StringPrep, NamePrep and IDNA:

Preparation of Unicode Strings for Domain Names

Raghuram (Ram) Viswanadha
ramv@us.ibm.com

Introduction

Unicode is a character repertoire with code position assignments for characters in most scripts in the world. Unicode contains assignments for many glyphically similar but semantically unique characters. Comparing strings in a consistent manner becomes imperative when such a large repertoire is used in network protocols. This document describes the StringPrep framework which facilitates this function. StringPrep provides sets of rules for use of Unicode and syntax for prevention of spoofing. The implementation of StringPrep and IDNA services in International Components for Unicode (ICU) and use of the Application Programming Interfaces (APIs) are also discussed. Knowledge of Unicode, concepts such as case mapping and normalization is assumed.

StringPrep

StringPrep, the process of preparing Unicode strings for use in network protocols is defined in RFC 3454 (http://www.rfc-editor.org/rfc/rfc3454.txt). The RFC defines a broad framework and rules for processing the strings.

Protocols that prescribe use of StringPrep must define a profile of StringPrep, whose applicability is limited to the protocol. Profiles are a set of rules and data tables which describe the how the strings should be prepare. The profiles can choose to turn on or turn off normalization, checking for bidirectional characters. They can also choose to add or remove mappings, unassigned and prohibited code points from the tables provided.

StringPrep uses Unicode Version 3.2 and defines a set of tables for use by the profiles. The profiles can chose to include or exclude tables or code points from the tables defined by the RFC.

StringPrep defines tables that can be broadly classified into:

1. Unassigned Table: Contains code points that are unassigned in Unicode Version 3.2. Unassigned code points may be allowed or disallowed in the output string depending on the application. The table in Appendix A.1 of the RFC contains the code points.
2. Mapping Tables: Code points that are commonly deleted from the output and code points that are case mapped are included in this table. There are two mapping tables in the Appendix namely B.1 and B.2

3. Prohibited Tables: Contains code points that are prohibited from the output string. Control codes, private use area code points, non-character code points, surrogate code points, tagging and deprecated code points are included in this table. There are nine mapping tables in Appendix which include the prohibited code points namely C.1, C.2, C.3, C.4, C.5, C.6, C.7, C.8 and C.9.

The procedure for preparing strings for use can be described in the following steps:

1. **Map:** For each code point in the input check if it has a mapping defined in the mapping table, if so, replace it with the mapping in the output.
2. **Normalize:** Normalize the output of step 1 using Unicode Normalization Form NFKC, it the option is set. Normalization algorithm must conform to UAX 15.
3. **Prohibit:** For each code point in the output of step 2 check if the code point is present in the prohibited table, if so, fail returning an error.
4. **Check BiDi:** Check for code points with strong right-to-left directionality in the output of step 3. If present, check if the string satisfies the rules for bidirectional strings as specified.

NamePrep

NamePrep is a profile of StringPrep for use in IDNA. This profile in defined in the RFC 3491 (http://www.rfc-editor.org/rfc/rfc3491.txt).

The profile specifies the following rules:

1. **Map:** Include all code point mappings specified in the StringPrep.
2. **Normalize:** Normalize the output of step 1 according to NFKC.
3. **Prohibit:** Prohibit all code points specified as prohibited in StringPrep except for the space (U+0020) code point from the output of step 2.
4. **Check BiDi:** Check for bidirectional code points and process according to the rules specified in StringPrep.

Punycode

Punycode is an encoding scheme for Unicode for use in IDNA. Punycode converts Unicode text to unique sequence of ASCII text and back to Unicode. It is an ASCII Compatible Encoding (ACE). Punycode is described in RFC 3492 (http://www.rfc-editor.org/rfc/rfc3492.txt).

The Punycode algorithm is a form of a general Bootstring algorithm which allows strings composed of smaller set of code points to uniquely represent any string of code points from a larger set. Punycode represents Unicode code points from U+0000 to U+10FFFF by using the smaller ASCII set U+0000 to U+0007F. The algorithm can also preserve case information of the code points in the lager set while and encoding and decoding. This feature, however, is not used in IDNA.

