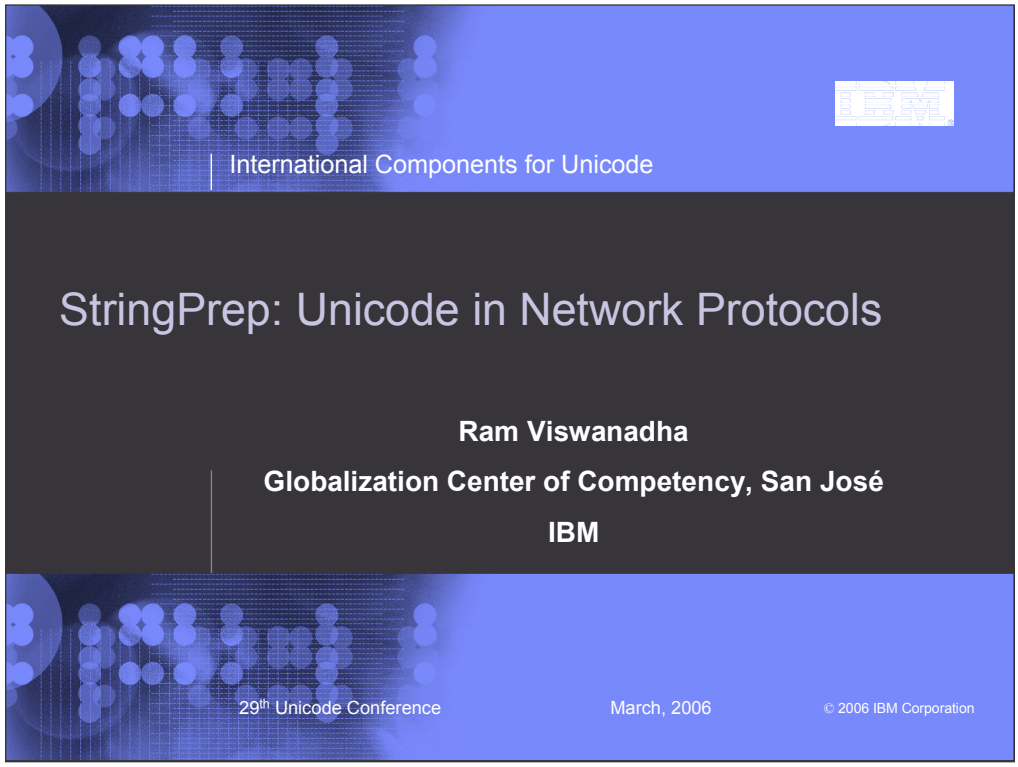
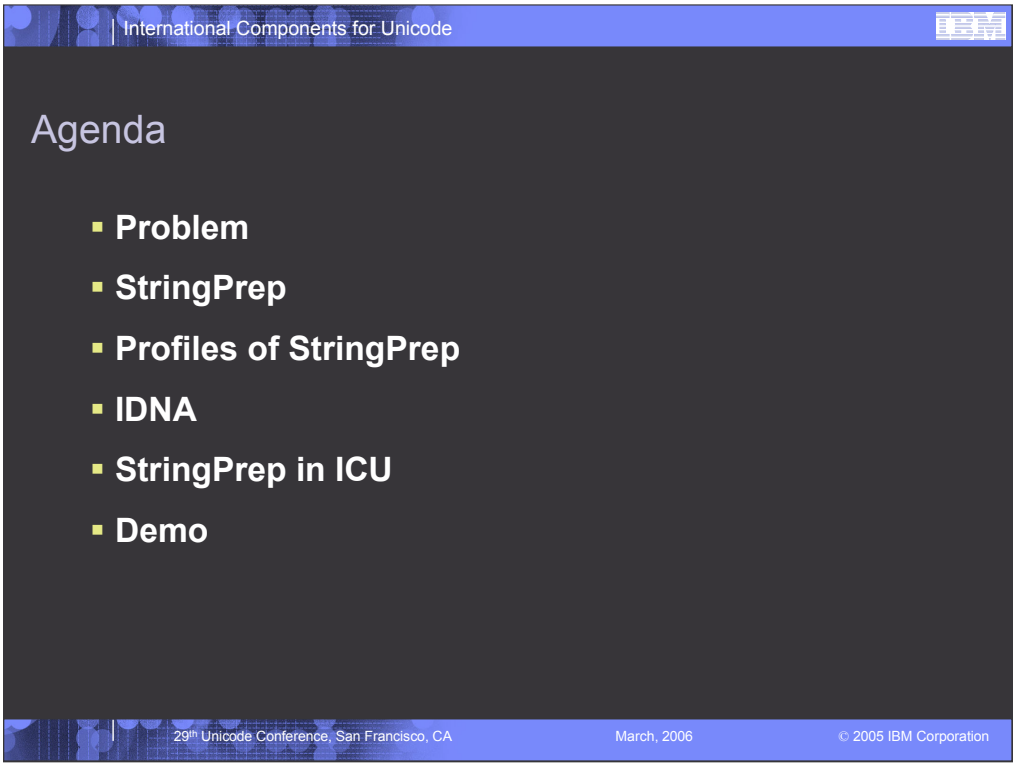
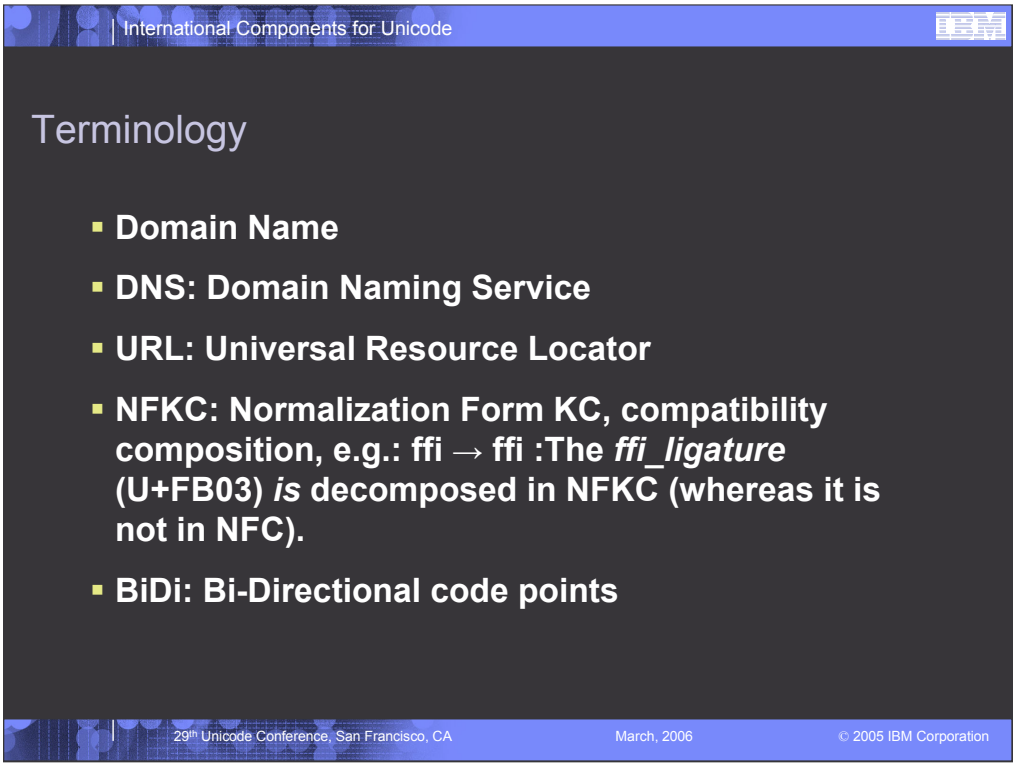


Raghuram (Ram) Viswanadha joined the ICU team in 2000. He has worked with projects at FedEx and Lotus in distributed learning management systems, and been associated with Lotus LearningSpace 4. His contributions in ICU include various character set converters, Indic transforms, StringPrep & IDNA implementation, tools for XLIFF support and UResourceBundle. He is also actively involved in Common Locale Data Repository (CLDR) project. He holds B.E in Electrical Engineering, M.S in Robotics and is currently pursuing a MBA at Santa Clara University.



Network protocols require consistent comparison of strings. The StringPrep framework (RFC 3454) facilitates this function. It provides sets of rules that can be applied to strings to prepare them for use in any protocol or program. Each system sets up a profile of StringPrep by selecting a set of rules. This presentation describes StringPrep and important profiles such as NamePrep, NFS, ResourcePrep, NodePrep. The usage of StringPrep and IDNA frameworks is illustrated by implementation in International Components for Unicode (ICU).



