Searching Beyond the Tower of Babel:
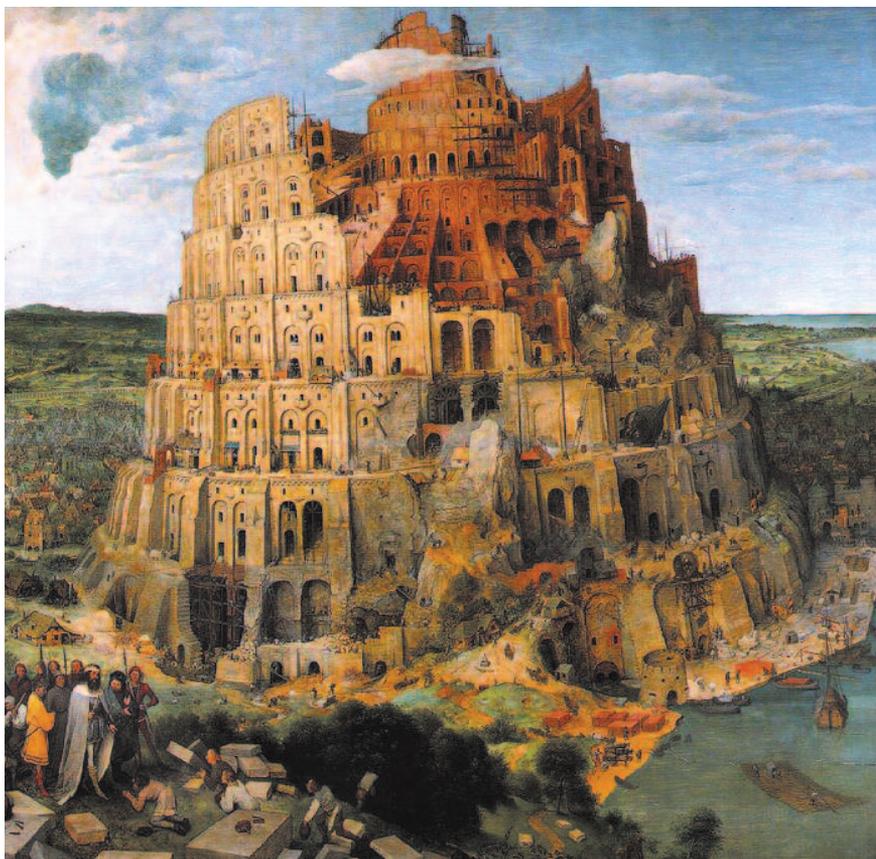# Unicode and Text Retrieval

The Tower of Babel fell, and all of the earth's languages were "confused." Nevertheless, while understanding each other's speech remains a confusing prospect, we have made great progress in sifting through each other's computer texts.

This article uses the dtSearch® product line and text searching examples to discuss international language search issues. Many of the concepts relating to Unicode, however, have broad applicability.

> Programming in a certain level of insensitivity as the default for Unicode searching is a technique that dtSearch,® for example, uses to avoid false misses resulting from Unicode anomalies.

The Tower of Babel, by Pieter Bruegel in 1563.

Unicode

In the beginning (not the Biblical beginning, but the DOS and early Windows beginning), the ASCII character set was the vehicle for expressing different languages. Each character had an allocation of a single byte, or eight bits. Taking eight bits to express each character made 256 (or 2 to the 8th power) characters available in the ASCII character set. English characters, punctuation, control characters (carriage return, backspace, etc.) and numbers took

And the Lord came down to see the city and the tower, which the sons of men had built. And the Lord said, "Behold, they are one people, and they have all one language; and this is only the beginning of what they will do; and nothing that they propose to do will now be impossible for them. Come, let us go down, and there confuse their language, that they may not understand one another's speech."

So the Lord scattered them abroad from there over the face of the earth, and they left off building the city. Therefore its name was called Babel, because there the Lord confused the language of all the earth; and from there the Lord scattered them abroad over the face of the earth.

— *Genesis*

up the first 128 characters, leaving only 128 characters to work with for non-English text.

The major successor to the ASCII character set is Unicode, as defined by the Unicode Consortium, *www.unicode.org*. Unicode is now as close to a universal standard as there is in the computer world. All recent major operating systems, Web browsers, Office software and the like now support the Unicode standard.

