Internet Draft
draft-ietf-idn-step-01.txt
Sept. 28, 2001
Obsoletes: draft-ietf-idn-step-01.txt
Expires in six months (March 2002)

Liana Ye
Y&D ISG

StepCode - A Mnemonic Internationalized Domain Name Encoding

Status of this memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsolete by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

This document describes an Internationalized Domain Name (IDN) Encoding method with US-ASCII [a-z0-9] characters, preserving the primary sound value of such names users want, and technically feasible, linguistically demanding once mechanism to represent the names of multi-scripts with language tags defined by [ISO 639] in the required DNS way, such that the encoded names can be used as valid domain name identifiers.

Table of Contents

1. Introduction

Symbolic representation of a concept takes on many forms. It can be encrypted to conceal from a human reader, it can be compressed for a mechanical program reader, and it can be an icon for any spoken language readers. For a domain name represents an entity as an individual, a product, or an organization, it has to be readable for human readers both in their native languages as well as for human readers not in that native languages, in addition to a computer program reader which only reads code points. To bridge the three types of requirement, StepCode is proposed to transform a native symbol to one or more universal ASCII symbols in a mechanical manner for a mechanical program reader.

1.1 Context

Although world-wide desire to use characters other than plain ASCII in hostnames is bubbling up and accelerating, ICANN has to take a cautious approach on adopting an international domain name system,

for the fear of duplicated or confused new domain names. The challenge of how to represent the names users want in the DNS in a way that is clear, technically feasible, and unique is still an open issue.

1.2. Issues in Multilingual Representation of DNS Host Names

A basic technical issue regarding a name is sorting and searching zone files or name servers of hostname identifiers containing different written languages for potentially very large numbers of users online, say 10% of the world's population. Hostname identification could become a bottleneck for internet traffic if sorting and searching has to be treated 1) in more than one set of partially overlapping or mixed or possibly mixed symbolic representations; and 2) mostly in compressed or semantically random ordered zone files scattered around the globe, as in the Shared Registration System (í SRSí) since 1999 installation.

Historically, Character-formed script such as CJK characters has inherent sorting and indexing difficulties and is used to be an intellectual activity just to use a dictionary. In fact, it has been a primary problem in computer processing of Oriental languages since the early development of computer industry. After almost importunate research and development in the past decades, the solution are all based on some types of table search, and the nature of such a processing has been well understood, and the techniques are ready to be applied to very large character set, such as Universal Character Set [UCS].

With the experiences we have obtained from Oriental languages processing, and suppose that we have solved such an indexing problem and have accommodated mixed scripts such as Japanese and Korean, and IDN goes to a character-form based system, then it is foreseeable that IDN system will have to support a text based DNS system as well for a long time. After all, the DNS system is a historically successful system. To throw such a system away is like asking people to stop shopping at supermarkets and pick up their lettuce on the Internet. Then it is certain, that we have to deal with two sets of domain name identifiers for a long time ahead.

The Romanized Pinyin, Jamo and On-kun systems for CJK character indexing has provided a feasible but partial solution. The currently used complete solution is to go through a software process of both searching tables for possible matches (not exact-match DNS lookups) and, where necessary, dialogue with the users, and arrive at strong candidates for the glyph representation. If this character selection process is organized in a similar way with book indexing system, alphanumeral-digits-digits..., used a North American library, then the indices can be codified using Latin alphabet. The dream of a complete Romanized character system will be reality, sorting and searching international domain names with one set of symbolic representation will be speedy, and exactly matched DNS lookups could

be a reality.

1.3. Romanized Multi-language Representation

Codifing a trade name representation process is not limited to codify a particular ASCII Compatible Encoding method or a particular code mapping from one code standard to another code standard in a technical context. It shall codify one set of symbols, or one representation system, and a number of efficient paths to let the users have some freedom to decide how to use the system to express their own trade names in the Internet context. Though this was the sprit of ASCII standard, it is the time to set more specific paths on how to use ASCII to represent different scripts of spoken languages, or to codify such a representation process, so that the number of paths does not head for combinatorial explosion, as it is the case in Chinese character encoding methods and for Japanese input systems. This is analogous to let students tread out a optimal path on campus before a concrete walk is poured, and it is our time to codify the paths.

Representation system for trade names is due to be unified. In fact, writing system unification has been seen with Arabic, Latin and Chinese. Many different spoken language groups use each of them. According to [DeFrancis 1989], human scripts can be organized into three groups for their phonetic characteristics:
1. Syllabic systems, for example, Chinese, Japanese, Maya and Yi;
2. Consonantal systems, such as Hebrew, Arabic and Indian languages;
and 3. Alphabetic systems, including Greek, Latin, Cyrillic, Korean Hangul and English. Alphabetic systems can be unified by embedding some differences under the hat of mnemonic representation of language symbols, so that the French 'u' is permitted to have a different sound value from the English 'u'.

Mapping a consonantal system to an alphabet symbol set is, essentially embedding some phonetic differences, using a Latin mnemonic hat. Additionally, there is the question on how to represent the vowels of the language. Turkey has provided an answer to this question, and Library of Congress has implemented extensive set of languages using the same principle [Translit 97].

As to unifying a syllabic system with an alphabet system, two issues need to be addressed. The first is the inclusion of additional character information which can not be expressed with an one-layer type of a flat alphabet system. The second issue is the reversibility from the alphabetic system back to the syllabic system.

1.4. StepCode Protocol to Represent Trade Names

The proposed solution is called StepCode, for its staircase type architecture in a transliteration procedure. First, it specifies the phonetic differences to be embedded in the representation, where an International Phonetic Alphabet [IPA] description of the embedded

differences shall be recorded. Second, if the Romanized embedding is not sufficient to cover the differences, such as tones, suprasegmentals and diacritics, then extend the mapping space to a 26x10 table for secondary phonetic elements which can not be embedded under the Latin mnemonic hat. Third, if the 26x10 space is not sufficient, then linearize the symbol by specifying each of its components. This last part may become recursive, or goes down for more steps.

This open-ended procedure not only provides a path to unify a large syllabic or character system with an alphabet symbol set, but also ensures that more semantically specific symbols, such as trademarks and logos, can be represented online and sorted for speedy referencing. In addition, the solution tolerates different viewpoints of the same glyph, such that a CJK character may be accessed by Mandarin Pinyin, Cantonese Wade, or Japanese On-kun, Korean Hangul as well as users of the same dialect creating different expressions in viewing the same glyph.

StepCode protocol does not open doors for trade name chaos. First, there are finitely many different scripts to support particular dialects and expressions. Second, the protocol provides locally available expressions for users to choose from, which also helps in conforming expressions especially in IDN context. Third, although the process allows users of the same dialect creating different expressions in viewing a glyph, as it has been experienced with over 600 variety of Chinese character encoding schemes in the past three decades, it limits the different views of a glyph to a matrix of one to ten cells on one fixed starting point [Ye95], where variations in such a process become predictable and manageable.

Due to its step nature, the representation can (and should) stop for each symbol, as soon as the symbol can be identified within its designated context. For example, the following list of StepCodes for four Chinese characters:

```
xin1qin1jin0        <new>
zhu2ge1ge0          <bamboo>
qing1shui1qing0     <clear>
hua2hua2shi0        <Chinese>
```

Each of these codes uniquely identify a CJK [CJK] character of a UCS [UCS] code point in CJK section using Pinyin spelling. They all have three parts: the first part is Pinyin spelling of the character; the second part is the digit following the Pinyin. The digit indicates the end of a character spelling and its tone mark. The two parts together is the transliteration of a character. The remaining alphanumeral string following the first digit is the third part of StepCode. They are in the same format of character transliteration, and is the radical part of transliteration.

When there is registration calls for the four characters, then the four characters may be combined into one new alphanumeral string:
    "xinzhuqinghua1212qin1jin0ge1ge0shui1qing0hua2shi0".
The list of StepCodes for the above four characters is resulted from two complete iterations of StepCode protocol.

Since it is enough for í xinzhuqinghuaí to identify a well-known name in DNS system, í xinzhuqinghua1212í for a not well-known name, and "xinzhuqinghua1212qin1jin0ge1ge0í for pin-pointing a rarely known name, it is up to the registrant and the a zone manager to register a DNS identifier to be just right length for the user, and to keep the full record for code reversal process, depends on IETF and ICANN decision to support a duel-record system [Uname][IDNmap].

1.5 StepCode Features

The StepCode protocol is fully compatible with DNS specification, yet is mnemonic, friendly multi-language accessible code points, and accommodates mixed script use.

1.5.1 Multi-language access of the same UCS code point

Similar to the method used for searching books in a library, such that CJK characters may be accessed by different language users. For example, the following four Korean characters may be coded as:

```
U+????      sim0sim0         Hanja <ten>
U+2fa5      ni0ni0           Hanja <inside>
U+351a      to0t2o0          Hanhul <to>
U+3747      mot0m2o2t0        Hangul <mot>
 (Note 1: the transliteration is used in [Translit 97], where the í tí ,
  in í motí  should be consistent to a jamo for a Korean sound value.
  Note 2: a hangul may not need to be treated as a CJK character. If
  it is the case, then í toí  and í motí  MUST be unique within all hangul
  symbols.)
```

The two Hanja character are CJK code points used by at least three languages, and Hangul is only used by Korean. When the four characters combined into an DNS name, it takes the following form as its full name:

```
  simnitomot0000sim0ni0t2o0m2o2t0
```

so kr-simnitomot0000sim0ni0t2o0m2o2t0.com can be the DNS name or it may be the full name record to be kept at local registrar and be registered with DNS as í kr-simnitomot.comí . StepCode permits different language tags to access the same glyph in [ISO10646].