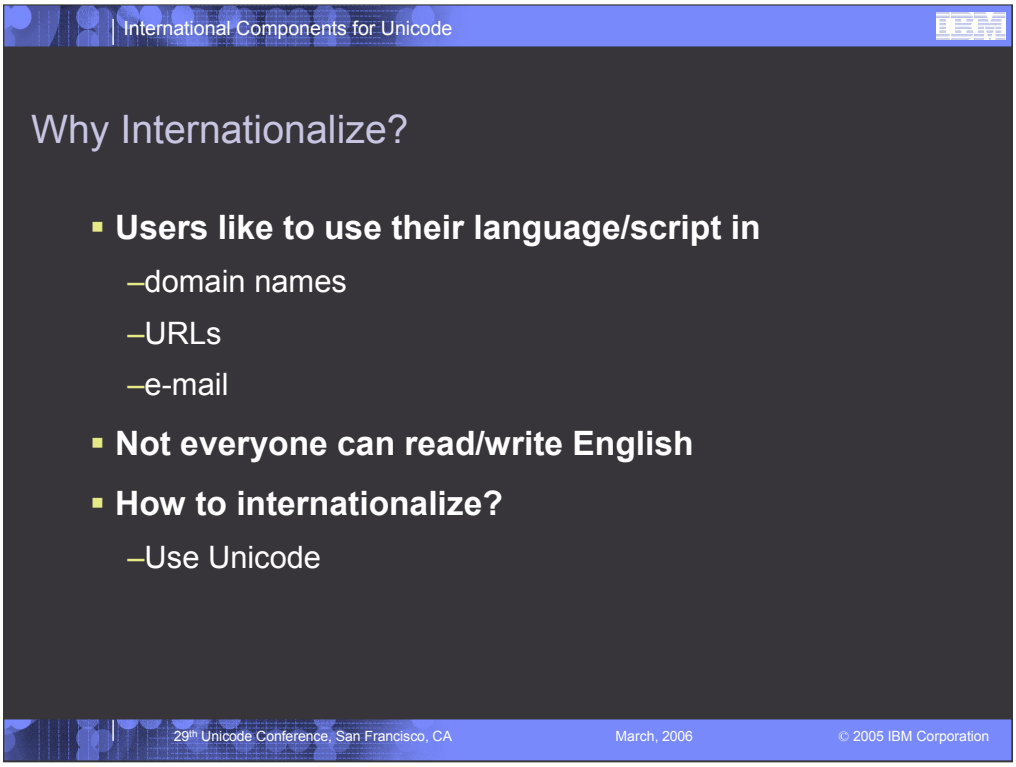
Domain Name: The human-friendly name that maps to a numeric identifier (IP address) of a computer.

DNS: The protocol that defines the method for mapping the domain name Internet Protocol (IP) address.

URL: www.IBM.com

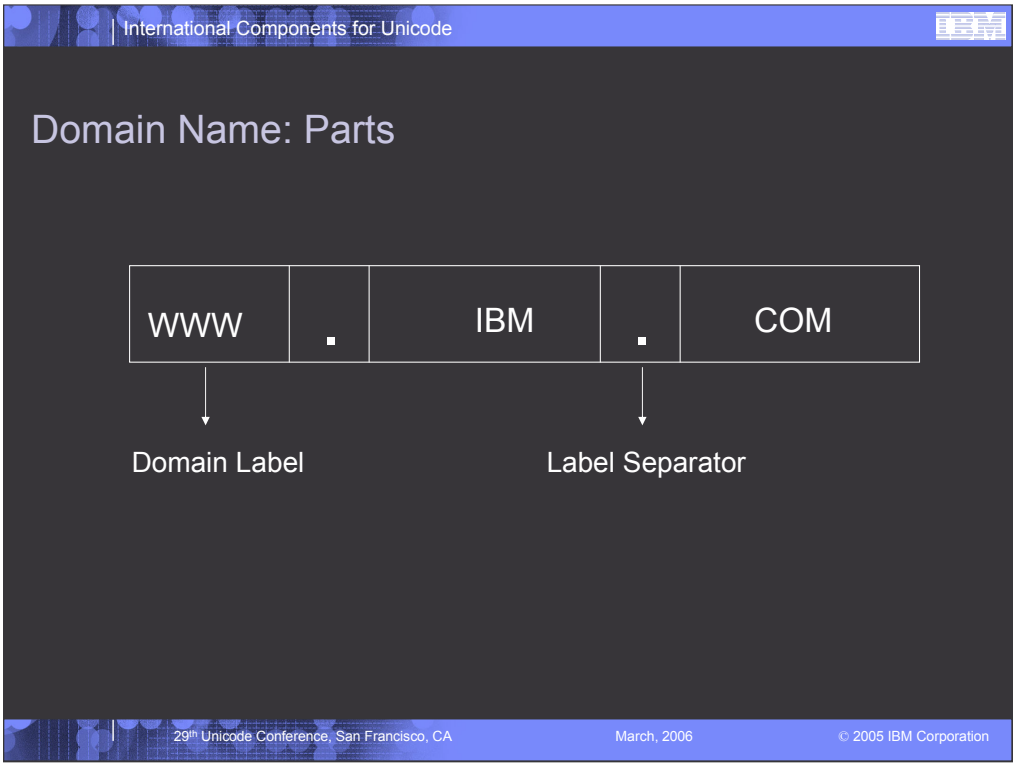
NFKC: Normalization Form Compatibility Composition (NFKC) is one of the four normalization forms specified in UAX#15.

BiDi: Directional Property of scripts in Unicode



Users of the internet needed the capability of using their own language and scripts to access resources on the internet and to communicate (e-mail) using the tools available. Fulfilling this request is not an easy task. The internet functions on the basis of not one but a stack of protocols each interacting with other. If the capability of using multiple scripts needs to be added to internet then it should be added to all layers in the stack that utilize strings to perform their functions starting at the bottom of the stack. The underlying protocol that is essential for accessing resources on internet and for communicating via e-mail is the Domain Naming Service (DNS) protocol. DNS protocol defines the method for associating a name string with an numeric Internet Protocol (IP) address.

The goal is to be able to support domain names such as these.



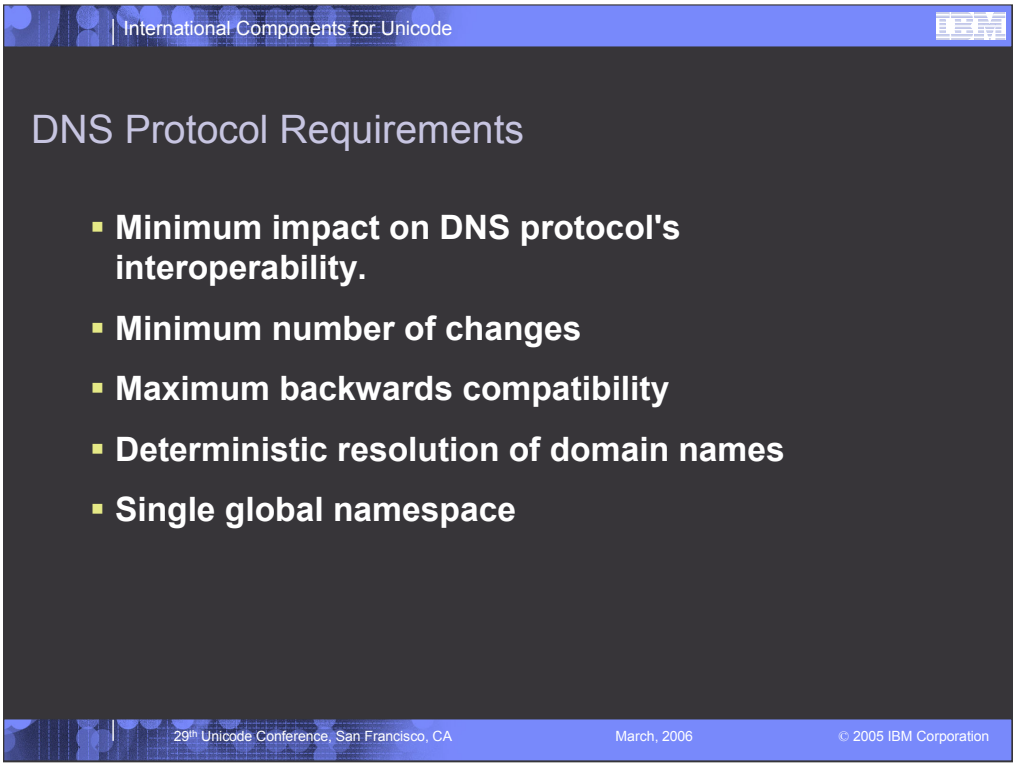
A domain name consists of one or more name parts

•Each name part can be up to 63 bytes in length

•The entire domain name can be up to 255 bytes in length

•A name part must begin and end with a letter or digit from the ASCII character set, and may contain letters, digits, or hyphens.

•Name parts are case-insensitive, so upper-case and lower-case characters are considered equal.



The DNS protocol is backbone of the internet. Adding multiple scripts capability without due consideration to existing hardware and protocol restrictions would be highly destructive and debilitating to the internet infrastructure. A set of requirements were put forth by the IDN working group as noted below:

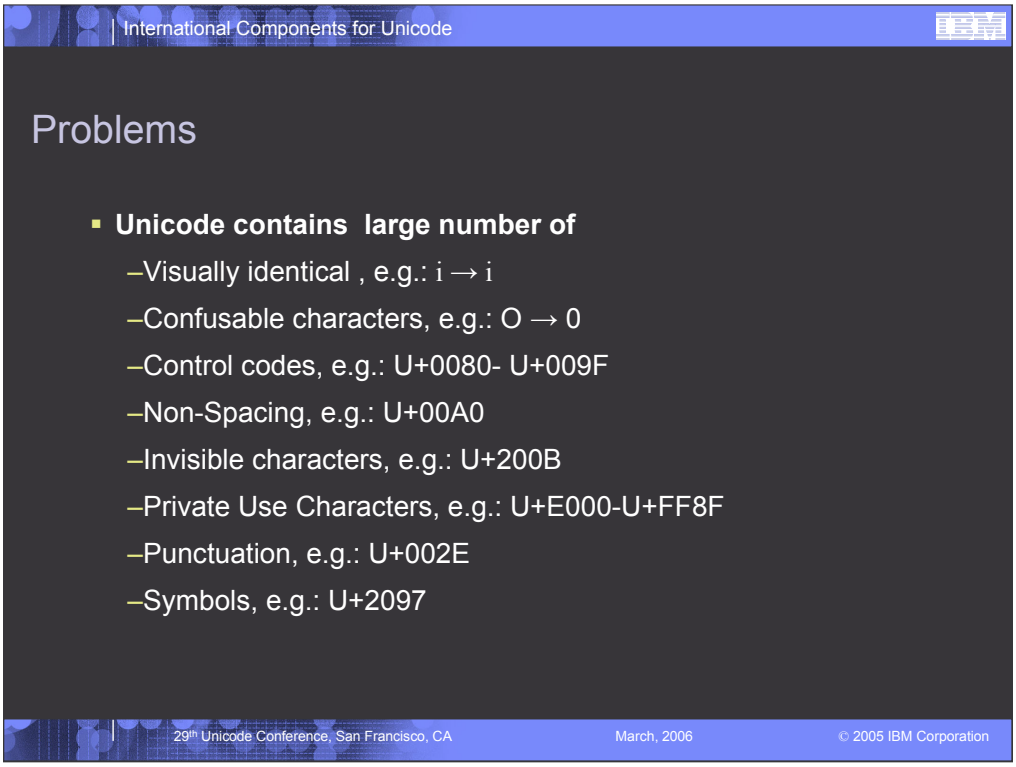
•The service designed must not damage present DNS protocol's interoperability.

