EMu Documentation

Unicode in EMu 5.0

Document Version 1

EMu 5.0

EMu
Museum
Management
System

KE Software
An AXIELL Group Company
emu.axiell.com

Contents

S ECTION 1

Unicode

Overview

EMu 5.0 sees implementation of support for the Unicode 8.0 (http://www.unicode.org/versions/Unicode8.0.0/) standard. While earlier versions of EMu allowed Unicode characters to be stored and retrieved, the system did not interpret the characters entered, leading to very limited searching functionality. In order to retrieve a Unicode character it was necessary to enter the search term in exactly the same case (upper or lower) along with the same diacritics. For example, a search for the name `Frederic` would not match `Fréderic` as the e acute character was not interpreted as an e character with a diacritic associated with it.

EMu 5.0 supports case folding and base character mapping:

- Case folding is similar to converting a character to its lower case equivalent except that it handles some special cases. The purpose of case folding is to make searching case insensitive. One special case is that the German lower case sharp s character (ß) is generally written in upper case as `SS`. So `Großen` would be converted to `GROSSEN` in upper case. When searching we would like to enter either of the previous terms and find all case variations. In order to do this the `ß` character needs to be folded to `ss` for searching purposes.

- The base version of a character is its most basic representation after all diacritics and marks have been removed. For example the base character of é is `e`.

The combination of case folding and base characters provides the basic mechanisms required to provide flexible searching over the full range of Unicode characters.

All data stored in EMu 5.0 is encoded in UTF-8 format. UTF-8 is a compact way of representing Unicode characters, particularly ASCII characters. The World Wide Web has adopted UTF-8 as the character encoding format to be used in web documents. EMu 5.0 enforces the use of UTF-8 by not allowing any invalid byte sequences to be stored in the system. The change has implications for data imports as all data imported **must** be encoded in UTF-8. In earlier versions of EMu, systems may have been configured to allow ISO-8859-1 (latin1) as the standard input format. ISO-8859-1 encoding is no longer supported.

Searching in EMu 5.0 has been extended to include punctuation characters. It is now possible to search for punctuation either as individual characters (`?`) or as part of a more complex string (`fred@global.com`). In EMu 4.3 and earlier certain punctuation characters have a special meaning when used in a search. For example a search for `fre*` will find all words beginning with the letters `fre`. The introduction of punctuation searching in EMu 5.0 means that these *special* characters need to be "escaped" to have their special meaning applied. Escaping a character involves

preceding the character with a backslash (\). Thus, an EMu 4.3 search for `fre*` becomes `fre\*` in EMu 5.0.

In the following sections we explore what changes have been implemented and how they impact usage of EMu 5.0.

Code Points

The basic unit of information in Unicode is known as a code point. A code point is simply a number between zero and $10FFFF_{16}$ that represents a single entity. Code points are generally represented as hexadecimal numbers, that is base 16. An entity may be a:

Entity	Description
graphic	A letter, mark, number, punctuation, symbol or space, e.g. the letter a.
format	Controls the formatting of text, e.g. soft hyphen (-) for breaking a word over lines.
control	A control character, e.g. the tab character (^I).
private-use	Not defined in the Unicode 8.0 standard but used by other non-Unicode scripts, e.g. unused cp 1252 character, 91_{16}.
surrogate	Used to select supplementary planes in UTF-16. Characters in the range $D800\text{-}DFFF_{16}$.
non-character	Permanently reserved for internal use. Characters in the range $FFFE\text{-}FFFF_{16}$ and $FDD0\text{-}FDEF_{16}$.
reserved	All unassigned code points, that is code points that are not one of the above.

