OpenTM2 for Windows

Technical Reference

Version 14.0

OpenTM2 for Windows

Technical Reference

Version 14.0

Note!: Before using this information and the product it supports, be sure to read the general information under "Notices," on page 191.

| **Fourth Edition, April 2017**

| This edition applies to OpenTM2 Version 1.4.0

Contents

About this book

This book is intended for users working with OpenTM2 under Windows.

This book is for all users of OpenTM2 who are already familiar with the basic functions of OpenTM2.

OpenTM2 basics are explained in *A Quick Tour* as well as in the *Translator's Workbook*. The *Translator's Reference* provides information on the more advanced topics of translating with OpenTM2. It provides comprehensive descriptions of all OpenTM2 components and their functions essential for doing the daily translation business. It also provides appendixes with detailed technical information.

This document describes the **APIs** (Application Programming Interfaces), which allow technically experienced users to automate processes.

An easy way to find information about a specific item is to look it up in the index. However, if you are not sure about the precise naming of a function, search the table of contents to find a topic where this function may belong to.

Related information

OpenTM2 for Windows: A Quick Tour. It teaches the basics of translating with OpenTM2.

OpenTM2 for Windows: Translator's Workbook. It helps to learn using OpenTM2.

OpenTM2 for Windows: Translator's Reference. It helps to understand details of OpenTM2.

Summary of Changes

This section provides a summary of changes compared to the previous version of the product. Changes in the book are marked with a vertical bar.

- **New** API-calls added to the document (see revision bars in the document, or see "New APIs in OpenTM2 v1.4.0" on page xv).

New APIs in OpenTM2 v1.3.0

This chapter is listing **new APIs** implemented in OpenTM2 v1.3.0.

New and Updated APIs in OpenTM2 v1.3.1

This chapter is listing **new and updated APIs** implemented in OpenTM2 v1.3.1.

- No updates in this release.

New and Updated APIs in OpenTM2 v1.3.2

This chapter is listing **new and updated APIs** implemented in OpenTM2 v1.3.2.

- Updated API “EqfAnalyzeDocEx”: see “EqfAnalyzeDocEx” on page 31.

New APIs in OpenTM2 v1.4.0

This chapter is listing **new APIs** implemented in OpenTM2 v1.4.0.

- API “EqfImportMemInInternalFormat”: see “EqfImportMemInInternalFormat” on page 119.
- API “EqfExportMemInInternalFormat”: see “EqfExportMemInInternalFormat” on page 88.
- API “EqfOpenMem”: see “EqfOpenMem” on page 126.
- API “EqfCloseMem”: see “EqfCloseMem” on page 44.
- API “EqfQueryMem”: see “EqfQueryMem” on page 132.
- API “EqfSearchMem”: see “EqfSearchMem” on page 138.
- API “EqfUpdateMem”: see “EqfUpdateMem” on page 141.
- API “EqfListMem”: see “EqfListMem” on page 120.

Part 1. Programming interfaces

Chapter 1. Application programming interface for adding editors

OpenTM2 provides an application programming interface (API) that lets you use various editors as translation editors. Using this API the editor can access all functions required for a translation, namely the Translation Memory, the automatic dictionary lookup, and the dictionary lookup dialog. OpenTM2 prepares the "Dictionary" and "Translation Memory" windows, establishes the communication links, handles all error conditions, and prepares and accesses the dictionaries and Translation Memory databases. The editor must provide the end-user interface to access the provided services and handle the retrieved data.

All API functions are provided as a dynamic-link library (DLL).

An editor that can be used as a translation editor must meet the following requirements:

- Run as a Presentation Manager application. A VIOwindowed application is not sufficient.
- Be programmable.
- Be able to access programs and DLLs written in C for multithread environments.
- Be able to recognize specific tags and extract and decompose text according to this information.

The following sections describe the data types used by the API interface, possible error conditions, and the individual API calls for the interface provided by OpenTM2.

Data types

The editor must use the following structure to communicate with OpenTM2. The C interface binding is available in the file `EQFTWBS.H`.

```
typedef struct _STEQFSTRUCT
       {
         HWND  hwndEdit ;              /* handle of editor window      */
         CHAR szSemaphore [EQF_NAME];  /* space for the semaphore name */
         HWND hwndEQFPropWnd;          /* handle of proposal window    */
         HWND hwndEQFDictWnd;          /* handle of dictionary window  */
         USHORT usIndustryCode;        /* industry code                */
         CHAR szProjPath [EQF_NAME];   /* path of file to be translated*/
         CHAR szFileName [EQF_NAME];   /* currently transl. file       */
         RECTL rectlEQFPropWnd;        /* coordinates of proposal wnd  */
         RECTL rectlEQFDictWnd;        /* coordinates of dictionary wnd*/
         SHORT sOS2;                   /* Error code                   */
       } STEQFSTRUCT;
```

Return codes

The following list contains all return codes provided by OpenTM2. If an operating-system error is found, the `EQFERR_SYSTEM` is set and the extended return code is updated in the line `stEQFStruct sOS2`.

EQFERR_TM_ACCESS
The Translation Memory could not be accessed.

EQFERR_DICT_ACCESS
The dictionary or the dictionary lookup program could not be accessed.

EQF_OKAY
The request completed successfully.

EQFERR_INIT
The system must first be initialized.

EQFERR_CLOSE_DICT
An error occurred during the closing of the dictionary.

EQFERR_CLOSE_TM
An error occurred during the closing of the Translation Memory.

EQFERR_ENTRY_NOT_AVAIL
The selected proposal is not available.

EQFERR_DISK_FULL
OpenTM2 detected that the disk is full.

EQFERR_TM_NOT_ACTIVE
The Translation Memory is not active.

EQFERR_SEG_EMPTY
The passed segment was empty and therefore was not stored in the Translation Memory.

EQFERR_TM_CORRUPTED
The Translation Memory is corrupted.

EQFERR_SEG_NOT_FOUND
The specified segment was not found.

EQFERR_DICTLOOK_NOT_FOUND
The dictionary lookup dialog could not be loaded.

EQFERR_DICT_LOOKUP_PENDING
The dictionary lookup request is pending.

EQFERR_NO_ENTRY_AVAIL
The dictionary entry is not available.

EQFERR_SYSTEM
A system error occurred.

API calls

The following sections describe the individual API calls for the interface provided by OpenTM2. The following calls are available:

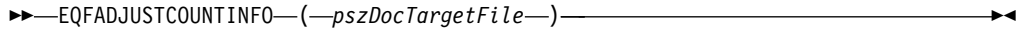
Call...	described on page...
EQFGETDOCFORMAT	“EQFGETDOCFORMAT” on page 12
EQFGETPROP	“EQFGETPROP” on page 12
EQFGETSEGNUM	“EqfGetSegNum” on page 100
EQFGETSOURCELANG	“EQFGETSOURCELANG” on page 14
EQFGETTARGETLANG	“EQFGETTARGETLANG” on page 14
EQFINIT	“EQFINIT” on page 14
EQFQUERYEXITINFO	“EQFQUERYEXITINFO” on page 15
EQFSAVESEG	“EQFSAVESEG” on page 17
EQFSEGFILECONVERTASCII2UNICODE	“EQFSEGFILECONVERTASCII2UNICODE” on page 17
EQFSEGFILECONVERTUNICODE2ASCII	“EQFSEGFILECONVERTUNICODE2ASCII” on page 18
EQFTRANSSEG	“EQFTRANSSEG” on page 19
EQFWORDCNTPERSEG	“EQFWORDCNTPERSEG” on page 20
EQFWRITEHISTLOG	“EQFWRITEHISTLOG” on page 21

EQFADJUSTCOUNTINFO

Purpose

EQFADJUSTCOUNTINFO writes the actual word-counting information for the specified document to the history log file and adjusts the count information stored in the document properties. This API call is quite expensive in resource usage and processing time and should only be called when the **STARGET** file has been changed massively during the **EQFPOSTTM** processing of the user exit..

Format



Parameters

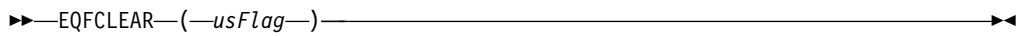
pszDocTargetFile(PSTRING) -- input
The fully qualified name of the document **STARGET** file

EQFCLEAR

Purpose

EQFCLEAR resets or clears the information stored.

Format



Parameters

usFlag(USHORT)
Can be either of the following:

EQFF_NODICTWND
The "Dictionary" window is hidden.

EQFF_NOPRDPWND
The "Proposals" window is hidden.

Return codes

EQF_OKAY
The request completed successfully.

EQFERR_INIT
The system must first be initialized.

Remarks

This call is used to initialize the buffers and clear the "Dictionary" and "Translation Memory" windows after a new document is loaded.

EQFCLOSE

Purpose

EQFCLOSE closes the session with OpenTM2.

