Knowledge Discovery

Software Version 25.1

Sentence Breaking API Technical Note

opentext™

Document Release Date: January 2025
Software Release Date: January 2025

Legal notices

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for updated documentation, visit https://www.microfocus.com/support-and-services/documentation/.

Support

Visit the MySupport portal to access contact information and details about the products, services, and support that OpenText offers.

This portal also provides customer self-solve capabilities. It gives you a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the MySupport portal to:

- View information about all services that Support offers
- Submit and track service requests
- Contact customer support
- Search for knowledge documents of interest
- View software vulnerability alerts
- Enter into discussions with other software customers
- Download software patches
- Manage software licenses, downloads, and support contracts

Many areas of the portal require you to sign in. If you need an account, you can create one when prompted to sign in.

Contents

Introduction

This technical note describes how to use the Knowledge Discovery Sentence Breaking API to create a custom sentence breaking library.

The purpose of a sentence breaking library in the Knowledge Discovery infrastructure is to take a byte stream containing a language that does not separate its words with whitespace, and to produce a modified stream that contains whitespace between tokens. The tokens that Knowledge Discovery uses internally are the whitespace-separated tokens in the modified stream. As such, if any morphological manipulation is required, this must appear in the modified stream.

> **NOTE:** If you produce a modified stream where you add, remove, or modify characters in addition to the whitespace augmentation, advanced functionality (such as highlighting) might not work fully.

Example

An example sentence breaking library is available for you to use as a starting point in your own implementations. You can find the project and code files at:

https://github.com/opentext-idol/idol-sentence-breaking-example

Interface

Sentence-breaking libraries are loaded as shared objects, dynamic libraries, or DLLs (the format is platform-specific), typically during initialization. Unloading of libraries is permitted, but does not always occur in normal usage.

Internally, the library is associated with a knowledge structure, which ties together information about known languages. Some applications maintain multiple knowledge structures, so multiple calls might be made to load the library. Similarly, a library that has been loaded in multiple places might be unloaded in some places, but still expected to run in the other places.

The library must support the following functions:

```
char * sentenceBreakGetInfo ()
```

```
long sentenceBreakInitialise (char *szLangDirectory, char **pszErrorDescription)
```

```
long sentenceBreakFreeErrorDescription (char **pszErrorDescription)
```

In addition, it must support one of the following options as the sentence breaking tokenization function:

- The automatically allocating interface:

  ```
  long sentenceBreakEx (char *szBufferIn, char *szBufferOut, t_
  lang2SentenceBreakingParam *pParam)
  ```

- The self allocating interface:

  ```
  long sentenceBreakSelfAllocating(char *szInput, char **pszOutput, size_t
  *pnOutputLength, t_lang2SentenceBreakingParam *pParam);
  void sentenceBreakFreeBuffer(char **pszBuffer);
  ```

> **NOTE:** If the library provides both the automatically allocating and the self allocating interface, the Content component uses only the self-allocating interface.

Common Parameters

The `t_lang2SentenceBreakingParam` structure is passed as the last argument to the sentence breaking tokenization function (`sentenceBreakEx` or `sentenceBreakSelfAllocating`).

```
typedef struct
{
   char *szOptions;
   char *szNonBreakCharacters;
   char *szFullPathDirectory;
   void (*fnLogFunction)(const char *fmt, va_list ap);
   int bQueryMode;
   int nLangCode;
```

```
}
t_lang2SentenceBreakingParam;
```

The following table describes the member variables in this structure.

Member	Description
`szOptions`	This string is blank unless you can configure options for the library (as defined in the `OPTIONS` tag). This string contains characters corresponding to the set options from this tag. For example, if `sentenceBreakGetInfo` includes the `OPTIONS` attribute `KANA="K"`, and this option is selected in the Content component configuration file (that is, by the `SentenceBreakingOptions` parameter) this string contains the character `K`. In addition, as long as there is at least one attribute to the `OPTIONS` tag, there is a plus (+) or minus (-) sign indicating whether stemming is activated or deactivated respectively.
`szNonBreakCharacters`	Some characters have a special meaning in Content component syntax and are attached to the tokens that they apply to. For example, at query time `DOG[123]` indicates that the term *DOG* has the APCM weight of `123`. Characters in this configuration string are treated as glue characters; that is, additional whitespace is never added to the left or right of them. Furthermore, the library must make decisions on how to break characters to the left of a glue character independently of those to its right. For example, the library might break the stream `DOG123` into `DOG1 23`, break the stream `DOG` as `DOG`, and break the stream `123` as `123`. If, however, an asterisk (*) was set in this configuration string, the token `DOG*123` should be returned as `DOG*123`; that is, the characters to the left (DOG) are treated independently of those on the right (123). The default characters in this set are `* ? : . _ [ ] ~ /`
`szFullPathDirectory`	This string contains the language directory path.
`fnLogFunction`	When `fnLogFunction` is not NULL, then the library can use this function to output messages to a log file. However, the library must not assume that `fnLogFunction` is non-NULL. `pParam->fnLogFunction("Processed text: %s\n", szInput);`
`bQueryMode`	This string indicates whether to process the text for indexing or for querying. When `bQueryMode` is set to **`0`**, the library must

