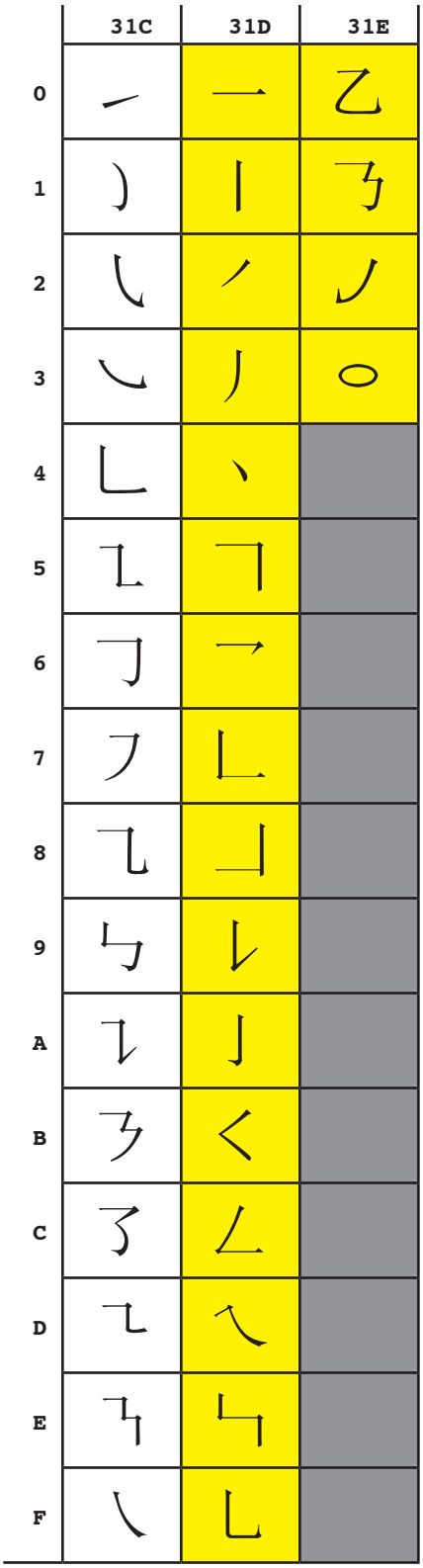


The **CJK Strokes** Block

U+31C0..U+31CF : Unicode 5.0
U+31D0..U+31E3 : Unicode 5.1?

Wenlin CDL: *Character Description Language*

by *Tom Bishop* & *Richard Cook*

The roughly 72,000 encoded CJK ("Chinese, Japanese, Korean, and Vietnamese", a.k.a. "Han") characters constitute by far the largest chunk of Unicode. And even that many code points isn't enough, since many thousands of CJK characters remain unencoded. Most of them are rare variants, but nevertheless important for certain purposes. Scholars frequently encounter strange characters in ancient texts, and government workers may need to write the names of people and places that use unusual characters.

This problem can *never* be solved once and for all by simply assigning a number to each character, since the set of CJK characters is open-ended.

In this article we introduce Wenlin's *Character Description Language* (CDL), an XML application that solves the unencoded CJK character problem once and for all. CDL is here shown to provide methods for handling *all* CJK characters, encoded and unencoded. And CDL methods will be seen as more generally applicable to other complex writing systems.

An Infinite Set of Characters

Historically, the most common way of creating new CJK characters has been to combine two or more existing characters and squeeze them into a square. Here is an unencoded character combining 車 'chariot' on the left, and 鳥 'bird' on the right:

Characters that can serve as pieces of more complex characters are called *components*. In the days when most writing was done by brush (or woodblock printing), it was easy and common for writers to invent new characters this way.

Usually one component serves to suggest the meaning, and the other to approximate the pronunciation. The combination of 'chariot' and 'bird' might represent a word for a kind of chariot whose name sounded similar to the word for 'bird', or vice-versa; or maybe a flying wagon. The truth is, we don't know! This character doesn't have a Unicode number yet, but it comes from the Adobe-Japan1 collection and was recently proposed (by Ken Lunde and Eric Muller) for future Unicode encoding.

In principle, any character, no matter how complex, can be used as a component in another (more complex) character. Likewise, any recurrent pattern of strokes can be identified and reused as a component.

Describing Characters, Not Just Numbering Them

The basic idea of CDL is that a character can be represented (described) as a sequence of what we call *basic script elements*. For CJK these basic script elements are *not* the characters themselves, but rather, the kinds of strokes used to write the characters. While components aren't basic in this sense, they often play a useful intermediate role in CDL, between individual strokes and entire characters.

A CDL description simultaneously identifies the character and also gives instructions for displaying it. The idea is similar to how the spelling of a word in an alphabetical script is described as a sequence of letters — for example, "cat" is simply described as "c", "a", "t".