Format

Parameters

fShutdown
Can be either of the following:

EQF_CLOSE_STANDBY
The services session is closed, the services remain active.

EQF_CLOSE_EXIT
The services are closed and destroyed.

Return codes

EQF_OKAY
The request completed successfully.

EQFERR_INIT
The system must first be initialized.

EQFERR_CLOSE_DICT
An error occurred during the closing of the dictionary.

EQFERR_CLOSE_TM
An error occurred during the closing of the Translation Memory.

EQFERR_SYSTEM
A system error occurred.

Remarks

This call must be the last OpenTM2 call, implicitly issued by OpenTM2.

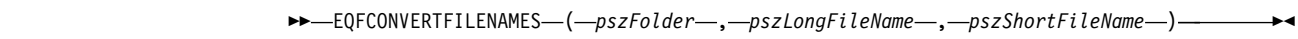
EQFCONVERTFILENAMES

Purpose

EQFCONVERTFILENAMES converts long file names into short file names, and vice versa.

If the long file name is an empty string, the long file name is created from the short file name, and vice versa. If the short file name meets the 8.3 DOS naming conventions, the long file name is returned as a null pointer.

Format



Parameters

pszFolder(PSTRING) **— input**
The name of the folder with path information, for example <*folder_drive*>:\otm\<*folder_name*>.f00. <*folder_name*> can be extracted from pSegTarget or pSegSource as defined in eqf_xstart.

pszLongFileName(PSTRING) **— input or output**
The long file name without path information. It is used to get the short file name. If pszLongFileName==NULL, *pszLongFileName* is output.

pszShortFileName(PSTRING) **— input or output**
The short file name (8.3 DOS naming convention) without path information. It is used to get the long file name. If pszShortFileName==NULL, *pszShortFileName* is output.

Return codes

A OpenTM2 return code as defined in the file OS2TOWIN.H. A return code of null indicates successful processing.

Remarks

If a long file name is to be created from a short file name and the result is an empty string for *pszLongFileName*, the short file name applies to the 8.3 naming conventions.

Notes

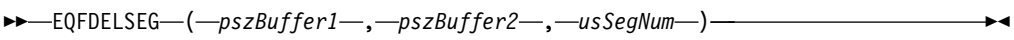
Either *pszLongFileName* or *pszShortFileName* must be an empty string. The non-empty string must be a valid file name, otherwise an error is recorded.

EQFDELSEG

Purpose

EQFDELSEG deletes the specified segment from the Translation Memory, together with its information.

Format



Parameters

pszBuffer1(PSTRING) **– input**
The buffer for the source segment to be deleted. It must have a length of EQF_BUFFERLEN. EQF_BUFFERLEN is defined in the file EQFTWBS.H.

pszBuffer2(PSTRING) **– input**
The buffer for the corresponding translation to be deleted. It must have a length of EQF_BUFFERLEN. EQF_BUFFERLEN is defined in the file EQFTWBS.H.

usSegNum(USHORT) **– input**
The segment number.

Return codes

EQFERR_SEG_NOT_FOUND
The specified segment was not found.

EQFERR_TM_ACCESS
The Translation Memory could not be accessed.

EQFERR_TM_CORRUPTED
The Translation Memory is corrupted.

EQF_OKAY
The request completed successfully.

EQFERR_INIT
The system must first be initialized.

Remarks

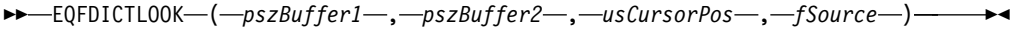
This call is useful if parts of combined segments are already translated. These parts are now meaningless and can therefore be deleted from the Translation Memory.

EQFDICTLOOK

Purpose

EQFDICTLOOK invokes the dictionary lookup dialog.

Format



Parameters

pszBuffer1(PSTRING) **– input**
The buffer for the active segment. It must have a length of EQF_BUFFERLEN. EQF_BUFFERLEN is defined in the file EQFTWBS.H.

pszBuffer2(PSTRING) **– input**
The buffer for the marked area. It must have a length of EQF_BUFFERLEN. EQF_BUFFERLEN is defined in the file EQFTWBS.H.

usCursorPos(USHORT) **– output**
The position of the input cursor.

fSource **– output**
Determines whether the term looked up is in the source or target language (not used in the current OpenTM2 version).

Return codes

EQFERR_DICTLOOK_NOT_FOUND
The dictionary lookup dialog could not be loaded.

EQF_OKAY
The dictionary term is selected and copied into the provided buffer.

EQFERR_INIT
The system must first be initialized.

EQFERR_DICT_LOOKUP_PENDING
The dictionary lookup request is pending.

EQFERR_NO_ENTRY_AVAIL
The dictionary entry is not available.

Remarks

EQFERR_DICT_LOOKUP_PENDING indicates that a dictionary lookup is active. After selecting an entry or leaving the dictionary lookup dialog, the return code is reset to either EQF_OKAY or EQF_NO_ENTRY_AVAIL.

From an editor's point of view, this call is handled in the same way as EQFGETDICT (see page "EQFGETDICT" on page 11).

EQFFILECONVERSIONEX

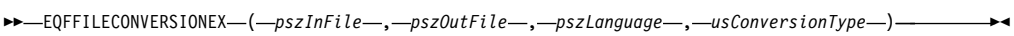
Purpose

EQFFILECONVERSIONEX is a helper function for user exits which require the files to be converted.

The new API function gives the possibility

- to convert an ASCII file into ANSI (EQF_ASCII2ANSI)
- to convert an ANSI file into ASCII (EQF_ANSI2ASCII)
- to convert an ASCII file into UTF8 (EQF_ASCII2UTF8)
- to convert an UTF8 file into ASCII (EQF_UTF82ASCII)
- to convert an ASCII file into UTF16 (EQF_ASCII2UTF16)
- to convert an UTF16 file into ASCII (EQF_UTF162ASCII)
- to convert an ANSI file into UTF8 (EQF_ANSI2UTF8)
- to convert an UTF8 file into ANSI (EQF_UTF82ANSI)
- to convert an ANSI file into UTF16 (EQF_ANSI2UTF16)
- to convert an UTF16 file into ANSI (EQF_UTF162ANSI)
- to convert an UTF8 file into UTF16 (EQF_UTF82UTF16)
- to convert an UTF16 file into UTF8 (EQF_UTF162UTF8)

Format



Parameters

pszInFile(PSZ) **– input**
the fully qualified filename of the input file. or as defined in .

pszOutFile(PSZ) **– input**
the fully qualified filename of the output file.

pszLanguage(PSZ) **– input**
the language of the file (e.g. it can be retrieved with EQFGETSOURCELANG/ EQFGETTARGETLANG).

usConversionType(USHORT) **– input**
identifier of type of conversion: ASCII2ANSI, ANSI2ASCII

- EQF_ASCII2ANSI
- EQF_ANSI2ASCII
- EQF_ASCII2UTF8
- EQF_UTF82ASCII
- EQF_ASCII2UTF16
- EQF_UTF162ASCII
- EQF_ANSI2UTF8
- EQF_UTF82ANSI
- EQF_ANSI2UTF16
- EQF_UTF162ANSI
- EQF_UTF82UTF16
- EQF_UTF162UTF8

usReturn(USHORT) **– output**

- EQFRC_OK successfully completed
- EQFS_FILE_OPEN_FAILED file cannot be opened
- ERROR_STORAGE allocation of memory failed
- ERROR_FILE_INVALID_DATA file contains data that cannot be converted
- EQFRS_INVALID_PARM in all other cases of error

Return codes

- EQFRC_OK successfully completed
- EQFS_FILE_OPEN_FAILED file cannot be opened
- ERROR_STORAGE allocation of memory failed
- ERROR_FILE_INVALID_DATA file contains data that cannot be converted
- EQFRS_INVALID_PARM in all other cases of error

Remarks

If the file pszOutFile exists already, it is overwritten.

The API EQFFILECONVERSION is not available any more in TM6.0.2. It has been replaced by the new API EQFFILECONVERSIONEX.

The pszInFile is converted according to the conversion type and written as the file pszOutFile. Output file and input file should be different files.

The input language is used to determine the ASCII and ANSI codepage for the conversion. Inside TM, exactly one ASCII /one ANSI codepage is attached to each possible language. The input language must be a valid TM source or target language.

If the language is NULL, the default target language of the system preferences is used for conversion.

If EQF_ASCII2ANSI is specified, it is assumed that the input file is in ASCII. If EQF_ANSI2ASCII is specified, it is assumed that the input file is in ANSI.

If EQF_UTF162ANSI or EQF_UTF162ASCII or EQF_UTF162UTF8 is specified, the input file is checked for the byte order mark. For UTF16 files, a byte-order-mark is required. If the input file does not contain such a mark, ERROR_FILE_INVALID_DATA is returned.