1.5.2 Multi-script Accommodation

SeptCode protocol allows mixed scripts to co-exist. For example, the five Kana, diacritic mark and Kanji from Japanese:

```
U+3055      sa            <kana>
U+30fc      1           <diacritic macron>
U+3073      bi            <kana>
U+3059      su            <kana>
U+????      gyo1go0       <Kanji business>
 (Note: Only one radical in a Kanji is coded, since the total number
  of Kanji is much smaller set than Han character set. Thus, one radical
  to be coded may be enough to guarantee a unique code within Kanji.)
```

Due to more complex decoding for Kanji than that of Kana, a delimiter for the two seems needed, so a digit 0 may be required to end the kana section. Thus the DNS name: í sa1bisu0gyo1go0í may be used. This shows, that StepCode protocol can be adapted to many different mix of scripts, and different languages needs different treatments on their scripts.

1.5.3 Fully Compatible with Current DNS

From the Chinese, Korean and Japanese example given above, the host parts have no international glyphs but US-ASCII, and can be a valid entry to DNS, and allows standard compression or security treatment compatible with existing hostnames.

1.5.4 One Mnemonic System

It is one mnemonic system for any scripts in UCS, such that whatever the language that the zone master understands, he can refer to, sort on, and support of a registered IDN name.

1.6 Author's Disclaimer

This document is a guide for implementing mnemonic StepCode protocol for IDN hostname identifiers in a language specific way. It is not a natural language dictionary of any decree. The sound value assignment of script symbol although balanced among several considerations are not intended in anyway to claim any linguistics expertise. The different scripts used by any one particular user group addressed in the document does not dictate the user groupsí¯ choice of any subsets of [ISO10646] symbols.

In addition, the document is bias on five issues:
1) The UCS symbol tabulation structure assumed is bias toward CJK users;
2) The mnemonic sound value is based on IPA classification;
3) The Latin letter value assignment is bias toward English usage;
4) The digits value assignment is bias toward Mandarin usage;
5) The language tag function is bias toward Indian languages.

1.7 Terminology

The key words "MUST", "SHALL", "REQUIRED", "SHOULD", "RECOMMENDED", and "MAY" in this document are to be interpreted as described in [RFC2119].

Examples in this document use the notation from the Unicode Standard [Unicode3] as well as the ISO 10646 names. For example, the letter "a" may be represented as either "U+0061" or "LATIN SMALL LETTER A".

A non-Roman character also is denoted in its Romanized form and followed by its English equivalent word in <>. For example, í zhong <heavy>í without reference to Unicode, due to difficulties in pin down all the code points used in this document from UCS table.

An IPA symbol is presented in [], while it is referred among text. For example, [c] is for IPA sound value í cí , not Latin letter í cí .

StepCode assumes its encoding is language specific, each language as it is defined in [ISO10646], has its mnemonic encoding and is a part of ACE encoding prefixed to ASCII host name only, for example, í krí for Korean, í jaí for Japanese. The encoding is called í language tagí of a DNS host name (for language tag implementation see [IDNmap] Section 3). The DNS host name with such a language tag is called a "language tagged ACE", or "T-ACE".

StepCode converts a list of internationalized characters at a client site into a string of US-ASCII that are acceptable as a host name in current DNS host naming usage. The former are called a list of í IDN identifiersí or a "glyph" for a symbol represented by one code point in [ISO10646] or "glyphs" for a string of glyphs and the post-converted ASCII string is called a "DNS identifier".

[Nameprep] defines Unicode characters mappings, normalizing and exclusions of internationalized host names. The characters from input and in mapping and normalization list is called í IDN-labelí , or IDN input, which includes symbols mapping to null. IDN-label is a super set of IDN identifiers in term of UCS code points.

The "IDN-label" at a client site may be represented by Unicode, GB code, JIS code, BIG5 and others which may contain equivalent information. These code forms are referred as language specific "localized code points", or í local display codesí .

A large script such as CJK or UCS can be classified into three glyph groups:

1) IDN letters: which can be directly mapped onto an alphanumeral symbol under the Latin mnemonic hat, for example, Bopomofo, Kana, Arabic, Bengali, Hebrew, Jamo, diacritics, etc.
2) IDN radicals: a minimum number of frequently used glyphs which are also used as radicals in other glyphs, and often has independent pronunciation, for example, U+2f00 to U+2fd5, U+2e80 to U+2ef3, and others scatted in CJK Plane 0 blocks;
3) IDN icons: the rest of the glyphs in the script, for example the majority code points of CJK, enclosed alphanumerices, enclosed CJK letters and ideographs.

The protocol uses US-ASCII to denote the phonetic elements of a script and calls for standardizing such a mapping for each language tag. The phonetic elements of a glyph is called "spelling" of the glyph and is called "stem" for that of a radical.

StepCode procedure may have more than two complete iterations. The first iteration is called í character transliterationí though it may take in more linguistic defined elements in such a conversion than a common term transliteration may imply. The second iteration is called í radical transliterationí , for it transcribes radicals of a glyph. The character to transliterated character table is called í tagged section mapí [IDNmap Sec. 2.2.3] or í tagged mapí , while a transliterated character is called a í StepCodeí . The process of converting an input string to T-ACE using a tagged map is called í language tagged proceduresí [IDNmap Sec. 4].

According to phonetic nature of world scripts, three groups are referred: Alphabet systems, including Latin, Cyrillic and Greek, Consonant systems, ie. Indian, Arabic languages), and Character Systems, ie. CJK languages.

1.8 IDN summary

The StepCode is a language dictated flexible ACE protocol and it is complement to the currently proposed, UCS flat treatment ACE. Its coding process reflects í Crowd Controlí concepts to better organize character and symbols before they are applicable in IDN system. To deliver itís full potential and to be more effective, it needs more consensus building among groups regarding code point treatment [Stone], which would be arguable points even a flat UCS code point treatment ACE is deployed alone in any case.

2. Host Name Transformation

According to [STD13], host parts must be case-insensitive, start and end with a letter or digit, and contain only letters, digits, and the hyphen character ("-"). This excludes any internationalized characters, any font variations, case variations, character set variations, as well as many other characters in the ASCII character repertoire. Further, domain name parts must be 63 octets or shorter in length including any language or other encoding tags.

User friendly encoding has to be coherent to usersí native languages, and consequently, host name transformation is dependent to the language tag [IDNmap Sec. 3] selected. As a StepCode encoding guide, the following discussion is focused on four different language groups: Alphabet systems, Consonant systems, Character systems and mixed script systems, from the simplest to more complex ones, and start with a general description of StepCode syntax.

2.1 StepCode Syntax

A Stepcode unit is a string of [A-Za-z0-9] letters without any white spaces, BLANK, in between. For each StepCode unit, there are data elements indicated by "", which is a MUST supplied element, and [] where the element is optional, and / where the data is selectable.

Sx stands for primary sound value or spelling of xth glyph;
Tx stands for secondary sound value or tone of xth glyph;
Ry stands for Stem for yth radical;
Ly stands for Layout relation from radical y to y+1;
Rx.y stands for Stem for Xth glyph and its yth radical;
Lx.y stands for Layout relation from Xth glyph and its radical y to y+1.

2.1.1 One glyph

A code point or a glyph in UCS can be an IDN letter, an IDN radical or an IDN icon. Where an IDN letter are phonetic symbols in its native language context marked by a language tag. For example Kana are IDN letters in Japanese context. An IDN radical is an independent glyph often used as a component of another glyph, or a glyph in a foreign language context. For example a simple Han character or a Han radical (U+2e90 ¿C U+2ef3), a Greek letter in Chinese context. An IDN icon is a composite glyph displayed in one display unit, normally a two dimensional square area. The majority of CJK characters are IDN icons. IDN icons can be viewed as compositions in terms of radicals, or IDN letters.

StepCode is language context sensitive transliteration of UCS code points. The The following is formal definition and examples of StepCode for a glyph.
The minimum code for a StepCode is one ASCII letter:

```
        "S"[T][P1][L1][P2][L2]...[Py][0/BLANK]
```

Thus, the following are examples of IDN letters, radicals and icons:
IDN letters:

```
        A        a       <Latin capital letter A>
        U+00c2   a6      <Latin letter A^>
      U+0a98   gha     <Gujarati letter gha>
      U+0a84   u1      <Gujarati letter uu>
```

IDN radicals:

```
      U+03b1   alf0            <Greek Small Letter Alfa>
      U+2f26 U+5b50   zi3z0    <CJK radical son>
      U+2f24 U+5937   da4d0    <CJK radical big>
      U+2f29 U+5b0f   xiao3x0  <CJK radical small>
      U+2f25 U+5973   nv3n0    <CJK radical female>
```

IDN icons:

```
      U+2639   :-(0    <White Frowning face>
      U+263a   :-)0    <White Smiling face>
           U+5b59   sun1zi1xiao0   <CJK character Grandson>
      U+597d   hao3nv1zi0     <CJK character good>
      U+5c16   jian1xiao2da0  <CJK character sharp>
```

Where the Unicode are IDN identifiers, the ASCII code column is corresponding transliterated StepCode, or DNS identifiers and the phonetic system used is in Chinese Pinyin.

2.1.2 Glyphs

A string of glyphs is considered as one unit with only alphanumeral:

```
"S1S2S3...Sx"[T1T2...Tx][P1.1][L1.1][P1.2][L1.2]...[P1.y][0]
               [P2.1][L2.1][P2.2][L2.2]...[P2.y][0]
                       ...
               [Px.1][Lx.1][Px.2][Lx.2]...[Px.y][0/BLANK]
```

Example of glyphs:

Latin	AaA^a		aaa6a
Gujarati	U+0a98	U+0a84	gha + u1 -> ghu1
Chinese	U+597d	U+5b0f U+5b50	haoxiaozi333nv1zi0x0z0

StepCodes are language specific. The above examples are from three language groups with common mix of symbols from the same languages. Where the Latin example has included capital letter A Circumflex, which is mapped to digit 6.

Gujarati letter GHA has an implicit vowel í aí , due to transliteration rule, when another vowel following the consonant the implicit vowel is replaced.