Unicode UCS-16, for example, uses a double-byte or 16-bits to express all language characters. With 16-bits, the number of characters available skyrockets from 256 to 65,536 (or 2 to the 16th power). The Unicode Consortium assigns the 65,536 available characters into groups called subranges. Each subrange has 256 characters. Subrange 0 includes English letters and numbers, along with commonly used punctuation. For example, the letter *A* is character 65 of subrange 0.

While *A* is the 65th character of subrange 0, Unicode expresses this position in base 16 hexadecimal format as U+0041. (Unicode characters by convention start with U+. And 65 in base 10 is the same as 41 in base 16.)

Although the U+ syntax is the convention for describing a Unicode character, the computer stores characters as individual bytes ranging from 0 to 255. For the English letter *A*, the representation under UCS-16 would consist of two bytes: one byte representing subrange 0, and another byte representing a 41 hexadecimal value. The last section in this article discusses issues relating to byte order: "big endian" vs. "little endian," and forensics.

The Unicode character designations distinguish between capital and lowercase letters. While *A* is U+0041, *a* is U+0061. Unicode also distinguishes accented characters, so, for example, *â* is U+00E2. What Unicode does not define is different fonts for the same character, or different ways of drawing a letter, such as Arial or Times Roman. Nor does Unicode define other letter attributes, such as italics, underlining, bold, size, or color.

While these are not part of the Unicode standard, applications such as Web browsers and word processors may nonetheless hold and display this information along with Unicode text.

Different alphabets generally have different subranges associated with them. French, German and Spanish share subrange zero with English. Arabic, Hebrew and Greek each have their own subrange. Japanese, Chinese and Korean each have a number of subranges because of the large number of ideographs in those languages. Unicode even has a subrange for mathematical symbols.

Unicode text is not without some odd properties. Each character contains a unique value. However, certain characters can look identical and yet have very different meanings. For example, π the mathematical symbol and π the letter of the Greek alphabet look alike visually, but have different Unicode encodings. Before clicking on that "special email-only offer" link, see *Unicode*

Unicode and Link Deception

When you move the cursor over a link in an email or a Web site, the URL appears in Unicode, offering reassurance that you know where the link is heading. But what may appear as a favorite Internet shopping site, *www.XYZshoppingsite.com*, may actually incorporate characters from other Unicode subranges masquerading as *www.XYZshoppingsite.com*.

Underlying this deception is the fact that very different Unicode characters can share the same visual representation. For example, the English lowercase letter *i* (U+0069) looks the same as the roman numeral *i* (U+2170).

Accordingly, if you click a link in an email, think twice before revealing any confidential information such as a credit card number. Instead, retype the URL in a browser address bar, so you know exactly where you are going.

> dtSearch's UK distributor has a Language Extension Pack for over a dozen very diverse noise word lists and stemming rules, including Belarusian, Czech, Danish, Dutch, Estonian, German, Finnish, French, Greek, Hungarian, Italian, Latvian, Lithuanian, Norwegian, Polish, Portuguese, Russian, Slovak, Spanish, Swedish, Turkish and Ukrainian.

and Link Deception for a caution that arises from this visual similarity issue.

Unicode UCS-16 radically increases the number of international language characters available relative to the old ASCII standard. However, it is less efficient than the ASCII 8-bit encoding in that it takes up twice the space to represent each character. A newer UCS-32 standard is even less efficient in that it takes four times the space as the old ASCII standard to represent a single character. This outcome led to the development of yet another Unicode encoding standard, UTF-8.

Under UTF-8, all English characters and basic punctuation—i.e. the characters up through 127 of subrange 0 under UCS-16—have single-byte representation. Other characters have two or three byte representation equivalent to their UCS-16 value. Because the majority of characters in most English text fall in the single-byte category, UTF-8 is generally a more efficient means of representing such text.

Unicode and Searching

While the Unicode standard is a great advance in making the text of any language easily representable, Unicode poses a unique set of issues for searching. One issue is Unicode's representation of the same numeric values with different encodings in different subranges. Performing a Unicode search for subrange 0 character *3* would not automatically retrieve the Gujarati number *3*. Likewise, a search for character *A* in subrange 0 would not find a very similar letter *A* in another subrange, or even an accented or lowercase version of *A* in subrange 0.