The table below lists some code points along with their representation, label and category:

Code point (hex)	Represent- ation	Label	Category
E9	é	Latin small letter e with acute	graphic (letter - lower case)
600	؀	Arabic number sign	format (other)
D6A1	횡	Hangul syllable hoeng	graphic (letter - other)
B4	´	Acute accent	graphic (symbol - modifier)
F900	豈	Chinese, Japanese, Korean (cjk) compatibility ideograph	graphic (letter - other)

A piece of text is logically just a sequence of code points, where each code point represents a part of the text. For example, the piece of text:

豈 ↔ how?

consists of the following code points:

Code point (hex)	Represent-ation	Label
F900	豈	Chinese, Japanese, Korean (cjk) compatibility ideograph
20		Space
2194	↔	Left right arrow
20		Space
68	h	Latin small letter h
6F	o	Latin small letter o
77	w	Latin small letter w
3F	?	Question mark

The code point sequence defines the text itself. There are a number of different ways that the code point sequence can be saved on a computer. One method, called UTF-32, represents each code point as a 32 bit (4 byte) quantity. Such a scheme uses a large amount of storage space as most text uses the Latin alphabet (ASCII), which can be represented in a single byte.

Another encoding is UTF-8. This allows ASCII characters to be stored as a single byte (code points 00-7F), with multiple bytes used for higher code points. UTF-8 is very efficient space wise where the text consists of mainly ASCII characters, and the World Wide Web has adopted it as the preferred encoding method for Unicode code points. EMu 5.0 also uses UTF-8 as the encoding method. Below, we show a string encoded in UTF-32 with a space between each code point:

豈 ↔ how?

```
0000F900  00000020  00002194  00000020  00000068  0000006F  00000077
0000003F
```

And the same string encoded in UTF-8:

```
EFA480 20 E28694 20 68 6F 77 3F
```

As you can see the UTF-8 encoding saves considerable space.

Prior to EMu 5.0 either UTF-8 or ISO-8859-1 could be configured as the encoding

used by EMu. EMu 5.0 drops support for ISO-8859-1 and only supports UTF-8 encoded characters. The change means that moving to EMu 5.0 requires all data to be converted from ISO-8859-1 to UTF-8 before the system may be used. The upgrade process performs this important function.

> EMu 5.0 will not allow non UTF-8 sequences to be input. If an illegal character is encountered, an error message is displayed. The enforcement of UTF-8 encoding means that all data entered into EMu, either by direct entry or by importing, must be in UTF-8 format. Data encoded in ISO-8859-1 cannot be loaded. If you receive import data from a third party source, ensure that it is in UTF-8 format otherwise errors will be generated for all non-ASCII characters. An ISO-8859-1 encoded data file can be converted to UTF-8 using the UNIX `iconv` utility.

Inputting Unicode Characters

Now that we understand that text is made up of a sequence of Unicode code points it is worth considering how these characters can be entered into EMu.

EMu supports two mechanisms:

- Escaped code point
- Raw characters

Escaped code point

The escaped code point mechanism allows an escape sequence to be placed in a text string to represent a Unicode code point. When the string is sent to the EMu server, the escape sequence is converted into a Unicode code point encoded in UTF-8.

For example, if the text `Fr\u{E9}deric` was input while creating or modifying a record, the data saved would be `Fréderic`. The format of the escape sequence is `\u{x}` where *x* is the code point in hexadecimal of the Unicode character required. The escape sequence may also be used when entering search terms:

The escape sequence may also be used in texql statements whenever a string constant is required. For example, the query statement:

```
select NamFirst
from eparties
where NamFirst contains 'Fr\u{E9}deric'
```

will find all Parties records where the First Name is `Fréderic` (and variations where diacritics are ignored). The escape sequence format may also be used for data imported into EMu via the Import facility.

Raw characters

The raw character method involves pasting Unicode characters into the required EMu field. There are a number of ways of adding Unicode characters to the Windows clipboard. One way is to use the Windows Character Map application. This can be found on a Window PC by selecting search on the Windows Start menu (or pressing the Windows Logo key and the letter `s` at the same time) and searching for `charmap`.