	process the text for indexing. For all other values of `bQueryMode`, the library must process the text for querying. For more information, see Index and Query Modes, below.
`nLangCode`	This integer indicates the language. If your library processes multiple languages, you must use the LANGCODE attribute of the sentence breaking library information string to specify a language code for each language. The Content component then uses the `nLangCode` variable to indicate the language that the library must use. For more information, see Use Multiple Languages, below.

NOTE: This structure is shared across various internal functions. You should not infer any additional information from their names. These configuration strings must not be modified by the library.

Index and Query Modes

When Content sends text to the sentence breaking library, it indicates whether to process the text for indexing, or for querying. It is useful to distinguish between these two situations because query text often consists of keywords or incomplete sentences. The library might need to process such query text differently, because it cannot rely on the presence of many linguistic features that help to resolve ambiguities in parsing, which are usually present at index time.

Content indicates the type of text that it is sending to the library to tokenize by setting the member variable `bQueryMode` of the structure `t_lang2SentenceBreakingParam`.

When `bQueryMode` is set to **`0`**, the library must process the text for indexing. For all other values of `bQueryMode`, the library must process the text for querying.

Use Multiple Languages

You can create a library to process text from multiple languages.

The sentence breaking library information string must contain a `PRIMARYLANGUAGE` entry for each language that the library supports. You must also set the `LANGCODE` attribute to a different integer for each language.

For example, if your library supports Japanese and Korean, the information string might contain:

```
<PRIMARYLANGUAGE NAME="JPN" DEFAULTENCODING="UTF-8" LANGCODE="1"/>
<PRIMARYLANGUAGE NAME="KOR" DEFAULTENCODING="UTF-8" LANGCODE="2"/>
```

Content then sets the `nLangCode` member variable of the `t_lang2SentenceBreakingParam` structure to define the language that the library must use.

For the previous example, if Content sends Korean text, it sets `nLangCode=2` for the `t_lang2SentenceBreakingParam` argument to the sentence breaking tokenization function. If it sends Japanese text, it sets `nLangCode=1`.

For any other language, or if `LANGCODE` is not present in the `PRIMARYLANGUAGE` tag of the sentence breaking library information string, Content sets `nLangCode` to **`-1`**.

For more information, see sentenceBreakGetInfo Function, below.

sentenceBreakGetInfo Function

```
char *sentenceBreakGetInfo ()
```

This function permits the sentence-breaking library to return information about itself to the calling application. You must return a pointer to a static string in memory, because no attempt is made to free the information that is passed out. The string returned is the *sentence breaking library information string*.

This information is returned in XML format. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ROOT>
   <VERSION NAME="J-Cha" MAJOR="2" MINOR="2" PATCH="0"
    DATE="6 June 2006" REVISION="12345"/>
   <PRIMARYLANGUAGE NAME="Japanese" DEFAULTENCODING="UTF8"/>
   <RUNTIME THREADSAFE="N"/>
   <OPTIONS KANA="K" OLDNEW="O" NUMBERS="N" DBCS="D"/>
</ROOT>
```

You must follow the format of this XML.

Tag	Description
`VERSION`	All attributes are required.
`PRIMARYLANGUGAGE`	• `NAME`. The `NAME` attribute must be an Autonomy-recognized name for the required language (that is, either the name of the language in English, or its three-letter ISO 639 code). This attribute is required. • `DEFAULTENCODING`. The `DEFAULTENCODING` attribute must be the Knowledge Discovery-recognized name for the required encoding (UCS2 is not permitted). In the application, text is converted into this encoding before being passed to the sentence-breaking library, and it is assumed that output from the library is also in this encoding. This attribute is required. **NOTE:** Because the Content component stores all text in UTF-8 internally, performance is affected if the library requires a different encoding. • `LANGCODE`. The `LANGCODE` attribute is the number of the `PRIMARYLANGUGAGE` tag. Use this attribute if your library processes multiple languages. In this case, set `LANGCODE` to a different integer for each language that your library can process. Content sets `nLangCode=N` in the `t_lang2SentenceBreakingParam` structure, where `N` is the appropriate `LANGCODE` for the language. If `LANGCODE` is absent from the `PRIMARYLANGUGAGE` tag, Content sets `nLangCode=-1`. See Use Multiple Languages, on the previous page.

RUNTIME	• THREADSAFE. Set THREADSAFE to **Y** if the library is thread-safe, otherwise set it to **N**. This value is required. It affects whether simultaneous requests are sent to the library. All libraries must still support multiple instances, as specified in Interface, on page 5. • HIGHLIGHT. Set HIGHLIGHT to **NQSP** for any sentence-breaking library that uses the special syntax. See Special Sentence-Breaking Syntax, on page 13. • BUFFERMULTIPLIER. Set BUFFERMULTIPLIER to an integer value if you want to adjust the size of the allocated buffer for szOutput in sentenceBreakEx. The default value is **7**.
OPTIONS	The OPTIONS tag is optional. Attributes in this section are library-specific. You can use any attribute name that uses the characters A-Z.The values for attributes can be any single character from A-Z; no two attributes can share the same character. These attributes specify options that can be set for the library using a Content component configuration file. See sentenceBreakEx Function, on the next page.

sentenceBreakInitialise Function