Programming in a certain level of insensitivity as the default for Unicode searching is a technique that dtSearch, for example, uses to avoid false misses resulting from Unicode anomalies. With the default insensitivity, a search for *3* would find the Gujarati *3*. A search for an *A* would retrieve lowercase and accented versions of that letter. While the defaults for search sensitivity are adjustable, generally the broader the formulation of a Unicode search request, the smaller the number of false misses.

The text searching process does not just look for an exact sequence of letters or numbers that form a word or phrase, but it also looks for word variants, Boolean operators, contextual relevancy-ranking, and the like.

What is most remarkable is what can work universally in a completely language-neutral capacity, using only Unicode and search engine functionality.

Language-Neutral Search Functionality

One of the most important issues for a search engine is sifting through typographical deviations that occur in words. Such deviations may result from different spellings of the same basic word, such as *defense* for an American spelling, and *defence* for a British spelling. They can also result from errors in scanning and OCR (Optical Character Recognition) processing. These character deviations can even result from mistyping, a problem that frequently plagues emails and email attachments.

In dtSearch, the fuzzy search algorithm adjusts from 0 to 10 at search time to look for typographical deviations. A search for *Tower of Babel* with a fuzziness of 1 would find *Tower of Babell*. With a fuzziness of 3, the search would also find *Tour of Bagel*. While subject to adjustment, the fuzzy search default is to locate the first couple of letters in a word, and then to look for deviations in letters beyond that.

Fuzzy searching is similar to Unicode sensitivity adjustment in that the fuzzier a search, the more false misses it will avoid. In the case of fuzzy searching, however, it is also true that the fuzzier the search, the more false hits it will retrieve. While *Tower of Babell* might be a desirable hit, a *Tour of Bagel* coffee shop might be completely off point. dtSearch,

for example, allows fuzziness adjustments upon entering a search request—as opposed to hardwiring it into the search index—to enable easy modification at search time. See *Indexed vs. Unindexed Searching* for a further discussion of search indexes.

Because fuzzy searching looks for deviations in any characters relative to a search request, it works for all international languages, without distinction. Similarly, searching with wildcard character placeholders, such as *?* to hold a single character space or * to hold any number of character spaces, also works universally. Another example of a fully language neutral search option is natural language searching.

dtSearch, for instance, processes natural language expressions through the vector-space method of assigning a relevancy rank to each document based on the density and rarity of search terms in that document relative to the entire document collection. In a search for *corpx takeover memo*, if *memo* appears in 10,000 documents, *takeover* appears in 2,500 documents, and *corpx* appears in only 3 documents, the *corpx* documents would come up as the most relevant.

Since dtSearch vector-space natural language processing depends on the density and rarity of search terms in a document collection, it is language independent. If anything, the key factor in the operation of natural language searching relates more to the requirements of the document collection, then the actual language. For example, if the same *corpx takeover memo*

Indexed vs. Unindexed Searching

Indexed searching is the most common method of searching through very large document collections, although exceptions exist. In forensics, an unindexed search may be useful for making an initial determination if a document collection is even relevant. In single-pass operations such as email filtering, unindexed searching is also the norm. Generally, however, indexed searching is the standard for the sole reason that even across a very large document collection, an indexed search is usually instantaneous.

The following excerpt from an article published in *PC AI Magazine* further explains indexed vs. unindexed searching:

Randomly opening documents to find the correct one is a form of unindexed searching. Advanced unindexed searching might iterate over files looking for specific key words. The main drawback of this searching technique is slow speed; it is particularly inefficient for successive searching.

The alternative to unindexed searching is indexed searching. As document collections grow from megabytes to gigabytes to terabytes, searching these large document sets usually mandates some form of indexing. Just as it is often faster to locate a particular topic using a book's index rather than thumbing through each page, it is faster to search for information using indexed computer files. Moreover, indexing with a modern full-text search program is easy — simply click on directories or drives and the search program does the work.

For complex document formats such as Word, Access, Excel, PowerPoint, PDF, ZIP, HTML, XML, etc., a code at the beginning of each document informs the program what formatting to expect as it parses the document. Good indexing programs can even fully index the occasional corrupt document (the one that a word processor suddenly refuses to open).