The Windows Character Map application allows you to select a character and copy it to the clipboard. By selecting **Advance view**, it is possible to search for a character by name. For example to find the `oe` ligature character (œ), enter `oe ligature` in the *Search for:* field and press **Search**. A grid of all matching characters is displayed:
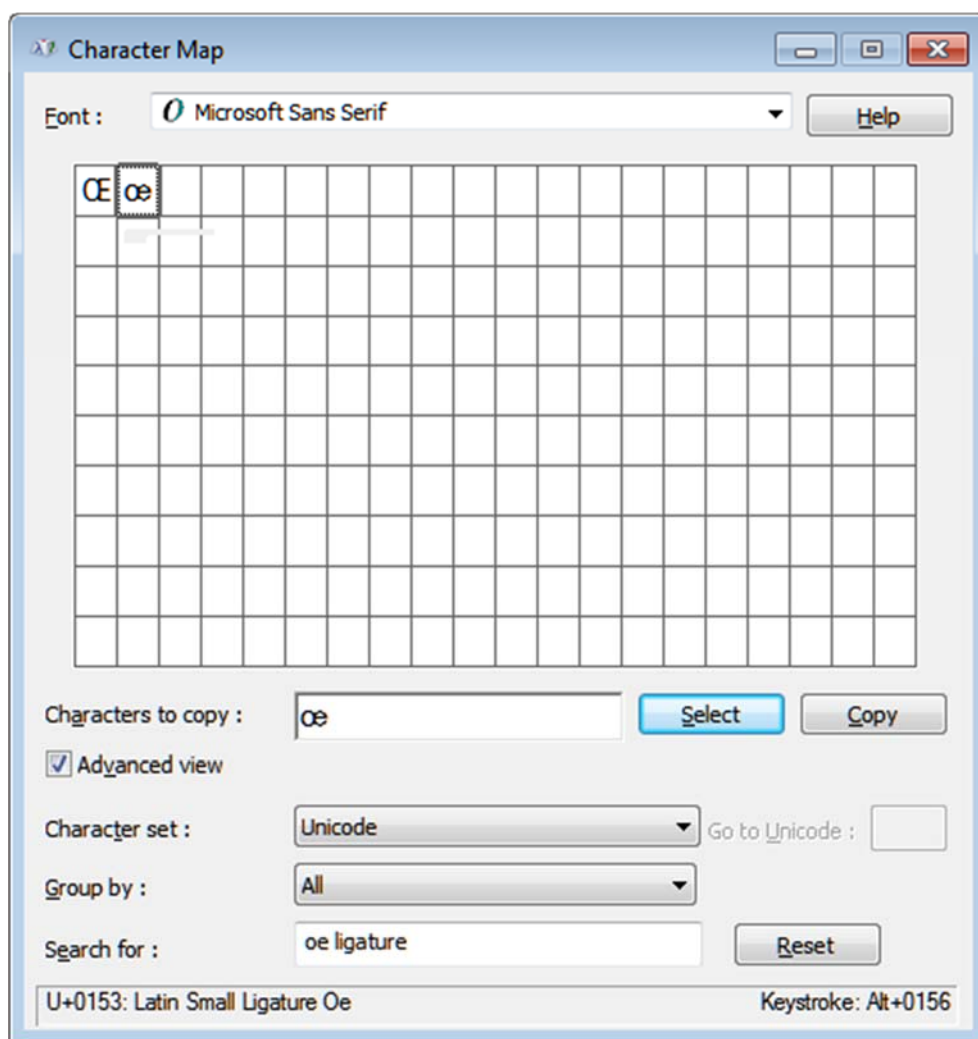

Double-click the required character, then press **Copy** to place it on the Windows clipboard. The character can then be pasted into EMu.

Alternative methods

Another way to add a Unicode character to the Windows clipboard is to use a website that allows Unicode characters to be searched for and displayed. Two useful sites are:

- graphemica.com (http://graphemica.com/)
- unicode-table.com (http://unicode-table.com/)