```
long sentenceBreakInitialise (char *szLangDirectory, char **pszErrorDescription)
long sentenceBreakFreeErrorDescription (char **pszErrorDescription)
```

This function is called after the library has been loaded so that it can perform any necessary initialization.

The following table describes the parameters for this function.

Parameter	Description
szLangDirectory	A string containing the configured language directory. You must not modify this string. This directory is typically set by the LanguageDirectory configuration parameter for the Content component, and is the directory where the application looks for the library. Any data files that the library requires must exist in this directory or one of its subdirectories. This parameter is the only piece of configuration that is passed to the library for initialization.
pszErrorDescription	Allows you to pass an error string from the library in the event of a problem. This string is used only when the return state of the function indicates a failure. After using the string placed in the second parameter, the sentenceBreakFreeErrorDescription function is called to free the memory allocated for this string if necessary. In the event of a success, the return value must be set to zero (0). For other permitted return codes, see Error Codes, on page 11.

In addition to any necessary initialization for the library, `sentenceBreakInitialise` must run a basic (internal) sentence-breaking test, and ensure that the expected results return.

sentenceBreakEx Function

```
long sentenceBreakEx (char *szBufferIn, char *szBufferOut, t_
lang2SentenceBreakingParam *pParam)
```

This function is called to add whitespace to a buffer.

The following table describes the parameters for this function.

Parameter	Description
`szBufferIn`	The original buffer in the encoding requested by the `DEFAULTENCODING` attribute, and NULL terminated. This string must not be modified by the library.
`szBufferOut`	A buffer for the output in the encoding requested by the `DEFAULTENCODING` attribute, and NULL terminated. This buffer is pre-allocated by the application. By default, the number of bytes allocated is 7 times the number of bytes in the original buffer, plus 55. You can use the `BUFFERMULTIPLIER` option in the sentence breaking library information string to modify this buffer. The library must not attempt to reallocate this buffer.
`t_ lang2SentenceBreakingParam`	This structure contains configuration information. See Common Parameters, on page 5.

Most configuration options do not change between calls to `sentenceBreakEx`. In particular, options listed using the `OPTIONS` tag are only set by the Content component `SentenceBreakingOptions` configuration parameter, which is static between re-initialization of the server.

In the event of a success, the return value should be set to zero (0). See Error Codes, on the next page for other permitted return codes.

sentenceBreakSelfAllocating Function

```
long sentenceBreakSelfAllocating(char *szInput, char **pszOutput, size_t
*pnOutputLength, t_lang2SentenceBreakingParam *pParam);
```

This function allows your library to allocate its own buffer, rather than relying on the Content component to allocate one of sufficient length.

The following table describes the parameters for this function.

Parameter	Description

`szInput`	This string contains the input text that the library must process.
`pszOutput`	This string contains the output string that the library returns.
`pnOutputLength`	This contains the length of the allocated buffer.

If you implement this function, you must also use `sentenceBreakFreeBuffer`. See sentenceBreakFreeBuffer Function, below.

> **NOTE:** Your library must provide either the self-allocating interface (`sentenceBreakSelfAllocating` and `sentenceBreakFreeBuffer`), or the automatically allocating interface (`sentenceBreakEx`).
>
> If you provide both interfaces, the Content component uses only the self-allocating interface.

sentenceBreakFreeBuffer Function