For UTF8 input files, a byte-order-mark is accepted, however it is not required. UTF8 output files are written without a byte-order-mark.

If the input file contains characters which are not valid in the codepage of the input language, the API EQFFILECONVERSIONEX may fail with the error return ERROR_FILE_INVALID_DATA.

EQFRS_INVALID_PARM is returned as error code if usConversionType is invalid.

Examples

Example

```
CHAR szInFile[145];
CHAR szOutFile[145];
CHAR szLanguage[20];
USHORT usRC = 0;
strcpy(szOutFile, "d:\temp\b.tst");
strcpy(szInFile,"d:\input\b.tst");
strcpy(szLanguage, "English(U.S)");

usRC = EQFFILECONVERSIONEX( szInFile, szOutFile, szLanguage, EQF_ASCII2ANSI );
```

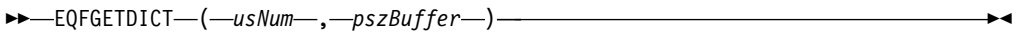
EQFGETDICT

Purpose

EQFGETDICT retrieves the selected dictionary word and copies it into the provided buffer.

EQF_UP or EQF_DOWN scrolls the contents of the “Dictionary” window in the selected direction, if possible. EQF_LOOKUP can be used to retrieve the selected dictionary lookup term. The appropriate return code is set if the dictionary lookup is pending or no term is selected. EQF_UP, EQF_DOWN, and EQF_LOOKUP are defined in the file EQFTWBS.H.

Format



Parameters

usNum(USHORT) **– input**
The number of the selected dictionary word (0...9, EQF_UP, EQF_DOWN, EQF_LOOKUP).

pszBuffer(PSTRING) **– output**
The buffer for the dictionary word. It must have a length of `EQF_BUFFERLEN`. `EQF_BUFFERLEN` is defined in the file `EQFTWBS.H`.

Return codes

EQFERR_ENTRY_NOT_AVAIL
The selected dictionary entry is not available.

EQF_OKAY
The selected dictionary term is available and copied into the provided buffer.

EQFERR_INIT
The system must first be initialized.

Remarks

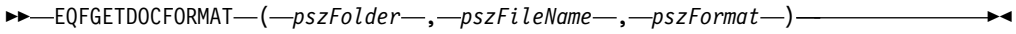
If the selected dictionary word is not available, a warning message is issued.

EQFGETDOCFORMAT

Purpose

EQFGETDOCFORMAT retrieves the format (markup language) of the specified document.

Format



Parameters

pszFolder(PSTRING) **– input**
The name of the folder with path information, for example `<folder_drive>:\otm\<folder_name>.f00`. `<folder_name>` can be extracted from `pSegTarget` or `pSegSource` as defined in `eqf_xstart`.

pszFileName(PSTRING) **– input**
The short file name (8.3 DOS naming convention) without path information.

pszFormat(PSTRING) **– output**
The format (markup language) of the specified document.

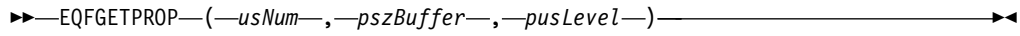
EQFGETPROP

Purpose

EQFGETPROP retrieves the selected proposal and copies it to the provided buffer.

EQF_UP or EQF_DOWN scrolls the contents of the "Translation Memory" window in the selected direction, if possible. EQF_UP and EQF_DOWN are defined in the file EQFTWBS.H.

Format



Parameters

usNum(USHORT) **– input**
The number of the selected proposal or match (0...9, EQF_UP, EQF_DOWN).

pszBuffer(PSTRING) **– output**
The buffer for the Translation Memory proposals. It must have a length of EQF_BUFFERLEN. EQF_BUFFERLEN is defined in the file EQFTWBS.H.

pusLevel(PUSHORT)
The pointer to the variable for the return match level.

Return codes

EQFERR_ENTRY_NOT_AVAIL
The selected proposal is not available.

EQF_OKAY
The selected proposal is available and copied into the provided buffer.

EQFERR_INIT
The system must first be initialized.

Remarks

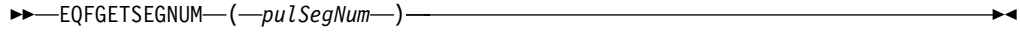
If the selected proposal is not available, a warning message is issued and the appropriate return code is set.

EQFGETSEGNUM

Purpose

EQFGETSEGNUM retrieves the segment number of the currently selected proposal (the segment that was used before by the *EQFGETPROP* call).

Format



Parameters

pulSegNum(PULONG) **– output**
The pointer to the ULONG variable receiving the segment number.

Return codes

One of the values listed in "Return codes" on page 3.

Remarks

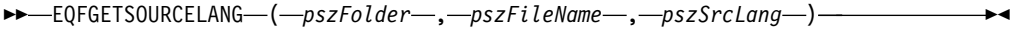
You can use the retrieved segment number, for example, as input parameter with the *EQFDELSEG* call.

EQFGETSOURCELANG

Purpose

EQFGETSOURCELANG retrieves the source language of the specified document.

Format



Parameters

pszFolder(PSTRING) **– input**
The name of the folder with path information, for example <*folder_drive*>:\otm\<*folder_name*>.f00. <*folder_name*> can be extracted from pSegTarget or pSegSource as defined in eqf_xstart.

pszFileName(PSTRING) **– input**
The short file name (8.3 DOS naming convention) without path information.

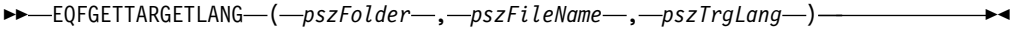
pszSrcLang(PSTRING) **– output**
The source language.

EQFGETTARGETLANG

Purpose

EQFGETTARGETLANG retrieves the target language of the specified document.

Format



Parameters

pszFolder(PSTRING) **– input**
The name of the folder with path information, for example <*folder_drive*>:\otm\<*folder_name*>.f00. <*folder_name*> can be extracted from pSegTarget or pSegSource as defined in eqf_xstart.

pszFileName(PSTRING) **– input or output**
The short file name (8.3 DOS naming convention) without path information.

pszSrcLang(PSTRING) **– output**
The target language.

EQFINIT

Purpose

EQFINIT initializes OpenTM2 for use by an editor. This means, it creates the "Dictionary" and "Translation Memory" windows, establishes the communication

links, attaches the Translation Memory and dictionaries, and allocates the internal structures required by OpenTM2.

Format

Parameters

pstEQFStruct(PSTEQFSTRUCT) **– input**
The number of sentences (0...9).

pszTranslationMemoryDatabases
The file name of the Translation Memory databases.

pszUserDictionaries
The name of the user-supplied dictionaries.

Return codes

EQFERR_TM_ACCESS
The Translation Memory could not be accessed.

EQFERR_TM_CORRUPTED
The Translation Memory is corrupted.

EQFERR_DICT_ACCESS
The dictionary or the dictionary lookup program could not be accessed.

EQF_OKAY
The request completed successfully.

EQFERR_SYSTEM
A system error occurred.

Remarks

The application must set the initial values for the position and size of the "Dictionary" and "Translation Memory" windows. If nothing is specified, the default values are used. If a problem occurs, a warning message is issued and the appropriate return code is set.

Notes

This call is implicitly issued by OpenTM2 and only listed for completeness reasons.

EQFQUERYEXITINFO

Purpose

The entry point *EQFQUERYEXITINFO*in QUERYEXIT_ADDFILES mode is called by OpenTM2 during folder export when a markup table having a user exit is added to the exported folder.

If the user exit requires other files beside the markup table (.TBL) file, the user exit DLL and the .markup table control file (.CHR) to be exported and imported with the folder it should place a list of these files in the supplied buffer area.

The list of files is a comma separated list of file names terminated by a null character (C string syntax).

The file names may not contain wildcard characters.

All files are specified with their relative path in the \EQF directory.

Files not located in the \EQF directory cannot be exported and imported using folder import.

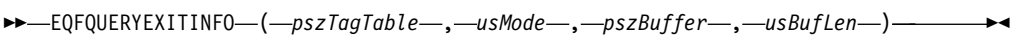
Example:

The file list "TABLE\ADDFILE.CHR,WIN\MYDLL.DLL,WIN\LOCALE\XYZ.CNV" will export the files \OTM\TABLE\ADDFILE.CHR, \OTM\WIN\MYDLL.DLL and the file \OTM\WIN\LOCALE\YXZ.CNV in the exported folder. OpenTM2 versions prior to TP603 will only import the files contained in the \OTM\TABLE directory, files in other directories will be ignored.

If the user exit places a list of additional files in the supplied buffer it should return a return code of zero all other values are assumed to be error codes.

In the future there will be other modes of the entry point EQFQUERYEXITINFO, so the requested mode should be checked by the user exit.