With both of these sites it is possible to search for a character by name or code point (in hex), e.g.:
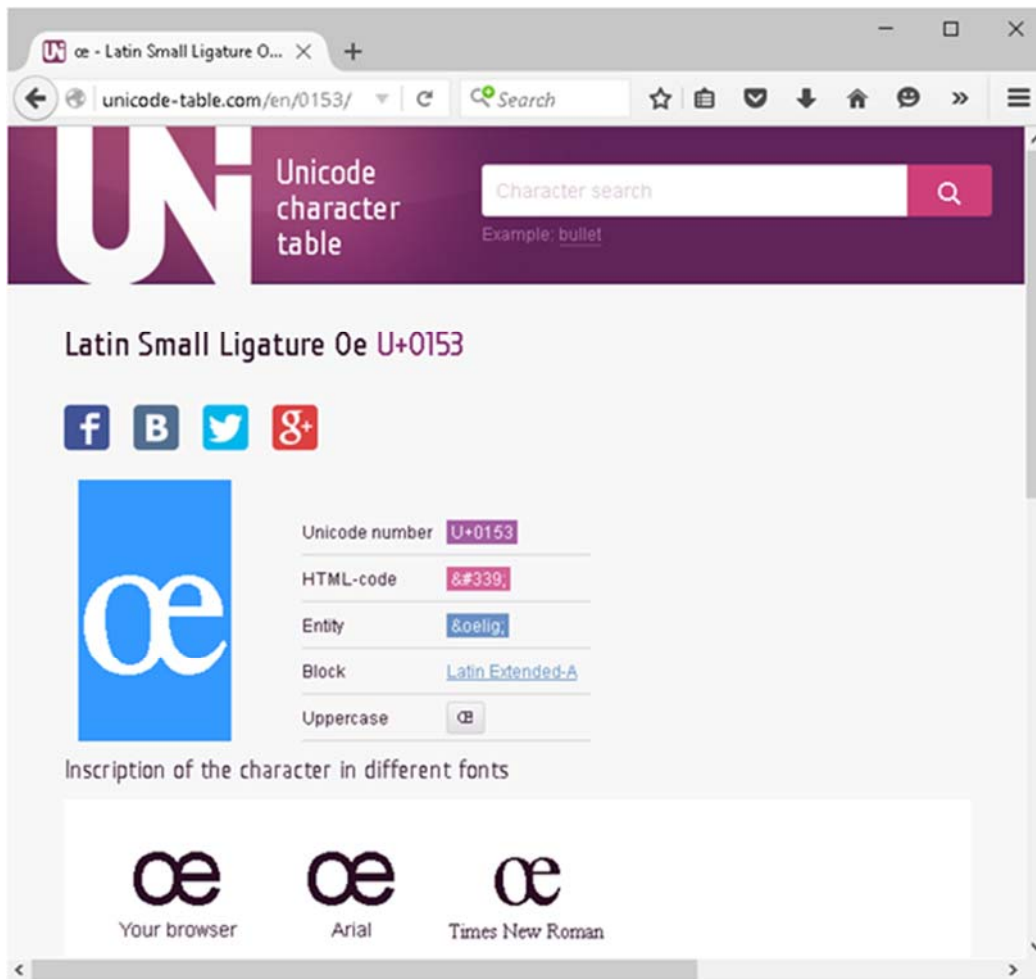
Highlight the character on the page and copy it to the clipboard. The character can then be pasted into the required EMu field.

Both of these websites display the code point for the character. In the picture above, the code point for œ is hex 153. If you wanted to use the escaped code point method, the escape sequence to use would be:

```
\u{153}
```

If you need to enter certain Unicode characters on a regular basis, you could create a WordPad (or Word) document that contains the characters. When you need a character, simply copy the character from the document and paste it into EMu, without the need to search for the character.

Graphemes

It is important to understand that what we think of as a character, that is a basic unit of writing, may not be represented by a single Unicode code point. Instead, that basic unit may be made up of multiple Unicode code points.

For example, "g" + *acute accent* (ǵ) is a *user-perceived character* as we think of it as a single character, however it is represented by two Unicode code points (67 301). A *user-perceived character*, which consists of one or more code points, is known as a grapheme. The use of graphemes is important for:

- collation (sorting);
- regular expressions;
- indexing; and
- counting character positions within text.

EMu 5.0 uses graphemes as the basic building block for text. Thus a text string is handled as a sequence of graphemes.

A grapheme consists of one or more base code points followed by zero or more zero width code points and zero or more non-spacing mark code points. In the case of "g" + *acute accent* (ǵ), the letter g is the base code point (67) and the *acute accent* is a non-spacing mark code point (301). The table below shows some multiple code point graphemes:

Grapheme	Code points
각	1100 (ㄱ) Hangul choseong kiyeok (base code point)
	1161 (ㅏ) Hangul jungseong a (base code point)
	11A8 (ㄱ) Hangul jongseong kiyeok (base code point)
ḍ́	64 (d) Latin small letter d (base code point)
	325 (̥) combining ring below (non-spacing mark)
	301 (´) combining acute accent (non-spacing mark)
á	61 (a) Latin small letter a (base code point)
	301 (´) combining acute accent (non-spacing mark)

Some common multiple code point graphemes have been combined into a single code point. For example, the last entry in the table above, á, can also be represented by the single code point E1. Hence we have two representations, or two graphemes, that represent the same character (á is this case).