Chinese phrase <good boy> in the above example shows a mix of IDN radicals and icons encoding, where the first three digits indicate three characters in the unit, and three radical transliterations immediately follow.

2.2 Glyph Boundary Marks

Most script transliterations are mapped to alphabet system consistent with consonant-vowel-terminal structure. The majority of í glyph to glyph sequenceí and í glyph sequence back to glyphí can be done with minimum amount of linguistic rules embedded in glyph sequence composing and decomposing procedures.

There are always exceptions to any rules in linguistics. For example, the uses of í -í in Chinese and Korean, the uses of í í¯í in French and Chinese, the uses of letters í ZWNJí in Arabic, and the use of í |í in Tibetan and Devangari to prevent two units to join, are complements to the consonant-vowel-terminal rule.

In DNS system, only hyphen í -í is allowed for this purpose, and there may be more than one levels of disjoints a host name of a script has to differentiate. It is RECOMMENDED to consider an unused or non-conflict letter first before the í -í has to be used in the transliteration of a language tagged script. For example, the í í¯í in Chinese Pinyin may be mapped to the letter í ví instead of a hyphen í -í .

2.3 Encoding Steps

StepCode starts at a phonetic representation of a glyph with ASCII letters and a digit when it in need. This character transliteration has two phases as in Sec. 2.1.1 IDN letter examples:

S1.1. Romanize the primary phonetic characteristic of a glyph/phrase;
S1.2. Supplement the secondary phonetic characteristic of the glyph with a digit/digits.

The second step of StepCode is applied to components of each glyph, radical transliteration, in the same way specified in S1.1, and shown in Sec. 2.1.1 IDN icon examples.

S2.1. Romanize the primary phonetic characteristic of a radical, B;
S2.2. Specify how the next radical is related to the current radical, B, with a digit;
S2.3. If the radical contains another radical, X of B, then go to S2.1 of X (and it is S2+1.1(X)); otherwise go to the next radical, B+1.

2.4 Transliteration Schemes

Language is creation of human thoughts, which wanders everywhere disregard boundary. StepCode above is a rigid passageway, which only let the properly formed traffic to go through. While an alphabetic script structurally appears closest to Latin alphabet, a few general issues are common to all transliterations. The first issue is which transliteration should be implemented. Unicode Consortium has given each symbol a Latin name for ease in reference. Such a name contains the main sound value of the symbol, but usually more than what is needed in a transliteration. For example, Cyrillic letter BE has sound value í bí in Latin, and it is transliterated in [Translit 97] as a í bí . This introduces transliteration modification #1 to Unicode, that the sound value of a glyph MAY be extracted from its Latin name from UCS standard.

2.4.1 Basic Phonetic Classifications

It is RECOMMENDED that when consulting publications on character transliteration, the IPA [IPA] definition SHOULD be the primary classes to be considered. IPA class is an artificial grid over an analog spectrum. For each class there is a focus sound with a Latin letter label, and its neighboring sound values slide into its neighboring sound classes. It has the best classification on human language sound values available and its focus sounds are labeled with Latin alphabet letters. [Translit 97] has provided 54 romanization and transliteration schemes, and SHOULD be one of the base transliteration document.

2.4.2 Fuzzy Sound Value to Base Class Mapping

Whence a sound value can be described with an IPA class, then a

proximate letter representation can be referred. Transliteration Modification #2 is to consider a letter assignment in term of IPA class. It is RECOMEMDED that when alphabet is used to represent a sound value in a script, a balance between the current use of a letter in the same script and common uses of the same letter in other languages shall be found. The following is a comparison table of fricative alveolar-palatal letter sound assignments of a group of sampled languages. The table is expended a little into Plosives, Post-Palatals and Approximants for different sound value comparison with Arabic, Hindi, Vietnamese and Chinese languages, and also is used as illustration of the nature of IPA classification.

Table header are IPA category represented as:

Alveolar	Alveo
Postalveolar	Postalv
Retroflex	Retrof
Alveolar-Palatal	Alv-Pal
Front Palatal	FrontP
Palatal	Pala
Plosive	Plos
Affricative	Affr
Fricative	Fric
Approximant	Approx

Languages tagged as:

Chinese	zh-
Arabic	ar-
Deutsch	de-
English	en-
Esperanto	eo-
French	fr-
Latin	la-
Hebew	he-
Japanese	ja-
Korean	ko-
Hindi	hi-
Lao	lo-
Russian	ru-
Spanish	es-
Serbo	sr-
Tamil	ta-
Urdu	ur-
Vietnamese	vi-

The IPA symbol entries:

U+0283	sh	Latin Letter esh
U+0292	zh	Latin Letter yogh
U+0282	s2	Latin Letter s hook
U+0290	z2	Latin Letter z Retroflex hook
U+0255	c3	Latin Letter c curl

U+0291 z3   Latin Letter z curl
U+029d j1   Latin Letter crossed-tail j
c U+0327 c1   Latin Letter c cedilla

	Alveo	Postalv	Retrof	Alv-Pal	FrontP	Pala
Plos	t d		t2 d2			c
			ar-T ar-D			
			he-T			
	hi-t hi-d		hi-T hi-D			
	hi-th hi-dh		hi-Th hi-Dh			hi-kh hi-gh
					sr-c	
					vi-ch	vi-c
			ur-T			
Affr	ts dz	tsh dzh	ts2 dz2	tc3 dz3	tc1 dj1	
	zh-z	en-ch en-j	zh-zh	zh-j		
	zh-c	ar-ch	zh-ch	zh-q		
		de-ch				
		eo-cx eo-gx				
	he-ts	he-ch				
	ja-ts	ja-ch				
		ko-ch	ko-tch/jj			ko-gg
		hi-c				
		hi-ch				
		lo-ch				
		es-ch				
	sr-dz	sr-ch	sr-dz2			
	sr-tsí̄					
	ru-ts	ru-ch ru-zh				
	ur-z		ur-zh			
Fric	s z	sh zh	s2 z2	c3 z3	c1 j1	
	zh-s	en-sh en-as	zh-sh zh-r	zh-x		
	ar-s ar-z	ar-sh ar-zh	ar-S ar-Z			ar-H
	de-s de-z	de-sch				
	eo-s eo-z	eo-sx eo-jx				
	fr-s fr-z	fr-sh fr-je				
	he-s he-z	he-sh				
	ja-s ja-z	ja-sh ja-j				
	ko-s ko-ss	ko-j				
	hi-s	hi-sh	hi-S			
	lo-s					
	es-s/c		es-z			
	sr-s sr-z	sr-sh sr-zh				
	ru-c	ru-sh				
	vi-x vi-d	vi-s				
	ur-s ur-z	ur-sh				

Approx					j
					hi-j
hi-r hi-l ta-l ta-r		ta-L ta-N ur-R			ta-l2 hi-jh
Alveo	Postalv	Retrof	Alv-Pal	FrontP	Pala

Table 1. Romanized Latin letter assignments found in contemporary text books, bilingual dictionaries and [Translit 97].

More notes on table entries:
 The entries under column headers are in unvoiced vs. voiced pairs.
 The entries of the same column with a same language tag are non-aspirated and aspirated pairs in two rows, for example:

```
            hi-c
         hi-ch
```

 The uppercase letter assignments are taken from certain text books, where the transliteration takes several forms: doubling letters (common in text books), a dot under a letter(Library of Congress) and a capital letter (IPA convention).

Particular languages often have several sounds falling into the same class, or under the neighboring classes of IPA table, but very few under other labels. This phenomenon is can be found in above, Table 1. It is RECOMMENDED to follow conventional use of neighboring labels to differentiate the value concentrated classes, provided it does not conflict with other sound values which are already stable assignments. Some language transliterations supplementing a secondary letter to the label in focus often achieve satisfactory results, for example í ja-shí .

From the tabulated 18 language transliterations in Table 1, and considering the conventional transliteration practice shown in the table, the following sound value convention is RECOMMENDED:

Doubling vowel for a long vowel sound, (mostly used in Arabic)
Doubling consonant for sound produced from back position (Arabic, Hindi)

```
sh    for U+0283, Latin Letter esh (All in the table)
j     for U+0292, Latin Letter yogh (most in the table)
zh    for dj/dz   as an alternative for conventional dj and dz, it appears
                  quite popular in non-Roman languages.
ch     t U+0283  (Almost all in the table have done so.)
c      c/ts      (Though existing TS is common, but a í«cí¯ is a clear favor
                  for simplicity, provided that [c] is covered under í«kí¯.)
h      as an attachment letter for aspirated sound (as in Hindi).
n      for nasalization, it is hard to separated from [n], as í«n-í«, so a
       diacritic is RECOMMENDED.)
k      for [c],[k],[q] (It is rare to differentiate all the three in a
        language. When it has such a need, a í«kkí¯ accomplishes the task
```

as itíˉs in Korean.)

Since most of the transliteration data of Table 1 is from English literature, the recommendation above clearly is bias toward English speakers. The bias is based on two reasons. The first is technical, that common English does not use diacritical marks, so that it is a better base scheme for adapting other language symbols which often use diacritics. The second reason is the fact shown, in Table 2, that English is the highest in number of population, as non-native language used in the world currently.

The principle languages of the world ¿C

Source: S. Culbert, NI-25, University of Washington, Seattle, WA 98195, USA; Data as of mid-1993 [WORLD 95]
Languages spoken by more than 100,000,000 people:

	Native	Non-native	Total
Mandarin -	836	126	952
Hindi -	333		418
Spanish -	332		381
English -	322	148	470
Bengali -	189		196
Arabic -	186		219
Russian -	170	118	288
Portuguese -	170		182
Japanese -	125		126
German -	98		121
French -	72		124
Malay-Indonesian -	50	105	155

Table 2. Top four non-native languages used in the world: English, Mandarin, Russian and Malay-Indonesian.