Similarly, the character 呆 can be described as a sequence of the two components "口", "木"; and those, in turn, can be described as sequences of strokes. That's not enough information to display or identify 呆, however, since the same components can be arranged differently, forming distinct characters such as 咻 and 困 (with completely different pronunciations and meanings). For this reason, CDL descriptions specify the two-dimensional positions of the components and strokes, as we'll explain below.

An Alphabet of Stroke Types

Since the set of components is unlimited, the components themselves need to have descriptions, and we don't stop describing until we've described (either directly or by reference) the individual strokes of each character.

Fortunately, we find that an "alphabet" of just 36 stroke types is enough for a very large class of CJK characters.

Wenlin Institute's CDL specification defines a set of stroke types, which has been translated into the block of 36 *CJK Strokes*, shown in the CJK Strokes table above. The strokes in the first column are already encoded in Unicode 5.0, while the rest will become a formal part of the standard presumably in version 5.1.

From Pictures To Strokes

One of the unique features of CJK characters is their high degree of dependence on norms determining stroke type, stroke order, and stroke count. These handwriting norms began to take form in the earliest known periods of Chinese writing more than 3,000 years ago, and underwent various changes and refinements in subsequent periods, as the result of changes in writing implement, writing media, writing method (printing), and language changes (vocabulary development).

A major script reform took place in the 秦 *Qín* Dynasty (c. 200 BCE). In the 東漢 Eastern *Hàn* period (c. 121 AD) an influential analysis codified the elements of the script. There were 9,353 Chinese characters at that time, and 1,163 variants, according to the most important 清 *Qīng* Dynasty recension of the 《說文解字》 *Shuō Wén Jiě Zì* text (that of 段玉裁 Duàn Yùcái, d. 1815).

As written texts grew in complexity, as libraries collected more and more books, and as the inventories of Chinese characters became larger and larger, it became urgent to define methods for organizing and indexing the elements of the complex character set.

Through refinements climaxing in the 宋 *Sòng* period (c. 1,000 AD), coinciding with the widespread use of woodblock printing, the Han glyphs attained the style evident in Unicode's code charts. Glyphs in the *Sòng* style are highly constrained in many ways, but most importantly in terms of the types of strokes which may occur.

Below is an archaic pictograph for the word meaning 'chariot' (such as is seen in *oracle bone inscriptions* [OBI = 甲骨文] from more than 3,000 years ago):

Compare the above with the Eastern *Hàn* Small Seal form (小篆, shown below), more than 1,000 years later, but having the relatively simple form attested much earlier (in middle and late 周 *Zhōu* Dynasty inscriptions):

In the *Sòng* printed style (below) many old curves are now sharp angles and the strokes are very regular and stylized:

The *Sòng-style* 'chariot' character above has *exactly seven* strokes (we'll count them later); it is the standard form in Japan, Taiwan, and Korea. In PRC and Singapore, the standard simple form (below) has only four strokes:

Like many simple forms, 车 is based on cursive forms in use for many centuries. Nevertheless, its strokes are all *Sòng* style. CDL is equally well suited to describing the simpler and more complex CJK variants that are preferred in different locales.

On the left below is an archaic pictograph for 'bird' (again, an early OBI form), and on the right is the Eastern *Hàn* Seal form:

Below are the modern *Sòng-style* descendants, full form (left, 11 strokes) and simple form (right, 5 strokes):

鳥 鸟

Han characters were not originally composed of a limited set of strokes, but only gradually came to be written that way. If the characters were still just pictures rather than being made up of standard strokes, maybe the best that could be done would be to assign a number to each picture. Fortunately, the set of stroke types evident in a *Sòng*-style typeface can be regarded in a very real sense as the alphabet of CJK writing.

The Origins of CDL

CDL was developed by *Wenlin Institute* as part of its software package for learning Chinese, to teach the fundamentals of traditional Chinese writing. The software provides animated stroke-by-stroke display of more than 50,000 CJK characters.

In the published versions of Wenlin, the CDL language itself is a hidden feature of the software implementation. The special abilities to view, create and modify CDL descriptions have so far only been included in unpublished versions of the software. The authors are working towards making these features, and the database of over 50,000 descriptions, available to a wider audience.

Stroke and Comp(onent)

The most important keywords in Character Description Language are **`stroke`** and **`comp`** (an abbreviation for *component*). We've called strokes the basic CJK "alphabet", and CDL can describe any Han character purely in terms of strokes. Nevertheless, the majority of characters can be described more briefly and more meaningfully in terms of components. Components then have their own descriptions, sometimes in terms of simpler components, until finally the sim-

plest components are described purely in terms of strokes.