Index Terms

An index term is the basic unit for searching. It is a sequence of one or more graphemes that can be found in a search but for which searching of sub-parts is not supported (except if regular expressions are used). EMu provides word based searching, so an index term corresponds to a word. You can search for a word, and records that contain that word will be matched. In languages that define a word as a sequence of letters separated by either spaces or punctuation, an index term corresponds to a word. In languages in which single (or sometimes multiple) letters make up a word, such as kanji, an index term corresponds to each individual letter. EMu 5.0 adds support for searching for punctuation, so each punctuation character is considered to be an index term.

Consider the following text:

```
香港 is Chinese for "Hong Kong" (香 = fragrant, 港 = harbour).
```

The index terms for the above text are:

| Index Term |
| --- |
| 香 |
| 港 |
| is |
| Chinese |
| for |
| " |
| Hong |
| Kong |
| " |
| (|
| 香 |
| = |
| fragrant |
| , |
| 港 |
| = |
| harbour |
|) |
| . |

Each of the above terms can be used in a search and the query will be able to use the

high speed indexes to locate the matching records. It is possible to use regular expression characters (e.g. `fra\*` to find all words beginning with `fra`) to search for sub-parts of words, however the high speed indexes will not be used in this case (unless partial indexing is enabled).

Each index term is folded and converted to its base form. The folding process, as described in the overview section (page 1), removes case significance from the term. The conversion to its base form involves removing all "mark" code points from the term and then converting the remaining code points to their compatible form (as defined by the Unicode 8.0 standard). The compatible form for a code point is a mapping from the current code point to a base character that has the same meaning. For example the code point for subscript 5 ($_5$) has a compatible code point of 5.

The table below shows some more examples for compatibility:

Type	Compatibility Examples		
Font variants	ℋ	→	H
	Ⅱ	→	H
Positional variants	ﻉ	→	ع
	ﻊ	→	ع
	ﻋ	→	ع
	ﻌ	→	ع
Circled variants	①	→	1
Width variants	ｶ	→	カ
Rotated variants	︷	→	{
	︸	→	}
Superscripts / subscripts	i⁹	→	i9
	i₉	→	i9

Unfortunately, some of the compatibility mappings in the Unicode 8.0 standard are narrower than we might expect when searching text. For example the oe ligature (œ) does not map to the characters "oe". So the French word cœur ("heart") does not have an index term of `coeur`, but remains as `cœur`. When searching you need to enter `cœur` as the search term otherwise cœur will not be found.

In order to *correct* some of the compatibility mappings, EMu 5.0 provides a mapping file where a code point can be mapped to its compatible code point(s), hence "œ" can be mapped to "oe". The mapping file is located in the Texpress installation directory in the `etc/unicode/base.map` file.

A sample file is (as distributed currently):

```
#
# The following file is used to extend the Unicode NFKD mappings
for
# characters not specified in the standard. The format of the file
is
# a sequence of numbers as hex. Each number represents a single
code
# point in UTF-32 format. The first code point is the code point
to map
# and the second and subsequent code points are what it maps to.
#
00C6 0041 0045         # Latin capital letter AE -> A E
00E6 0061 0065         # Latin small letter ae -> a e
00D0 0044              # Latin capital letter Eth -> D
00F0 0064              # Latin small letter eth -> d
00D8 004F              # Latin capital letter O with stroke -> O
00F8 006F              # Latin small letter o with stroke -> o
00DE 0054 0068         # Latin capital letter Thorn -> Th
00FE 0074 0068         # Latin small letter thorn -> th
0110 0044              # Latin capital letter D with stroke -> D
0111 0064              # Latin small letter d with stroke -> d
0126 0048              # Latin capital letter H with stroke -> H
0127 0068              # Latin small letter h with stroke -> h
0131 0069              # Latin small letter dotless i -> i
0138 006B              # Latin small letter kra -> k
0141 004C              # Latin capital letter L with stroke -> L
0142 006C              # Latin small letter l with stroke -> l
014A 004E              # Latin capital letter Eng -> N
014B 006E              # Latin small letter eng -> n
0152 004F 0045         # Latin capital ligature OE -> O E
0153 006F 0065         # Latin small ligature oe -> o e
0166 0054              # Latin capital letter T with stroke -> T
0167 0074              # Latin small letter t with stroke -> t
```

Compatible mappings may be added to the file as required.