2.5 Alphabetic Script Transformation ¿C Mechanical Methods

Transliteration is mostly table lookups with minimum rules to implement. Although alphabetic script transliteration is simplest, it is the place to specify transliteration table format and a few basic concepts and basic decision points in StepCode implementation, such as which phonetic system shall be selected, which foreign symbol set to be included in a language tagged script range [IDNmap] and how to include a foreign symbol or a symbol set.

2.5.1 Transliteration Tables

Transliteration table usually contains two columns. To make referencing easy for a layman, it is RECOMMENDED that transliteration tables contains at least four columns: ASCII symbol, UCS glyph, IPA sound value, and examples of spoken words of the language as shown in Table 3, with necessary comments.

ASCII ru-	UCS	IPA	Example
a	U+0430	U+0251 :	matb
b	U+0431	b	co6aka
v	U+0432	v	
g	U+0433	g	
d	U+0434	d	
e	U+0435	e	
j	U+0436	U+02a4	
z	U+0437	z	
i	U+0438	i:	
y	U+0439	i	
k	U+043a	k	
l	U+043b	l	
m	U+043c	m	
n	U+043d	n	
o	U+043e	U+0259	
p	U+043f	p	
r	U+0440	r	
c	U+0441	s	(í«sí¯ in [Translit 97])
t	U+0442	t	
w	U+0443	u:	
f	U+0444	f	
x	U+0445	x	(í«khí¯ in [Translit 97])
ts	U+0446	ts	
ch	U+0447	U+02a7	
sh	U+0448	U+0283	
sch	U+0449	U+0283 U+02a7	(í«shchí¯ in [Translit 97])
q	U+044a	(slilent)	
h	U+044b	U+0263	
q	U+044c	(soften the last consonant)	
a	U+044d	U+00e6	
iu	U+044e	ju:	
ia	U+044f	j U+0251 :	

Table 3. Russian Transliteration Table.

The third and forth columns are convenient references to phonetic data threads online.

Structurally, alphabetic script is similar with Latin, where some letters may represent different sound with Latin letter. For example, in Table 3 [Russian 44] the letter í«xí¯ and í«cí¯ are kept as Cyrillic letter, but in [Translit 97] they are transliterated to í«khí¯ and í«sí¯ respectively. Since the letters used here do not present conflict assignment with other letters, it is in the best interests of the native speakers to decide which version shall be used as DNS identifiers.

2.5.2 Mixed used of Alphabetical scripts

The major alphabetical scripts are Latin, Greek and Cyrillic, with very few cases using symbols from another script, for example í AGAPEí is Greek in Latin script, not in Greek script. It is RECOMMENDED to have three languages tags: la-, el- and ru- for Latin, Greek and Cyrillic, as three respective primary language tags [IDNmap] for alphabetic scripts.

If an English user wants to include a symbol from Greek, he has to wait for Latin tag to include Greek code block as its second script, if there is enough demand for such a service. In this case, there are two methods to include the transliteration table for Greek symbols in Latin tag.

The first one is to use a digit to indicate the second script set, as in column 1 of Table 4, and is called í Overflow Symbol Mappingí (Section 3.3), for simplicity in mechanical filling with a second set of symbols.

The second method is called í Radical mappingí is shown in column 2 of Table 4. The name í radicalí for Greek symbol is an analogy to radicals in CJK, for a Greek letter has a sound and a name and can not be decomposed. That is it is not a composite glyph, nor can it be sub-divided. They are treated in the similar way with CJK character set in a foreign language.

A secondary script attached to Latin language tagged section map:

```
la-
a       U+0061
b      U+0062
...
z      U+007a

a9     alf0     U+03b1
b9     bet0     U+03b2
c9     gam0     U+03b3
d9     del0     U+03b4
e9     eps0     U+03b5
f9     zet0     U+03b6
g9     eta0     U+03b7
h9     the0     U+03b8
i9     iot0     U+03b9
j9     kap0     U+03ba
k9     lam0     U+03bb
l9     mu0      U+03bc
m9     nu0      U+03bd
n9     xi0      U+03be
o9     omi0     U+03bf
p9     pi0      U+03c0
q9     pho0     U+03c1
r9     fsi0     U+03c2
s9     sig0     U+03c3
t9     tau0     U+03c4
u9     ups0     U+03c5
v9     phi0     U+03c6
```

w9	chi0	U+03c7
x9	psi0	U+03c8
y9	ome0	U+03c9

Table 4. Two methods to expend the Latin script.

The pros for Column 1 is short and regular, provided the digit 9 is not assigned to something else. The cons is hard to remember which letter of Greek is in that Latin letter position.

The second method shown in Column 2 is easy to remember since a Greek letter is mostly spelled out in a syllable (and can be mapped according to its sound value instead of the mechanical flooding as they are in Table 4), but is harder for a program to tell the character boundary. The few options are available for amending the radical mapping implementation:

1) Filling the short name up to make all the Greek symbols with uniform length, say 3 letters. By recognizing digit 0, the decomposing procedure can take preceding 3 letters as one symbol, this is called Protocol method.
2) Insert another digit 0 before the Greek symbol to mark a foreign symbol, and is called Marker method.
3) Insert a hyphen í«-í« before the Greek symbol, to make an independent sub-name unit, and is also a Marker method.

The pros for the above IDN radical symbol treatment is it is flexible, in terms of the number of symbols to be introduced, and in terms of naming such a symbol that a native reader understand, also it can be used for trademark encoding when there is such a request. The cons for it is lacking market data to support such an implementation. It is RECOMEMMDED a radical mapping is selected for introduce foreign symbols into a language tag.

Assuming the above recommendation is accepted, it is RECOMMENDED to use Method 2) to mark a foreign symbol in a language tag, for it accommodates variable length description of a foreign symbol, it is consistent with CJK symbol treatment discussed in Section 2.7 and it preserves method 3) for users to make individual decisions on their naming.

2.6 Consonant Script Transformation ¿C Developmental Issues

The name for this group of scripts may not be accurate, it just as well be called as the í rest of scriptsí besides Euro and Han scripts. The main concern in treating this group of scripts is treating each script independently and not let any rules made now develop into extreme in a near future. For example, one extreme is to forbid any new symbols to enter a language tagged range, the other is open up the whole UCS for one language tag. The Hindi language section map is selected here to examine implementation issues, since it reflects some of the reality in that user sector as well as in the engineering sector regarding language tag design issues [Stone].

hin-

7	**U+0901**	(nasalization)
	U+0902	(no decision)
	U+0903	(no decision)
a	U+0905	U+028c
aa	U+0906	U+0251 :
i	U+0907	I
ii	U+0908	i:
u	U+0909	U+028a
uu	U+090a	u:
ri	U+090b	ri
lri	U+090c	lri
e	U+090d	e
e	U+090e	e
e	U+090f	e
ai	U+0910	U+00e6/aI
o	U+0911	U+0259 U+028a
o	U+0912	U+0259 U+028a
o	U+0913	U+0259 U+028a
au	U+0914	U+0254 : / a U+028a
k	U+0915	k
kh	U+0916	x
g	U+0917	g
gh	U+0918	g'
ng	U+0919	U+014b
c	U+091a	U+02a7
ch	U+091b	U+02a7 '
j	U+091c	j
jh	U+091d	j'
ny	U+091e	ni
tt/T	U+091f	U+0288
tth	U+0920	U+0288'
dd	U+0921	U+0256
ddh	U+0922	U+0256'
nd	U+0923	nd
t	U+0924	t
th	U+0925	t'
d	U+0926	d
dh	U+0927	d'
n	U+0928	n
nn	U+0929	n (for Tamil n)
p	U+092a	p
ph	U+092b	p'
b	U+092c	b
bh	U+092d	b'
m	U+092e	m

y	U+092f	y
r	U+0930	r
rr	U+0931	r (for Tamil r)
l	U+0932	l
ld	U+0933	ld
ll	U+0934	l (for Tamil l)
v	U+0935	v
sh	U+0936	U+0283
ss	U+0937	U+0282
s	U+0938	s
h	U+0939	h
q	U+0958	q
khh	U+0959	q'
ghh	U+095a	G'
z	U+095b	z
dddh	U+095c	U+0256 d'
rh	U+095d	U+0280
f	U+095e	f
yy	U+095f	y:
	U+093a	
	U+093b	
	U+093c	
	U+093d	
aa	U+093e	U+0251 :
i	U+093f	I
ii	U+0940	i:
u	U+0941	U+028a
uu	U+0942	u:
ri	U+0943	rI
rii	U+0944	ri:
e	U+0945	e
e	U+0946	e
e	U+0947	e
ai	U+0948	U+00e6 / aI
o	U+0949	U+0259 U+028a
o	U+094a	U+0259 U+028a
o	U+094b	U+0259 U+028a
au	U+094c	U+0254 : / a U+028a

Table 5. IDN Hindi section Map [Hindi 98].

Observations of Table 5:
1) It has no example word column;
2) It has not made decisions on several code points;
3) It has adopted three Tamil symbols;
4) the extra long vowel sound is indicated by doubling the vowel letter;
5) the retroflex sound is indicated by doubling the consonant letter,
   while other forms exist, such as uppercase letter or an under letter

mark as they are shown in Table 1 and [Translit 97];
6) the aspirated sound is indicated by letter í«hí¯ instead of an apostrophe
í í¯í used in [IPA];
7) the symbol transliteration is not mechanical mapping, it needs linguistic rules to composing and decomposing a transliterated Latin string for Hindi.
8) the nasalizing sign, Devangari Sign Candrabindu, is mapped to digit 7, since it is the last diacritical mark used in [Translit 97]. The under-letter marks either have been reflected in Table 5, or ignored due to implicit transliteration of Table 5;
9) the section í U+093e - U+094cí are equivalent to section í U+0905 ¿C U+0914í , the section of symbols are not treated separately in [Translit 97]. These symbols could be included in canonicalizing procedure specified in [Nameprep] but dependent to input code processing.