•A minimum number of changes to existing protocols on all layers of the stack must be made.

•The service must continue to allow any system anywhere to resolve any internationalized domain name.

•The service must preserve the basic concept and facilities of domain names as described in RFC 1034.

•The same name resolution request must generate the same response and in any server involved in the resolution process.



•Some characters are visually identical thus making it impossible to accurately compare the strings.

•Horizontal and vertical spacing characters would not be visible, so strings that are identical in display may not be equal.

•Non-Spacing characters are invisible, their presence does not effect display but effect the equality operations on strings

•Control characters, formatting characters, tagging characters are also invisible but also effect the equality operations.

•Private use characters are reserved for private use of implementers of Unicode. These characters have no visual or semantic representation. Since the semantic representation is implementation dependent, equality operation of the strings with PUA code points is undefined.

•Some punctuation characters are significant in the syntax of network protocols.

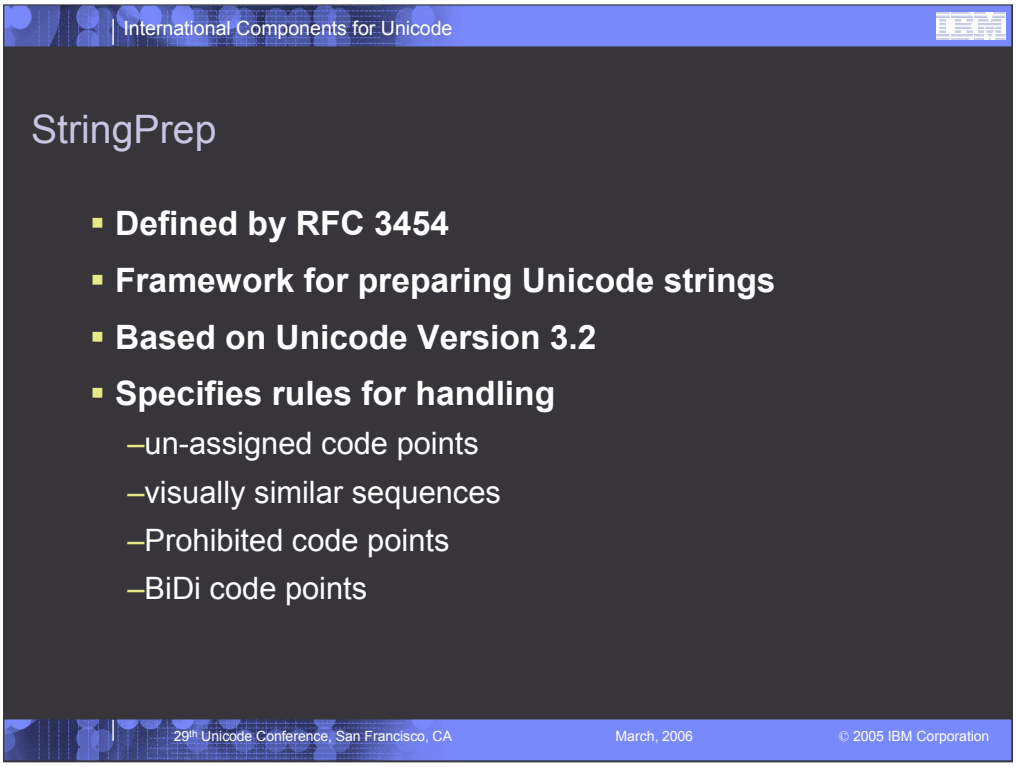
•Symbols and non-name punctuation are disallowed in protocols

Can anyone tell the difference between the above strings?

Both strings are visually equal but when you try to compare them they are not equal.

The first string contains e with a macron.
The second string contains e with a combining macron.

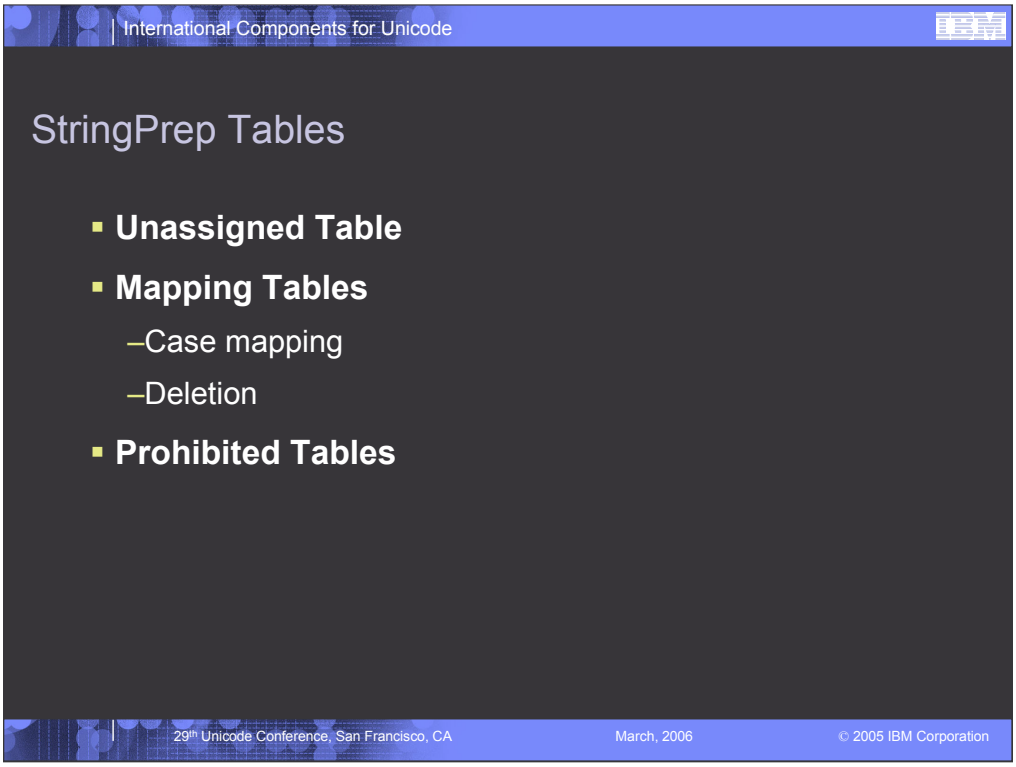
So the first step is to figure out :
•code points/sequences are legal
•code points/sequences produce consistent comparison results



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 prepared. 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.

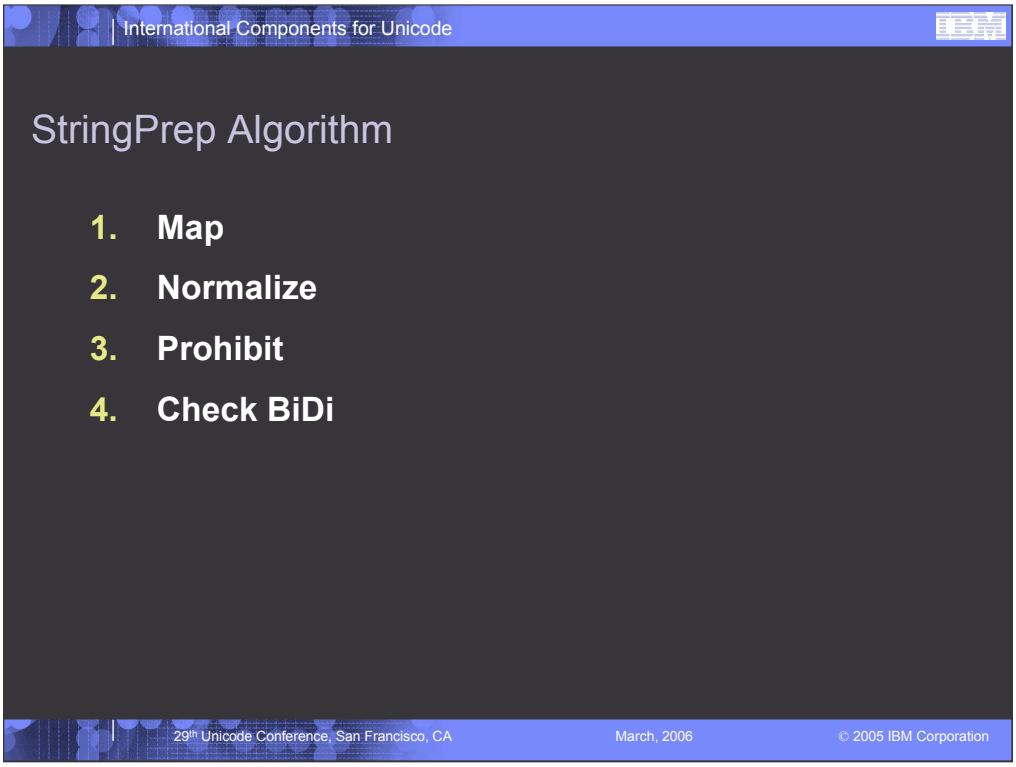
StringPrep uses Unicode Version 3.2 and defines a set of tables for use by the profiles. The profiles can chose to include or exclude tables or code points from the tables defined by the RFC.