> If the file is modified, a complete reindex of the system is required in order for the new mappings to be used to calculate the index terms.

If you consider the French phrase:

`Sacré-Cœur est situé à Paris.`

the index terms after folding and conversion to base form are:

Index Term
sacre
coeur
est
situe
a
paris
.

When a record is saved in EMu all index terms are folded and converted to their base form before indexing occurs. Similarly, when a search is performed, the query terms are folded and converted to their base form before the search commences. Hence a search for "coeur" or "Cœur" or even "COEUR" will still match the text in the French phrase above.

S E C T I O N 2

Searching

Now that we understand what an index term is we can talk about searching. The incorporation of Unicode into EMu has resulted in the searching mechanism being extended to handle all code points that have a base representation. In essence this is all *graphic* (page 3) code points except for marks and spaces, namely:

- letters
- numbers
- punctuation
- symbols

The inclusion of punctuation as an index term means that punctuation may now be included in searches and the high speed indexes will be used to locate matches.

An issue arises in EMu versions prior to 5.0 where certain punctuation characters were used to adjust the type of searching performed. For example, in EMu 4.3 a search for `@John` would find all records containing words that sound like John (phonetic searching). Similarly a search for `^joh*` would match records where the first word starts with the letters `joh` (case ignored). A search for `=John` would locate records containing John with case significance (that is an upper case `J` and lower case `ohn`). Since EMu 4.3 and earlier removed punctuation and symbols from searching (only letters and numbers were supported) there was no ambiguity about the punctuation associated with search terms (as in the previous examples). As EMu 5.0 allows symbols and punctuation to be searched for, some ambiguity can creep in. For example, what does searching for `fred@global.com` mean? In EMu 4.3 it would have meant finding:

- "fred"
- AND the phonetic of "global"
- AND "com"

However, in EMu 5.0 is the `@` character to be treated as punctuation or does it mean the phonetic of the word "global"?

When searching for a word prior to EMu 5.0 you simply entered the word and performed the search. We have taken the same approach in EMu 5.0 with punctuation characters. In other words, when you have punctuation in a search, only records containing the punctuation are matched. Thus, in the previous example the `@` character is treated as punctuation and so must appear in matching records.

How then do we indicate that the `@` character means we want the phonetic version of the following word? We proceed the character with a special marker indicating the character is to take on its phonetic meaning. The marker character used is the backslash (\) character. The introduction of a marker character to alter the meaning of a character is not new in EMu. For example, `\n` can be used in strings to represent

the newline character; similarly `\u{}` is used to introduce the escape sequence for a Unicode code point.

EMu 5.0 has a simple rule to determine how to format a search:

> All *graphic* (page 3) characters, expect for spaces and marks, in a search are matched as the character. Where the special meaning of a character (e.g. @) is required, the character must be preceded by the backslash (\) escape character. The only exception to this rule is that the backslash character itself must be entered twice (\\) where the actual character is required.

The table below compares some searches in EMu 4.3 and their equivalent in EMu 5.0:

Find	EMu 4.3	EMu 5.0
Records containing `Fred`	`fred`	`fred`
Records where `Fred` is the only word in the field	`^fred$`	`\^fred\$`
Records that contain `Fred` phonetically	`@fred`	`\@fred`
Records containing `Fred` with matching case	`=Fred`	`\=Fred`
Records containing the phrase `Sacré-Cœur`	`"sacré cœur"`	`\"sacre-coeur\"`
Records where `blue` and `sky` are within five index terms of each other	`(blue sky) <= 5 words`	`\(blue sky\) <= 5 words`

In the following sections we will look at all available special search operators and show examples of their use in EMu 5.0. Each of the operators is displayed with its leading escape character, the backslash character.