Format



Parameters

pszTagTable(PSZ) **– input**
The name of the active tag table; e.g. "IBMHTM32"

usMode(USHORT) **– input**
Mode of the function, currently on "QUERYEXIT_ADDFILES" is being used

pszBuffer(PSZ) **– input**
Points to a buffer which will receive the list of additional markup table files

*usBufLen(USHORT)***– input**
Length of the supplied buffer area in number of bytes

Return codes

USHORT (zero = function completed successfully)

EXAMPLE:

```
USHORT APIENTRY16 EQFQUERYEXITINFO
(
  PSZ pszTagTable,                    // name of the markup table, e.g. "IBMHTM32"
  USHORT usMode,                      // type of information being queried
  PSZ pszBuffer,                      // buffer area receiving the information
                                         returned by the exit
  USHORT usBufLen                     // length of buffer area
)
{
switch( usMode )
{
case QUERYEXIT_ADDFILES:
  strcpy( pszBuffer, "TABLE\MYINFO.CTL,WIN\MYDLL.DLL" );
  break;
default:
  usRC = 1;                      // mode is not supported by user exit
```

```
  } /* endswitch */
} /* end of function EQFQUERYEXITINFO */
In this sample the files "\OTM\TABLE\MYINFO.CTL" and "\OTM\WIN\MYDLL.DLL"
are exported within the exported folder package.
```

EQFSAVESEG

Purpose

EQFSAVESEG saves the passed segment information in the Translation Memory.

Format

```
►►—EQFSAVESEG—(—pszBuffer1—,—pszBuffer2—,—usSegNum—)———————————————————►◄
```

Parameters

pszBuffer1(PSTRING) **– input**
The buffer for the source segment. It must have a length of EQF_BUFFERLEN. EQF_BUFFERLEN is defined in the file EQFTWBS.H.

pszBuffer2(PSTRING) **– input**
The buffer for the translated segment. It must have a length of EQF_BUFFERLEN. EQF_BUFFERLEN is defined in the file EQFTWBS.H.

usSegNum(USHORT) **– input**
The segment number.

Return codes

EQFERR_DISK_FULL
OpenTM2 detected that the disk is full.

EQFERR_TM_NOT_ACTIVE
The Translation Memory is not active.

EQFERR_SEG_EMPTY
The passed segment was empty and therefore was not stored in the Translation Memory.

EQF_OKAY
The dictionary term is selected and copied into the provided buffer.

EQFERR_INIT
The system must first be initialized.

Remarks

The editor must ensure that only correct data is saved in the Translation Memory. This means that the application must first check the spelling of the data.

EQFSEGFILECONVERTASCII2UNICODE

Purpose

EQFSEGFILECONVERTASCII2UNICODE gives the possibility to convert the segmented ASCII file to UTF16-Unicode (EQFSEGFILECONVERTASCII2UNICODE).

EQFSegFileConvertASCII2Unicode are helper functions for user exits which require the segmented files to be in ASCII whereas OpenTM2 expects the segmented files to be saved in Unicode.

The pszInFile is converted from ASCII to Unicode and written as the file pszOutFile. If the file pszOutFile already exists, it is overwritten. Only files which are correctly segmented, can be converted with this API.

Format

```
►►—EQFSEGFILECONVERTASCII2UNICODE—(—pszInFile—,—pszOutFile—,—)———————►◄
```

Parameters

pszInFile(PSZ) **– input**
The fully qualified filename of a segmented file in ASCII format which should be converted .

pszOutFile(PSZ) **– input**
the fully qualified filename of the file to which pszInFile should be converted.

usReturn(USHORT) **– output**

Return codes

EQFRC_OK
successfully completed

ERROR_FILE_OPEN_FAILED
file read error

ERROR_STORAGE
allocation of memory failed.

ERROR_FILE_INVALID_DATA
segmentation of file is erraneous

EQFRS_INVALID_PARM
table cannot be accessed

Remarks

If the file pszOutFile exists already, it is overwritten.

EQFSEGFILECONVERTUNICODE2ASCII

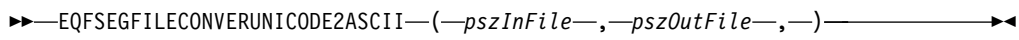
Purpose

EQFSEGFILECONVERTUNICODE2ASCII gives the possibility to convert the segmented UTF16 -Unicode file to ASCII (EQFSEGFILECONVERTUNICODE2ASCII)

EQFSegFileConvertUNICODE2ASCII are helper functions for user exits which require the segmented files to be in ASCII whereas OpenTM2 expects the segmented files to be saved in Unicode.

The pszInFile is converted from Unicode to ASCII and written as the file pszOutFile. If the file pszOutFile already exists, it is overwritten. Only files which are correctly segmented, can be converted with this API.

Format



Parameters

pszInFile(PSZ) **– input**
The fully qualified filename of a segmented file in UTF16 Unicode format which should be converted .

pszOutFile(PSZ) **– input**
the fully qualified filename of the file to which pszInFile should be converted.

usReturn(USHORT) **– output**

Return codes

EQFRC_OK
successfully completed

ERROR_FILE_OPEN_FAILED
file read error

ERROR_STORAGE
allocation of memory failed.

ERROR_FILE_INVALID_DATA
segmentation of file is erraneous

EQFRS_INVALID_PARM
table cannot be accessed

Remarks

If the file pszOutFile exists already, it is overwritten.

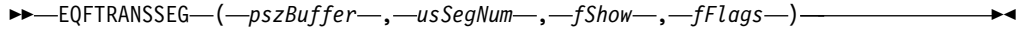
EQFTRANSSEG

Purpose

EQFTRANSSEG retrieves the information available for the current segment and puts it into the internal waiting list.

OpenTM2 handles the layout and scrolling of the “Dictionary” and “Translation Memory” windows and the selection of entries.

Format



Parameters

pszBuffer(PSTRING) **– input**
The buffer for the source segment. It must have a length of EQF_BUFFERLEN. EQF_BUFFERLEN is defined in the file EQFTWBS.H.

usSegNum(USHORT) **– input**
The segment number.

fShow(BOOL) **– input**
Determines whether the segment must immediately be displayed in the "Dictionary" or "Translation Memory" window:

TRUE Put the segment into the "Dictionary" or "Translation Memory" window.

FALSE
Use the segment information as sentence.

fFlags(FLAG) **– input**
Determines what is displayed:

EQF_NODICTWND
No "Dictionary" window is displayed.

EQF_NOPROPWND
No "Translation Memory" window is displayed.

EQF_NOAUTODICT
The automatic dictionary lookup is disabled.

Return codes

EQFERR_DISK_FULL
OpenTM2 detected that the disk is full.

EQFERR_TM_CORRUPTED
The Translation Memory is corrupted.

EQFERR_TM_ACCESS
The Translation Memory could not be accessed.

EQFERR_DICT_ACCESS
The dictionary or the dictionary lookup program could not be accessed.

EQF_OKAY
The request completed successfully.

EQFERR_INIT
The system must first be initialized.

Remarks

If *fShow* is set to FALSE, the success indicator is immediately set to TRUE. In addition, the sentence is treated as a sentence and processed in the background. Any error information produced during background processing is stored and displayed when this segment is displayed.

If *fShow* is set to TRUE, this call first checks if the segment information is already prepared and can be immediately retrieved. If this is not the case, it is processed in the foreground.

The EQF_NOAUTODICT flag is used to determine if the dynamic dictionary lookup, which consumes a lot of performance, should be skipped.

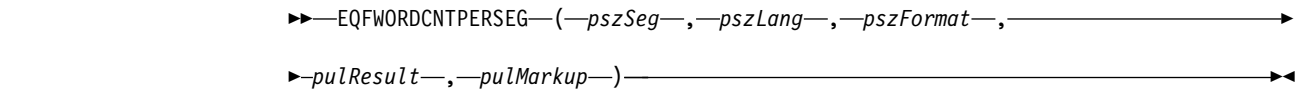
EQFWORDCNTPERSEG

Purpose

EQFWORDCNTPERSEG counts the number of words and markup tags in the specified segment using the specified language and markup. To count the number

of words in a document, the words must be counted segment by segment.

Format



Parameters

pszSeg(PSTRING) **— input**
The segment of which the number of words and markup tags must be counted.

pszLang(PSTRING) **— input**
The source or target language as provided by *EQFGETSOURCELANG* (see page "EQFGETSOURCELANG" on page 14) or *EQFGETTARGETLANG* (see page "EQFGETTARGETLANG" on page 14).

pszFormat(PSTRING) **— input**
The format of the document as provided by *EQFGETDOCFORMAT* (see page "EQFGETDOCFORMAT" on page 12).

pulResult(PULONG) **— output**
The result of word counting.