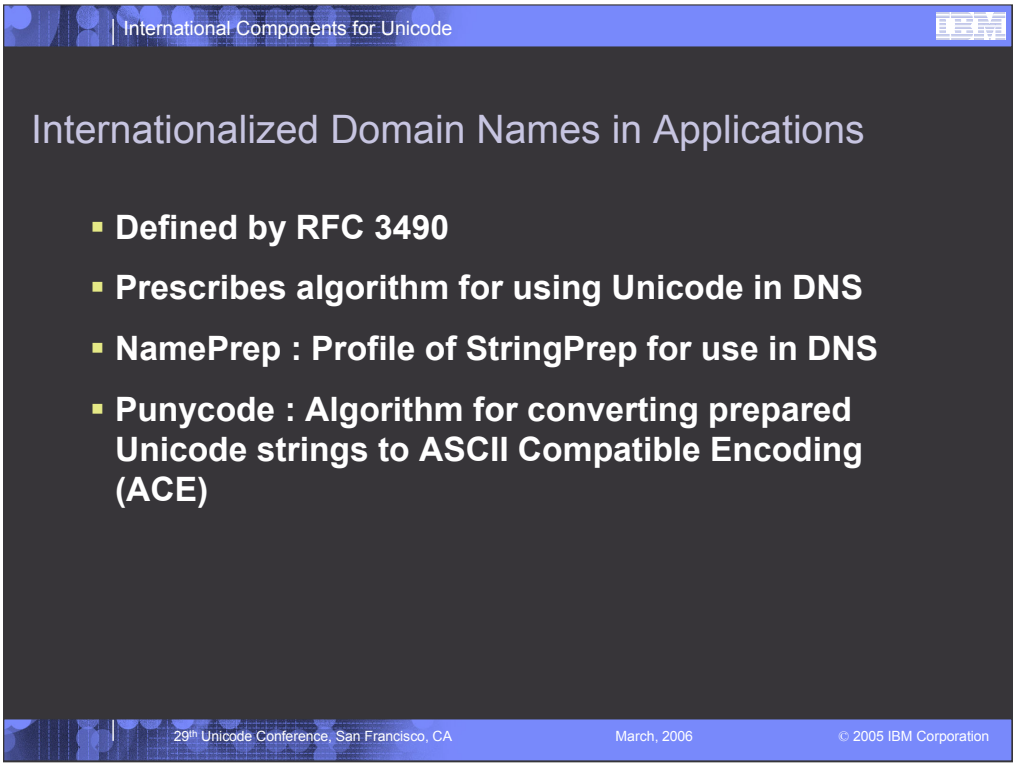
•Unassigned Table: Contains code points that are unassigned in Unicode Version 3.2. Unassigned code points may be allowed or disallowed in the output string depending on the application. The table in Appendix A.1 of the RFC contains the code points.

•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

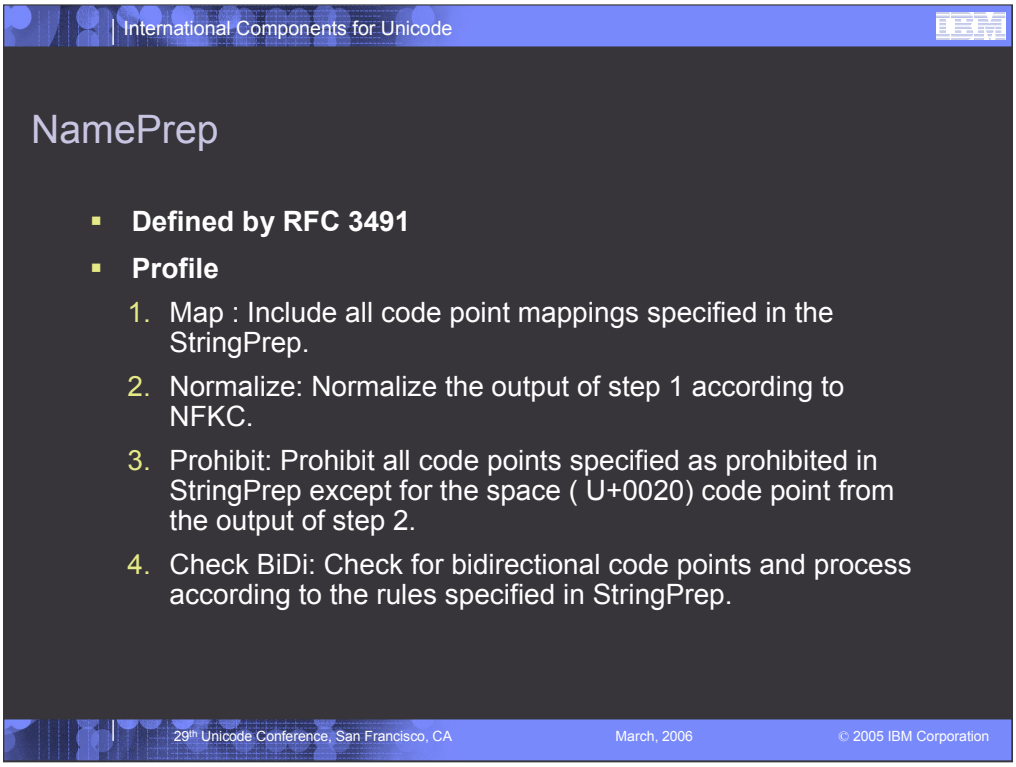
•Prohibited Tables: Contains code points that are prohibited from the output string. Control codes, private use area code points, non-character code points, unmatched surrogate code points, tagging and deprecated code points are included in this table. There are nine mapping tables in Appendix which 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.



- 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.
- Normalize: Normalize the output of step 1 using Unicode Normalization Form NFKC, it the option is set. Normalization algorithm must conform to UAX 15.
- 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.
- 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.

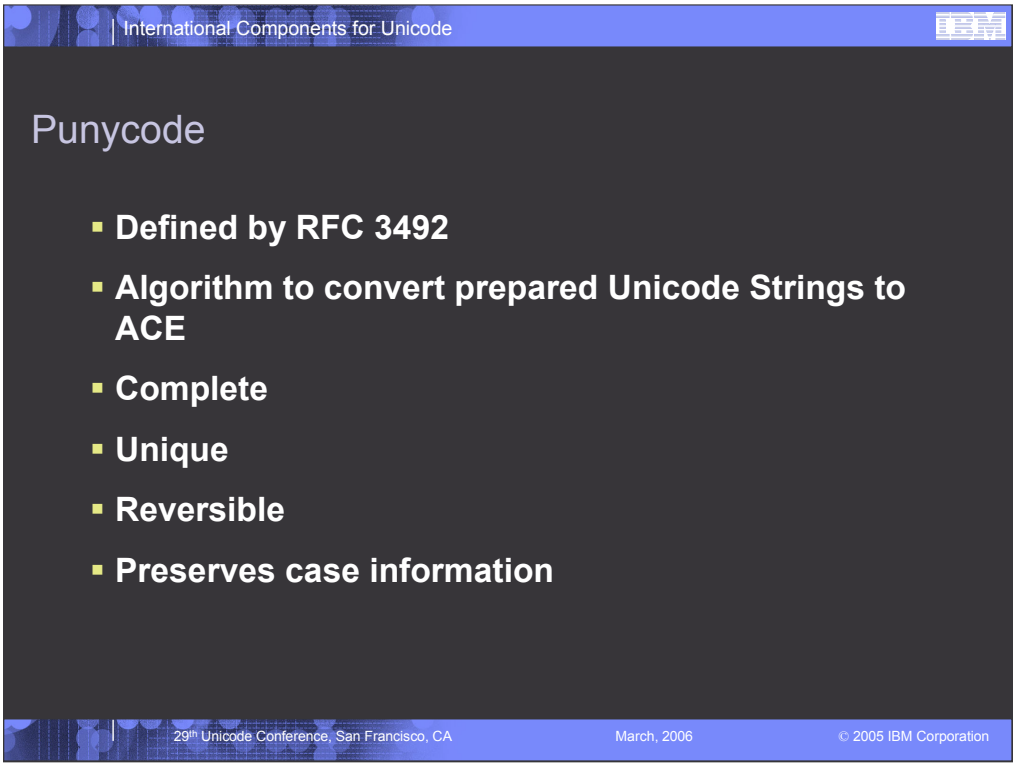


The Domain Name Service (DNS) protocol defines the procedure for matching of ASCII strings case insensitively to the names in the lookup tables containing mapping of IP (Internet Protocol) addresses to server names. When Unicode is used instead of ASCII in server names then two problems arise which need 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.



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

The profile specifies the rules described above.



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 casing feature, however, is not used in IDNA.

According to the RFC, a bootstring algorithm exhibits the following characteristics

•Completeness: Every extended string (sequence of arbitrary code points) can be represented by a basic string (sequence of basic code points). Restrictions on what strings are allowed, and on length, can be imposed by higher layers.