Indexed search engines also operate over a network. If EnterpriseX has a 5,000-page policy manual, each employee could separately index it, saving and searching the resulting index. Alternatively, for improved efficiency, the network administrator could build one policy manual index for shared access. Thousands of people could then simultaneously search the index, with the network software simultaneously updating the index.

The search techniques that this article describes, including Unicode, fuzzy, Boolean, wildcard, etc., at least for the dtSearch product line, relate equally to indexed and unindexed searching. The major exception to this rule is natural language searching with vector-space relevancy rankings, as this option requires indexing to determine word frequency in the documents.

search took place across a *corpx memo* document collection, then *corpx* and *memo* might be the most prevalent terms, and *takeover* documents would receive the highest relevancy rank.

A variant of natural language searching, which automatically ranks retrieved documents according to naturally occurring search term relevancy, is variable term weighting. This also works across different languages in the same way. For example, a search for the equivalent of *corpx:7 takeover:-3 memo:2* would work in any language, with *corpx* receiving an artificially high relevancy score, and *takeover* receiving a negative relevancy score.

Other search types that work universally across any language include: Boolean (and/or/not), phrase, proximity, and numeric range (subject to the Unicode considerations mentioned above). In contrast to these language-neutral search types, a few search features, most notably relating to noise word lists, stemming rules and concept/synonym/thesaurus definitions, do benefit from language-specific customization.

Language Customization

A search engine keeps its index size more compact and its general operations more efficient by essentially ignoring a few dozen words that are so common in a particular language as to be, for purposes of searching, mere "noise." English-language examples would include *the*, *of* and *for*. This English-language noise word list does not work, for example, for French, which would require adjustments to *le*,

la, *de*, *pour*, etc, for optimal performance.

Another item that requires language-specific adjustment is stemming, a search feature that finds linguistically related words, such as in English *learn*, *learned*, *learns* and *learning*. Different languages have very diverse root word endings. Covering the pan-European area, dtSearch's UK distributor (*www.dtsearch.co.uk*) has assembled a Language Extension Pack for over a dozen very diverse noise word lists and stemming rules, including

Belarusian, Czech, Danish, Dutch, Estonian, German, Finnish, French, Greek, Hungarian, Italian, Latvian, Lithuanian, Norwegian, Polish, Portuguese, Russian, Slovak, Spanish, Swedish, Turkish and Ukrainian.

However, even simple stemming adjustments will not work for all languages. For example, in Arabic, the surrounding context for a word (*my*, *your*, *the*, *a*, masculine/feminine, etc.) can be expressed as characters in front of or behind the word. To use an English-language analogy, *the apple* or *my*

Search Types	General Language-Neutral Application
Fuzzy, adjustable from 0 to 10 to sift through typographical deviations	**Yes**
Natural language with vector-space relevancy ranking	**Yes**
Variable term weighting	**Yes**
Phrase	**Yes**
Boolean (and/or/not)	**Yes**
Proximity and directed proximity	**Yes**
Wildcard	**Yes**
Macros	**Yes**
Numeric range	**Yes**
Fielded data (alone or combined with full-text searching)	**Yes**
Phonic/Soundex	**No**
Stemming and noise word adjustment	**No; requires a Language Extension Pack or equivalent to work (see text discussion)**
Advanced language-specific morphology (e.g. for Asian languages)	**No; requires language analyzer plug-in through API**
Synonym/concept/thesaurus	**No; requires language-specific plug-in through API**

All terminology relates to the dtSearch product line.

apple are different prefixes or suffixes added to *apple*. To search for text in these languages requires adding the wildcard * in front and back of the root word as in *apple*.

For even more extreme morphological issues, such as those in Asian text, an API (application programming interface) allows a developer to plug-in full-fledged language-specific analyzers. For thesaurus/concept/synonym searching (going beyond Boolean searching, macros, and user-defined synonym rings) an API allows a developer to plug in a complete replacement to the built-in English-language thesaurus.

Unicode and Forensics

Most documents contain a heading indicating the language system, enabling easy search engine interpretation. Forensically recovered data is often an exception to the rule. Examples of forensically recovered data would be data recovered through a file undelete process, data recovered from unallocated computer space, or data recovered from partially recovered file fragments.