Internationalizing Domain Names in Applications (IDNA)

The Domain Name Service (DNS) protocol defines the procedure for matching of ASCII strings case insensitively to the names in the lookup tables containing mapping of IP (Internet Protocol) addresses to server names. When Unicode is used instead of ASCII in server names then two problems arise which need to be dealt with differently. When the server name is displayed to the user then Unicode text should be displayed. When Unicode text is stored in lookup tables, for compatibility with older DNS protocol and the resolver libraries, the text should be the ASCII equivalent. The IDNA protocol, defined by RFC 3490 (http://www.rfc-editor.org/rfc/rfc3490.txt), satisfies the above requirements.

Server names stored in the DNS lookup tables are usually formed by concatenating domain labels with a label separator, e.g.:

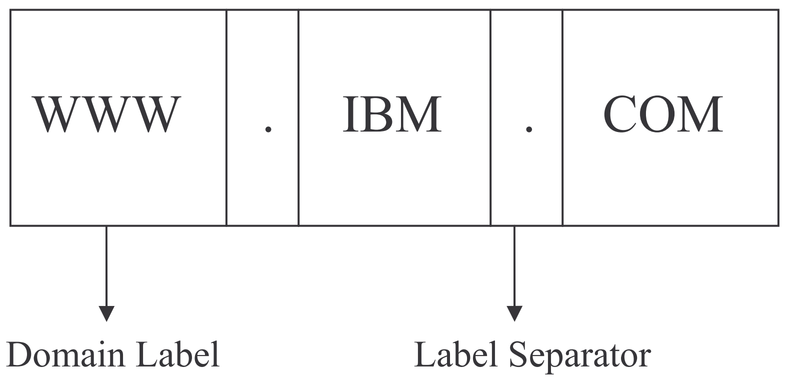

Figure (i)

The protocol defines operations to be performed on domain labels before the names are stored in the lookup tables and before the names fetched from lookup tables are displayed to the user. The operations are:

- **ToASCII:** This operation is performed on domain labels before sending the name to a resolver and before storing the name in the DNS lookup table. The domain labels are processed by StringPrep algorithm by using the rules specified by NamePrep profile. The output of this step is then encoded by using Punycode and an ACE prefix is added to denote that the text is encoded using Punycode. IDNA uses “xn--” before encoded label.

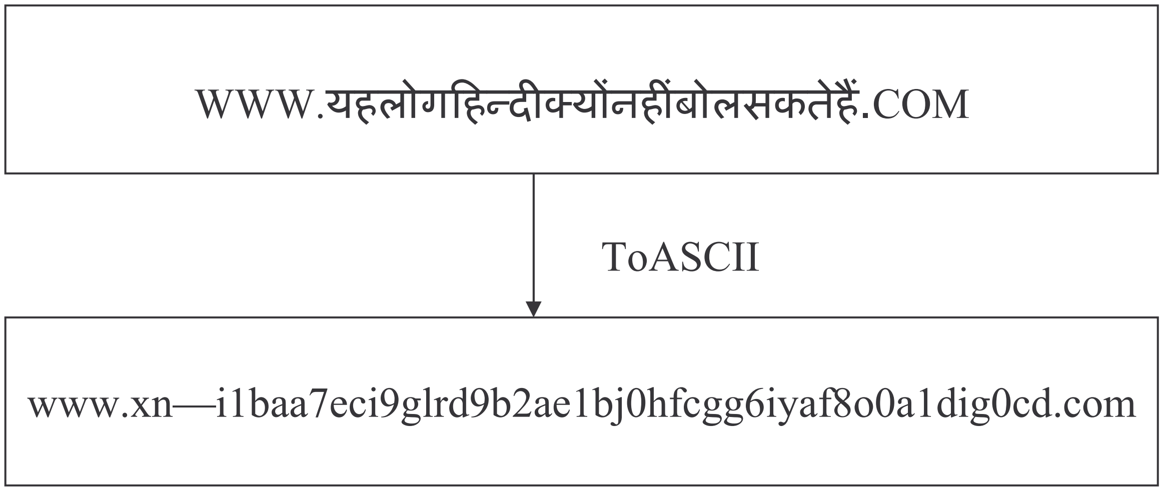
WWW.यहलोगहिन्दीक्योंनहींबोलसकतेहैं.COM
ToASCII
www.xn—i1baa7eci9glrd9b2ae1bj0hfcgg6iyaf8o0a1dig0cd.com