Since that is the usual situation, we'll demonstrate the usage of **comp** first, and **stroke** later.

A *Bird* in the *Chariot* (is worth ...)

A CDL description for our "flying chariot" (or "wheeled bird"?) character might look almost like this:

```
<cdl>
 <comp char='車'
   position='left' />
 <comp char='鳥'
   position='right' />
</cdl>
```

Just looking at the code, it isn't hard to see that the character being described is built from two components: 車 'chariot' on the left, and 鳥 'bird' on the right.

However, we generally want more precise control over the positions of components and (especially) strokes. (Otherwise, CDL would be similar to an older technology known as *Ideographic Descriptions Sequences*, which is useful but lacks many of the capabilities of CDL.)

Most often when we create descriptions, we use a graphical interface to position the elements visually, by dragging on "control points" with the mouse. We usually don't need to read or type any numbers; the CDL description is automatically updated to contain the exact coordinates.

Therefore, the form of CDL that we actually use doesn't have a **position** attribute. Instead, it has a **points** attribute with numerical (x, y) coordinates, and looks like this:

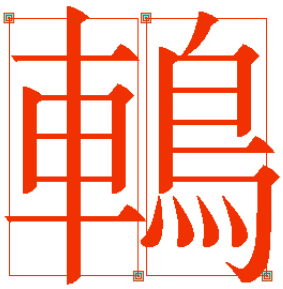
```
<cdl>
 <comp char='車'
  points='0,0 64,128' />
 <comp char='鳥'
  points='68,0 128,128' />
</cdl>
```

[Technical note: Any coordinate system could be used. In the current implementation, we have found it convenient and efficient to use coordinates from 0 to 128, where (0,0) is the (left, top) corner and (128,128) is the (right, bottom) corner. Decimal points are allowed for fractional coordinates (such as 3.14159), so the precision is practically unlimited and does not depend on the number (such as 128) that is chosen for the maximum coordinate value.]

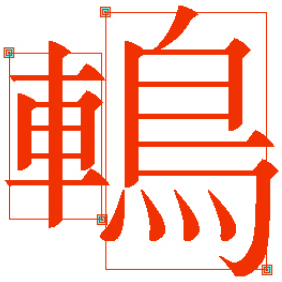
The Wenlin Stroking Box

The illustrations below are screenshots from the *Stroking Box* in an unpublished CDL-development version of the Wenlin software. The interface looks like this:

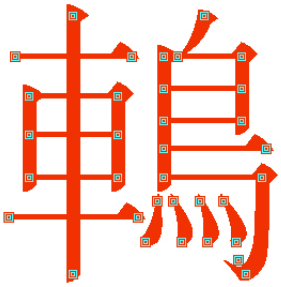
It's the same interface that the published educational version uses when displaying stroke-by-stroke animation, but with two additional capabilities: one for displaying the CDL code, and one for modifying the positions by pointing with the mouse and dragging on "control points". The illustration below shows the control points at the corners of the bounding boxes:



Here's what it looks like if we drag some points around to make the *chariot* smaller and the *bird* bigger:



It's possible to convert automatically any description that uses **comp** elements into one that uses only **stroke** elements (eighteen of them in this case). When this is done, all control points are at the stroke level, as in the following illustration:



The all-strokes description allows us to adjust the position of any stroke, add or remove strokes, and so forth, perhaps to describe a peculiar variant like this one:

Seven-Stroke *Chariot*

To demonstrate the **stroke** element, here's a description of 車:

```
<cdl>
 <stroke type='h' />
 <comp   char='日' />
 <stroke type='h' />
 <stroke type='s' />
</cdl>
```

That's pretty simple. In fact, it's too simple for some purposes, and we'll add some important details shortly, but first let's explain what we have so far.

The character 車 is described as a sequence of four elements: a stroke, a component, and two more strokes. (This is the order in which the character is traditionally written: basically from top to bottom, but with the long vertical stroke last.)

The component 日 needs to have its own description, which we'll supply further below. As we'll see, 日 itself has four strokes, so 車 has a total of seven strokes. Now, what do **h** and **s** stand for?

H = Heng = Horizontal

Each of the **stroke** elements has a stroke **type** attribute, specifying one of the 36 types, represented by the ASCII abbreviation that is used in the Unicode CJK Stroke name. 車 starts with an **h** and ends with an **s**.

The first stroke in 車, **h**, goes horizontally from left to right. Its name is an abbreviation for Mandarin *héng* 横 'horizontal', which is the traditional name for this stroke type. (It's a coincidence that *héng* and *horizontal* both start with the letter h.)

The last stroke in 車, **s**, starts at the top center and goes straight down. The letter **s** is an abbreviation for *shù* 竖 'vertical stroke'.