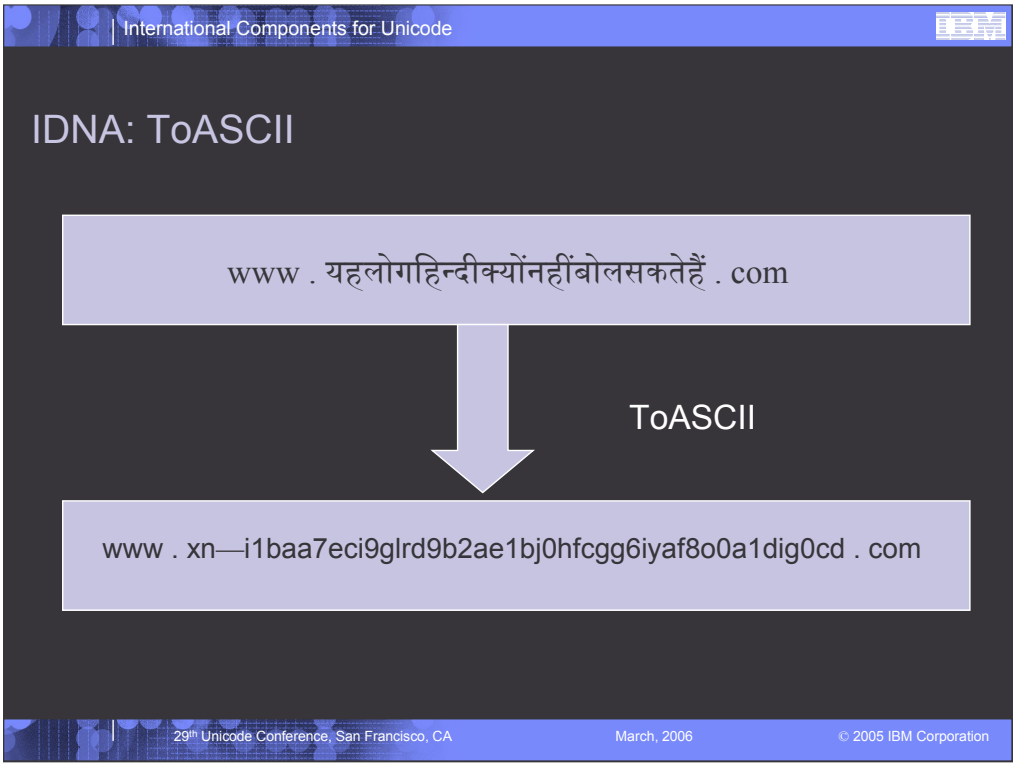
•Uniqueness: There is at most one basic string that represents a given extended string.

•Reversibility: Any extended string mapped to a basic string can be recovered from that basic string.

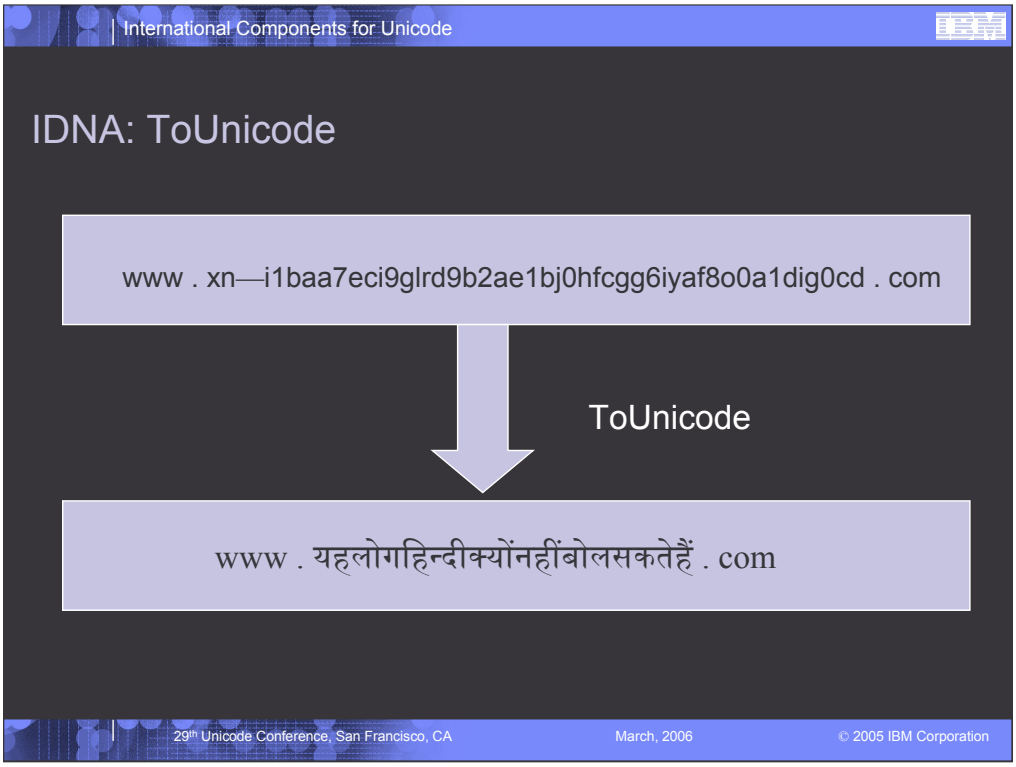
•Efficient encoding: The ratio of basic string length to extended string length is small.

•Simplicity: The encoding and decoding algorithms are reasonably simple to implement.

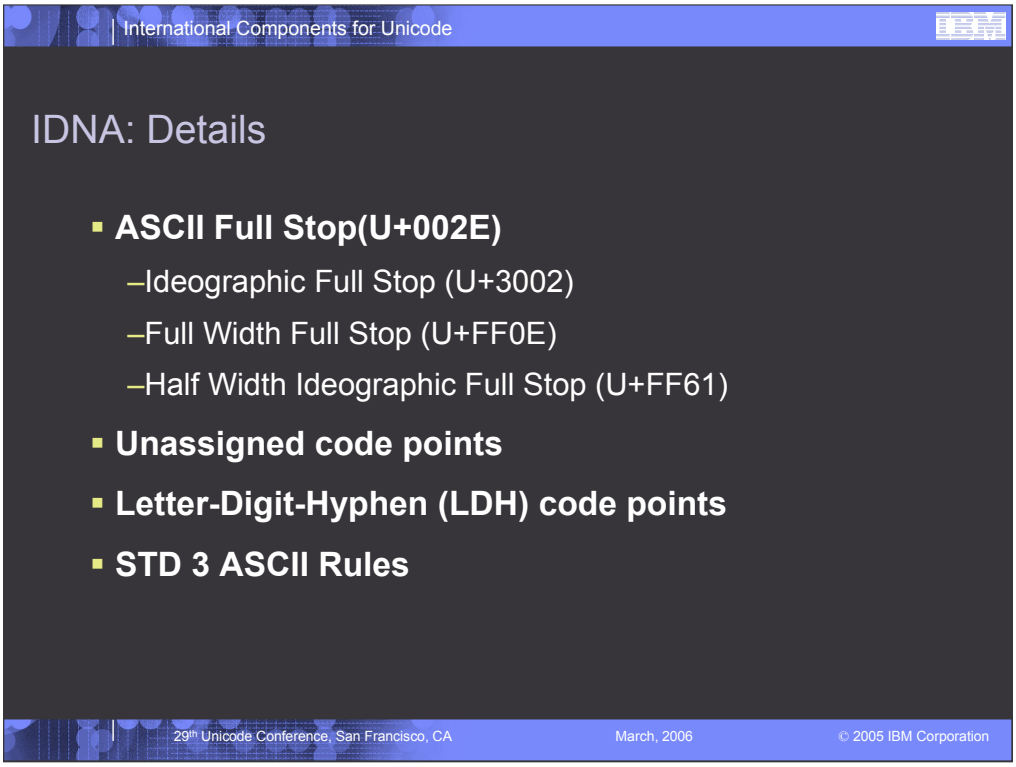
•Readability: Basic code points appearing in the extended string are represented as themselves in the basic string



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 the encoded label.



ToUnicode: This operation is performed on domain labels before displaying the names to 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 applyin g ToASCII operation and comparing the output with the input to the ToUnicode operation.



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)

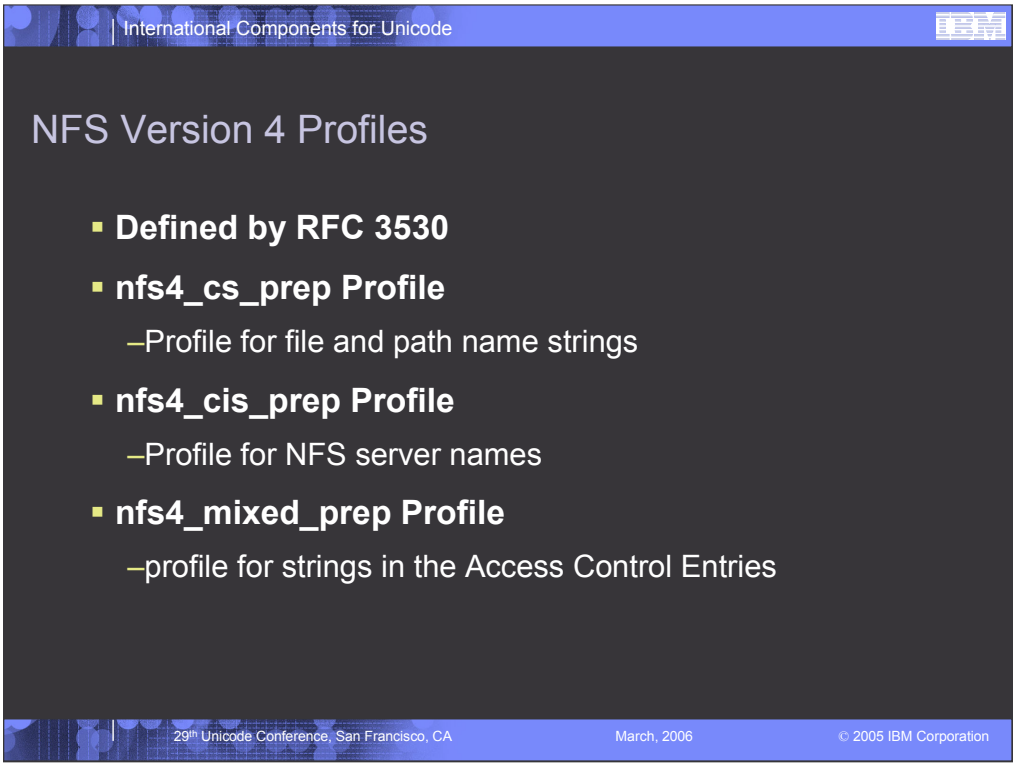
Full Width Full Stop (U+FF0E)

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.