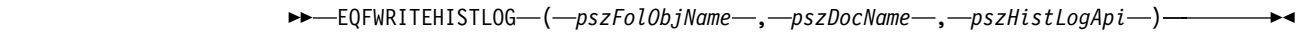
pulMarkUp(PULONG) **— output**
The result of markup-tag counting.

EQFWRITEHISTLOG

Purpose

EQFWRITEHISTLOG writes the word-counting information to the history log file of the specified folder. The word-counting information for the entire document is needed.

Format



Parameters

pszFolder(PSTRING) **— input**
The name of the folder with path information, for example <*folder_drive*>:\otm\<*folder_name*>.f00. <*folder_name*> can be extracted from pSegTarget or pSegSource as defined in eqf_xstart.

pszFileName(PSTRING) **— input**
The short file name (8.3 DOS naming convention) without path information.

pszHistLogApi(PAPIDOCSAVEHIST) **— input**
The structure of the history log file:

```
typedef struct _APISumPerClass
   {
      USHORT  usNumSegs;                // number of segments in this class
      ULONG   ulSrcWords;               // sum of all source words
      ULONG   ulTgtWords;               // sum of all target words
   }  APISUMPERCLASS, *PAPISUMPERCLASS;
```

```
typedef struct _APICriteriaSum
{
   APISUMPERCLASS  SimpleSum;         // number of segments in this class
   APISUMPERCLASS  MediumSum;         // number of segments in this class
   APISUMPERCLASS  ComplexSum;        // number of segments in this class
}  APICRITERIASUM, *PAPICRITERIASUM;
  typedef struct _APIDocSaveHist
{
   APICRITERIASUM  EditAutoSubst;     // sums for segments translated by
                                      // Edit Auto
   APICRITERIASUM  ExactExist;        // sums for segments with exact
                                      // proposals
   APICRITERIASUM  ExactUsed;         // sums for segments with exact
                                      // proposals used by translator
   APICRITERIASUM  FuzzyExist;        // sums for segments with fuzzy
                                      // proposals
   APICRITERIASUM  FuzzyUsed;         // sums for segments with fuzzy
                                      // proposals used by translator
   APICRITERIASUM  FuzzyExist_1;      // sums for segments with fuzzy
                                      // proposals
   APICRITERIASUM  FuzzyUsed_1;       // sums for segments with fuzzy
                                      // proposals used by translator
   APICRITERIASUM  FuzzyExist_2;      // sums for segments with fuzzy
                                      // proposals
   APICRITERIASUM  FuzzyUsed_2;       // sums for segments with fuzzy
                                      // proposals used by translator
   APICRITERIASUM  FuzzyExist_3;      // sums for segments with fuzzy
                                      // proposals
   APICRITERIASUM  FuzzyUsed_3;       // sums for segments with fuzzy
                                      // proposals used by translator
   APICRITERIASUM  MachExist;         // sums for segments with machine
                                      // proposals
   APICRITERIASUM  MachUsed;          // sums for segments with machine
                                      // proposals used by translator
   APICRITERIASUM  NoneExist;         // sums for segments with no proposal
   APICRITERIASUM  NotXlated;         // sums for TOBE, ATTR, CURRENT
}  APIDOCSAVEHIST, *PAPIDOCSAVEHIST;
```

The various classes are described in .

For this structure the thresholds of the standard editor were used, namely:

```
// defines for fuzzy classes
    #define FUZZY_THRESHOLD_1      0.7   // Threshold for different fuzzy
                                         // classes
    #define FUZZY_THRESHOLD_2      0.9   // Threshold for different fuzzy
                                         // classes


    // defines for the classes: simple sentences, medium sentences,
    // complex sentences
    #define SIMPLE_SENT_BOUND       5
    #define MEDIUM_SENT_BOUND      15
```

Chapter 2. The general application programming interface

OpenTM2 provides an application programming interface (API) that enables an application to directly communicate with the OpenTM2 functions without OpenTM2 running. However, it is required that OpenTM2 is installed, all OpenTM2 drives are configured, and shared resources are connected. The application can communicate with all functions currently covered by the dynamic data exchange (DDE) interface (that is, the OTMBATCH command area). In addition, it can use all functions concerning dictionary and Translation Memory handling, namely retrieving dictionary and Translation Memory proposals and updating dictionaries and Translation Memory databases.

Overview and terminology

Each OpenTM2 function includes a generic data block, which is encapsulated in the session handle. This session handle is created by the *EqfStartSession* call (see page "EqfStartSession" on page 141). It ensures that several OpenTM2 functions can run concurrently. The functions are delivered as a library and a dynamic-link library (DLL) following the standard PASCAL calling conventions. The include file `EQFFUNC.H` contains the prototypes of all available functions.

The long-running tasks, such as the export or the organization of a Translation Memory, are split into small units of work. The return code indicates if the task has completed successfully or if data is pending. The calling application must allocate the memory and free it when no longer used. In this way, the interface is independent of any compiler or runtime libraries used.

The term "folder" in the following descriptions also implies subfolders. Whenever a function requires the specification of a folder as a parameter, for example `"folder_main"`, you can also specify a subfolder, for example `"folder_2001\\ folder_sub1"`. You can even expand subfolder specifications, up to the limits of the operating system, for example `"folder_2001\\folder_sub1\\sub_sub\\ sub_sub_sub\\..."`.

Data types

The non-DDE interface for OpenTM2 functions uses the following data types for parameters and return codes:

HSESSION	The session handle that is created by *EqfStartSession*. It must be specified in all other functions of the non-DDE interface.
PHSESSION	The pointer to a HSESSION variable.
LONG	A long (32-bit) signed integer. In the non-DDE interface, this data type is used for option flags. Use 0L if no options are to be specified.
PSZ	The pointer to a zero-terminated string (C-language string). Use `NULL` if no parameter is specified.
USHORT	A short (16-bit) unsigned integer value. This data type is used for return codes.
PUSHORT	The pointer to a variable of type USHORT.

FORMLIST	A structure consisting of two length fields and a memory block. The byte ch indicates the start of the memory block:
	`typedef struct`
	`{`
	`  ULONG ulAllocated;`
	`  ULONG ulUsed;`
	`  BYTE ch;`
	`} FORMLIST, *PFORMLIST;`

Sample code

The following sample is written in the C programming language. It shows how to **create a new folder** using the API-call "EqfCreateFolder", how to **import documents** using the API-call "EqfImportDoc", and how to **analyze documents** using the API-call "EqfAnalyzeDoc".

```
USHORT usRC = 0;
HSESSION hSession = 0L;

// start the Eqf session
usRC = EqfStartSession( &hSession );

// create the folder SAMPLE1
if ( !usRC )
{
  usRC = EqfCreateFolder( hSession, "SAMPLE1", NULL, '\0', "MEM1",
                          "EQFASCII", NULL, NULL, "English(U.S.)",
                          "German(national)" );
}

// import the documents TEST1.DOC and TEXT2.DOC into folder SAMPLE1
if ( !usRC )
{
  do
  {
      usRC = EqfImportDoc( hSession, "SAMPLE1", NULL,
                           "C:\\TEXT1.DOC,C:\\TEXT2.DOC",
                           NULL, NULL, NULL, NULL, NULL, NULL, 0L );
  } while( usRC == CONTINUE_RC );
}

// Analyze all documents of folder SAMPLE1
if ( !usRC )
{
  do
  {
      usRC = EqfAnalyzeDoc( hSession, "SAMPLE1", NULL, NULL, 0L );
  } while( usRC == CONTINUE_RC );
}

// end the Eqf session
if ( hSession != 0L )
{
  EqfEndSession( hSession );
}
```

Calling interface reference

The following sections describe the individual calls provided by OpenTM2.

