SQL Server Full Text Search Language Features

21 April 2006
by Hilary Cotter

SQL Server Full Text Search: Language Features

SQL Full-text Search (SQL FTS) is an optional component of SQL Server 7 and later, which allows fast and efficient querying when you have large amounts of unstructured data. This is the first of a two-part article that explores the full-text language features that ship with SQL Server versions 7, 2000, and 2005, with particular focus on the new language features in SQL 2005.

Here, in part 1, we examine the **index time** language options: how words or tokens are broken out and stored in a full text index. In part 2, we will switch focus to the query time language options.

SQL FTS architecture

SQL FTS builds a **full-text index** through a process called **population**, which fills the index with words and the locations in which they occur in your tables and rows. The full text indexes are stored in **catalogs**. You can define multiple catalogs per database, but the catalog cannot span databases; it is database specific. Similarly a table can be full-text indexed in a single catalog; or to put it another way - a table's full-text index cannot span catalogs. However, in SQL 2005 you can full-text index views which may reside in different catalogs than the underlying base tables. This provides performance benefits and allows partitioning of tables.

There is a useful depiction of the SQL FTS architecture in BOL. I won't repeat it here, but I do encourage you to familiarize yourself with it, and how the various FTS components interact. Very briefly, during the population process the indexing engine (MSSearch in SQL 7 & 2000 and MSFTESQL in SQL 2005) connects to SQL Server using an OLE-DB provider (PKM-Publishing and Knowledge Management) and extracts the textual content from your table on a row by row basis. MSSearch uses the services of COM components called **iFilters** to extract a text stream from the columns you are indexing.

iFilters

The iFilter used depends on the data type of the column in which the data is stored and on how you have configured your full-text index:

- For columns of the `CHAR`, `NCHAR`, `VARCHAR`, `NVARCHAR`, `TEXT`, and `NTEXT` data types the indexing engine applies the **text** iFilter. You can't override this iFilter.
- For the columns of the XML data type the indexing engine applies the **XML** iFilter. You can't override the use of this iFilter.
- For columns of the `IMAGE`, and `VARBINARY` data type, the indexing engine applies the iFilter that corresponds to the document extension this document would have if stored in the file system (i.e. for a Word document, this extension would be doc, for an Excel Spreadsheet this would be xls). Please refer to the later section on *Indexing BLOBs* for more information on this.

NOTE: It is important to understand that an iFilter can launch a different language word breaker from the one specified as the default in the full-text language setting for your Server, or from the

word breaker you set for the column in the table you are full-text indexing. I cover this in more detail in the Word Breaker section.

A complete list of built-in iFilters is accessible from: http://www.indexserverfaq.com/sql2005iFilters.htm. You can also obtain a list of the signed iFilters that are currently accessible on your SQL Server 2005 by issuing the following query in any database:

```
select document_type, path from sys.fulltext_document_types
```

The iFilters are listed in the **path** and the document types that are indexed by the iFilters are listed in the **document_type** column. Note that although there are iFilters for asp and aspx pages, these pages will not be generated by the asp or aspx rendering engines, rather the HTML MetaTags and textual content between the body tags will be indexed and searchable.

*NOTE: To see the text (and properties) that the iFilters emit from the documents, you can run them through FiltDump a utility available from the Platform SDK in the **binn** directory. It will show not only the text and properties the iFilters extract from the documents, but also the language tags applied to the text streams.*

A note on iFilters and BLOBs

For SQL 2000, the correct iFilter for the Binary Large Object (BLOB) column you wish to index must be installed on your OS (use the filtreg utility, available from the Platform SDK in the binn folder, to tell which iFilters will be applied for a specific document type).