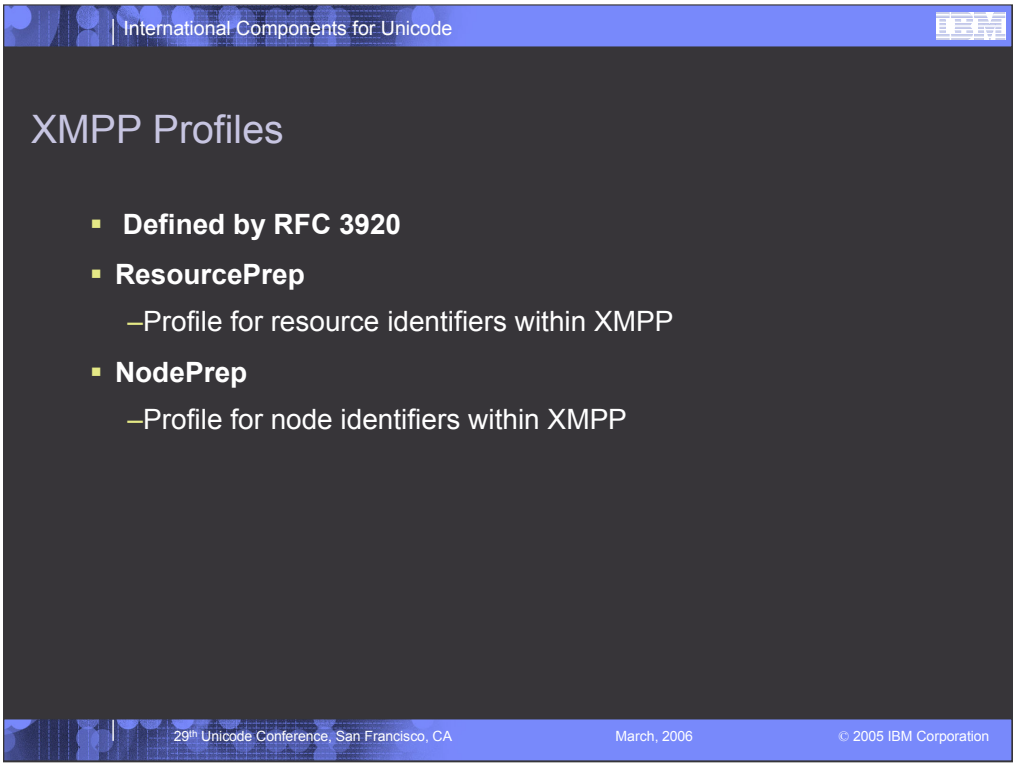
IDNA specifies that domain labels are equivalent if and only if the output of ToASCII operation on the labels match using case insensitive ASCII comparison.



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.

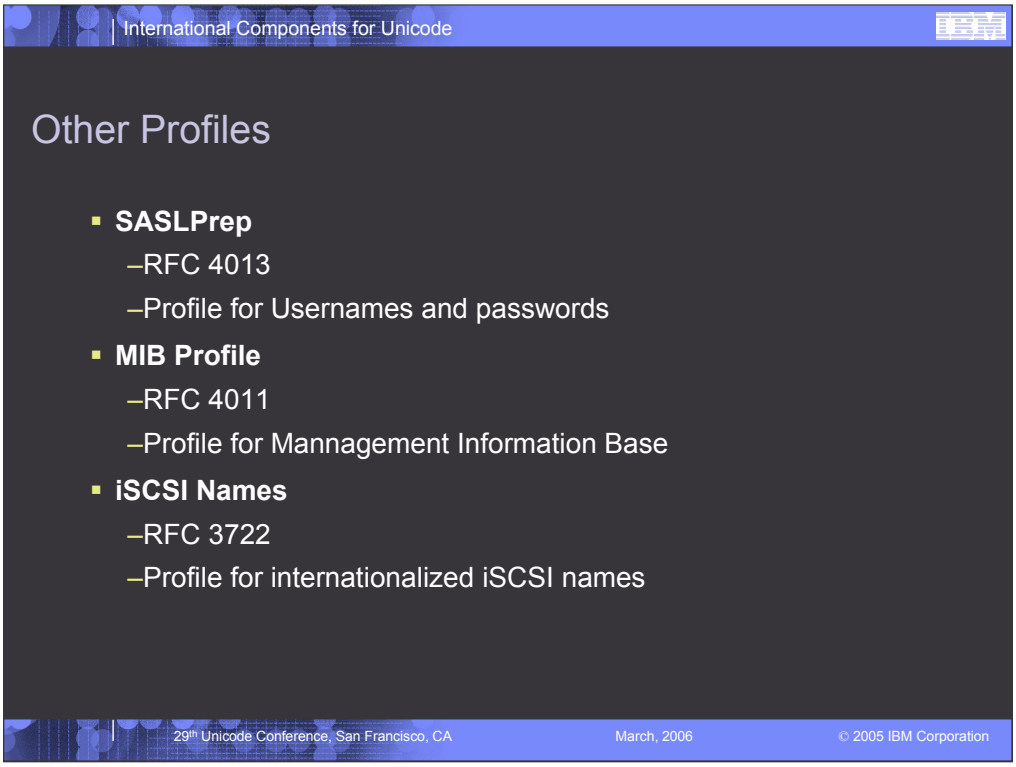
The RFC defines three profiles :

1. 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.
2. 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.
3. 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.



Extensible Messaging and Presence Protocol (XMPP) is an XML based protocol for near real-time extensible messaging and presence and defined by RFC 3920 (http://www.rfc-editor.org/rfc/rfc3920.txt) This protocol defines use of two StringPrep profiles:

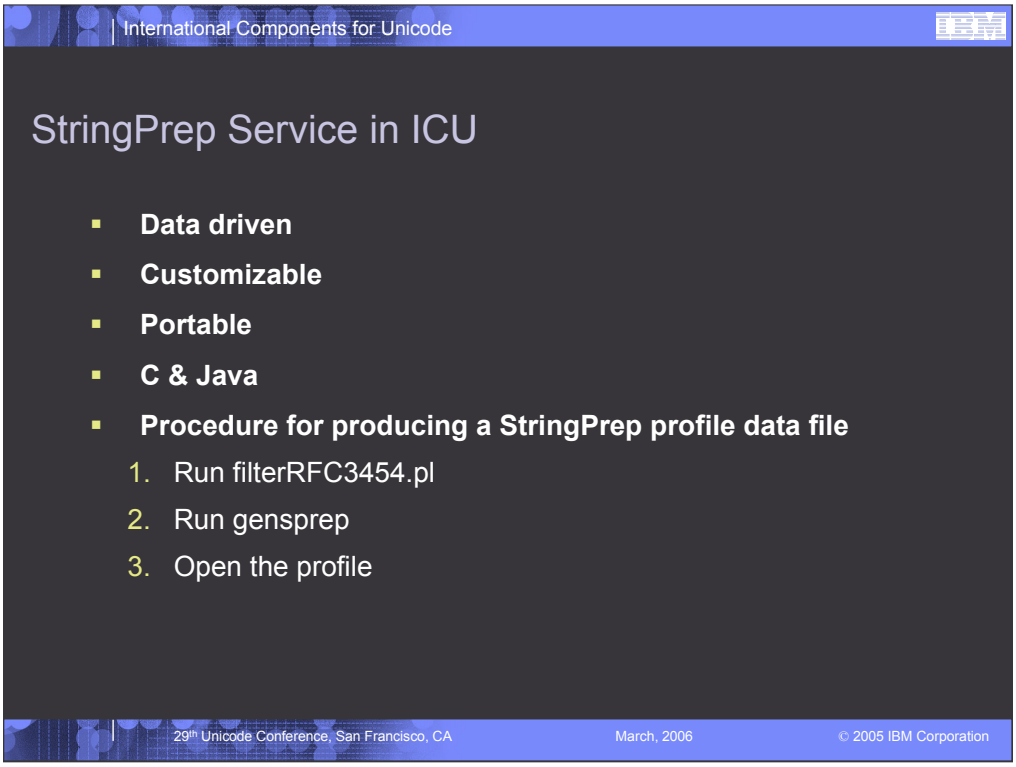
1. ResourcePrep Profile: This profile is used for processing the resource identifiers within XMPP. Normalization of code points and checking of bidirectional code points are turned on. Case mappings are excluded. The space code point (U+0020) is excluded from the prohibited code points set. The resource identifier is an optional tertiary identifier placed after the domain identifier and separated from the latter by the '/' character.
2. 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. The node identifier is an optional secondary identifier placed before the domain identifier and separated from the latter by the '@' character.



SASLPrep: This profile is intended to be used by Simple Authentication and Security Layer (SASL) mechanisms (such as PLAIN, CRAM-MD5, and DIGEST-MD5), as well as other protocols exchanging simple user names and/or passwords.

MIB Profile: Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. MIB objects are generally accessed through the Simple Network Management Protocol (SNMP). Objects in the MIB are defined using the mechanisms defined in the Structure of Management Information

iSCSI Name: The Internet Small Computer Systems Interface (iSCSI) protocol provides a way for hosts to access SCSI devices over an IP network. The iSCSI end-points, called initiators and targets, each have a globally-unique name that must be transcribable, as well as easily compared.