Figure (ii)

- **ToUnicode:** This operation is performed on domain labels before displaying the names to users. If the domain label is prefixed with the ACE prefix for IDNA, then the label excluding the prefix is decoded using Punycode. The output of Punycode decoder is verified by applying ToASCII operation and comparing the output with the input to the ToUnicode operation.

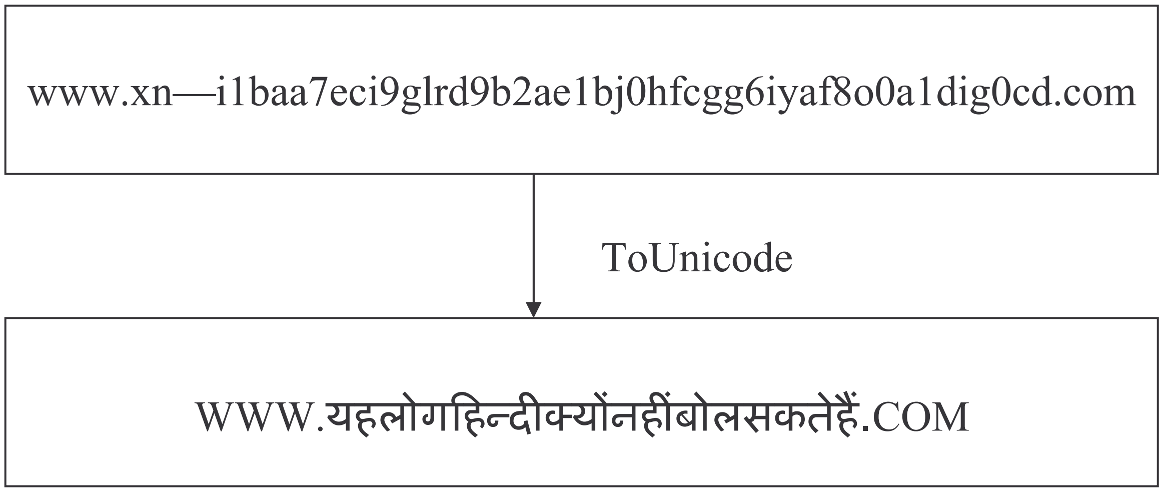
www.xn—i1baa7eci9glrd9b2ae1bj0hfcgg6iyaf8o0a1dig0cd.com
ToUnicode
WWW.यहलोगहिन्दीक्योंनहींबोलसकतेहैं.COM

Figure (iii)

Unicode contains code points that are glyphically similar to the ASCII Full Stop (U+002E). These code points must be treated as label separators when performing ToASCII operation. These code points are:

- Ideographic Full Stop (U+3002)
- Full Width Full Stop (U+FF0E)
- Half Width Ideographic Full Stop (U+FF61)

Unassigned code points in Unicode Version 3.2 as given in StringPrep tables are treated differently depending on how the processed string is used. For query operations, where a registrar is requested for information regarding availability of a certain domain name, unassigned code points are allowed to be present in the string. For storing the string in DNS lookup tables, unassigned code points are prohibited from the input.

IDNA specifies that the ToUnicode and ToASCII have options to check for Letter-Digit-Hyphen code points and adhere to the STD3 ASCII Rules.

IDNA specifies that domain labels are equivalent if and only if the output of ToASCII operation on the labels matches using case insensitive ASCII comparison.

StringPrep Service in ICU

The StringPrep service in ICU is data driven. The service is based on Open-Use-Close pattern. A StringPrep profile is opened, the strings are processed according to the rules specified in the profile and the profile is closed once the profile is ready to be disposed.