For SQL 2005, if you wish to index a document type for which a pre-installed iFilter does not exist, you must install the appropriate iFilter on the OS, and then load it using the following commands:

```
exec sp_fulltext_service 'load_os_resources', 1;
exec sp_fulltext_service 'verify_signature', 0;
go
```

So, for example, if you wish to store and index PDFs in columns of the `IMAGE` or `VARBINARY(MAX)` data type you must install the Adobe PDF iFilter and issue the two commands above. Otherwise you will get errors of the following form in the full-text gatherer log (found at C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Log):

```
2006-01-10 13:42:59.29 spid28s
Warning: Failed to get MSFTESQL indexer interface for full-text catalog
'ix_STS_ctcsppweb01_1' ('5') in database 'MyDatabase' ('11').
Error: 0x80000018.
```

SQL FTS Overview

In essence, in order to perform full text searching on a table you need to:

1. Ensure that the table has a unique, not null column (e.g. primary key)
2. Create a full text catalog in which to store full text indexes for a given table
3. Create a full text index on the text column of interest

For a brief tutorial on how to create a full text index on SQL 2005 please click on the **CODE DOWNLOAD** link in the box to the right of the article title.

You can build full-text indexes on textual data stored in `CHAR`, `NCHAR`, `VARCHAR`, `NVARCHAR`, `TEXT`, `NTEXT` columns as we as on `IMAGE` (SQL 200x), `VARBINARY(MAX)` (SQL 2005), and `XML` (SQL 2005) data type columns that contain formatted binary data (such as this word document, for example).

Indexing Text

The sophisticated language features of full-text search then allow you to perform a range of advanced searches on your textual data, using the `CONTAINS` and `FREETEXT` T-SQL predicates (as well as the `CONTAINSTABLE`, and `FREETEXTTABLE` functions), returning a list of rows that contain a word or a phrase in the columns that you are searching on. You can perform:

- Simple searches for specific words or phrases
- Thesaurus searches for synonymous forms of word – a search on *IE* might return hits to *Internet Explorer* and *IE* (thesaurus based expansion search); a search on *Bombay* might also return hits to *Mumbai* (thesaurus based replacement search)
- Searches that will return all different linguistic forms of a word (called generations) – a search on bank would return hits to *banking*, *banked*, *banks*, *banks'* and *bank's*, etc. (all declensions and/or conjugations of the search term bank)
- Accent insensitive searches – a search on *café* would return hits to *cafe* and *café*

NOTE: there are other forms of searches too, which we don't cover in this article – for example weighted searches and proximity searches. See MSDN2 for further details.

In short, SQL FTS provides a very powerful text searching mechanism. It is many orders of magnitude faster than a `LIKE` operator; especially for larger tables. This is because when you use the `LIKE` operator it does a byte-by-byte search through the contents of a row looking for a match to the search phrase. SQL FTS references the full text index to instantly return a list of matches. SQL FTS also supports a large number of different languages and language characters. As well as accents, it also seamlessly handles: compound words (in German and Chinese) and compound characters which occur in Chinese.

You may have noticed in some of the earlier CTPs (Community Technology Previews) for SQL Server 2005, 23 languages were supported, in the RTM version of SQL Server 2005 only 17 languages are supported. Please refer to http://support.microsoft.com/kb/908441/en-us for more information on how to enable the missing 6 languages.

Indexing BLOBs

With SQL 2000 and later, you also have the ability to full-text index BLOBs (Binary Large Objects or binary files) stored in the `IMAGE` (SQL 200x) and `VARBINARY` data type column (SQL 2005). SQL 7 did not have the capability to index BLOBs.

For BLOBs to be successfully full-text indexed you must ensure that the document extension is stored in an additional column (referred to as a **document type** column), and that you have configured your full-text index to use this document type column (i.e. the content of the document type column would be **doc** for rows containing Word documents, **xls** for rows

containing Excel Spreadsheets, **xml** for rows containing XML documents, and so on).