The following calls are available:

API call name	Described on page
EqfAddCTIDList	"EqfAddCTIDList" on page 27
EqfAddMatchSegID	"EqfAddMatchSegID" on page 28
EqfAnalyzeDoc	"EqfAnalyzeDoc" on page 29
EqfAnalyzeDocEx	"EqfAnalyzeDocEx" on page 31
EqfArchiveTM	"EqfArchiveTM" on page 35
EqfBuildSegDocName	"EqfBuildSegDocName" on page 36
EqfChangeFolProps	"EqfChangeFolProps" on page 37
EqfChangeFolPropsEx	"EqfChangeFolPropsEx" on page 39
EqfChangeMFlag	"EqfChangeMFlag" on page 41
EqfCleanMemory	"EqfCleanMemory" on page 42
EqfClearMTFlag	"EqfClearMTFlag" on page 43
EqfCloseMem	"EqfCloseMem" on page 44
EqfCountWords	"EqfCountWords" on page 46
EqfCountWordsInString	"EqfCountWordsInString" on page 48
EqfCreateCntReport	"EqfCreateCntReport" on page 49
EqfCreateCntReportEx	"EqfCreateCntReportEx" on page 54
EqfCreateCountReport	"EqfCreateCountReport" on page 58
EqfCreateCountReportEx	"EqfCreateCountReportEx" on page 60
EqfCreateControlledFolder	"EqfCreateControlledFolder" on page 62
EqfCreateFolder	"EqfCreateFolder" on page 64
EqfCreateITM	"EqfCreateITM" on page 66
EqfCreateMarkup	"EqfCreateMarkup" on page 69
EqfCreateMem	"EqfCreateMem" on page 70
EqfCreateSubFolder	"EqfCreateSubFolder" on page 71
EqfDeleteDict	"EqfDeleteDict" on page 72
EqfDeleteDoc	"EqfDeleteDoc" on page 73
EqfDeleteFolder	"EqfDeleteFolder" on page 74
EqfDeleteMem	"EqfDeleteMem" on page 75
EqfDeleteMTLog	"EqfDeleteMTLog" on page 76
EqfDictionaryExists	"EqfDictionaryExists" on page 76
EqfDocumentExists	"EqfDocumentExists" on page 77
EqfEndSession	
EqfExportDict	"EqfExportDict" on page 78
EqfExportDoc	"EqfExportDoc" on page 79
EqfExportFolder	"EqfExportFolder" on page 81
EqfExportFolderFP	"EqfExportFolderFP" on page 83

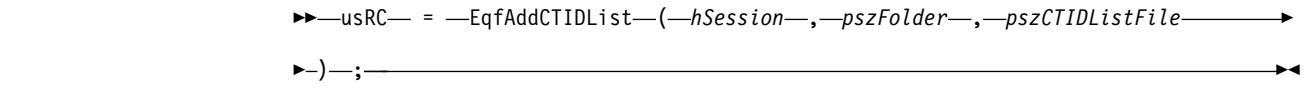
API call name	Described on page
EqfExportFolderFPas	“EqfExportFolderFPas” on page 85
EqfExportMem	“EqfExportMem” on page 86
EqfExportMemInInternalFormat	“EqfExportMemInInternalFormat” on page 88
EqfExportSegs	“EqfExportSegs” on page 89
EqfFilterNoMatchFile	“EqfFilterNoMatchFile” on page 90
EqfFolderExists	“EqfFolderExists” on page 92
EqfFreeSegFile	“EqfFreeSegFile” on page 92
EqfGetFolderProp	“EqfGetFolderProp” on page 94
EqfGetFolderPropEx	“EqfGetFolderPropEx” on page 95
EqfGetLastError	“EqfGetLastError” on page 96
EqfGetMatchLevel	“EqfGetMatchLevel” on page 97
EqfGetProgress	“EqfGetProgress” on page 99
EqfGetSegNum	“EqfGetSegNum” on page 100
EqfGetSegW	“EqfGetSegW” on page 101
EqfGetSegmentNumber	“EqfGetSegmentNumber” on page 102
EqfGetShortName	“EqfGetShortName” on page 103
EqfGetSourceLine	“EqfGetSourceLine” on page 104
EqfGetSysLanguage	“EqfGetSysLanguage” on page 105
EqfGetVersion	“EqfGetVersion” on page 106
EqfGetVersionEx	“EqfGetVersionEx” on page 107
EqfImportDoc	“EqfImportDoc” on page 107
EqfImportDict	“EqfImportDict” on page 109
EqfImportFolder	“EqfImportFolder” on page 111
EqfImportFolderAs	“EqfImportFolderAs” on page 114
EqfImportFolderFP	“EqfImportFolderFP” on page 112
EqfImportMem	“EqfImportMem” on page 115
EqfImportMemEx	“EqfImportMemEx” on page 116
EqfImportMemInInternalFormat	“EqfImportMemInInternalFormat” on page 119
EqfListMem	“EqfListMem” on page 120
EqfLoadSegFile	“EqfLoadSegFile” on page 121
EqfMemoryExists	“EqfMemoryExists” on page 122
EqfOpenDoc	“EqfOpenDoc” on page 123
EqfOpenDocByTrack	“EqfOpenDocByTrack” on page 124
EqfOpenDocEx	“EqfOpenDocEx” on page 125
EqfOpenMem	“EqfOpenMem” on page 126
EqfOrganizeMem	“EqfOrganizeMem” on page 127
EqfProcessNomatch	“EqfProcessNomatch” on page 128
EqfProcessNomatchEx	“EqfProcessNomatchEx” on page 130
EqfQueryMem	“EqfQueryMem” on page 132
EqfReduceToStemForm	“EqfReduceToStemForm” on page 134

EqfAddCTIDList

Purpose

EqfAddCTIDList associates a global memory filter file with a OpenTM2 folder.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolder	The name of the folder.
PSZ	pszCTIDListFile	The fully qualified file name of the global memory option file

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The analysis has not completed yet. Call *EqfAnalyzeDoc* again.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );
  // asociate the global memory option file C:\GlobMem\CFM\GlobMemOptions.xml with
     folder ATestFolder
  if ( !usRC ) {
```

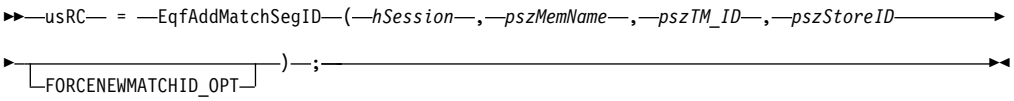
```
    usRC = EqfAddCTIDList( hSession, "ATestFolder", "C:\GlobMem\CFM\GlobMemOptions.xml" );
  } /* endif */
  // terminate the session    EqfEndSession( hSession );
}
```

EqfAddMatchSegID

Purpose

EqfAddMatchSegID adds match segment IDs to all entries of a translation memory.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemName	The name of an existing OpenTM2 translation memory.
PSZ	pszTM_ID	Identifier for the translation memory within the StoreID.
PSZ	pszStoreID	Identifier of the origin of the translation memory.
LONG	lOptions	FORCENEWMATCHID_OPT

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  // Add match segment IDs to all entries of translation memory "MEMDB1"
  // using the provided TM_ID "ACP005AV2" and the StoreID "TMB"
  if ( !usRC )
  {
  usRC = EqfAddMatchSegID ( hSession, "MEMDB1", "ACP005AV2", "TMB",
    FORCENEWMATCHID_OPT );
  } /* endif */
```

```
  // terminate the session
  EqfEndSession( hSession );
  }
}
```

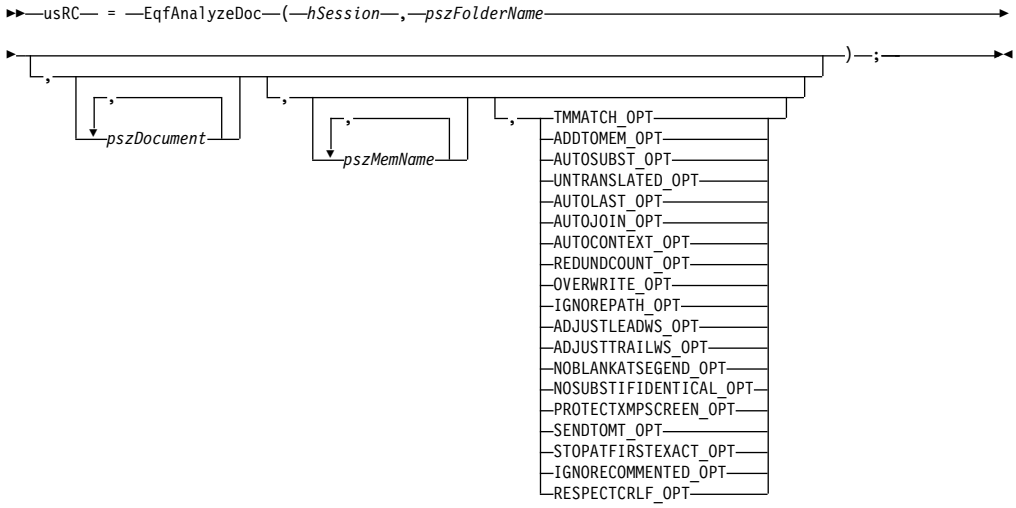
EqfAnalyzeDoc

Purpose

EqfAnalyzeDoc analyzes one or more documents. If no documents are specified, the function analyzes all documents in the selected folder.

This function performs the analysis in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents.
PSZ	pszDocument	The name of one or more documents. If you want to analyze all documents in the folder, specify NULL or an empty list.
PSZ	pszMemName	The name of the Translation Memories to be used as search memories.