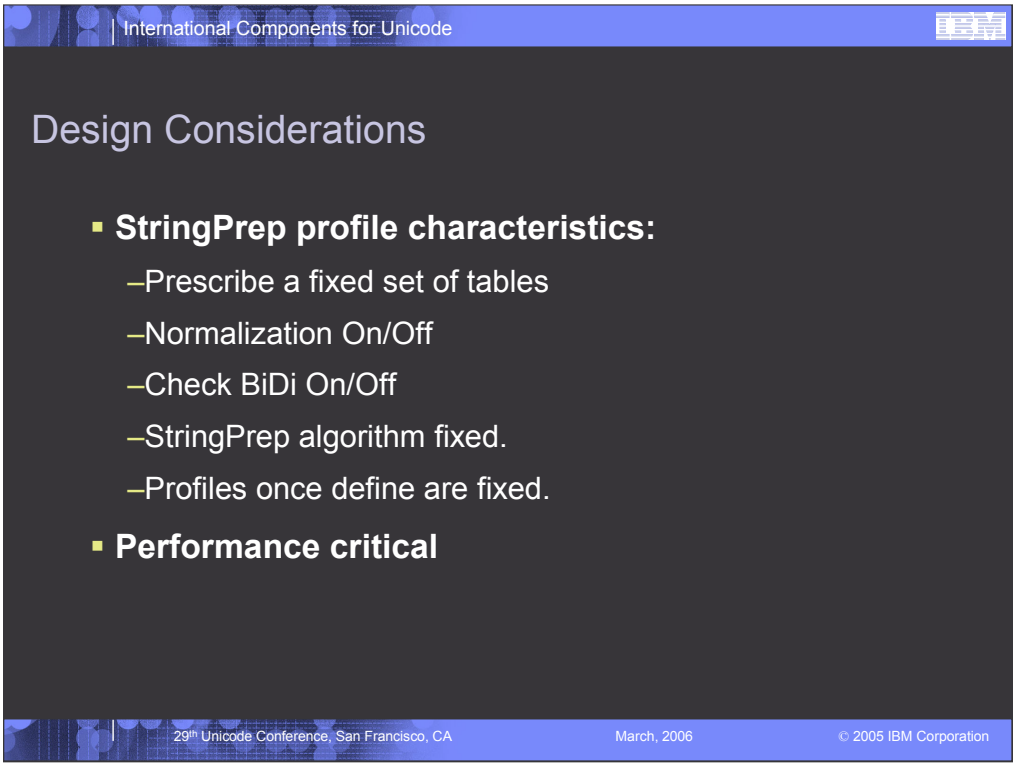


The StringPrep service in ICU is data driven. The service is based on Open-Use-Close pattern. A StringPrep profile is opened, the strings are processed according to the rules specified in the profile and the profile is closed once the profile is ready to be disposed.

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 are 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.



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.

C

```
UErrorCode status = U_ZERO_ERROR;
UParseError parseError;
/open the StringPrep profile */
UStringPrepProfilenameprep = 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
public  NFSCSIStringPrep (){
    try{
        InputStream  nfscsiFile = this.class.getResourceAsStream("nfscsi.spp");
        nfscsi = new StringPrep(nfscsiFile);
       nfscsiFile.close();
    }catch(IOException e){
       //handle the exeption
     }
}
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");
}
```

A web application at http://www.ibm.com/software/globalization/icu/demo/domainillustrates the use of IDNA API. The source code for the application is available at

http://dev.icu-project.org/cgi-bin/viewcvs.cgi/icuapps/idnbrowser/.

News article

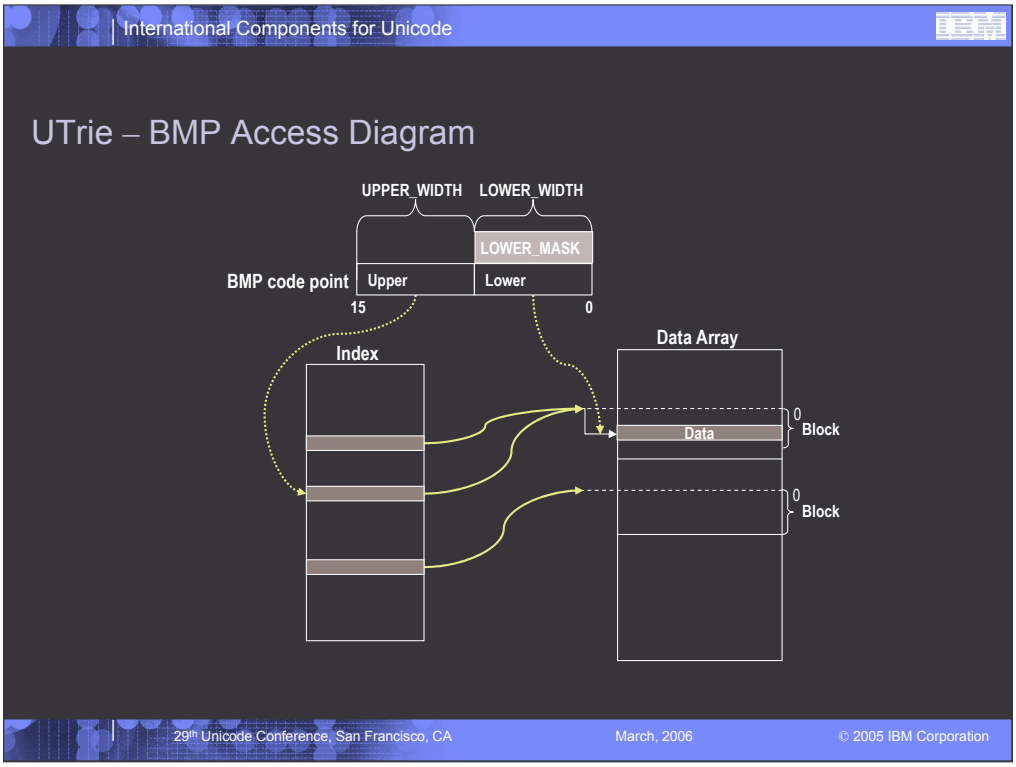
China's Ministry of Information Industry revamps Internet domain names system

Shanghai. February 27. INTERFAX-CHINA - China's Ministry of Information Industry (MII) announced Monday it will reform the country's Internet domain name system, which will be enforced from March 1, 2006. The revamp was launched in accordance with China's administrative measures on Internet domain name system.

中国　公司　网络

dot china, dot com and dot net in Chinese language part of new domain name system revamp

The new domain names system consists of a total of 4 Country Code Top Level Domains (ccTLD) including the English language domain .CN and 3 Chinese-character top-level domains "中国" (.China), "公司" (.com)- in China .com is used to refer to companies, as previously only companies could register .cn domains –the .co.cn sub domain system is not in use in China, and "网络"(.net).



This kind of trie consists of an index array and a data array. The index array holds indexes of data blocks in the data array. The upper part of the code point is used to access the index array. The result is an offset of a data block from the start of the data table. The lower part of the code point is an offset from the start of the data block where the data for the given code point is stored.

The number of elements in the index array is 2^UPPER_WIDTH. The size of each data block is 2^LOWER_WIDTH. The widths of upper and lower parts can be adjusted for the best compression. A popular value is 11 bits for the upper part and 5 bits for the lower part, giving a reasonable compromise of index size versus data block size.

Access to the data corresponding to a particular code point is achieved as :

```
value = data[index [cp>>LOWER_WIDTH]+(cp&LOWER_MASK)]]
```

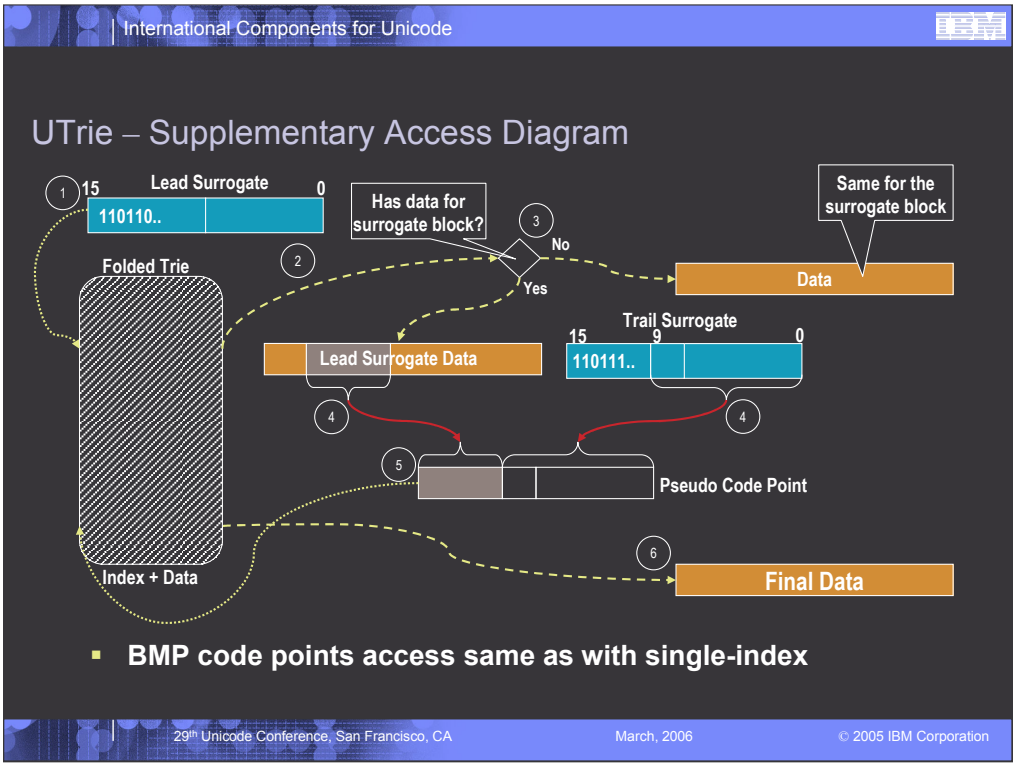
The access time is slightly higher than the access time for arrays.