Use `VARCHAR(3)` or `VARCHAR(4)` as the length of the document type column because, in some of the earlier versions of SQL FTS, if your document extension was less than the length of your document extension column the `CHAR` data type would pad the row contents with spaces, and consequently MSSearch would not launch the iFilter correctly. For example if your content was a Word document with the doc file extension, stored in a `CHAR(4)` document type column, MSSearch would attempt to launch the iFilter corresponding to "doc " instead of "doc" and fail. The `VARCHAR` data type does not pad the data with white space, and so this problem is circumvented.

You configure your full-text table to reference this document type column by using the `sp_fulltext_column` stored procedure (in SQL 2000) or by the `CREATE FULL TEXT INDEX` statement in SQL 2005.

Script 1 in the download code for this article provides a worked example of indexing a BLOB.

Further information on how to index BLOBs in SQL 2000 and SQL 2005 can be found at http://www.indexserverfaq.com/Blobs.htm and at http://www.indexserverfaq.com/BlobsSQL2005.htm.

SQL FTS Language Features

The language features of SQL FTS come into play both at "Indexing Time" and "Query Time". At index time we are essentially building an index of words or **tokens** on the textual data. A token is a character string, delineated from other character strings by white space or punctuation, which may or may not have linguistic significance but is not normally construed as a word, e.g. MSFT, XXX, PartNo 1231, 159-23-4364, or even http://www.microsoft.com.

During the indexing process, linguistic analysis is performed. Word boundaries are located by application to the text data of language specific components called word breakers, thus identifying our tokens. Basically, the word breakers will control how the textual content is broken into words and how these words are stored in the full-text index. Word breakers may tag the word as currency, a date, and for some languages (especially German, and Far East languages) will sometimes analyze the word and extract constituent words or characters.

At query time, the user supplies a query, such as the following:

```
SELECT * FROM TableName WHERE CONTAINS(ColumnName,'bank')
```

When the query is submitted, a language specific component called a stemmer is used to expand the search phrase, depending on language specific rules that are in place. These rules are controlled by SQL Server settings, or your SQL Full-text query arguments. So, in the above example, the search term, *bank*, would be expanded to incorporate variants such as *banking*, *banked*, *banks*, *banks'* and *bank's*, etc (all conjugations of the search term bank). The index will then be searched for these words and the matching rows will be returned.

Word Breakers

The iFilters return a text stream to the indexing engine which then applies language specific

word breakers to the text stream to break the stream into words or tokens typically at white space or punctuation boundaries.

Specifying the Word Breaker

Different language word breakers are loaded depending on how you configure SQL Server, or how you configure full-text indexing or, sometimes, even on the content being indexed.

Via Full-Text Index Settings

The language word breaker will be used that corresponds to the **LCID** (LoCale IDentifier) specified in the `sp_addfulltext_column` (for SQL 7 and 2000) or the language specified in the `CREATE FULL TEXT INDEX` statement in SQL 2005, as illustrated below:

```
CREATE FULLTEXT INDEX ON FullTextTableImage
(imageCol TYPE COLUMN DocumentTypeColumn LANGUAGE 'FRENCH')
```

See Script 2 in the download code for a full worked example

For SQL 2005, you can get a list of supported full-text languages in SQL 2005 by issuing the following query:

```
select * from sys.fulltext_languages
```

In SQL 2000 you would issue this query to `xp_msfulltext` which will return the language name and its LCID. If no language word breakers exist for an LCID, the neutral word breaker is used (LCID=0).

Via SQL Server Settings

If no language is specified in the `sp_addfulltext_column` or `CREATE FULLTEXT INDEX` statement then the default full-text language setting for your SQL Server is used. You can check/modify the default full-text language setting using:

```
sp_configure 'default full-text language'
```

You may have to set advanced options to "on" to see or use this option.

Via language tags in the content being indexed

Some documents have language tags and, if the iFilter understands these language tags, it might (depending on the iFilter implementation) launch a different word breaker than the one specified by any the Full-Text index or SQL Server settings.