(The 36 members of Unicode's new CJK Strokes block [see the table above] make it possible to use the Unicode characters or numbers as alternatives to the Mandarin-ASCII stroke-name abbreviations.)

Here Comes The Sun

We promised to show the description of 日, which is needed for a complete interpretation of the description we gave for 車. The character 日 by itself means 'sun' or 'day'; however, as a component in 車, it's simply a reusable arrangement of four strokes.

```
<cdl>
 <stroke type='s' />
 <stroke type='hz' />
 <stroke type='h' />
 <stroke type='h' />
</cdl>
```

The only new stroke type here is **hz**, which stands for Mandarin *héng-zhé* 横折 'horizontal-turning'. This is a stroke that turns a corner; in this case, the top-right corner of 日.

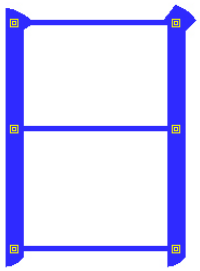
We could choose to describe 車 directly in terms of seven strokes, and avoid using 日, but it would be a longer description (with seven elements instead of four). The reuse of components makes a collection of descriptions not only more efficient in terms of space, but also more internally consistent and easier to develop and maintain.

Getting To The Points

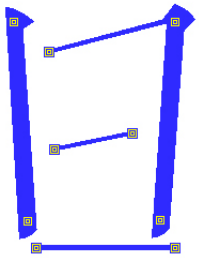
To enable the full power of CDL to describe exactly how to display a character, each **stroke** element has a **points** attribute specifying placement of its control points.

Just as with components, a graphical interface enables us to move the control points around visibly on the screen, by dragging on them with a mouse. Therefore, it's often unnecessary for the person who creates, edits, or uses a CDL description to be concerned with the actual numerical coordinates.

This is how the control points are displayed in the Stroking Box:



By dragging them apart, we can see the four individual strokes:



The number of control points depends on the stroke type. The simple **h** and **s** types each require only a starting point and an ending point. The more intricate **hz** uses three points: the start, the corner, and the end. (The most intricate stroke type is named **hzzzg** and uses six points. It occurs in the character 乃.)

For those of you who are interested, here is a description of 日 with **points** included:

```
<cdl>
 <stroke type='s'
  points='0,0 0,128' />
 <stroke type='hz'
  points='0,0 128,0
          128,128' />
 <stroke type='h'
  points='0,60 128,60' />
 <stroke type='h'
  points='0,128 128,128' />
 </cdl>
```

Information Overload?

Before your eyes start to glaze over, we recommend that you skip this section and move on to the next one, unless you're really wanting more technical details right now! This is the last chunk of code we'll show in this article.

Here is our most complete description of 日 'sun':

```
<cdl char='日' uni='65e5'
   variant='0'
   points='24,8 104,120'>
 <stroke type='s'
  points='0,0 0,128'
  tail='long' />
 <stroke type='hz'
  points='0,0 128,0
          128,128'
  head='cut' tail='long' />
 <stroke type='h'
  points='0,60 128,60'
  head='cut' tail='cut' />
 <stroke type='h'
  points='0,128 128,128'
  head='cut' tail='cut' />
 </cdl>
```

The first line opens the **cdl** element, and the last line ends it with **</cdl>**. This time we've included four optional attributes for the **cdl** element: **char** and **uni**, which simply say this is a description of 日 (**U+65e5**); also **variant**, which assigns the description to a particular variant identifier, in case we have multiple descriptions for the "same" character. (We discuss variation and unification further below.)

The fourth optional attribute is the **points** for the character as a whole. (That is, the first **points** attribute, before the two **stroke** elements.) As we've used it in 日, it has the effect of making 日 slightly narrower and shorter than the regular square, and this improves its appearance, partly by avoiding having strokes running along the outer edges of the square. (By the way, this first **points** attribute only affects the display of 日 as a standalone character. When 日 is used as a **comp** in another character, such as 車, the **points** attribute of the **comp** tag in that character overrides this one.)

Finally, the strokes have optional attributes such as **tail='long'**, which affect subtle decorative features of the *Sòng* style outlines, especially where strokes meet. CDL descriptions can be displayed

in a "plain" style without decorations, in which case these decorative attributes are superfluous.

Applications

CDL began as a way to implement a learning tool, yet applications of CDL technology are more general and far-reaching, and provide solutions to difficult problems facing the international encoding community. Some of the most important applications for CDL include:

(1) Teaching stroke order
(2) Representing unencoded characters
(3) Distinguishing unified variants
(4) A font format with built-in indexing
(5) Handwriting recognition
(6) Indexing by strokes & components
(7) Maintaining & Building Unicode
(8) Optical Character Recognition (OCR)

CDL has already been put to all of these uses, in varying degrees. Most of the remainder of this article will discuss these applications.

