

[11] This specification does not cover recent efforts to define internationalized email addresses.

[12] RFC 2368: The mailto URL scheme <http://tools.ietf.org/html/rfc2368>.

Listing 9: A mailto: URI Containing JID-Disallowed Characters

```
mailto:here%27s_a_wild_%26_%2Fcr%zy%2F_address@example.com?subject=
    that%20is%20crazy%21
```

Listing 10: The Resulting Mailbox

```
here's_a_wild_&_/cr%zy/_address@example.com
```

Listing 11: The Transformed JID

```
here\27s_a_wild_\26_\2fcr%zy\2f_address@example.com
```

Listing 12: The JID as Presented to a User

```
here's_a_wild_&_/cr%zy/_address@example.com
```

The foregoing examples showed how to transform an email address or mailto: URI into a JID. However, it also may be necessary to convert a JID into an email address or mailto: URI, as shown in the following example.

Listing 13: User Enters Address, Including Disallowed Characters

```
here's_a_wild_&_/cr%zy/_address@example.com
```

Listing 14: Client Transforms Address Using JID Escaping

```
here\27s_a_wild_\26_\2fcr%zy\2f_address@example.com
```

Listing 15: Application Converts Escaped JID to Mailbox

```
here's_a_wild_&_/cr%zy/_address@example.com
```

Listing 16: Application Converts Mailbox to mailto: URI

```
mailto:here%27s_a_wild_%26_%2Fcr%zy%2F_address@example.com
```

5.3 SIP Addresses

As specified in RFC 3261 [13], a SIP address (i.e., a sip: or sips: URI) can be quite complex if URI parameters or headers are included. However, a basic SIP address (the combination of the optional "userinfo" and required "hostport" constructions) is essentially similar to an email address (e.g., the same characters & ' / allowed in an email address but disallowed in an XMPP node identifier are also allowed in a basic SIP address).

[13] RFC 3261: Session Initiation Protocol (SIP) <http://tools.ietf.org/html/rfc3261>.

Listing 17: A Basic sip: URI Containing JID-Disallowed Characters

```
sip:here%27s_a_wild_%26_%2Fcr%zy%2F_address@example.com
```

Listing 18: The URL-Decoded Address

```
here's_a_wild_&_/cr%zy/_address@example.com
```

Listing 19: The Transformed JID

```
here\27s_a_wild_\26_\2fcr%zy\2f_address@example.com
```

Listing 20: The JID as Presented to a User

```
here's_a_wild_&_/cr%zy/_address@example.com
```

The foregoing example showed how to transform a sip: or sips: URI into a JID. However, it also may be necessary to convert a JID into a sip: or sips: URI, as shown in the following example.

Listing 21: User Enters Address, Including Disallowed Characters

```
here's_a_wild_&_/cr%zy/_address@example.com
```

Listing 22: Client Transforms Address Using JID Escaping

```
here\27s_a_wild_\26_\2fcr%zy\2f_address@example.com
```

Listing 23: Application Converts Escaped JID to sip: URI

```
sip:here%27s_a_wild_%26_%2Fcr%zy%2F_address@example.com
```

5.4 IM and Presence Addresses

The im: and pres: URI schemes are specified in RFC 3860 [14] and RFC 3859 [15] respectively. With the exception of headers, an im: or pres: URI is simply a mailbox (as specified in RFC 2822) prepended with the im: or pres: scheme. Thus a basic IM or PRES address (not including optional headers) is essentially similar to an email address (e.g., the same characters & ' / allowed in an email address but disallowed in an XMPP node identifier are also allowed in a basic IM or PRES address).

Listing 24: A Basic im: URI Containing JID-Disallowed Characters

```
im:here%27s_a_wild_%26_%2Fcr%zy%2F_address@example.com
```

[14] RFC 3860: Common Profile for Instant Messaging (CPIM) <http://tools.ietf.org/html/rfc3860>.