The language-aware document formats that I know of are Word, XML, and HTML. illustrates these features. These language tags will override all of the above settings. For Word, the entire document, or a portion of the document, can be tagged as having a different language than the default for the machine on which they were created, and there can be multiple different language tags in the same document (i.e. French, German, and Chinese). You set the Word language tags in two ways:

1. At the document level: open up your document in Word and click Format, in the Show: drop down box, select Custom, and click Styles, Custom, and select Modify. Click Format and select Language.
2. At the text level: highlight a portion of text, and click Tools, click Language and Set Language and select the language with which you wish this portion of text to be tagged.

For HTML documents, you can use the **ms.locale** meta tag. For example, for Greek the tag would be:

```
<meta name="MS.LOCALE" content="el-gr" >
```

For XML documents, you can use the **xml:lang** tag. For example, for Ukrainian use:

```
<xml:lang="uk">
```

Note that the use of the Greek and Ukrainian languages is for illustrative purposes only. These languages are not currently supported in SQL Full-Text search.

Script 3 from the code download illustrates the use of language tags to set the word breaker.

How Words are stored in the index

Generally, the words broken at index time are stored in the catalogs exactly as broken by the word breakers. However, there are some exceptions. For example:

- *c#* is indexed as *c* (if *c* is not in your noise word list, see more on noise word lists later), but *C#* is indexed as *C#* (in SQL 2005 and SQL 2000 running on Win2003 regardless if *C* or *c* is in your noise word list). It is not only *C#* that is stored as *C#*, but any capital letter followed by *#*. Conversely, *c++* (and any other lower-cased letter followed by a ++) is indexed as *c* (regardless of whether *c* is in your noise word list).
- Dates are stored in multiple formats for example *1/1/06* is stored as *1/1/06* and *2006/1/1* (and also 1906/01/01! – 1/1/2006 is stored correctly as 1/1/2006 and 2006/01/01).
- For some languages the word is stored more than once in the catalog – the first time as it appears in the content, and then with alternative word forms of that word. For example, using most word breakers, *L'University* is stored as *University* and also as *L'University*.
- German compound words are stored as a compound word and the constituent words (for example, *wanderlust* is stored as *wanderlust*, *wander*, and *lust*) when indexed using the German word breaker. Using other word breakers it is stored as *wanderlust*.
- Chinese and other Far East language "words" are stored as the word, characters, and constituent characters.

You can determine how the word breaker will break a word in your content by running the word through the **lrtest** (language resource test) tool. Lrtest ships with the SharePoint Portal Server resource kit, which you can obtain from SharePoint Portal Server CD in the Support\Tools directory.

You can find a description of how to use this tool here: http://support.microsoft.com/kb/890613. For SQL 2000 the language resource location is:

```
HKLM\SYSTEM\CurrentControlSet\Control\ContentIndex\Language
```

The relevant language resource location for SQL 2005 is:

```
HKLM\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL.1\MSSearch\Language
```

Regrettably, lrtest does not work for most SQL 2005 word breakers except for Thai, Swedish, and Japanese, so the following examples use the SQL FTS 2000 word breakers. For the time being, you will have to resort to trial and error to determine if the behavior revealed using lrtest on the SQL 2000 work breakers still applies on SQL 2005.

For index time behavior the relevant key for your language is **WBreakerClass**. So, to see how C# is broken for US English you get the CLSID of the following key:

```
HKLM\SYSTEM\CurrentControlSet\Control\ContentIndex\Language\
English_US\WBreakerClass
```

And run the following:

```
lrtest /b /c:{80A3E9B0-A246-11D3-BB8C-0090272FA362} "C#"
```

The result (abbreviated) should look as follows:

```
IWordSink::PutWord: cwcSrcLen 2, cwcSrcPos 0, cwc 2, 'C#'
```