When a search engine encounters a document, a key issue is the determination of the text language. Absent Unicode, characters below 128 indicate the old ASCII English-language character set. A character of 128 or above, however, may represent any other language in the world. Without a clear indication in the document as to which language is in use, it takes a fair amount of detective work for a text retrieval program to figure out even the broad category of language(s)—Middle Eastern, Western European, Eastern European, etc.—that the text of the document is in.
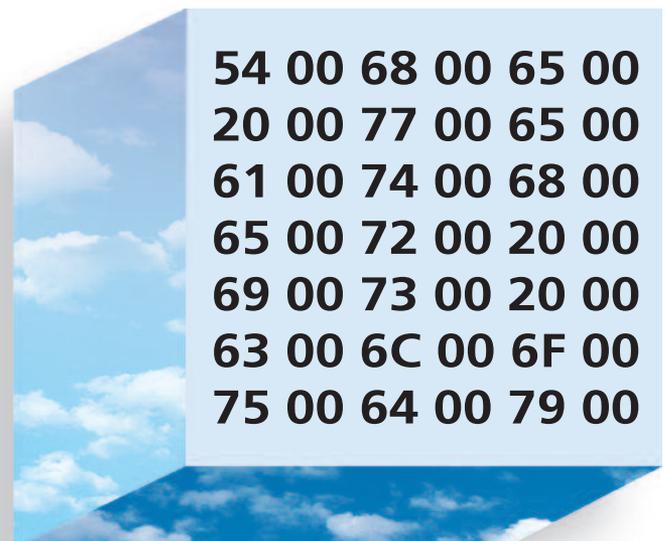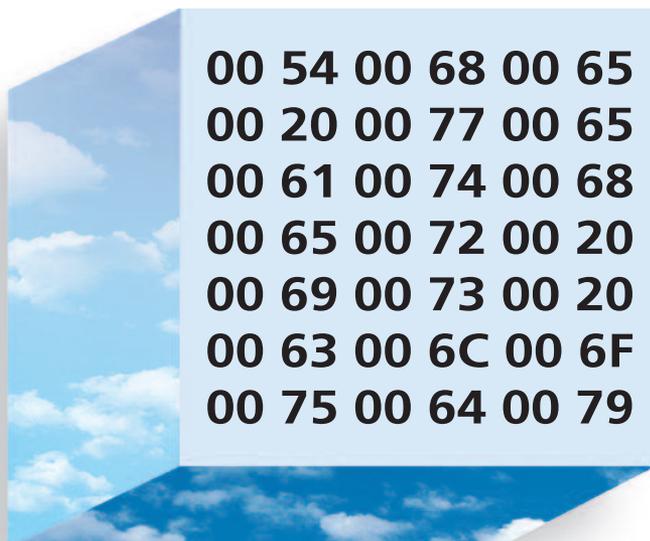
Unicode greatly reduces this ambiguity. Most Unicode documents embed information clearly identifying the Unicode encoding in use. When Unicode ambiguity remains at all, it typically relates only to byte order.

If the subrange designation byte comes first, the Unicode representation goes by the name "big endian." If the character position byte comes first, the Unicode representation goes by the name "little endian." In either case, to determine big endian or little endian is a simple either/or search engine decision, in stark contrast to the complex linguistic analysis that forensically-retrieved non-Unicode text can require.

In sum, while the spoken languages of the world may indeed remain confused, through Unicode and other text retrieval techniques, searching after the fall of the Tower of Babel has made some genuine progress.

Please visit dtSearch online at www.dtsearch.com

```
00 54 00 68 00 65
00 20 00 77 00 65
00 61 00 74 00 68
00 65 00 72 00 20
00 69 00 73 00 20
00 63 00 6C 00 6F
00 75 00 64 00 79
```

The weather is cloudy in big-endian Unicode.

```
54 00 68 00 65 00
20 00 77 00 65 00
61 00 74 00 68 00
65 00 72 00 20 00
69 00 73 00 20 00
63 00 6C 00 6F 00
75 00 64 00 79 00
```

The weather is cloudy in little-endian Unicode.