[15] RFC 3859: Common Profile for Presence (CPP) <http://tools.ietf.org/html/rfc3859>.

Listing 25: The URL-Decoded Address

```
here's_a_wild_&_/cr%zy/_address@example.com
```

Listing 26: The Transformed JID

```
here\27s_a_wild_\26_\2fcr%zy\2f_address@example.com
```

Listing 27: The JID as Presented to a User

```
here's_a_wild_&_/cr%zy/_address@example.com
```

The foregoing example showed how to transform an im: or pres: URI into a JID. However, it also may be necessary to convert a JID into an im: or pres: URI, as shown in the following example.

Listing 28: User Enters Address, Including Disallowed Characters

```
here's_a_wild_&_/cr%zy/_address@example.com
```

Listing 29: Client Transforms Address Using JID Escaping

```
here\27s_a_wild_\26_\2fcr%zy\2f_address@example.com
```

Listing 30: Application Converts Escaped JID to pres: URI

```
pres:here%27s_a_wild_%26_%2Fcr%zy%2F_address@example.com
```

5.5 IMPS Addresses

The Instant Messaging and Presence Service (IMPS) protocol was originally defined by the Wireless Village consortium and is now maintained by the Open Mobile Alliance (OMA) [16]. An IMPS address is formatted as a wv: URI, as specified in WV Client-Server Protocol v1.1 [17]. A basic address (not including a private resource) is of the form <wv:user-id@domain> and an address with a private resource is of the form <wv:user-id/resource@domain>.
The ”User-ID” construction is either a mobile phone number (beginning with ”+1” for international numbers and a digit for national numbers) or an ”Internet-Identity”. An ”Internet-Identity” may contain any US-ASCII character other than / @ + SP TAB and thus may include the following characters that are disallowed in the node identifier portion of a JID: ” & ' / : < > (which characters MUST be escaped when transforming an IMPS address into a JID). However, some of those characters are also reserved in URI syntax (namely the & ' / characters) so those characters will be found in escaped form within a wv: URI.

[16] The Open Mobile Alliance is the focal point for the development of mobile service enabler specifications, which support the creation of interoperable end-to-end mobile services. For further information, see <http://www.openmobilealliance.org/>.

[17] Wireless Village Client-Server Protocol v1.1 <http://www.openmobilealliance.org/tech/affiliates/wv/wvindex.html>.

Listing 31: A Basic wv: URI Containing JID-Disallowed Characters

```
wv:here%27s_a_wild_%26_%2Fcr%zy%2F_address_for%3A%3Cwv%3E%28%22IMPS
    %22%29@example.com
```

Listing 32: The URL-Decoded Address

```
here’s_a_wild_&_/cr%zy/_address_for:<wv>(”IMPS”)@example.com
```

Listing 33: The Transformed JID

```
here\27s_a_wild_\26_\2fcr%zy\2f_address_for\3a\3cwv\3e(\22IMPS\22)
    @example.com
```

Listing 34: The JID as Presented to a User

```
here’s_a_wild_&_/cr%zy/_address_for:<wv>(”IMPS”)@example.com
```

Unlike the foregoing address types, IMPS addresses are allowed to contain backslashes. This implies that it is possible for an IMPS address to contain a character sequence that corresponds to one of the escaped character representations for code points that are disallowed in XMPP node identifiers. An example would be the IMPS address <wv:\3and\2is\5cool@example.com>, where the character sequence ”\3a” could be interpreted as the : character (and the character sequence ”\5c” as ”\”) if that IMPS address is directly converted into a JID. Therefore, the leading \ character MUST be transformed to ”\5c” (and the source character sequence ”\5c” to ”\5c5c”) in order to avoid possible ambiguity. Thus the transformed JID would be <\5c3and\2is\5c5cool@example.com>, which would be presented to a user as <\3and\2is\5cool@example.com>.

If an IMPS address contains a private resource, a gateway between XMPP and IMPS should process the resource and append it to the end of the JID; however, such gateway behavior is out of scope for this document.