Each of the observations flags a developmental issue:
1) Concerning the IDN as a long term solution or a short term fix. If this is a long term solution, then to fill up the column will benefit long term reference, there is no need to revisit the same issue when the reference is organized for later comers.
2) The assignment of 10 digits has to consider its common meaning to other languages so that, there is conformity semantics for less confused implementation and long term use.
3) Implies that Tamil language often appears among Hindi speakers. It is RECOMEMMDED to consider inclusion of one to two other scripts for each of languages in Consonant language group in the future IDN releases.
4), 5) and 6) are differences with [Translit 97] implementation. Advantages of this implementation is not over-load diacritical marks and is more reader friendly, with easier linguistic interpretation. Disadvantage is using variable length of Latin letters for each Hindi symbol.
7) As result of 4) 5) and 6), more linguistic understanding is required in implementation of a language tagged procedures.
8) With the more reader friendly treatment of Devanagari shown in 4)-7), there are enough digits to be used for other aspects of the linguistic issues, such as boundary, nasal, tonal or stress marks.
9) Case mapping is a common issue, which can be applied equally to Latin, Chinese, Japanese, Hindi as well as whatever there are such requests, and which have been defined by their primary users. In any case, the Hindi case mapping requires a better understanding of how the symbols are used at the user end both from keyboard, as well as keyboard signal to text transformation and local code exchange standard. When such an expertise is not available, there is still no base for exclusion for such a case mapping in IDN.

2.7 Character Script Transformation ¿C Feasibility

The commonly used symbol set for Chinese, Japanese and Korean is around **4000 characters each, with some differences in forms, while majority of** the symbols in each set over lap with the other two. Access of the 4,000 characters is a headache if one has to select from a table of 4,000

character without some efficient indexing system. For UCS CJK character set, the issue is to address over 21,003 characters using one primary language tag.

For languages with a large number of glyphs, such as CJK set and is impossible to map onto a Latin alphabet directly, a three layered scheme is RECOMMENDED, and a minimum set of glyphs of a script which are often used as parts of other glyphs are CJK radicals SHOULD be derived.

In the IDN system, the IDN letters include Bopomofo, Kana, and Jamo phonetic symbol sets. Since these systems all have been used, has stable transliterations standards to refer to, and have been discussed in previous sections, in this section the discussion will be focused on radical transliteration.

2.7.1 Character transliteration Scheme for IDN Radicals

Radical are building blocks of CJK character set. Radicals are independent symbols with semantics and pronunciation or names. For example,

Unicode	Short form	Long form	
U+03b1	alfa0		<Greek Small Letter Alfa>
U+5b50	zi0	zi3z0	
U+5937	da0	da4d0	
U+5b0f	xiao0	xiao3x0	
U+5973	nv0	nv3n0	

are five radicals, where the first part of each code is the name of the radical, the second part as they are shown in the last column is its primary sub-radical name letter. Mandarin has 417 sounds with average 4 tones each, total covers basic radical set of 1,500. With 25 letters before the delimiter 0, theoretically it is enough to give 23,000 UCS characters unique index. However, it is not enough to give each character a unique mnemonic name to facilitate usersí¯ access.

With the fast expansion of memory chips and transmission speed in the last **10 Years, vast amount of data can be stored at any local chips for fast** references. It is doubtful to design an index system concurs to above theory is wise. Instead, user friendly configuration should have the highest priority, and a complete set of data at ease of access shall be the base for a new IDN design philosophy.

Considering the radical encoding above, although it is enough to have Pinyin with tone indicator as its transliteration, as zi3, xiao3, da4, and nv3, it creates a different coding format, such that when they are mixed with an IDN icon, two different formats require more rules in processing. For simplicity, IDN radicals takes the same StepCode format as IDN icons, as shown in the last column on above four examples, which all end with a digit 0 as delimiter, but include only one letter as their sub-radical encoding to indicate a simple character with no further decomposing.

Thus, the longer form of IDN radical transliteration applies when 1) the radical set is large within a language tag, and the diacritical marks play a part in the transliteration; 2) the radicals are used with large IDN icon set, such as CJK, a uniform format with the larger set is Preferred over code complexity, so the radical is treated as an IDN icon.

The short form of IDN radical transliteration applies, when 1) the radicals are small set of foreign symbols under a concerned language tag, 2) a radical is used as radical transliteration of an IDN icon transliteration, as radical í xiao0í in Han character Sharp, í jian1xiao1da0í .

2.7.2 Radical Naming Convention

Some glyphs in the IDN radical set are most frequently used glyphs by themselves, some are used by themselves only in a particular language, yet some of them never stand alone, and their names follow naming convention which is listed bellow:

"pang" - a radical on the left, í pí for short;
"bian" - a radical on the right, í bí for short;
"tou" - a radical on the top, í tí for short;
"di"   - a radical on the bottom, í dí for short;
"xin" - a radical in the middle, í xí for short;
"kuang"- a container or an enclosure radical, í kí for short.

Since CJK characters are written from left to right and top-down, often the "pang" is the first radical of a character to be used as the key for searching into dictionaries and is partially listed in UNICODE, so "pang" has the most number of them appear in an index table in a regular Han dictionary.

2.7.3 CJK Character Coding Process

CJK Character coding process reflects í Crowd Controlí concepts: 1)survey Requests ¿C sorting, 2) select leaders ¿C identify equivalent cases, 3) mark directions ¿C mnemonic encoding, and 4) divert traffic ¿C leave out individual issues out for other applications. The principle applies to other UCS symbol transliteration encoding processes as well.

The naming process SHOULD reflect a user¯s viewpoint, not a programmer¯s viewpoint. The following radical transliteration procedure is RECOMMENDED:
1) Sort all the characters, include IDN icons, by Romanized names, which is Pinyin for Chinese, or a Latin symbol name in UCS;
2) Delete all polyphones of a character but leave one as the IDN identifier;
3) Sort all the homophones by frequency of usage counting both as a radical and as an IDN icon, and obtain a sorted list on frequency of usage, for example:
   fei-20 fei-8 fei-3 fei-2 fei-1
4) Move the hard to decompose character to the front, and suppose fei1-3

is such a character, then
fei1-20 fei1-3 fei1-8 fei1-2 fei1-1
5) Adjust homophone and polyphone characters as needed for easy coding discrimination;
6) Code each of the above symbol in the order prepared above:
fei1-20 fei1f0 <fly> (radical)
fei1-3 fei1b0 <not> (radical)
fei1-8 fei1nv1yi0 <concuban>
fei1-2 fei1caot2fei0 <poor>
fei1-1 fei1ko1fei0 <fei>
such that the front radical or character gets a shorter name;
7) Identify semantically equivalent character set, and assign only one character per set to IDN identifier.

Additional care MUST be applied in above process for future application system developments:
1) Reserve the polyphones opted out from Naming Process 2) and 5) above for other applications, for example user input processing, not discussed in IDN-map [IDNmap] but indicated in [SLS].
2) Reserve the members of semantically equivalent character set from Naming Process 7) above for other applications, for example IDN name display processing, which are not discussed in IDN-map [IDNmap], but indicated in [SLS].
3) For non-character radicals one may fall onto in Naming Process 6), a multi-syllabic name may be shorten with conventions specified in Section 2.7.2, for example, í cao zi touí is shorten to í caotí in í fei1-2 fei1caot2fei0 <poor>í above.

It is RECOMMENDED that the glyph transliteration process of CJK Characters DOES NOT bind by any particular radical list, which are only references as historical character decompositions. This introduces Transliteration modification #3 to UNICODE document, CJK radicals and radical supplement: U+2f00 to U+2fd5 and U+2e80 to 2ef3.

Other limitations posted by IDN system application are discussed in [Stone] Section 3. Observing limitations and follows the above coding process and sort out equivalent character set phonetically and semantically is REQUIRED as the first step to tame í A Tangled Webí [RFC 2825].

2.7.4 Use of character transliteration

It was a struggle to decide to put a full description of a Han character as its encoding or as its index, until the recent release of a wrist watch sized computer. It is clear that such a full description of a character will benefit symbolic processing greatly. For example, an automated voiced teaching tool may generate instructions on characters directly from the transliteration. IDN registration software can extract a DNS identifier from a full character description if such a holocode is available for access. For example, from the following IDN radicals and icons:

```
          U+5b59   sun1zi1xiao0
      U+597d   hao3nv1zi0
      U+5c16   jian1xiao2da0
      U+5b50   zi3z0
      U+5937   da4d0
      U+5b0f   xiao3x0
      U+5973   nv3n0
```

It is easy to extract a transliterated word from the first part of the above listed StepCode, and get the word í haoxiaozií . It is just as easy to match the second part, the radical transliteration only, to refer back to the characterí¯s pronunciation. This is a hint for another type of user friendly input glyph processing.

2.8 Mixed Script Transformation ¿CImplementing Japanese Tag

Japanese using different phonetic system, its homophone list would be different with that of Chinese, but the coding procedure described in Section 2.7.3 SHOULD be the same.

Section 2.7 concerning keeping one format for two types of characters, the radicals and icons of the same script. Japanese uses two different scripts from two script groups, kana and Kanji. Since Kana are IDN letters, and digits are diacritical marks of the letter preceded and appear at non-regular places, only digit 0 is reserved as delimiter. To include a Kanji among IDN letters, the rule of delimiter 0 SHOULD be applied as discussed in Section 2.5. For example, the Japanese section map:

```
U+3055      sa            <kana>
U+30fc      1           <diacritic macron>
U+3073      bi            <kana>
U+3059      su            <kana>
U+????      gyo1go0       <Kanji business>
```

Thus the DNS name í sa1bisu0gyo1go0í is readily available to be composed from these transliterated glyph codes.