Tools for filtering RFC 3454 and producing a rule file that can be compiled into a binary format containing all the information required by the service are provided.

The procedure for producing a StringPrep profile data file is as given below:

1. Run filterRFC3454.pl Perl tool, to filter the RFC file and produce a rule file. The text file produced can be edited by the clients to add/delete mappings or add/delete prohibited code points.
2. Run the gensprep tool to compile the rule file into a binary format. The options to turn on normalization of strings and checking of bidirectional code points are passed as command line options to the tool. This tool produces a binary profile file with the extension “spp”.
3. Open the StringPrep profile with path to the binary and name of the binary profile file as the options to the open call. The profile data files are memory mapped and cached for optimum performance.

Code Snippets:

Note: The code snippets demonstrate the usage of the APIs. Applications should keep the profile object around for reuse, instead of opening and closing the profile each time.

C++

```
UErrorCode status = U_ZERO_ERROR;
UParseError parseError;

/* open the StringPrep profile */
UStringPrepProfile* nameprep = usprep_open("/usr/joe/mydata",
                                        "nfscsi", &status);
if(U_FAILURE(status)){
     /* handle the error */
}
/* prepare the string for use according
 * to the rules specified in the profile
 */
int32_t retLen = usprep_prepare(src, srcLength, dest, destCapacity,
                                USPREP_ALLOW_UNASSIGNED, nameprep,
                                &parseError,&status);
/* close the profile*/
usprep_close(nameprep);
```

Java

```
private static final StringPrep nfscsi = null;
//singleton instance
private static final NFSCSIStringPrep prep=new NFSCSIStringPrep();

private NFSCSIStringPrep (){
      try{
            InputStream nfscsiFile =
                                 TestUtil.getDataStream("nfscsi.spp");
            nfscsi = new StringPrep(nfscsiFile);
            nfscsiFile.close();

      }catch(IOException e){
            throw new RuntimeException(e.toString());
      }
}

private static byte[] prepare(byte[] src, StringPrep prep)
                              throws StringPrepParseException,
                              UnsupportedEncodingException{

      String s = new String(src, "UTF-8");

      UCharacterIterator iter = UCharacterIterator.getInstance(s);

      StringBuffer out = prep.prepare(iter,StringPrep.DEFAULT);

      return out.toString().getBytes("UTF-8");
}
```

IDNA API in ICU

ICU provides APIs for performing the ToASCII, ToUnicode and compare operations as defined by the RFC 3490. Convenience methods for comparing IDNs are also provided. These APIs follow ICU policies for string manipulation and coding guidelines.

Code Snippets:

Note: The code snippets demonstrate the usage of the APIs. Applications should keep the profile object around for reuse, instead of opening and closing the profile each time.

ToASCII operation

C++

```
UChar* dest = (UChar*) malloc(destCapcaity* U_SIZEOF_UCHAR);

destLen = uidna_toASCII(src, srcLen, dest, destCapacity,
 UIDNA_DEFAULT, &parseError, &status);

if(status == U_BUFFER_OVERFLOW_ERROR){

      status = U_ZERO_ERROR;

      destCapacity = destLen + 1/* for the terminating Null */;

      free(dest); /* free the memory */

      dest = (UChar*) malloc( destLen * U_SIZEOF_UCHAR);

      destLen = uidna_toASCII(src, srcLen, dest, destCapacity,
                              UIDNA_DEFAULT, &parseError, &status);
}
if(U_FAILURE(status)){
      /* handle the error */
}
/* do interesting stuff with output*/
```

Java

```
try{

      StringBuffer out= IDNA.convertToASCII(inBuf,IDNA.DEFAULT);

}catch(StringPrepParseException ex){

      /*handle the exception*/
}
```