The foregoing example showed how to transform a wv: URI into a JID. However, it also may be necessary to convert a JID into a wv: URI, as shown in the following example.

Listing 35: User Enters Address, Including Disallowed Characters

```
here’s_a_wild_&_/cr%zy/_address_for:<wv>(”IMPS”)@example.com
```

Listing 36: Client Transforms Address Using JID Escaping

```
here\27s_a_wild_\26_\2fcr%zy\2f_address_for\3a\3cwv\3e(\22IMPS\22)
    @example.com
```

Listing 37: Application Converts Escaped JID to wv: URI

```
wv:here%27s_a_wild_%26_%2Fcr%zy%2F_address_for%3A%3Cwv%3E%28%22IMPS
    %22%29@example.com
```

5.6 LDAP Distinguished Names

Within the Lightweight Directory Access Protocol (see RFC 2251 [18]), a ”distinguished name” (DN) is a hierarchically-organized string representation that uniquely identifies a user, system, or organization. It is possible that some messaging systems use LDAP distinguished names to identify entities that can communicate using the system (e.g., this is reputed to be the case for certain releases of the Lotus Sametime system sold by IBM), and in any case it may be helpful to transform an LDAP distinguished name into an XMPP address for identification or addressing purposes.

As previously mentioned, a UTF-8 string representation of LDAP distinguished names is specified in RFC 2253. This representation specifies that the characters , + ” \ < > ; are to be escaped with the backslash character (e.g., the character sequence ”\,” would be used to escape the , character) and that any other non-US-ASCII characters are to be escaped using a character sequence of the form ”\xx”.

The following example shows a distinguished name (and transformations thereof) for a person whose common name is ”D’Artagnan Saint-André” and who is associated with an organization called ”Example & Company, Inc.” whose domain name is ”example.com”:

Listing 38: A Distinguished Name

```
CN=D'Artagnan␣Saint-André,O=Example␣&␣Company,␣Inc.,DC=example,DC=com
```

Listing 39: UTF-8 Representation of Distinguished Name

```
CN=D'Artagnan␣Saint-Andr\E9,O=Example␣&amp;␣Company\,␣Inc.,DC=example,
    DC=com
```

This example assumes that the specified user is identified with a gateway running at st.example.com (note that the backslash escaping the , character in the organization name is removed during the transformation).

Listing 40: The Transformed JID

```
CN=D\27Artagnan\20Saint-Andr\E9,O=Example\20\26\20Company,\20Inc.,DC=
    example,DC=com@st.example.com
```

Listing 41: The JID as Presented to a User

```
CN=D'Artagnan␣Saint-André,O=Example␣&␣Company,␣Inc.,DC=example,DC=
    com@st.example.com
```

Naturally, a more intelligent gateway could use the Domain Components to construct a more readable JID, such as <D\27Artagnan\20Saint-André@example.com>; however, such gateway behavior is out of scope for this document.

The foregoing example showed how to transform an LDAP distinguished name into a JID.

[18] RFC 2251: Lightweight Directory Access Protocol (v3) <http://tools.ietf.org/html/rfc2251>.

However, it also may be necessary to convert a JID into an LDAP distinguished name, as shown in the following example.

Listing 42: User Enters Address, Including Disallowed Characters

```
CN=D’Artagnan␣Saint-Andr&#xe9;,O=Example␣&amp;␣Company,␣Inc.,DC=
    example,DC=com@st.example.com
```

Listing 43: Client Transforms Address Using JID Escaping

```
CN=D\27Artagnan\20Saint-Andr\E9,O=Example\20\26\20Company,\20Inc.,DC=
    example,DC=com@st.example.com
```

Listing 44: Application Converts Escaped JID to UTF-8 Representation of LDAP Distinguished Name

```
CN=D’Artagnan␣Saint-Andr\E9,O=Example␣&amp;␣Company\,␣Inc.,DC=example,
    DC=com
```