3. Numerical Symbol Value Assignments

Though, it can be argued even among native speakers regarding a sound value of a symbol, the domain name identifiers only have 26 letters and some reasonable combinations within a script. These are the primary sound elements of a script in any case. Some changes to the primary sound elements are conventionally represented by modification marks to a primary symbol. Some modifications are significant and can be transcribed by a vowel from an alphabet system such as in Arabic. Others may be represented by a diacritics, as they are in French. UNICODE has provided clear separation along this line and some instructions on the functions of modification marks.

Unicode also has listed more than 64 general diacritical marks, U+0300 to U+0340, while the use of them in a language is not more than 12 by [Translit 97], (Hindi 12, Ottoman Turkish 11, Azerbaijani and Telugu both have used 10). Among the usage, the under-letter diacritical marks can be reflected in letters by conventional transliteration methods used in dictionaries and text books as shown in Hindi transliteration Table of Sec. 2.6, so that none of them will need more than 9 diacritical marks. It is REQUIRED that digit 0 is reserved as icon delimiter from diacritical mark functions.

Transliteration modification #4 to UNICODE document is to use a digit to represent diacritic like features, or secondary sound values of a script.

A digit has no universal sound value associated to it like that of a Latin letter. It is a good word separator and a less confusing diacritical mark than that of a letter. For scripts have frequent use of diacritics, it is RECOMMENDED to use digit in place of a diacritic mark in a normalized string. For syllabic scripts, it is RECOMMENDED to use digits at the end of an IDN identifier to indicate a semantic unit and the number of IDN identifiers in a transliterated string as shown in Section 2.

Although 26x10 is a two dimensional map, it can be filled with more than two phonetic aspects of a script. With increased complexity, the mnemonic value diminishes gradually. For simplicity, four phonetic mapping rules SHOULD be observed: R1. Diacritic mark mapping; R2. Phoneme Mapping; R3. Overflow consecutive slot mapping; R4. Priority elements mapping.

3.1 Diacritic Mark Mapping

[R1] Graphic based Diacritics mapping. For some scripts a secondary phonetic elements have to be marked for their users. For example European scripts, a simple diacritics mapping is RECOMMENDED, where the digits MAY denote common diacritics, tones and suprasegmentals.

	Tone mark	Diacritics
0	**no tone**	voiceless (o)
1	**flat/high(-)/long**	macron (-)
2	**global rise (/)**	acute (/)
3	**dip and rising (v)**	breve (v)
4	**global fall (\) grave (\)**	
5	**thrill (~)**	tilde (~)
6	**rising-falling(^)**	circumflex (^)
7		**umlaut(")**
8	**user assign**	cedilla (hook)
9	**user assign**	user assign

Table 6. General Diacritics Mapping Table

The assignment depends on four factors: 1) current user base with respect to keyboard assignment, 2) the number of marks in a script from a published dictionary, 3) IPA [IPA] value, 4) first come and first serve.

The above assignments due to:
1) #0 is reserved as icon delimiter;
2) #1 ¿C 4 due to common naming as first, second, third and fourth tone in Chinese;
3) #6 for common Qwerty keyboard assignment;
4) #5 and 7 for frequent appearance in Russian, German, Spanish and Vietnamese;
5) #8 a place holder for under-letter diacritic mark for Arabic and Hindi languages.
6) #9 for possible inclusion of Overflow symbol set assignment shown in Section 2.5.

The position of a similar marks are RECOMMENDED to stay in its respective position for ease interoperation cross script boundary and also for users looking for replacement marks. A French diacritical mark assignment is in Table 6.

French has less than eight but more than four diacritic marks, it is an example of phonetic mapping [R1].

fr-	
0	**no tone**
1	**Silent or Liaison '**
2	**rise/acute (/)**
3	**(dip/breve is not used)**
4	**drop/grave (\)**
5	**thrill/tilde (~)**
6	**throw/circumflex (^)**
7	**dieresis (")**
8	**Supercript or nasal n**
9	**(not used for French)**

Table 7. French Example of Using Diacritics mapping.

The French diacritical mark assignment is an example to demonstrate the usage of Table 6, not a French tag implementation. The fr- tag format is used for consistent presentation in this document.

For scripts in consonant system, a subset of marks is RECOMMENDED to be mapped to ASCII letters as its first choice, while the rest MAY be assigned a digit. Letters have associated sound values and easier for a non-native speaker to attach its IPA label association. A digit is better used for separating a secondary property from its primary sound based on IPA definitions. An Arabic example assignment is provided in [Mnemonics].

3.2 Phoneme Table

[R2] Sound based phoneme table mapping, where each digit specifies a variant of a base phoneme, and a maximum of nine variants may be accommodated. This rule has a best mnemonic result cross different scripts. For example, IPA symbol mapping for English in Table 8.

ipa- 0	**1**	2	3
a	U+0251	ae U+00e6	U+0292
b			
c	ch U+02a7		
d			
e	U+025b	.e U+0259	.e: U+025c
f			
g			
h			
i			
j	U+02a4		
k			
l			
m			
n	ng U+014b		
o	U+0252	o: U+0254	
p			
q			
r			
s	sh U+0283		
t	th U+03b8	U+00f0	
u	U+028c	U+028a	U+0075
v			
w			
x			
y			
z	zh U+0292		

Table 8. Exampe English Phoneme Mapping

IPA symbol mapping for English has used four variants. The Unicode code point indicates the IPA symbols where an ASCII symbol can not be found.

A full set of IPA symbol Phoneme mapping is provided in [Mnemonics] for references.

3.3 Overflowing

[R3] Overflow Symbol mapping - where the symbols SHOULD fill in only consecutive slots in the opposite directions in the 26 x 10 table for ease of index computation, where the middle section of the table SHOULD be left for user selected definitions. This rule is suited for two sets of corresponding

symbols of the similar scripts, for example Latin and Greek, Indian scripts. A Chinese version is shown in Table 9 for the method only, not in any way to suggest such an assignment.

```
zh-
        0       no tone
        1       flat/macron (-)
        2       rise/acute (/)
        3       dip/breve (v)
        4       drop/grave (\)

        5       classic character drop/grave (\)
        6       classic character dip/breve (v)
        7       classic character rise/acute (/)
        8       classic character flat/macron (-)
        9       classic character no tone
```

Table 9. Example use of Overflowing slot mapping.

The above Overflow and Tone Mark mapping architecture, [R1-R3], partitions the 26 x 10 table to symmetric two different glyph sets.

3.4 Priority List

[R4] Priority elements mapping - Selecting a set of often used symbols to be placed in the table. For example:

```
[R1-R3-R4]
en-
0       a-z
1       flat/macron (-)
2       rise/acute (/)
3       dip/breve (v)
4       drop/grave (\)
5       thrill/tilde (~)
6       throw/circumflex (^)
7       dieresis (")
8       Dingbats
9       A-Z

        0       8 (Dingbats)
        a       U+2604  /*areo or comet*/
        b
        c       U+24b8  /*copyright*/
        d       U+25ca  /*diamond*/
        e       U+24d4  /*eletron*/
        f       U+2709  /*fly*/
        g
        h       U+2624  /*health or Caduceus*/
        i       U+261e  /*index or white right pointing index*/
        j
```

```
k       U+2654  /*king*/
l       U+2661  /*love or white heart suit*/
m       U+2709  /*mail or envelope*/
n       U+266b  /*note or Barred eighth note*/
o
p       U+262e  /*peace symbol*/
q       U+2655  /*queen*/
r       U+2602  /*rain or umbrella */
s       U+263a  /*smile*/
t       U+231a  /*time or watch*/
u       U+2328  /*utility or keyboard*/
v       U+260e  /*voice or phone*/
w       U+270d  /*writing*/
x
y       U+262f  /* yinyang */
z
```

Table 10. Example use of Priority Mapping.

In fact, example Table 10 is general Latin script assignment, except the dingbats mnemonic values are keyed on English. DNS name resolver treats uppercase same as lower case, it provides no additional way for users to assign any specific value to upper case letters. One way to expand the symbol set allowed in DNS is to use [R3] as in Table 10. The English mapping assignment above takes rules [R1-R3-R4].

The above assignment rules MAY be used in a combination according to an order of weights in such an assignment. Such an order of weights SHOULD be specified in the form [Rx-Ry-Rz-R4] in front of a transliteration table of a language tag in form of comments.

3.5 Digits as Radical Layout Indicators

A unified CJK character is often a composition of several independent symbols from the script. It is possible to describe a CJK character by representing a character with only its radicals. Although it can identify a character uniquely, normally it is accompanied with a number of rules with too many exceptions for the majority of users to comprehend. StepCode encoding has reduced the complexity of the rules by considering a CJK character as a simple grid of 1 to 10 units. Naming the 1 to 10 units in a linear fashion results a linear representation of the glyph or its encoding.

The order of prioritizing radicals of a character is important. In general, the radical that one writes it with a pen containing the first stroke of a symbol in printing manner, which is publicized as part of a national education system is the í primary radicalí of the symbol. For example the character í xin <new>í (the digit is the tone of the character, hereafter) has two radicals:

1) í qin1 <intimate>í + í jin1 <a half kilogram>í

Since í qin1í may be considered as two radicals as well, the radicals list may be in the following form too:

2) í li4 <stand>í + í jin1<a half kilogram>í + í mu4 <wood>í

or with different radical ordering:

3) í li4 <stand>í + í mu4 <wood>í + í jin1 <a half kilogram>í

In this case the í qin1í or í li4í both may be the primary radical dependent to which viewpoint of the user takes, which may be address in a different document. StepCode protocol favors 1) as discussed in Section 2.7.

Variation in Radical transliteration can result in multiple StepCodes to one character within the same tagged map. It is due to 1) Radical transliteration is usually used as secondary representation of a character, however sometimes it may be used as its primary representation, when the correct sound of a character is not available to the user. 2) When viewing a character as a grid, there are disagreements on the number of units in a character. For domain names, the point of views in describing compositions of a character for a domain name MUST be limited to only one major viewpoint. The minor viewpoints SHOULD be converted to the major viewpoint, and radical transliteration MAY be the key to locate its character transliteration part through user interface when a name is registered.