ToUnicode operation

C++

```
UChar* dest = (UChar*) malloc(destCapacity* U_SIZEOF_UCHAR);

destLen = uidna_toUnicode(src, srcLen, dest, destCapacity ,
                          UIDNA_DEFAULT &parseError, &status);

if(status == U_BUFFER_OVERFLOW_ERROR){
      status = U_ZERO_ERROR;

      destCapacity= destLen + 1/* for the terminating Null */;

      /* free the memory */
      free(dest);

      dest = (UChar*) malloc( destLen * U_SIZEOF_UCHAR);

      destLen = uidna_toUnicode(src, srcLen, dest, destCapacity,
                                UIDNA_DEFAULT, &parseError,
                                &status);
}
if(U_FAILURE(status)){
   /* handle the error */
}
/* do interesting stuff with output*/
```

Java

```
try{

    StringBuffer out= IDNA.convertToUnicode(inBuf,IDNA.DEFAULT);

}catch(StringPrepParseException ex){

    // handle the exception

}
```

Compare operation

C++

```
int32_t rc = uidna_compare(source1, length1,
                            source2, length2,
                            UIDNA_DEFAULT,
                            &status);

if(rc==0){
     /* the IDNs are same ... do something interesting */
}else{
     /* the IDNs are different ... do something */
}
```

Java

```
try{

    int retVal = IDNA.compare(s1,s2,IDNA.DEFAULT);

    // do something interesting with retVal

}catch(StringPrepParseException e){

   // handle the exception
}
```

Design Considerations

StringPrep profiles exhibit the following characteristics:

- The profiles contain information about code points. StringPrep allows profiles to add/delete code points or mappings.
- Options such as turning normalization and checking for bidirectional code points on or off are the properties of the profiles
- The StringPrep algorithm is not overridden by the profile.
- Once defined, the profiles do not change.
- The StringPrep profiles are used in network protocols so runtime performance is important.

Many profiles have been and are being defined, so applications should be able to plug-in arbitrary profiles and get the desired result out of the framework.

ICU is designed for this usage by providing build-time tools for arbitrary StringPrep profile definitions, and loading them from application-supplied data in binary form with data structures optimized for runtime use.

Demo

A web application at http://oss.software.ibm.com/cgi-bin/icu/idnademo illustrates the use of IDNA API. The source code for the application is available at http://oss.software.ibm.com/cvs/icu/icuapps/idnbrowser.

Conclusion

The StringPrep and IDNA algorithms demonstrate that use of Unicode in protocols designed for ASCII is possible without a replacing the hardware currently in use.

Appendix

NFS Version 4 Profiles

Network File System Version 4 defined by RFC 3530 (http://www.rfc-editor.org/rfc/rfc3530.txt) defines use of Unicode text in the protocol. ICU provides the requisite profiles as part of test suite and code for processing the strings according the profiles as a part of samples.

The RFC defines three profiles:

nfs4_cs_prep Profile: This profile is used for preparing file and path name strings. Normalization of code points and checking for bidirectional code points are turned off. Case mappings are included if the NFS implementation supports case insensitive file and path names.

nfs4_cis_prep Profile: This profile is used for preparing NFS server names. Normalization of code points and checking for bidirectional code points are turned on. This profile is equivalent to NamePrep profile.

nfs4_mixed_prep Profile: This profile is used for preparing strings in the Access Control Entries of NFS servers. These strings consist of two parts, prefix and suffix, separated by '@' (U+0040). The prefix is processed with case mappings turned off and the suffix is processed with case mappings turned on. Normalization of code points and checking for bidirectional code points are turned on.

XMPP Profiles

Extensible Messaging and Presence Protocol (XMPP) is an XML based protocol for near real-time extensible messaging and presence. This protocol defines use of two StringPrep profiles:

ResourcePrep Profile: This profile is used for processing the resource identifiers within XMPP. Normalization of code points and checking of bidirectional code points are turned on. Case mappings are excluded. The space code point (U+0020) is excluded from the prohibited code points set.

NodePrep Profile: This profile is used for processing the node identifiers within XMPP. Normalization of code points and checking of bidirectional code points are turned on.

Case mappings are included. All code points specified as prohibited in StringPrep are prohibited. Additional code points are added to the prohibited set.