```
CALL sa_text_index_stats( );
```

To retrieve text index creation options

- When a text index is created, the current database options are stored with the text index. To retrieve the option settings used during text index creation, execute the following statement:

  ```
  SELECT b.object_id, b.table_name, a.option_id, c.option_name,
  a.option_value
  FROM SYSMVOPTION a, SYSTAB b, SYSMVOPTIONNAME c
  WHERE a.view_object_id=b.object_id
  AND b.table_type=5;
  ```

  A table_type of 5 in the SYSTAB view is a text index.

See also

- "sa_text_index_stats system procedure" [*SQL Anywhere Server - SQL Reference*]
- "SYSMVOPTION system view" [*SQL Anywhere Server - SQL Reference*]
- "SYSMVOPTIONNAME system view" [*SQL Anywhere Server - SQL Reference*]
- "SYSTAB system view" [*SQL Anywhere Server - SQL Reference*]

How to use external term breaker and prefilter libraries

Why use an external term breaker or prefilter library

Full text search in SQL Anywhere is performed using a text index. Each value in a column on which a text index has been built is referred to as a **document**. When a text index is created, each document is processed by a built-in term breaker algorithm to determine the terms contained in the document, and the position of the terms in the document. The term breaker algorithm also applies these operations to the documents (text components) of a query string. For example, the query string 'rain or shine' consists of two documents, rain and shine, connected by the OR operator.

Depending on the needs of your application, you may find limitations to the built-in term breaker. For example, the SQL Anywhere term breakers do not offer language-specific term breaking. Here are some other reasons you may want to implement a custom term breaker:

- **Handling of words with apostrophes** The word "they'll" is treated as "they ll" by the GENERIC term breaker. However, you could design a custom term breaker that treats the apostrophe as part of the word.

- **No support for term replacement** You cannot specify replacements for a term. For example, when indexing the word "they'll", you might want it stored as two terms: they and will. Likewise, you may want to use term replacement to perform a case insensitive search on a case sensitive database.

SQL Anywhere allows you to use your own **external term breaker library**. That way, when your text index is built, the external term breaker is used instead of the SQL Anywhere term breaker.

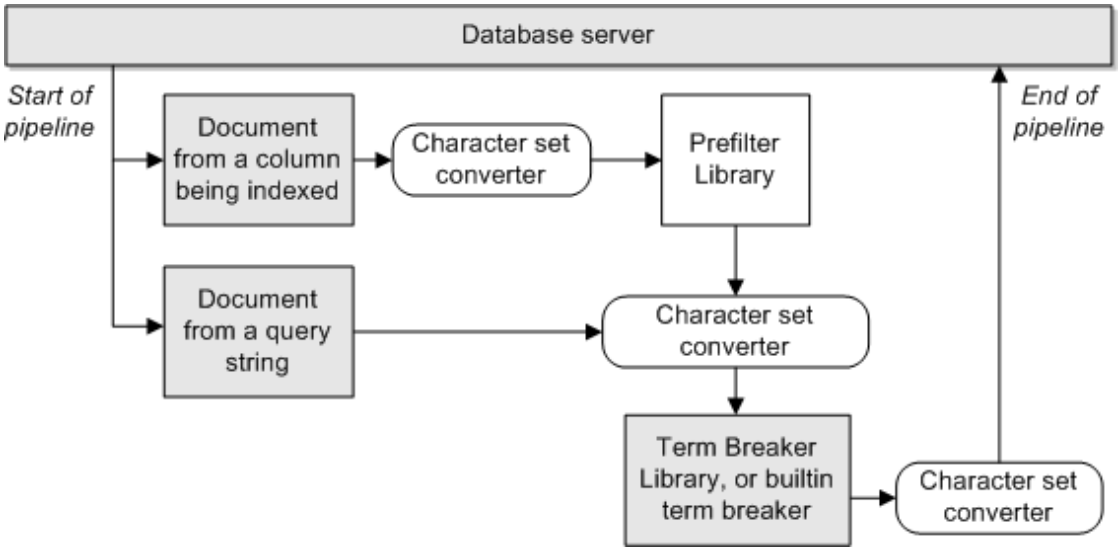
SQL Anywhere also allows you to use your own **external prefilter library** to perform prefiltering on data before it is indexed. Prefiltering allows you to extract only the textual content you want indexed from

Copyright © 2010, iAnywhere Solutions, Inc.

a document. For example, suppose you want to create a text index on a column containing XML values. A prefilter would allow you to filter out the XML tags so that they are not indexed with the content.

Full text pipeline workflow

The following diagram shows how data is converted from a document to a stream of terms to index within SQL Anywhere. The workflow for creating a text index, updating it, and querying it, is referred to as the **pipeline**. The mandatory parts of the pipeline are depicted in light gray.



High level view of how the pipeline works

1. The processing of each document is initiated by the database server by calling the begin_document method on the end of the pipeline, which is either the term breaker or the character set converter. Each component in the pipeline calls begin_document on its own producer, in order to satisfy, and before returning from its, begin_document method invocation.

2. The database server calls get_words on the end of the pipeline after the begin_document completes successfully.

   - While executing get_words, the term breaker calls get_next_piece on its producer to get data to process. If a prefilter is available, the data is filtered by it during the get_next_piece call.

   - The term breaker breaks the data it receives from its producer into terms according to its term breaking rules.

3. The database server applies the minimum and maximum term length settings, followed by the stoplist restrictions to the terms returned from get_words call.

Copyright © 2010, iAnywhere Solutions, Inc.

4. The database server continues to call get_words until no more terms are returned. At that point, the database server calls end_document. This call is propagated through the pipeline in the same manner as the begin_document call.

Note
Character set converters are transparently added to the pipeline by the database server where necessary.

External prefilter libraries

How to configure SQL Anywhere to use the external prefilter

SQL Anywhere does not provide a built-in prefilter algorithm. To have data pass through an external prefilter library, you declare the library using the ALTER TEXT CONFIGURATION statement, similar to the following statement:

```
ALTER TEXT CONFIGURATION my_text_config
   PREFILTER EXTERNAL NAME 'my_prefilter@prefilter'
```

This example tells the database server to use the my_prefilter function in the prefilter library to obtain a prefilter instance to use when building or updating a text index using the my_text_config text configuration object. The interface for a prefilter is described here: “a_text_source interface” on page 364.

See also: “ALTER TEXT CONFIGURATION statement” [*SQL Anywhere Server - SQL Reference*]

How to design an external prefilter library

The prefilter library must be implemented in C/C++, and must have:

- #include of the prefilter interface definition header file, *extpfapi1.h*
- an implementation of the a_text_source interface containing function pointers as described in a_text_source. See “a_text_source interface” on page 364.
- an entry point function that sets up and returns an instance of a_text_source (prefilter) and the label of the character set supported by the prefilter.

Calling sequence for the prefilter

The following calling sequence is executed by the consumer of the prefilter for each document being processed:

```
begin_document(a_text_source*)
get_next_piece(a_text_source*, buffer**, len*)
get_next_piece(a_text_source*, buffer**, len*)
...
end_document(a_text_source*)
```

Copyright © 2010, iAnywhere Solutions, Inc.