The digits in radical transliteration specifying how a radical of a glyph on its grid is related to the next radical, are called layout digits. Layout digits specify the relation to the next radical in line. The left and right direction are defined by a user's left or right hand while sitting in front of a display screen or a piece of paper.

The glyph layout digits are:

- 0 - end of a character or a radical
- 1 - to its right
- 2 - to its underside
- 3 - to contain the following
- 4 - to divide the following
- 5 - to its left
- 6 - to its top

The following selectable digits are to specify additional glyphs of the script and directions of layout.

- 7 - to overlay itself with X then to its right;
- 8 - to overlay itself with X then to its left;
- 9 - to overlay itself with X then to its underside.

Table 11. Glyph Layout Numeral Values

The radical layout scheme trades complexity of a glyph with code length, such that the complexity can be left out when an application only needs the character transliteration.

4. Language Specific Procedures

Either, StepCode may be obtained directly from local display codes to StepCode phrase conversion tables or to be taken from IDN identifier of language tagged section maps. Or, it inputs directly from keyboards, where an input processing module verifies correctness of intended glyphs and normalizes a StepCode. [Appendix] is an example of such cached input processing procedure.

Different scripts have different transliterations published worldwide. These publications are the base for implementing tagged maps and tagged conversions as discussed in previous sections.

4.1 IDN Input Normalization Procedures

The protocol contains two pairs of conversion and reversion procedures per language tag supported(See [IDNmap] Section 4.3) and calls for a minimum number of semantic independent symbols of a language to be mapped onto a Latin alphabet in a mnemonic manner (Section 2.7.3). The first pair of conversion and reversion procedures are convert language specific presentation form to a normalized form and vice versa, named as Normalize and Present procedures respectively and have been described for Latin, Arabic and Chinese script implementation in [UAX 15][Bidi][Icdn].

4.2 DNS Fitting Procedures

The second pair of language specific procedures converts a list of transliterated symbols to a name unit, either it is a word or a phrase or an identifier of any kinds, to fit into a desired format for any artificial goals with restrictions that format has to be reversible back into the list of transliterated symbols in its corresponding decomposing procedure. The pair of procedures is called Fitting and Decompose respectively.

The purpose of assembling a StepCode is to be disassembled at its end of wire travel and indexing back into a tagged map, such that the pre-converted local display codes can be retrieved in an equivalent local display code worldwide. For some StepCode, when a list of character transliteration is combine into a string, it blurs the pre-converted symbol boundary, which is significant in their semantic differences, and interferes with correctly disassembling a StepCode string. It is RECOMMENDED in such a case, a hyphen, í -í , is added as the last reserved character separator.

When a post-converted string contains mixed scripts, for example Japanese domain names, exceeds maximum label length, it is only the

characters with radical transliteration MAY be dropped. The truncated radical transliteration SHOULD reinsert a digit í«0í¯ to mark the end of radical transliteration, or using transmission protocols decided by network group among servers on how to deal with code length exceeding the DNS label maximum, or other protocols specific to a language tag to recover, partial recover or intelligent guesses in preventing confusion when it is decomposed.

Possible protocols for Fitting/Decomposing procedures depend on the scale of such format to be placed.
1) Zone records: IDN zoned record keeping at IDN name registration locale;
2) Caches: Cached traffic records at client sites;
3) Exceptions: Exception handling rules implemented by protocols;
4) Markers: Symbolic marker interpretation for specific language tag;
5) Models: Embedded linguistic rule interpretation in Fitting/Decomposing programming languages.

It is RECOMMENDED, that each language tagged procedure SHOULD specify which protocol type is implemented and what their effects are for world wide basic code maintenance.

StepCode string is assembled with orders consistent with keyboard input, regardless it how it would be displayed on a screen or in URI [URI]. For some scripts, its character display order may be rearranged. Such a display order is implied by tagged display procedure, and is not a part of character transliteration nor a part of radical transliteration. Layout digits apply to layout directions within a character space as defined by UNICODE, NOT between characters.

5. Embodiment of StepCode Protocol

Symbolic representation in machine format with mnemonic label for human readers is a basic technique to improve human control over programs. With such a control of large name base, many artificial intelligence type of applications can benefit from it. For example, the mnemonic indexing system for UCS discussed in [IDNmap] may be extended to sort and index on trademarks and icons for automatic access needed in [WIPO].

A very much needed universal keyboard access to the full spectrum of code points in UCS becomes feasible. Imagine that a user pickups a language tag from a pull-down window, and then types in the keys from a Latin alphabet labeled keyboard, gets the typed alphabet showing on the screen for the first level of input verification, and then looks at the transliteration to symbol conversion to get the second level, í spellingí verification. (A dream that the author has had for more than 15 years.)

Since StepCode preserves the complete character information, it is a holocode scheme of a symbol. From which one may extract a set of radicals to infer the content of a discourse. For example, by recognizing large presence of í shui3 <water>í radical, one may infer a water body context.

With such type of inference, a semantic net is not too far for reach.

6. Security Considerations

Much of the security of the Internet relies on the DNS. Thus, any change to the characteristics of the DNS can change the security of much of the Internet. Thus, StepCode makes no changes to the DNS itself.

Hostnames are used by users to connect to Internet servers. The security of the Internet would be compromised if a user entering a single internationalized name could be connected to different servers based on different interpretations of the internationalized hostname. Thus the restriction of DNS names to a small symbol set is necessary and effective, where adding any other data format only opens the security gate to complications.

7.Internationalization considerations

StepCode is designed so that every internationalized hostname part can be represented as one and only one DNS-compatible string. If there are two different ways to obtain the same glyph on a display device, then they are still two distinct hostnames, with no bearing on DNS security issues. If there is any way to follow the steps in this document and get two or more different results, it is because of an error in the domain name registration process, where one domain name registrar fails to update other domain name registrar servers about a newly registered and well researched hostname.

StepCode using only [a-z0-9] as the basic symbol set is linguistics sounding choice. Since the base classification used by IPA is Latin symbol set, the only authoritative study on the subject. The symbol set has been successfully applied to majority of languages on earth, and have been proven an effective set of symbols for people of many native tones to remember and to map to, shown by existing vast quantity of national standards and dictionaries. Thus [a-z0-9] is the best set of symbols to be used for universal mnemonic applications of any kind involving human records. StepCode is a symbol organization scheme to connect the symbol set to these applications.

8. References

[ASCII] American National Standards Institute (formerly United States of America Standards Institute), X3.4, 1968, "USA Code for Information Interchange". (ANSI X3.4-1968)

[CJK] James SENG and etc. í Han Ideograph (CJK) for Internationalized Domain Namesí , draft-ietf-idn-cjk-01.txt, 11th Apr 2001.

[DeFrancis 1989] John DeFrancis, "Visible Speech - The Diverse Oneness of Writing Systems", 1989, ISBN 0-8248-1207-7.

[Dictionary79] Beijing Foriegn Language Dept., "A Chinese-English
Dictionary", 1979, BK# 9017.810.

[Icdn] Xiang Deng and Yan Fang Wang, "The Implementation of Chinese character
in IDN", draft-ietf-idn-icdn-00.txt, July 2001.

[IDNReq] Zita Wenzel and James Seng, "Requirements of Internationalized
Domain Names", draft-ietf-idn-requirements. May 2001.)

[IPA] The International Phonetic Alphabet, http://www2.arts.gla.ac.uk/IPA
1996.

[ISO639][ISO639-2/T] ISO/IEC 639-2 2001 Codes for the Representation of
Names of Languages.

[ISO10646] ISO/IEC 10646-1:2000 (note that an amendment 1 is in
preparation), ISO/IEC 10646-2 (in preparation), plus
corrigenda and amendments to these standards.

[Hindi 98] "Hindi & Urdu Phrase Book", Lonely Planet Publications, 1998,
ISBN 0-86442-425-6.

[Translit 97] Barry, Randall K. 1997. ALA-LC romanization tables:
transliteration schemes for non-Roman scripts. Washington: Library
of Congress Cataloging Distribution Service. ISBN 0-8444-0940-5

[PinyinCon] Library of Congress Pinyin Conversion Project, í New Chinese
Romanization Guidelinesí ,
http://lcweb.loc.gov/catdir/pinyin/romcover.html#7

[Macmillan93] The Macmillan Visual Desk Reference, 1993,
ISBN 0-02-531310-x.

[Mnemonics] Liana Ye, í Mnemonic Symbol Mapping of UCSí .

[RFC 2026] S. Bradner, í The Internet Standards Process -- Revision 3í ,
1996, RFC 2026.

[RFC2119] Scott Bradner, "Key words for use in RFCs to Indicate
Requirement Levels", March 1997, RFC 2119.

[RFC2277] "IETF Policy on Character Sets and Languages",
rfc2277.txt, January 1998, H. Alvestrand.

[RFC2396] Tim Berners-Lee, et. al., "Uniform Resource Identifiers (URI):
Generic Syntax", August 1998, RFC 2396.

[Russian 44] "New Russian-English and English-Russian Dictionary", Dover
Publications, New York, 1944, ISBN 0-486-20208-9.

[SIS] M. Mealling & L. Daigle, í Service Lookup System (SLS)í

draft-mealling-sls-00.txt

[STD13] Paul Mockapetris, "Domain names - implementation and specification", November 1987, STD 13 (RFC 1035).

[RFC2825] L. Daigle, Ed. í A Tangled Web: Issues of I18N, Domain Names, and the Other Internet protocolsí , May 2000, RFC 2825.