Type	Parameter	Description
LONG	lOptions	The options to be used for the analysis: TMMATCH_OPT ADDTOMEM_OPT AUTOSUBST_OPT UNTRANSLATED_OPT AUTOLAST_OPT AUTOJOIN_OPT AUTOCONTEXT_OPT REDUNDCOUNT_OPT OVERWRITE_OPT IGNOREPATH_OPT ADJUSTLEADWS_OPT ADJUSTTRAILWS_OPT NOBLANKATSEGEND_OPT NOSUBSTIFIDENTICAL_OPT PROTECTXMPSCREEN_OPT SENDTOMT_OPT RESPECTCRLF_OPT STOPATFIRSTEXACT_OPT IGNORECOMMENTED_OPT These options correspond to those on the "Analyze Documents" window. OVERWRITE_OPT must be specified if the translation of the documents has already started. You can combine the constants using OR.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The analysis has not completed yet. Call *EqfAnalyzeDoc* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Analyze all documents of folder SAMPLE1 and
  // substitute exact matches automatically
  if ( !usRC )
```

```
  {
    do
    {
      usRC = EqfAnalyzeDoc( hSession, "SAMPLE1", NULL, ("Mem1", "Mem2"),
                            AUTOSUBST_OPT | OVERWRITE_OPT );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

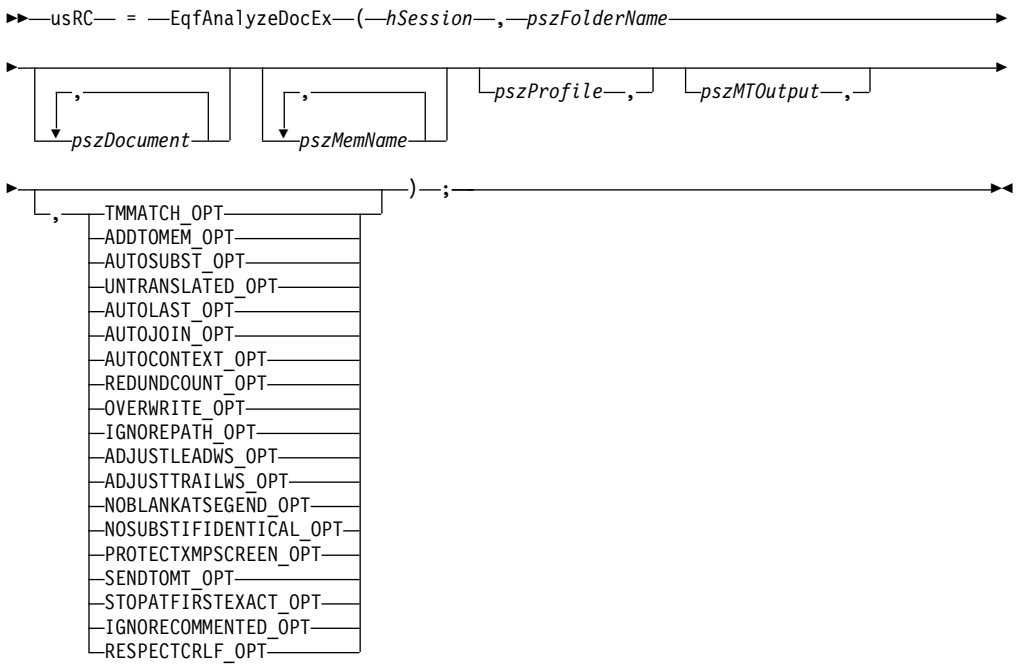
EqfAnalyzeDocEx

Purpose

EqfAnalyzeDocEx analyzes one or more documents. If no documents are specified, the function analyzes all documents in the selected folder.

This function performs the analysis in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents.
PSZ	pszDocument	The name of one or more documents. If you want to analyze all documents in the folder, specify NULL or an empty list.

Type	Parameter	Description
PSZ	pszMemName	The name of one or more Translation Memories to be used as search memories. Use a comma separated list if more than one memory is specified. Specify `NULL` if no search memory is to be used.
PSZ	pszProfile	The name of the analysis profile. Specify `NULL` if no analysis profile is to be used.
PSZ	pszMTOutput	This parameter can be used to control which MT-output files are to be created, and which segments are written to these output files. Specify `NULL` if no MT-output is to be created.
LONG	lOptions	The options to be used for the analysis: TMMATCH_OPT ADDTOMEM_OPT AUTOSUBST_OPT UNTRANSLATED_OPT AUTOLAST_OPT AUTOJOIN_OPT AUTOCONTEXT_OPT REDUNDCOUNT_OPT OVERWRITE_OPT IGNOREPATH_OPT ADJUSTLEADWS_OPT ADJUSTTRAILWS_OPT NOBLANKATSEGEND_OPT NOSUBSTIFIDENTICAL_OPT PROTECTXMPSCREEN_OPT SENDTOMT_OPT RESPECTCRLF_OPT STOPATFIRSTEXACT_OPT IGNORECOMMENTED_OPT These options correspond to those in the "Analyze Documents" window. OVERWRITE_OPT must be specified if the translation of the documents has already started. You can combine the constants using OR.

pszMTOutput option description

The option **SENDTOMT_OPT** is used to control the MT output creation.

When the option **SENDTOMT_OPT** has been specified, the parameter **pszMTOutput** can be used to control which MT output files are created, and which segments are written to these output files. When the parameter is not used, the EQFNFLUENT.TRG file controls the MT output files being created.

The **pszMTOutput** parameter contains a comma separated list of MT output files to be created. For each output file additional options can be specified.

The MT output files and the options are specified following the following scheme:

```
Outputfile-1(options for output file1),outputfile-2(options for output file2),...,
outputfile-n(options for output file n)
```

Keywords for pszMTOutput

Output Type	Description
NOMATCH	Creates a NOMATCH output file which contains all translatable segments for which there is no 100% proposal available (100% proposals include exact matches, Hamster matches, and Machine Translation matches).
ALLSEGS	Creates an ALLSEGS output file containing all translatable segments regardless of the available proposals for the segment.
ALLWMATCH	Creates an ALLWMATCH output file containing all translatable segments regardless of the available proposals for the segment, and adds the list of available proposals.
ALLWMATCHSOURCE	Creates an ALLWMATCHSOURCE output file containing all translatable segments regardless of the available proposals for the segment, adds the list of available proposals, as well as the corresponding source segment.
NOPROPOSAL	Creates a NOPROPOSAL output file which contains all translatable segments for which there is no 100% proposal available, and no fuzzy proposal with a fuzziness of 50% or better (100% proposals include exact matches, Hamster matches (= global memory matches), and Machine Translation matches).
XLIFF	Creates a XLIFF output file which contains ALL source-segments, along with all proposals.

Parameters for pszMTOutput

Type	Values
Output format	The options to be used for the analysis: • **EXP** (default) - Create a file in the EXP format. • **XML** - Create a file in the nFluent XML format. • **TMX** - Create a file in the TMX format.
Handling of duplicates	The options to be used for the analysis: • **DUPLICATES** (default) - Do not filter duplicate segments. • **NODUPLICATES** - Suppress duplicate segments (i.e. add only one occurrence of each segment).
Handling of Hamster matches	The options to be used for the analysis: • **NOHAMSTER** (default) - suppress segments having a Hamster proposal. • **HAMSTER** - add segments having a Hamster proposal to the output file.
Handling of MT matches	The options to be used for the analysis: • **NOMACHINEMATCH** (default) - suppress segments having a MT match. • **MACHINEMATCH** - include segments having a machine match proposal.

Type	Values
Handling of fuzzy matches	The options to be used for the analysis: **NOFUZZYABOVE=dd** - suppress segments fuzzy proposals which a fuzziness above the specified value "dd" (e.g. to suppress all segments with a fuzziness above 33% specify NOFUZZYABOVE=33).
Include wordcount information	The options to be used for the analysis: • **NOWORDCOUNT** (default) - do not add word count information to the output. • **WORDCOUNT** - add word count information to the output.

Samples

pszMTOutput parameter to create a NOMATCH file in EXP format which does not contain segments with exact matches, Hamster matches, MT matches and fuzzy matches with a fuzziness of 20% or better and which does include word count information and which does not contain duplicates:

```
NOPROPOSAL(EXP,NODUPLICATES,NOHAMSTER,NOMACHINEMATCH,NOFUZZYABOVE=20,WORDCOUNT)
```

As some of the options are active by default the following string will achieve the same result:

```
NOPROPOSAL(NODUPLICATES,NOFUZZYABOVE=20,WORDCOUNT)
```

The order in which the keywords are specified does not matter.

Using this enhanced API call, the MT output files can be created as needed and with the required granularity.