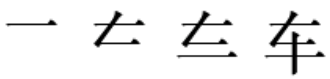
• Application 1: Teaching Stroke Order

We've already shown pictures of the Stroking Box, an educational software feature which was (and still is) the original application for CDL. It displays animation of a character being written stroke-by-stroke. Each stroke is written gradually so that the learner can see where it begins and ends. There are controls for setting the speed, colors, style, and stroke thickness. Thousands of students have used the Stroking Box since *Wenlin Software for Learning Chinese*, version 1.0, was published in 1997.

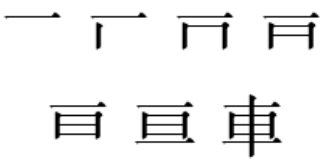
In addition to the dynamic Stroking Box, Wenlin displays static stroke-order diagrams like the following.

日 'sun/day' (4 strokes):

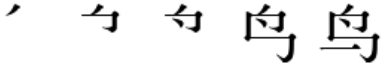
车 'car' (simple form, 4 strokes):



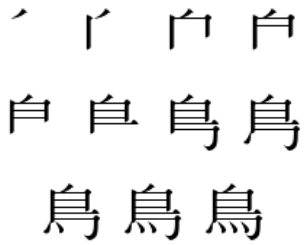
車 'car' (full form, 7 strokes):



鸟 'bird' (simple form, 5 strokes):



鳥 'bird' (full form, 11 strokes):



Our primary reference for stroke order and other orthographic questions is a PRC national standard published under the title of 现代汉语通用字笔顺规范 (ISBN 7-80126-201-8).

Of course, CDL descriptions can and should be created as needed to represent orthography in other locales, whenever there is variation between the standards. (Cf. the **`variant`** attribute mentioned above under "too much information"; and the discussion below of "unification and variation".)

• Application 2: Representing Unencoded Characters

We've already demonstrated how CDL can represent any combination and arrangement of *Sòng*-style strokes and components, regardless of whether there is a Unicode number assigned to it.

Such a CDL description can be included directly in a text file, interspersed with ordinary text (perhaps including other XML-based markup). CDL-enabled software will display the character simply by interpreting the CDL, without needing a customized font, a *Private-Use Area* code point, or an embedded graphical image. Users won't see codes, they'll just see characters.

So far, Wenlin is the only software with this ability to display embedded CDL (and its CDL features are mostly unpublished or undocumented), but we expect this ability eventually to be included by the mainstream web browsers, word processors, operating systems, and other software.

When end-users are able to create CDL descriptions of characters, encoded or not, and can embed these in online documents, web spiders crawling the internet will be able to collect them automatically, and use associated metadata to feed these descriptions into encoding and variant-mapping processes.

CDL: the Elements of CJK(V)

Imagine that instead of typing the letters of the alphabet on your computer keyboard to write English words, you needed a keyboard with one key for each *word*. This would be a big keyboard indeed! Even if the size were not a problem, no matter how may words it included, there would always be missing words that couldn't be typed.

Another way to think about it is to imagine an inflexible spell-checker that absolutely refuses to let you write any word not in its dictionary. The situation in CJK encoding is rather similar to this, to the extent that the Han *character* (comprised of strokes) is like the English *word* (comprised of letters).

CJK input methods, as a rule, only allow inputting a limited set of characters. They don't allow constructing new (missing or unencoded) characters by combining strokes or components. In this way, they really are like the absurdly enormous English keyboard we were imagining that has no letters, or like a mad(dening) spell-checker that won't allow exceptions and can't be turned off.

We should clarify that for many purposes the situation isn't really quite as bad as these analogies suggest, because most CJK "words" are in fact written as strings of two or more characters. Thus, ten thousand or so characters are enough to write hundreds of thousands of words. New words are created all the time, but, partly due to technical limitations for typesetting, new Han characters are rarely created anymore. The current Unicode CJK set is sufficient for most ordinary modern vocabulary used in newspapers, etc.

Nevertheless, these analogies do accurately portray the difficulties people are up against when they wish to go beyond the limitations of a finite set of Han

characters and represent, on a computer, the characters that were so freely and frequently created in the past with handwriting and woodblock printing.

Jabberwocky Unencoded?

If English writers were encumbered by the restrictions currently afflicting CJK writers, the situation would be obviously intolerable. For example, in production of editions of Shakespeare, or Chaucer, or Lewis Carroll, it would be impossible to use the original spellings, unless they happened to be in a dictionary used as a source for a computer encoding.