Trying c#:

```
lrtest /b /c:{80A3E9B0-A246-11D3-BB8C-0090272FA362} "c#"
```

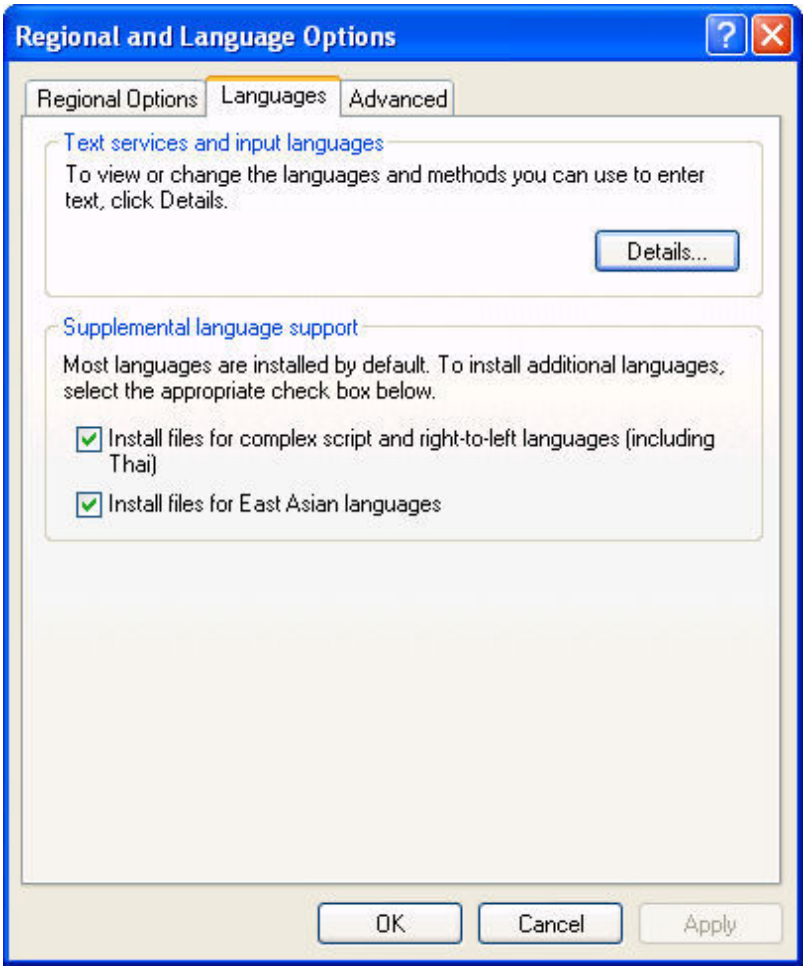
The result (abbreviated) should look as follows:

```
IWordSink::PutWord: cwcSrcLen 2, cwcSrcPos 0, cwc 2, 'c'
```

Adding support for your own language word breaker

In SQL 2005 you can add support for your own language by adding a key to HKLM\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL.1\MSSearch\Language. For example in order to add support for Arabic, follow these steps:

1. Download and install the Arabic word breaker from: http://www.microsoft.com/middleeast/arabicdev/beta/search/
2. Run the installation program
3. Install Complex Script components – open up Regional Settings, click on the Languages tab, and select Install files for complex script and right to left languages.



4. Add the Arabic key: HKLM\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL.1 \MSSearch\Language\Arabic-EG with the following values:

   Locale (dword) 1025 (decimal)

   NoiseFile (String) c:\temp\Arabic.txt

   StemmerClass (String)

   TsaurusFile (String) c:\temp\ArabicThesaurus.txt

   WordBreakerClass (String) {3E0C67A6-38F8-43b6-BD88-3F3F3DAC9EC1}

5. Add the following key HKLM\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL.1 \MSSearch\CLSID\{3E0C67A6-38F8-43b6-BD88-3F3F3DAC9EC1} with a string value of lrar32.dll.
6. Copy lrar32.dll and lrar32.lex from C:\Program Files\SharePoint Portal Server\Bin\ to C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn
7. Issue a call to `exec sp_fulltext_service 'update_languages'` and restart SQL Server.