Transformations

Transformations are an operator that is applied to a search term to alter its interpretation. The table below lists all valid transformations:

Transformation	Description
\~	Search for all variations of a word. For example, searching for `\~elect` will match `elect`, `election`, `electing` and `elected`, but not `electricity` (its base word is `electric`)
\&	Ignore the case (upper or lower) of the search term. This is the default transformation if one is not specified explicitly.
\@	Use phonetic or sounds like searching for the specified word.
\=	Perform the search using case significance for the following word.
\==	Perform the search not only matching the case but also matching any marks (diacritics).

A transformation is always applied to a word and is placed immediately before the word to which it applies. Some examples are:

Find	Search
Records containing all tenses of the word `locate`.	`\~locate`
Records where `melbourne` is all in lower case.	`\=melbourne`
Records with `Sacré` and `Cœur` exactly as specified, that is matching case and diacritics, but not necessarily next to each other.	`\==Sacré` `\==Cœur`
Records containing words similar to `smythe` phonetically.	`\@smythe`

Regular Expressions

Regular expressions provide a mechanism for searching for patterns in a word. With regular expressions, sub-parts of a word may be matched. In general the high speed indexes cannot be used with regular expression searches. The only exception is trailing regular expressions (that is a regular expression that has leading letters), where partial indexing has been enabled.

Regular expressions can be intermixed with the `\=` and `\==` transformations to enforce case and diacritic significance.

The table below lists all valid regular expressions:

Regular Expression	Description
`\?`	Matches any single grapheme.
`\*`	Matches zero or more graphemes.
`\[`*range*`\]`	Matches only one of a sequence of graphemes specified in *range.* *range* may consist of individual graphemes or a beginning and end grapheme may be specified separated by a minus sign (e.g. `a-z`).
`\{`*range*`\}`	Matches one or more of a sequence of graphemes specified in *range*. *range* may consist of individual graphemes or a beginning and end grapheme may be specified separated by a minus sign (e.g. `0-9`).

Some examples are:

Find	Search
Records containing words starting with `abs`.	`abs\*`
Records containing Arabic numbers.	`\{٠-٩\}`
Records with a three grapheme word.	`\?\?\?`
Records with `organisation` spelt with either an `s` or `z`.	`organi\[sz\]ation`
Records with at least one word containing a capital `S`.	`\=\*S\*`
Records containing either an upper case or lower case `é`.	`\==\*\[éÉ\]\*`

Anchors

Anchors are used to indicate that a search term should be located as either the first or last word in a piece of text. Anchors can be used in combination with all other types of search operators, namely transformations, regular expressions, phrases and proximity.

The table below lists all valid anchors:

Anchors	Description
`\^`	The search term following must be the first word in the text.
`\$`	The search term following must be the last word in the text.

Some examples are:

Find	Search
Records that have text ending in a question mark.	`?\$`
Records with text beginning with the word `the`.	`\^the`
Records where the text contains only the word `Unknown`.	`\^Unknown\$`
Records with text where the first word starts with a lower case Latin letter.	`\^\==\[a-z\]\*`

Proximity

Proximity searching provides a mechanism for finding a list of words within a specified distance (either words, sentences or paragraphs). EMu supports two types of proximity searches:

- The first is phrase searches where the words must appear next to each other and in the order they are specified. The words in a phrase search may have transformations, regular expressions and anchors applied.
- The second is a regular proximity search. Proximity searches may include transformations, regulars expressions, anchors and phrases.

The table below lists all valid proximity operators:

Proximity	Description
\"*search terms*\"	The *search terms* enclosed within the phrase operator (\") must appear next to each other and in the order they are specified.
\(*search terms*\) *distance*	The *search terms* may appear in any order unless otherwise specified. The *distance* between the terms indicates the range within which the search terms must appear. The syntax for *distance* is: [ordered] *relop number type* where: • *relop* is one of the relational operators <, <=, =, >, >= • *number* is the distance to use • *type* is one of words, sentences or paragraphs The keyword ordered is optional, but if given, requires the search terms to be in the order specified.