When the famous linguist Y. R. Chao (趙元任) translated Lewis Carroll's poem *Jabberwocky* into Chinese, he invented some new Han characters and words to match the invented English words. Some modern avant-garde artists have produced beautiful calligraphy using made-up or nonsense Han characters. When such creative people start using CDL, for better or worse, they're going to have a lot of fun breaking the Unicode barrier. CJK *Finnegans Wake* here we come!

• Application 3: Distinguishing Encoded Variants

By design (and by *necessity*), Unicode unifies (i.e. lumps together under a single code point) many variants of CJK characters that differ in small (and presumably, mostly non-distinctive) ways, such as the presence or absence of a dot, or whether stroke segments are joined or separated.

Such variations can be very important for some purposes. For example, different national standards sometimes assign different official stroke counts to the "same" character, and the stroke counts determine the organization of dictionaries, etc.

For another example, historical Chinese texts can often be dated on the basis of a single stroke being intentionally omitted in taboo avoidance of the name of the reigning emperor.

Shakspere, Shaksper, Shaxper: Unified Variants?

The Bard himself reportedly spelled his name variously in his own day as Shakspere, Shaksper, Shaxper, and Shakespeare. If English words needed to be encoded like CJK characters, and the variant spellings were unified, then we'd be unable to write the preceding sentence (without resorting to some non-standard text representation practice). The digitization of original editions of important historical texts would be hindered by the lack of prior lexicographic and encoding work. This is the *status quo* for CJK texts, and it presents an unacceptable and unnecessary bottleneck.

Given that a goal of Unicode is to provide an international CJK character encoding, transcending temporal and political boundaries, Unicode doesn't have the luxury of narrow scope. In fact the problems faced in Unicode's CJK encoding are greatly compounded, requiring in effect innovative and comprehensive lexicographic work to be the gating constraint on the digitization of texts.

Clearly, an adequate system needs to be developed and promoted for wide adoption, to make possible the short-term progress and long-term success of CJK digitization projects. CDL enables computerized encoding and structured combination of the minimally distinctive features that are lost in Unicode unification.

Unicode has recognized the need for identifying variants that share a code point, and for this purpose has established the "variant selector" mechanism. The need remains, however, to assign a precise meaning to each combination of code point and variant selector (see Hiura & Muller 2006: *UTS #37*, <http://www.unicode.org/reports/tr37/>). CDL is ideal for this purpose. Each code point can have multiple CDL descriptions, one description for each variant, uniquely identified by a selector.

• Application 4: CDL Fonts

A database of CDL descriptions serves as a font that can be used by a CDL-aware application to display any CJK text. A CDL font has some valuable characteristics.

A CDL font can be compressed to an extremely small size, much smaller than any other format with comparable quality. This is because the vast majority of characters can be described with just two **comp** tags each, and for such a character all that needs to be stored is two Unicode numbers and a few coordinates. The reuse of component descriptions means that each component only needs to be described once in the font. Wenlin's CDL font file of over CJK 50,000 characters occupies less than one megabyte of storage (about 18 bytes per character).

A CDL font displays very quickly, at a speed comparable to conventional font formats (TrueType, etc.).

We have implemented utilities for converting CDL descriptions into other font/graphics formats including Postscript, Scalable Vector Graphics (SVG), and METAFONT.

Since the descriptions are standards-based, and components are reused, the CDL font is inherently very consistent. Contrast the CJK fonts that have been used, for example, to print editions of *The Unicode Standard*, which by necessity have been combinations of fonts that don't really go together and therefore are inconsistent in the ways that particular components and strokes are displayed in different characters.

Adding a new glyph to a CDL font is generally just a matter of combining a few elements and adjusting a few coordinates. That's much easier than adding a new glyph to an ordinary font, especially if you are trying to match the style.

A single CDL font can be displayed in different styles and weights. Thus it really serves as a font family. (In this respect, CDL has something in common with the METAFONT language, and with Multiple Master fonts.) The CDL descriptions themselves determine the essential shape or "skeleton" of a character in a general way. The CDL interpreter is responsible for deciding stroke thickness, and whether to include optional decorations. Here are two examples of the same text, displayed using the same CDL descriptions, but different choices of style:

• Application 5: Handwriting Recognition

Since CDL describes the way each CJK character is actually written by hand, stroke by stroke, it naturally has applications for handwriting recognition. This capability of CDL is demonstrated by the brush tool input method of the published Wenlin software. When the user selects

the brush tool, a window is displayed containing a square region in which the user can write a character by hand, using a mouse or a pen input device. If the handwritten character is similar enough to any of the 50,000 characters in the CDL database, then the character is inserted (as Unicode) into the text which the user is editing. The recognition accuracy rate is high, provided that characters are written with standard stroke count and stroke order. There is an option to ignore stroke order, but accuracy is higher if that option is turned off and the standard order is used. There is another option for showing a list of characters similar to the handwritten character, from which the user can choose.