It is critical that you review your license, or contact your local Microsoft office, with regard to

possible licensing issues surrounding use of the Arabic word breaker in your SQL Server search application. The Arabic word breaker seems to be in perpetual beta and it is not clear whether it is licensed for SQL Server 2005. It appears that there are no licensing restrictions on other third-party supplied SQL 2005 word breakers, but they are not supported by Microsoft.

This post on blogs.msdn.com provides more information on how to add custom language support for SQL 2005 FTS.

Token	SQL 2000 on Win2k (SP4)	SQL 2000 on Win2003	SQL 2005
AT&T	AT, T	AT, T	AT&T
C#	C	C#	C#
C++	C	C++	C++
c#	C	C	C
c++	C	C	C
.Net	NET	NET	NET
<Title>rats<Title>	<TITLE>R, ATS, <TITLE>	TITLE, RATS, TITLE	TITLE, RATS, TITLE

Note that the anomalies you see with the HTML tags, or the > and < characters, only occur when you are indexing text; they do not occur if you are using the HTML iFilter. Unless otherwise indicated, these tokens are all searchable using a placeholder in place of the non-alphanumeric character – so a search on *AT&T* will match with *AT&T*, *AT!T*, *AT*T*, *AT T*, and so on. To correct this behavior you may need to replace the problem token in your content with a non-breaking version. For example if you want searches on AT&T to be handled properly you need to e. replace *AT&T* with *ATANDT* wherever they occur in the tables you are full-text indexing and and then trap for a search phrases which include *AT&T*, and replace it with *ATANDT*. The replacement feature in the thesaurus option is ideal for this.

Chinese, Japanese, Korean and Thai

The Far East word breakers (Chinese, Japanese, Korean and Thai) break the text stream into individual characters, or even break the characters into constituent characters which correspond to words. The Chinese writing system is not phonetically based. Most ancient writing systems (Chinese, Sumerian, Egyptian, and Aztec for example) started as pictograms, whereby each character was represented by a picture. Due to the large number of characters required, most pictorial systems collapsed and/or evolved in to phonetic systems. However, Chinese retained a pictogram component and then evolved to incorporate ideograms, whereby the graphical symbols (characters) represent ideas.

There are actually five types of Chinese characters. The simplest are pictograms and ideograms, but these two types comprise only a small percentage of the total number of characters. Most Chinese characters are semantic-phonetic compounds where part of the character indicates the sound of the word, and another part indicates its meaning. The other two types are compound ideograms (where two or more characters are compounded into one, and their individual meanings contribute to the meaning of the whole) and phonetic loans (in which a pre-existing character is borrowed to represent a word whose sound is similar to that of the word the character originally represented).

Consider that the character for tree in Chinese is (木), forest is (森), and (林) is a wood. If you look closely you will see the character for tree is present in the characters for wood and forest; in wood you see two tree characters stuck together, in forest you see three stuck together.

In Japanese or Chinese a character string has to be broken into its component characters to determine what the word is about and to determine what it matches (as the phrase among Natural Language researchers goes – you can tell a word by its neighbors – similarly you can tell a character by its neighbors). So if you were doing a FreeText search on the Chinese character wood you would expect to get hits to rows containing wood and forest.

However to index Chinese characters you need to derive their meaning, which requires further decomposition. Chinese characters consist or one or more strokes, termed radicals. There are 214 radicals used to build characters. By decomposing a character into its constituent characters, or radicals, you can derive its meaning. For more information on radicals consult: http://www.chinese-outpost.com/language/characters/char0035.asp.

The word breakers for Chinese, Japanese, Korean, and Thai will take the text stream, break at white space, and punctuation, and then make multiple passes of the tokens using context to extract the constituent characters. Indexing these languages is a CPU intensive process because of the multiple passes required in the word breaking process for these languages.