Listing 45: Application Converts UTF-8 Representation to LDAP Distinguished Name

```
CN=D’Artagnan␣Saint-André,O=Example␣&␣Company,␣Inc.,DC=example,DC=com
```

5.7 IRC Addresses

RFC 2812 [19] defines the address format for Internet Relay Chat (IRC) entities, which can be servers, channels, or users. The ”user” portion of an IRC address may contain any octet except NUL, CR, LF, SP, and ”@”; this includes the characters ” & ’ / : < > \ (which are disallowed in XMPP node identifiers and therefore MUST be escaped when transforming an IRC address into a JID).

Listing 46: A Basic IRC address Containing JID-Disallowed Characters

```
somenick!user”&’/:<>\3address@example.com
```

Listing 47: The Transformed JID

```
somenick!user\22\26\27\2f\3a\3c\3e\5c3address@example.com
```

Listing 48: The JID as Presented to a User

```
somenick!user”&’/:<>\3address@example.com
```

Like IMPS addresses, IRC addresses are allowed to contain backslashes. This implies that it is possible for an IMPS address to contain a character sequence that corresponds to one of the escaped character representations for code points that are disallowed in XMPP node identifiers. An example is shown above.

[19] RFC 2812: Internet Relay Chat: Client Protocol <http://tools.ietf.org/html/rfc2812>.

6 Determining Support

If an entity needs to determine whether another entity supports JID escaping, it MUST send a disco#info request to the other entity as specified in Service Discovery (XEP-0030) [20].

Listing 49: Client requests features

```
<iq type='get'
    from='porthos@musketeers.lit/gate'
    to='irc.shakespeare.lit'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

If the queried entity supports JID escaping, it MUST return a **jid\20escaping** [sic] feature in its reply.

Listing 50: Service responds with features

```
<iq type='get'
    to='porthos@musketeers.lit/gate'
    from='irc.shakespeare.lit'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
...
    <feature var='jid\20escaping'/>
  </query>
</iq>
```

7 Security Considerations

If an entity (e.g., a client or a gateway) performs JID escaping, it MUST do so consistently (for example, a client or server MUST consistently apply JID escaping and unescaping to the JIDs it handles) so that the entity does not present the same JID in two different ways or present two different JIDs in the same way.

Naturally, if one entity performs JID escaping and another entity does not perform JID escaping, the same JID could be presented differently by those entities (e.g., the JID d\27artagnan@musketeers.lit would be presented as d'artagnan@musketeers.lit by a client that performs JID escaping but as d\27artagnan@musketeers.lit by a client that does not perform JID escaping). By the same token, two different JIDs could be presented in the same way by those entities (e.g., the JID foo\5cbar@example.com would be presented as foo\bar@example.com by a client that performs JID escaping and the JID foo\bar@example.com would be presented as foo\bar@example.com by a client that does not perform JID escaping). These differing presentations could be a source of confusion (e.g.,

[20] XEP-0030: Service Discovery <http://xmpp.org/extensions/xep-0030.html>.

the same human user could use two different clients, one of which performs JID escaping and one of which does not). This confusion may have security implications since in rare instances messages and other information could be directed to an entity other than the intended recipient; unfortunately, this is unavoidable until all XMPP clients support JID escaping.

An entity that performs JID escaping MUST NOT compare unescaped versions, otherwise messages and other information could be directed to an entity other than the intended recipient.

An entity that transforms a non-XMPP address into a JID MUST follow the algorithm specified in the Address Transformation Algorithm section of this document, otherwise messages and other information could be directed to an entity other than the intended recipient.

8 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [21].

9 XMPP Registrar Considerations

9.1 Service Discovery Features

The XMPP Registrar [22] includes the **jid\20escaping** [sic] feature in its registry of service discovery features.

10 Acknowledgements

The authors would like to thank Robin Redeker for his feedback.

[21] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

[22] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <http://xmpp.org/registrar/>.