The recognition algorithm currently employed is essentially simple. It counts the strokes in the handwritten character, then compares it with all the characters in the CDL database with the same number of strokes, and finds the most similar character(s), based on the distance (sum squared difference) between the corresponding stroke coordinates. Perhaps surprisingly, this is not only very fast (a fraction of a second), but also accurate.

Currently only the beginning and ending coordinates of each stroke are used. Even higher accuracy could be achieved by taking into account intermediate points along the path of a stroke, rather than only the beginning and ending points. The algorithm could be extended to handle cursive handwriting, in which strokes are not necessarily separated by lifting the pen. We have experimented with using the CDL database for training a neural network to recognize stroke types.

Theoretically, it should be possible to implement a recognizer which would create a complete CDL description based on a handwritten character, even if there were no matching description in the CDL database. In that situation, the recognizer could insert the CDL description itself (as XML), rather than a Unicode character code, directly into the text. In this way one could input unencoded characters such as our first example of the "chariot plus bird" combination.

• Application 6: Indexing by Strokes and/or Components

Many CJK dictionaries and other reference works include indexes for locating characters according to their strokes and/or components. (Pronunciation is another popular key to organizing CJK dictionaries, but of course it requires knowledge of the pronunciation, which for someone consulting a dictionary is sometimes like putting the cart before the horse.)

Components used for organizing dictionaries and indexes are known in English as radicals. A conventional set of 214 radicals has been used to organize many dictionaries, starting with *Zihui* in the 15th century, and including the famous *Kangxi Dictionary* published in 1716. Our "chariot plus bird" character might be listed under either the *chariot* radical or the *bird* radical. (You would have to guess which of the two, and generally try both; but don't bother since this character probably isn't in your dictionary!)

Thousands of characters may share the same radical, just as thousands of words written alphabetically may start with the same letter. For alphabetical indexing, the order depends on the non-initial letters in a word. Early radical-based dictionaries listed same-radical characters in essentially random order, but eventually it became common to sub-group using residual stroke count, which is the number of strokes a character would have if its radical were removed. Characters with the same radical and same residual stroke count were still listed in random order, and this practice survives in the Unicode standard. Some 20th-century dictionaries, however, started using a third organizing principle to determine the order of characters with the same radical and same residual stroke count. This principle takes into account the types of the residual strokes, using classification systems related to what are now the CDL stroke types **h**, **s**, etc. The system which has become standard in PRC distinguishes five categories of stroke types with a conventional ordering. The CDL specification includes the mapping of the 36 CDL stroke types to the five stroke categories.

Owing to the difficulties with radicals, many reference works employ an alternative method instead of (or in addition to) radicals. This method simply orders characters by their total stroke counts, and orders characters with the same stroke count by the type of the first stroke. If the stroke count and first stroke type are the same, the type of the second stroke is taken into account, and so forth.

The Wenlin software uses its CDL database to provide lists of characters ordered by radical, residual stroke count, and residual stroke type; and also lists ordered by total stroke count and stroke types.

We used CDL to prepare the radical and stroke indexes for the *ABC Chinese-English Comprehensive Dictionary*, edited by John DeFrancis, published in book form by the University of Hawaii Press, 2003 (ISBN 0-8248-2766-X). Wenlin Insitute also provided the Unicode Consortium with CDL-based data to assist in the construction of a radical index for a subset of the Han characters in the Unicode Standard 5.0.

• Application 7: Maintaining and Extending Unicode

Organizing the more than 70,000 CJK characters already in Unicode, and the thousands more destined for future encoding, is an extremely challenging task. This is especially true since the main method of organization used so far has been the venerable *Kangxi Dictionary* system just described. Characters are first grouped by radicals, but different lexicographers classify the same character under different radicals (like chariot versus bird). Then characters are subgrouped by residual stroke count, but different lexicographers often count slightly different numbers of strokes in the same character. Finally, the order of characters with the same radical and same residual stroke count is random. (Unicode follows the random order of *Kangxi Dictionary* for characters that are included in that dictionary.)

Under these circumstances, it would be unfair and pointless to blame anybody for the fact that some characters have accidentally been encoded twice. (That is, sometimes two different Unicode numbers have been assigned to the same character.) Such mistakes were bound to occur when it is so easy not to find a character which is in a different location from where one expects to find it. When an obscure character isn't found among the already encoded characters, after searching several times, the conclusion is naturally reached that the character hasn't been encoded yet. This problem has been greatly exacerbated by the fact

of variation. Two forms of the "same" character may be different enough that they are not easily recognized as the same character in spite of unification guidelines.