```
void sentenceBreakFreeBuffer(char **pszBuffer);
```

This function releases memory allocated by the library in `pszBuffer` and sets `*pszBuffer` to NULL.

> **NOTE:** Your library must provide either the self-allocating interface (`sentenceBreakSelfAllocating` and `sentenceBreakFreeBuffer`), or the automatically allocating interface (`sentenceBreakEx`).
>
> If you provide both interfaces, the Content component uses only the self-allocating interface.

sentenceBreakUninitialise Function

```
long sentenceBreakUninitialise(char **pszErrorDescription);
```

The Content component calls this function on shutdown if the library has `KEEPRESIDENT="Y"` as an attribute of the `RUNTIME` element in the sentence breaking library information string. Use this function to uninitialize the library.

You can use the `pszErrorDescription` parameter to store any error message encountered during uninitialization. However, the library must not rely on `pszErrorDescription` being non-NULL, and must ignore the parameter if it is set to NULL.

Error Codes

The `sentenceBreakInitialise`, `sentenceBreakFreeErrorDescription`, and `sentenceBreakEx` functions return a long containing an error code. In the event of a success, the functions should return the value zero (0). Other acceptable error codes are:

```
-1   General error during initialisation
-2   General error during breaking
-3   Unknown error
```

```
-4   Encoding error
-5   Invalid input
-6   Assertion error
-7   Directory not found
-8   Library is already initialised
-9   Memory error
-10  File not found
```

Special Sentence-Breaking Syntax

You can use a sentence-breaking library to perform stemming, allowing the user to plug in their own stemming algorithm to the OpenText Knowledge Discovery Content component.

To highlight against the original text, the original form of the token must be kept for comparison to the original (unbroken) text. Additionally, for advanced search, the library must return the unstemmed token. The unstemmed token is often identical to the token in the unbroken buffer, but might be different if, for example, the same word has multiple spellings.

To enable both term highlighting and advanced search, you must use a special syntax for stemmed tokens in the output buffer from the sentence breaking tokenization function. There are two forms of this syntax. The new form is a superset of the old form.

Output Syntax for Stemming and Advanced Search

Use the old sentence-breaking syntax when the unstemmed token is identical to the token in the unbroken buffer. This is the case most of the time. The new syntax allows normalization in cases where the same word has multiple spellings, such as in the example realize and realise given below.

Old Syntax

The old syntax is of the form `1X2X3X4`, where X is the special syntax-breaking character (see Special Syntax-Breaking Character, on the next page). Any of 1, 2, 3, and 4 can be omitted, though it is more usual to omit 1 and 4. The stemmed form of the token is `124`, the unstemmed form is `134`, and the term for matching to the unbroken buffer is also `134`. For example, for the word *men* stemming to *man*, you can break the word in any of the following ways:

`mXaXeXn`
`XmanXmenX`
`mXanXenX`
`XmaXmeXn`

New Syntax

The new syntax is of the form `1X2X3X4X5`, where X is the special syntax-breaking character (see Special Syntax-Breaking Character, on the next page). The stemmed form is `124`, and the term for matching to the unbroken buffer is `134`, which is the same as the old syntax. However, the unstemmed form is now `154`.

The new syntax allows the unstemmed and original forms to be different. This might be needed if the original form was spelled differently or incorrectly. For example, for the original form *realise* stemming to *realiz* with the unstemmed form *realize*, you can break the word as

`realiXzXseXXze`

Special Syntax-Breaking Character

The special character is a 3-byte UTF-8 character with byte sequence `E2 8A A1`.

Notifying the Content component

Any sentence-breaking library that uses the special syntax must specify `HIGHLIGHT=”NQSP”` in the `RUNTIME` section of the XML string returned by the function `sentenceBreakGetInfo()`. For example, for a threadsafe library that uses the special syntax, the `RUNTIME` section reads:

```
<RUNTIME THREADSAFE ="Y" HIGHLIGHT="NQSP">
```

If you are using a library for stemming, you must set `Stemming` to **`false`** in the relevant language section in the Content configuration file. For example:

```
[german]
Encodings=ASCII:germanASCII,UTF8:germanUTF8
Stoplist=german.dat
SentenceBreaking=germanbreaking
Stemming=false
IndexNumbers=1
```

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on OpenText Knowledge Discovery 25.1 Sentence Breaking API Technical Note

Add your feedback to the email and click **Send**.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to MFI-swpdl.idoldocsfeedback@opentext.com.

We appreciate your feedback!