[UAX15] Mark Davis and Martin Duerst. Unicode Standard Annex #15: í Unicode Normalization Formsí , Version 3.1.0. <http://www.unicode.org/unicode/reports/tr15/tr15-21.html>

[UNICODE] The Unicode Consortium, "The Unicode Standard". Described at http://www.unicode.org/unicode/standard/versions/.

[UNICODE30] The Unicode Consortium, "The Unicode Standard -- Version 3.0", ISBN 0-201-61633-5. Same repertoire as ISO/IEC 10646-1:2000. Described at http://www.unicode.org/unicode/standard/versions/Unicode3.0.html.

[URI] Roy Fielding et al., "Uniform Resource Identifiers: Generic Syntax", August 998, RFC 2396.

[Versions] Marc Blanchet, í Handling versions of internationalized domain names protocolsí , draft-ietf-idn-version-00.txt, October 26, 2000.

[WIPO] í The Role of Technical Measuresí , RFC3, http://wipo2.wipo.int/process2/rfc/rfc3/index.html

[WORLD 95] í The world Almanac and Book of Facts 1995í , ISBN 0-88687-766-0

[Ye95] Liana Ye, "A Language Oriented Chinese Encoding for Multilingual Computing Environments", in "Proceeding of the 1995 International Conference on Computer Processing of Oriental Languages", Page 323.

9. Acknowledgements

The author has benefited from special comments and suggestions from Aaron Irvine, John C Klensin, Eric Brunner-Williams, Erik Nordmark and William Davis and relevant discussions from IDN Working Group to improve this document.

10. IANA Considerations

This document requires IANA action for availability of script tag, and registration for each tag and possibly its sub-field for phonetic system used, and readiness of associated language specific procedures.

11. Authors' Contact Information

Liana Ye
Y&D ISG

2607 **Read Ave.**
Belmont, CA 94002, USA.
(650) 592-7092
liana.ydisg@juno.com

Expires March 2002

[Appendix] StepCode keyboard input process for Chinese

```
/* buff.c  StepCode processor interface   Copyright Y&D ISG, Inc. 1994
 *-------------------------------------------------------------------------*
 *  find_gly  find a glyph online.
 *  find_wd   find a word online.
 */

#include <stdio.h>
#include <ctype.h>
#include "steplib.h"

int auto_learn= TRUE;
int udic_large= FALSE;
int udic_database= FALSE;
int odic_expand = FALSE;
int dic_saved = FALSE;
int keyboard_in = TRUE;
int alt_memb = 2;       /* extra members of a poly-code to be recorded */

/*
 * find_gly  using a StepCode to find the GB code for display a glyph.
 */
int find_gly(step, stepcd, infor, gb, key)
        char *step, *stepcd, *infor, *gb;
        int *key;
{
        FILE *bufp;
        int linecnt, bytes;
        char line[MAXdatalen], *p;
        char bufname[FILENAMSIZ];

        strncpy(stepcd, step, strlen(step)+1);
        if (hit_gly(stepcd, gb))
                { *key=GB; return(A_to_B);}

        strncpy(bufname, BUFFILE, FILENAMSIZ);
        bufp = (FILE *)fopen(bufname, "w+b");
        if( bufp == NULL )
        {
                strcpy( message, "Buffer file unavailable.");
                typo(message, word);
                return(ERROR);
```

```
        }
        search_dic(STEP, 1, stepcd, bufname, &bufp, &linecnt);
        if (linecnt<=0)
        {
                if(verbose)
                typo("No entry found in GB table. You may create one.", step);

                fclose(bufp);
                return(A_to_ZIL);
        }
        fseek( bufp, 0L, 0 );           /* to beginning sake read */
        if(fgets(line, MAXdatalen,  bufp)== NULL)
        {       if(verbose)
                fprintf(stderr, "ERROR- buffer file read error.\n");
                fclose(bufp);
                return(ERROR);
        }
        sscanf(line, "%s%d%s%s\n", stepcd, key, gb, infor);
        hash_gly(stepcd, gb);
        fclose(bufp);
        if (linecnt>1)
        {
                return( A_to_N);
        }else {
                return( A_to_B);
        }
}

int find_wd(step, stepcd, infor, gb, cnt, key)
        char *step, *stepcd, *infor, *gb;
        int cnt, *key;
{
        FILE *bufp;
        int linecnt;
        char line[MAXdatalen], *p;
        char bufname[FILENAMSIZ];

        strncpy(stepcd, step, strlen(step)+1);
        if ( hit_wd(stepcd, gb))
                { *key = GB; return(A_to_B);}

        strncpy(bufname, BUFFILE, FILENAMSIZ);
        bufp = (FILE *)fopen(bufname, "w+b");
        if( bufp == NULL )
        {
                fprintf( stderr, "Buffer file unavailable.");
                return(ERROR);
        }
        search_dic(STEP, cnt, stepcd, bufname, &bufp, &linecnt);
        if (linecnt<=0)
        {       if (!auto_learn)
```

```
                {
                   if(verbose)
                        typo("Not found.  You may create the word.", step);
                   fclose(bufp);
                   return(A_to_ZIL);
                }else
                {
                        neww = learnword(cnt, stepcd, gb);
                        /* Do whatever with neww here */
                        if(dic_saved)
                                {
                                        hash_wd(stepcd, gb);
                                        dic_saved = FALSE;
                                }
                        else
                        {
                           typo("The new word has not saved.", stepcd);
                        }
                        fclose(bufp);
                        neww = reset_word(neww);
                        return(ZIL_to_A);
                }
        }
        fseek( bufp, 0L, 0 );           /* to beginning sake read */
        fgets(line, MAXdatalen,  bufp);
        if(line == NULL)
        {
                if (ferror(bufp)!=0 && verbose)
                        fprintf(stderr, "Error during buffer read.\n");
                if (feof(bufp) !=0 && verbose)
                        fprintf(stderr, "Buffer file ended.\n");
                clearerr(bufp);
                fclose(bufp);
                return(A_to_ZIL);
        }
        sscanf(line, "%s%d%s%s\n", stepcd, key, gb, infor);
        hash_wd(stepcd, gb);
        fclose(bufp);
        if (linecnt>1)
        {
                return( A_to_N);
        }else {
                return (A_to_B);
        }
}



/* ---------------------------------------------------------------------
 * Figure out the number of glyphs in a word. The next two routines are
 * based on PINYIN system.
```

```
 */
int one_letter_sound(word)
        char *word;
{
        int cnt=0;
        char *w, *v;

        w=word;
        while (*w=='m'||*w=='M'||*w=='n'||*w=='N')
                        { ++cnt; ++w;}
        if (cnt>0)
        {
                v = w; --v;
                if((*w=='g'||*w=='G')&& (*v=='n'||*v=='N'))
                        ++w;    /*ex: mng nnng*/
        }
        if(cnt==0) while (*w=='a'||*w=='A'){ ++cnt; ++w;}
        if(cnt==0) while (*w=='o'||*w=='O'){ ++cnt; ++w;}
        if(cnt==0) while (*w=='e'||*w=='E'){ ++cnt; ++w;}
        if (!isalpha(*w))
                return(cnt); /*ex:a aa ooo eee- mmm nmn*/
        else cnt=0;             /*ex: an hhh oong */
        return(cnt);
}

int tell_word(word)
        char *word;
{
        char *w, *v;
        int  cnt;
        cnt=0;

        if(!isalpha(*word)) return (NULL);

        for (w=word;isalpha(*w);++w); /*skip Pinyin */
        while (isdigit(*w)) {cnt++; ++w;} /*count the number of tone marks*/

        if (cnt<1)              /*special sigle letter glyph cases*/
        {
                cnt = one_letter_sound(word);
                if (cnt>=1) return(cnt); /* else do syllable analysis */
        }
        else return(cnt);

        /*
         * find the number of syllables by vowel rules
         * This implementation is accuate even without using apostrophe
         */
        w=word;
        while (isalpha(*w)) /*check the Pinyin only*/
        {
```

```
                switch (*w)
                {
                case 'a':
                case 'i':
                case 'e':
                case 'o':
                case 'u': v=w; ++w; cnt++; /*one vowel case*/
                        switch (*w)
                        {
                        case 'i':
                        case 'e':
                        case 'o':
                        case 'u': ++w;break; /*two vowels sound*/
                        case 'a': ++w;
                                if (*v=='u' && *w=='i') break;/*uai*/
                                if (*v=='i' && *w=='o') break;/*iao*/

                                else {
                                        --w;     /*still two vowels*/
                                        break;
                                }
                        default: break;
                        }
                default:
                        /*already get out off the compound vowel*/
                break;
                }
                ++w;
        }/*check syllables*/
        return(cnt);
}

/*
 * ------------------------------------------------------------------
 * Interactive input process procedure
 * ------------------------------------------------------------------
 */
inputp(char *word, char *gb)
{
        int  i,  glyphcnt;
        char c, *w;
        int cnt, key, stat;
        char dump[MAXdatalen];

        for (;;)
        {
                *word='\0';
                fgets(word, MAXlinelen, stdin);
                if (isspace(*word))
                        break;
```

```
                /* Check if the entry is a glyph string by */
                glyphcnt = tell_word(word);
                if (glyphcnt == NULL)
                {
                        printf("%s", *word);
                        fflush(stdin);
                        continue;
                }

                w=word;
                while (isalnum(*w)) ++w;
                *w = '\0';
                if(verbose)
                        printf("tell_word figure:  %d glyphs\n", glyphcnt);

                /* Determin the entry is known through dictionary
                 * and cache lookup.
                 */
                if(glyphcnt >=2)
                        stat = find_wd(word, stepcd, dump,gb,glyphcnt, &key);
                else stat = find_gly(word, stepcd, dump,gb, &key);

                /* Print out with GB code */
                if (!stat==ERROR) font_code(stepcd, gb, &key, stderr);
                if(verbose) printf("%s\n", stepcd);
                fflush(stdin);
                fflush(stderr);
        }
        return(0);
}
```