If the process continues without better methods, we can only expect the frequency of mistakes to increase, since the larger the haystack, the harder it will be to find any particular needle.

Another problem with the never-ending addition of thousands of Han characters to Unicode, is that each new version of Unicode causes existing fonts and input methods to become out of date.

CDL can help to solve these problems, or at least alleviate them. First of all, when CDL technology becomes widely available, there will be less need or urgency for encoding obscure Han characters. Digitization of ancient Buddhist manuscripts, for example, will be able to proceed using CDL to represent unencoded or rare variant characters, rather than waiting years for those characters to become part of Unicode, or creating incompatible, temporary fonts and documents using private-use code points.

When addition of rare Han characters to Unicode does continue, a complete CDL database of the already-encoded Han characters can be searched exhaustively to make sure that each candidate character really is unencoded, and not just hiding under a non-obvious radical. To facilitate this kind of searching, each encoded character should have not only *one* CDL description in the database, but *multiple* descriptions corresponding to different ways of writing or analyzing the "same" character. The search algorithm should use robust and "fuzzy" matching to find similar or identical descriptions. Similarity can be based on either components or strokes, or both. CDL's ability to automatically convert a component-based description into a strokes-only description can be exploited, to avoid the problem of variant component-analysis. For example, 章 can be described as 音 over 十, or as 立 over 早, or as 立 over 日 over 十: when converted to a sequence of strokes, the descriptions are identical.

The problem of fonts becoming out of date can be alleviated in two ways: by using CDL to display a character that is missing from a font; and by using CDL descriptions to assist in font creation. Similarly, input methods based on strokes could be updated much more easily using CDL.

All these solutions will be greatly facilitated if publication of at least one CDL description for a character is made a prerequisite for adding it to Unicode.

• Application 8: Optical Character Recognition

Another exciting (and still experimental) application of CDL is to the problem of CJK *optical character recognition* (OCR). Current OCR technologies, when applied to CJK text, yield largely binary results: either a printed character is identified with an encoded character, or it is not. Where matching fails or is imperfect, either a completely wrong character is selected, or else OCR fails completely. Using CDL, the results of partial or failed OCR matches become meaningful. Just as the inability to recognize a single letter in OCR of English text might not result in failure to recognize the word, and just as the absence of an English word from a spell-checker's dictionary needn't signal complete OCR failure, so too OCR of unencoded or damaged CJK characters can succeed where current CJK OCR fails, by including CDL in the OCR output.

Even with a well-developed system for semi-automatically converting highly constrained bitmaps to CDL, the exact stroke-order information may be missing, and the resulting description will need human checking anyway. And initial creation of the highly constrained bitmaps is a real bottleneck: with unconstrained bitmaps (i.e., scans of variable-quality printed examples), automatic CDL creation is going to be difficult at best. But if the writer/inputter can stroke over a bitmap template of whatever quality, build the description stroke by stroke with reference to an original (and optionally convert the strokes to components somewhere down the line), this would be ideal, a very natural CDL creation process for CJK writers.

Other Scripts: Cuneiform, Etc.

The applications of CDL technology for scripts beyond the CJK world are just as important. We have discussed this with researchers working on other scripts, contributing to the on-going development of Unicode to handle digital text representation of the scripts of the world. For example, there has been growing interest in developing CDL schemes for Cuneiform scripts, Tangut (西夏 *Xīxià*, an extinct CJK offshoot recently proposed to Unicode), Egyptian Hieroglyphs, Mayan, and other complex scripts with componential basis and special component and character positioning requirements.

Such scripts are similarly limited by the open-endedness of their character sets due to historical and local variation, and due to the ill-defined nature of the bounds of the higher-level units of writing. There are, nevertheless, basic script elements identifiable in each of these scripts, that can be employed in conjunction with CDL to structure text, to remove the encoding bottleneck, and to empower the paleographers, lexicographers and linguists who must ultimately seek to resolve these problems.

In the End ...

CDL is a core infrastructure technology, providing a rock-solid framework for data structure, data storage and data interchange, and CDL should be adopted internationally in work to digitize and preserve humanities collections and paper and archeological archives.

Because CDL is *pure Unicode* and *pure XML*, and because its applications for CJK are clearly based on *traditional orthographic standards* active across CJK scripts, *CDL is completely standards-based*. CDL technology bridges an important gap, putting real intelligence into fonts, and giving CJK digitization projects the freedom which roman-based orthographies take for granted. The returns from this technology will be large indeed, and shared globally, as improvements are made to data stability, content, and access.

We are working to bring the CJK part of the CDL database online, so that everyone can use this technology. And we are working to promote and refine CDL methods to make them more flexible, more generally applicable.

... CDL Unlimited!

To learn more about the CDL project, please visit us online:

<http://wenlin.com/cdl/>