However, the main value of a single-index trie is its size when storing sparse and/or repetitive data - which is the nature of Unicode data. The index array allows the data blocks to be anywhere in the data array. Therefore, all the blocks that are filled with exactly the same value (for example combining class zero) will physically be represented as one single block. Also, the blocks that overlap some of their values can overlap in the data array.

Single-index tries were almost ideal for storing Unicode data until the supplementary space was defined. However, the introduction of supplementary code points made their usage complicated and/or infeasible. First, with the same **LOWER_WIDTH**, the index array is 17 times as large. A larger **LOWER_WIDTH** could counter that at the expense of compressibility. The growth of index size is problematic since most of the supplementary space is not populated, therefore only about one fifth of the space taken by the index table carries meaningful data. Also, the index array used by ICU is 16 bits wide, which is enough for BMP data, but with supplementary characters there may be more than 64K values in the data array.

The folded trie keeps the low access time for frequent code points, while keeping the table compact. Its design rests on two important properties of the code point distribution in the Unicode standard. First, the BMP is densely populated, while the supplementary space is sparsely populated. Second, in the vast majority of the processes that involve Unicode data, BMP code points are accessed much more frequently than supplementary code points.

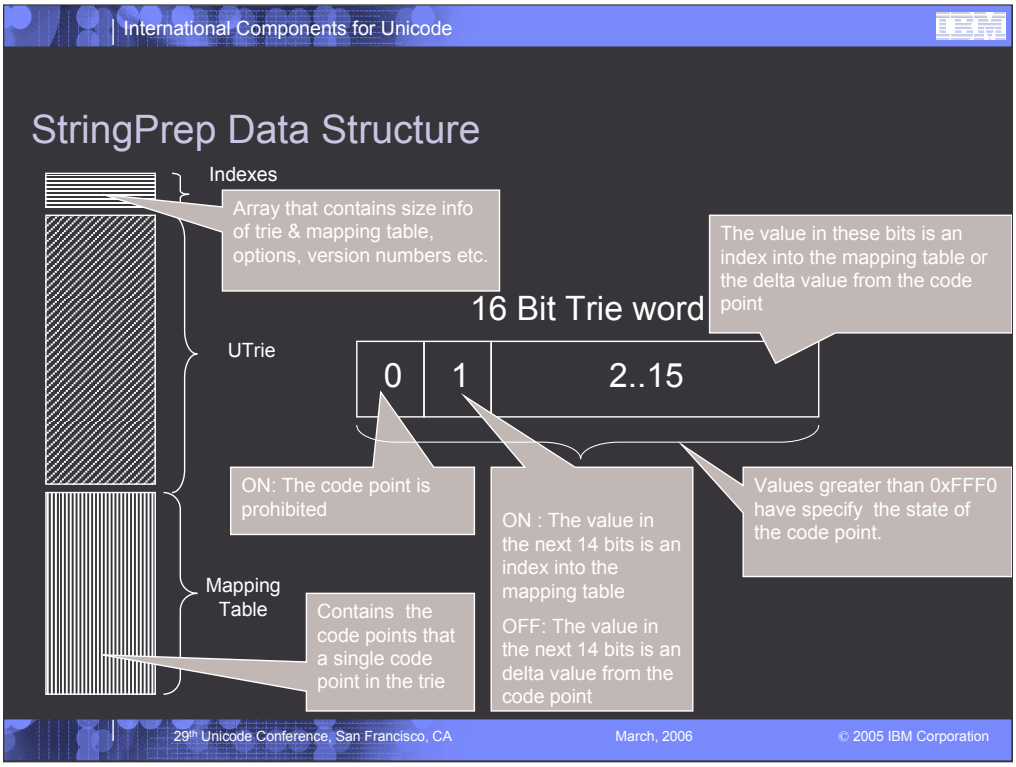
The folded trie structure is a single indexed trie structure. It is addressed by the UTF-16 code units. The key advantage of the folded trie is that it encodes supplementary values using a pair of surrogate values. The value addressed by the lead surrogate contains a new offset to the index table that is combined with the value of the trail surrogate to address the value for that particular code point. The size gain is in the index array, as it is no longer necessary to keep around indexes for all the unassigned supplementary code points, but only for ones that actually contain useful data.



UTrie clearly separates the build and the run time stage. During the build stage, added data is addressed by UTF-32 code points. The whole index is allocated. During the build phase, UTrie can be queried if the data lookup is needed. After the data entry is finished, the process of compaction and folding takes place.

Access to the data for BMP code points is the same as in single-index trie.

Access for supplementary code points is in two phases. First, the lead surrogate is used to access the data (as in a normal single-index trie). The data for the lead surrogate contains a flag that says that it is in fact a special piece of data (in order to distinguish it from the first surrogates that have no data in the trail surrogate block) and piece of data that gets combined with the lower ten bits of the trail surrogate in order to get a new pseudo-code point that addresses the data for the supplementary code point.



StringPrep Trie :
The StringPrep tries is a 16-bit trie that contains data for the profile.
Each code point is associated with a value (trie-word) in the trie.

- structure of data words from the trie

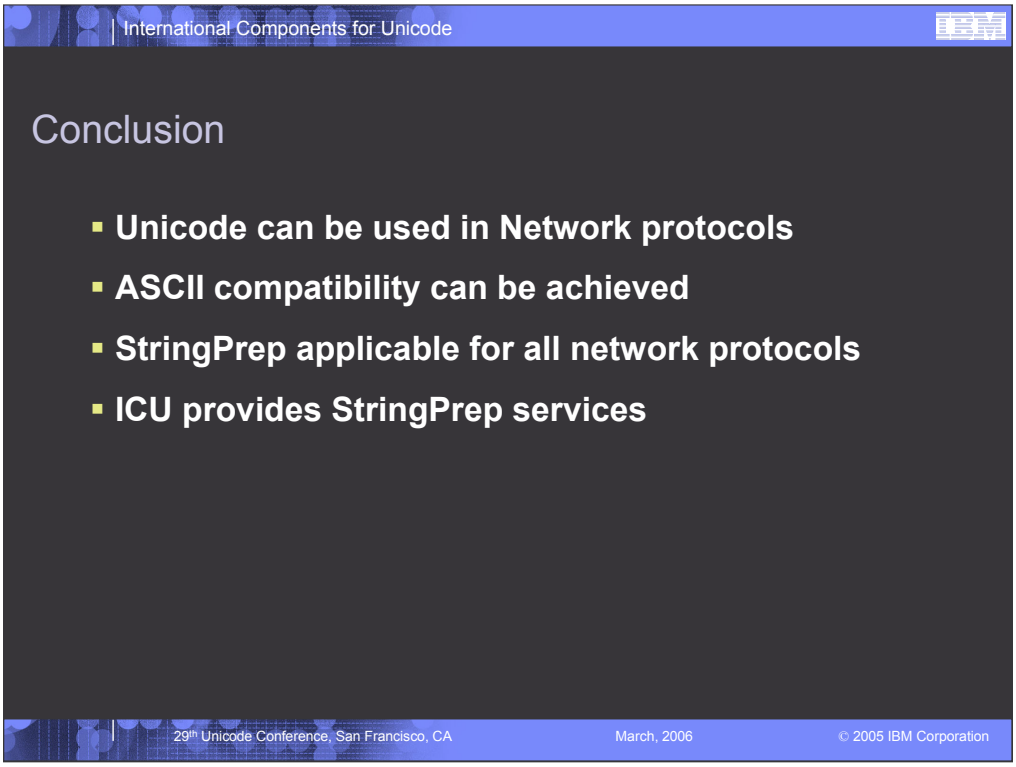
i) A value greater than or equal to _SPREP_TYPE_THRESHOLD (0xFFF0)
represents the type associated with the code point
if(trieWord >= _SPREP_TYPE_THRESHOLD){
type = trieWord - 0xFFF0;
}
The type can be :
USPREP_UNASSIGNED
USPREP_PROHIBITED
USPREP_DELETE

ii) A value less than _SPREP_TYPE_THRESHOLD means the type is USPREP_MAP and
contains distribution described below

0 - ON : The code point is prohibited (USPREP_PROHIBITED). This is to allow for code points that are both prohibited and mapped.
1 - ON : The value in the next 14 bits is an index into the mapping table
OFF: The value in the next 14 bits is an delta value from the code point
2..15 - Contains data as described by bit 1. If all bits are set
(value == _SPREP_MAX_INDEX_VALUE) then the type is USPREP_DELETE

Mapping Table:
The data in mapping table is sorted according to the length of the mapping sequence.
If the type of the code point is USPREP_MAP and value in trie word is an index, the index
is compared with start indexes of sequence length start to figure out the length according to
the following algorithm:
*

```
if(     index >= indexes[_SPREP_ONE_UCHAR_MAPPING_INDEX_START] &&
        index < indexes[_SPREP_TWO_UCHARS_MAPPING_INDEX_START]){
    length = 1;
}else if(index >= indexes[_SPREP_TWO_UCHARS_MAPPING_INDEX_START] &&
        index < indexes[_SPREP_THREE_UCHARS_MAPPING_INDEX_START]){
```



The StringPrep algorithm demonstrates that use of Unicode in protocols designed for ASCII is possible without a replacing the hardware currently in use.

The StringPrep framework is applicable not only for DNS protocols but also to a number of other protocols that need to use Unicode instead of ASCII.

ICU provides services that implement the StringPrep framework and algorithm. Clients can implement profiles of StringPrep easily by using ICU.

References

- **Moving Towards Internationalized Domain Names**
  - –Paul E. Hoffman
- **A Tangled Web: Issues of I18N, Domain Names, and the Other Internet protocols**
  - –RFC 2825
- **Multilingual Domain Name Race**
  - –Suzzanne Topping

Q & A