Some examples are:

Find	Search
Records where the phrase the black cat occurs.	\"the black cat\"
Records containing only the phrase Not Applicable.	\"\^Not Applicable\$\"
Records where Fred occurs case significantly in the same sentence as the phonetic of Smith where Fred appears first.	\(\=Fred \@Smith\) ordered = 1 sentence

Find	Search
Records where the kanji character 豈 appear within 5 characters of the phrase 香 港.	`\(豈 \"香 港\"\) <= 5 words`

Conditionals

EMu provides support for one conditional operator, NOT. The NOT operator reverses the sense of the next search term. The NOT operator can be applied to any of the other search operators, that is transformations, regular expressions, anchors and proximity.

The table below lists the valid conditional operator:

Conditionals	Description
\!	The sense of the next search term is reserved.

Some examples are:

Find	Search
Records that do not contain the kanji 豈.	\!豈
Records that contain anything apart from the single word Unknown.	\!\^Unknown\$
Records that do not contain the phrase Not Applicable.	\!\"Not Applicable\"
Records containing the phrase Sacré Cœur with case and diacritic significance but not Paris.	\"\==Sacré \==Cœur\" \!Paris

S ECTION 3

Auto-phrasing

Unicode graphemes are broken down into one of three categories for use in EMu 5.0. The categories are:

Category	Description
combining	A grapheme that is a simple letter or number. It is not a word in its own right but requires other characters to form words. Examples are the Latin, Arabic and Hebrew letters and numbers.
single	A single grapheme is used to represent a base word or meaning. Examples are Kanji and punctuation characters.
break	A character that delineates words, typically a space character.

Consider the following text:

香港 = "Hong Kong".

The graphemes along with categories are:

Grapheme	Category
香	*single*
港	*single*
	break
=	*single*
	break
"	*single*
H	*combining*
o	*combining*
n	*combining*
g	*combining*
	break
K	*combining*
o	*combining*
n	*combining*
g	*combining*
"	*single*
.	*single*

EMu uses the category to determine what is an index term. Each *single* grapheme is treated as a separate index item, while *combining* graphemes are joined together to form a "word" up to a *break* or *single* category grapheme. A *break* grapheme is not an index term and is discarded.

In general, a phrase-based search must be performed where you want to find records where a list of index terms occur sequentially. For example, to find the two kanji characters 香港 (Hong Kong) next to each other, the query \"香 港\" may be used. Where a grapheme is part of the *single* category (like the two kanji characters), the system knows what the index term is and is able to treat them as a phrase provided a *break* character is not found. In fact EMu 5.0 treats a combination of *combining* and *single* graphemes as a phrase without the need for the phrase operator until a *break* grapheme is encountered. This process is known as auto-phrasing.

Auto-phrasing means that a query of 香港 is equivalent to \"香 港\" without the need to add the quotes or space. Another example is an email address such as `fred@global.com`. In this case the index terms `fred`, `@`, `global`, `.`, `com` must be located sequentially. Auto-phrasing effectively allows you to enter non-space separated terms and EMu will retrieve records where the terms are adjacent. If you do not want the terms to appear next to each other, for example if you want to find 香 (fragrant) 港 (harbour), then simply placing a space between the two kanji characters will disable auto-phrasing.

SECTION 4

Collation

Collation is the general term for the process of determining the sorting order of strings of characters. EMu 5.0 uses the Default Unicode Collation Element Table (DUCET), as defined in the Unicode 8.0 standard, to determine how text should be sorted. DUCET provides a locale independent mechanism for ordering values.

If you are interested in the ordering used by DUCET, please consult the Unicode Collation Charts (http://unicode.org/charts/collation/).

S ECTION 5

Lookup Lists

The addition of support for searching on punctuation in EMu 5.0 has flowed through to other parts of the system. The most notable change is that punctuation is now significant in Lookup List values.

When comparing Lookup List entries prior to EMu 5.0, punctuation was removed before the entries were processed. Hence a Lookup List entry of `Smith (?)` was treated the same as an entry for `Smith`, so only one value (the first one entered in the system) would be stored. The problem is that these two entries are very different in meaning. The first implies a level of uncertainty with the name which is not present in the second.

EMu 5.0 retains punctuation when comparing Lookup List values, meaning that the two entries in our example are treated as separate and we end up with two entries in the Lookup List itself.

Index