UK vs. US English at Index Time

SQL FTS supports two versions of English: UK English and US English. One of the questions I most frequently get asked is ""What is the difference between the UK and US English word breakers?" There are four points to keep in mind here:

- In actual fact UK English does not refer to the Queen's English or the English used in the United Kingdom, but **International English**; the English that is used in all other English speaking countries other than US English.
- They use the same word breaker.
- Most frequently the UK word breaker is used when you want to use a different noise word list than the US one to implement a less precise search (we'll cover this on the section in noise words), or a different thesaurus.
- Significant differences do occur at query time between the two language stemmers (we'll cover this in the next article, in the section on UK vs. US English at Query Time), but not in the word breakers. In fact in SQL 2000 FTS the UK English and US English word breakers are the same. In SQL 2005 they have a different CLSID. I did speak with a Microsoft developer who worked on many of the Microsoft Search products and he speculated that the differences are there to handle different word endings in UK English, i.e. Catalog versus catalogue; check versus cheque; bank versus banc, or banque; orientated versus oriented.

German

As mentioned previously, the German language has a large number of compound words. When these words are indexed they are stored as the compound word as well as its constituent words. The German word breaker also handles umlauts (the dots that appear in *Mötley Crüe*). Umlauts denote plural forms of nouns and sometimes different verb forms (e.g. present tense).

Words with umlauts are indexed with the umlauts and the non variant of the word without the umlaut, for example *häuser* (German for houses) is indexed as *häuser* and *haeuser*. *Mötley Crüe* is stored as *Mötley Crüe*, and *Moetley Cruee*.

Noise Words

When computer scientists first started work in the field of information retrieval (the branch of computer science which involves search technology) hard disks were very expensive. Search scientists did anything they could to save on disk space. They recognized that there were a large number of words which occurred very frequently and which were not helpful in resolving searches. Such words are referred to as **noise words**. For example words which linguists class as articles, such as *the*, *a*, and *at* are for the most part irrelevant in a search. They add nothing to the search and don't help to resolve searches – for instance a search on *University of California at Berkley* returns the same results as a search on *University of California Berkley*. Similarly pronouns like *me*, *my* and *mine* don't help in a search either. Articles and pronouns (and other similar parts of speech) occur frequently in bodies of text, and by not indexing them search scientists were able to save disk space and improve search speed (very slightly), with only a marginal decrease in accuracy. Google does not index noise words, but MSN Search does to provide greater accuracy in searches!

Another example of a noise word is a word which may not be a part of speech, but occurs frequently in the content and does not help in resolving searches. For example, the word Microsoft occurs on every page of Microsoft.com. A search on Microsoft Windows XP is indistinguishable from a search on Windows XP. By adding such words to your noise word list your catalog size will decrease.

The problem with noise words is that they will break searches which contain them. So a search on the band "*the the*" will result in the following message on SQL 2005:

```
Informational: The full-text search condition contained noise word(s).
```

But results will be returned for matches to other search arguments, so a search on "the the albums" would return rows which contained the word albums in the full-text indexed columns. On SQL 2000 however, the message is:

```
Server: Msg 7619, Level 16, State 1, Line 1

Execution of a full-text operation failed. A clause of the query contained only
ignored words.
```

And the query will not return any results. BOL states that this behavior can be fixed in SQL 2005 by using:

```
exec sp_configure 'transform noise words', 1

reconfigure
```

But as far as I can tell, this command has no effect.

Summary

In this article, we looked at the language options of SQL Full-Text Search which occur at index time. We started off by looking at the SQL full-text architecture with special emphases on iFilters and word breakers. We also looked at how language aware iFilters can override the server language settings or the word breaker you specify to break the text in these documents. We then discussed how the words are stored in the full-text catalogs and had a look at some of the language idiosyncrasies. In the next article in the series we will look at language options at query time; i.e. when you query. Till next time, I hope you find what you are looking for!