Adding word count information to the MT output file is already implemented, and can be triggered by specifying the keyword INCLUDEWORDCOUNT in the EQFNFLIUENT.TRG trigger file (located in \OTM\PROPERTY\).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The analysis has not completed yet. Call *EqfAnalyzeDocEx* again.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Analyze all documents of folder SAMPLE1 and
  // substitute exact matches automatically, use analysis profile Profile1
```

```
  if ( !usRC )
  {
    do
    {
      usRC = EqfAnalyzeDocEX( hSession, "SAMPLE1", NULL, ("Mem1", "Mem2"), "Profile1",
                              NULL, AUTOSUBST_OPT | OVERWRITE_OPT );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

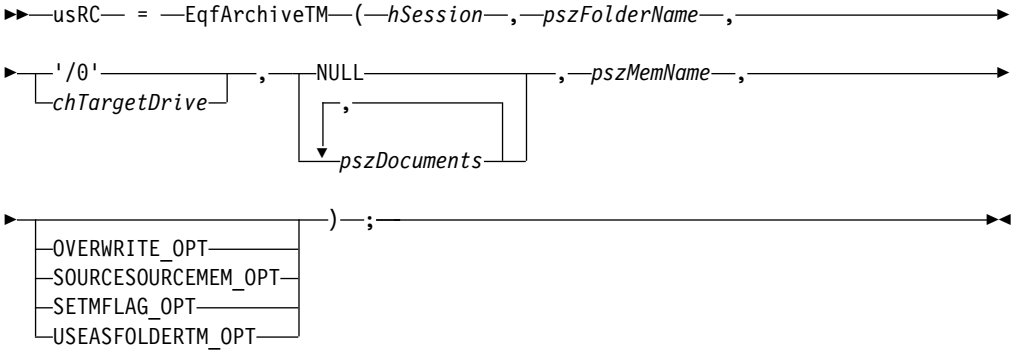
EqfArchiveTM

Purpose

EqfArchiveTM builds an Archive Translation Memory from an existing Translation Memory. At least one segment of at least one document you want to archive must have been translated (when SOURCESOURCEMEM_OPT option is not specified).

The SOURCESOURCEMEM_OPT option can be used to create a source-source Translation Memory. If the option is specified all translatable segments of the document are written to the specified Translation Memory. Without the option only segments already translated are processed.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
CHAR	chTargetDrive	The target drive where the folder is located, or `'/0'` if it is the drive where the `eqf` directory is located.
PSZ	pszDocuments	List of the documents that are searched for translated segments to be included in the Translation Memory, or `NULL` to search in all documents of the folder.
PSZ	pszMemName	The name of an existing Translation Memory.

Type	Parameter	Description
LONG	lOptions	The options used for the Archive Translation Memory: • OVERWRITE_OPT (overwrites the contents of an existing Translation Memory) • USEASFOLDERTM_OPT (uses the Translation Memory as the new folder Translation Memory) • SOURCESOURCEMEM_OPT (creates a source-source Translation Memory containing all translatable segments of the document) • SETMFLAG_OPT (sets the machine translation flag of the segments written to the Translation Memory)

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The Archive Translation Memory has not completed yet. Call *EqfArchiveTM* again.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

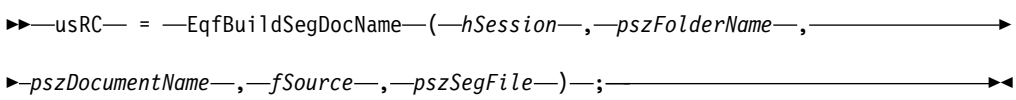
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  // Build Archive Translation Memory "MEM1" for the folder
  // "TEST" (including document "test.txt")
  if ( !usRC )
  {
    do
    {
          usRC = EqfArchiveTM(hSession, "TEST",'i',
                                        "test.txt",
                                        "MEM1",
                                         OVERWRITE_OPT|USEASFOLDERTM_OPT);
     } while ( usRC == CONTINUE_RC );
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfBuildSegDocName

Purpose

Builds the fully qualified file name of a segmented document within a OpenTM2 folder.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	Long name of the folder
PSZ	pszDocumentName	Long document name
USHORT	fSource	Flag selection source or target document • 0 = build segmented source file name • 1 = build segmented target file name
PSZ	pszSegFile	Points to a buffer receiving the fully qualified document file name, must have a width of at least 60 characters

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use EqfGetLastError to retrieve the complete error information.

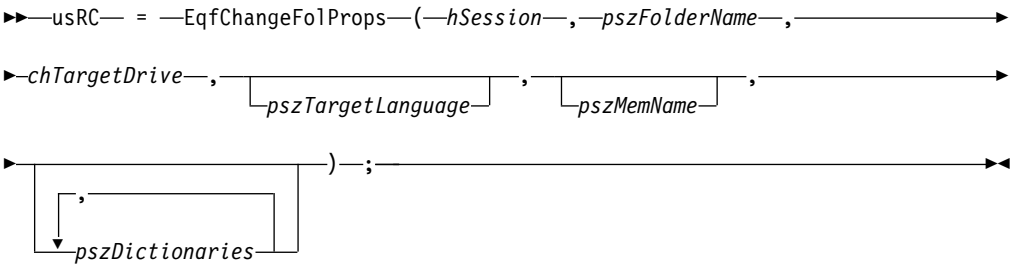
```
{
  USHORT  usRC = 0;
  CHAR   szFileName [60];
 HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  if ( !usRC )
  {
    usRC = EqfBuildSegDocName( hSession, "SAMPLE1", "Document1", 1, szFileName );
  } // endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfChangeFolProps

Purpose

EqfChangeFolProps lets you change the following folder properties: the target language, the folder Translation Memory, and the dictionaries.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
CHAR	chTargetDrive	The target drive where the folder is located, if it is not the drive where the `eqf` directory is located. If you do not specify a drive, specify `'/0'`.
PSZ	pszTargetLanguage	The target language for the documents in this folder, or `NULL` if the target language should not be changed. Specify the language exactly as it appears in the "Language List" window, for example `English(U.S.)`. The target language must be different from the source language.
PSZ	pszMemName	The name of the Translation Memory, or `NULL` if the Translation Memory should not be changed.
PSZ	pszDictionaries	The list of dictionaries to be used during translation. You can specify up to 10 dictionaries. If the dictionaries should not be changed, specify `NULL`.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  // Change the properties (target language, Memory, Dictionaries)
```

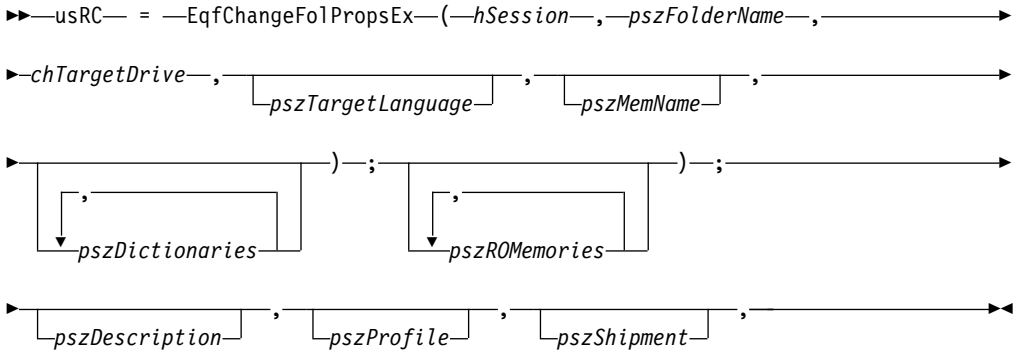
```
  // of the folder named "test"
  if ( !usRC )
  {
    usRC = EqfChangeFolProps(hSession, "test", 'e',
                             "English(U.S)",
                             "MEM1", "DICT1,DICT2");
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfChangeFolPropsEx

Purpose

EqfChangeFolPropsEx lets you change the following folder properties: the target language, the folder translation memory, the dictionaries, the search translation memory databases, the folder descriptiption, the analysis profile name and the shipment number.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
CHAR	chTargetDrive	The target drive where the folder is located, if it is not the drive where the `eqf` directory is located. If you do not specify a drive, specify `'/0'`.
PSZ	pszTargetLanguage	The target language for the documents in this folder, or `NULL` if the target language should not be changed. Specify the language exactly as it appears in the "Language List" window, for example `English(U.S.)`. The target language must be different from the source language.
PSZ	pszMemName	The name of the translation memory, or `NULL` if the translation memory should not be changed.

Type	Parameter	Description
PSZ	pszDictionaries	The list of dictionaries to be used during translation. You can specify up to 10 dictionaries. If the dictionaries should not be changed, specify NULL.
PSZ	pszROMemories	The list of search translation memory databases to be used during analysis and translation. You can specify up to 10 translation memory databases. If the search translation memory databases should not be changed, specify NULL. When you prefix the list of memories with a plus sign, the specified translation memories are added to the existing list of folder search memories instead of replacing them.
PSZ	pszDescription	The folder description or NULL when folder description should not be changed.
PSZ	pszProfile	The folder analysis profile, or NULL when no profile name is used.
PSZ	pszShipment	The shipment number, or NULL when no shipment number is used.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  // Change the properties (target language, Memory, Dictionaries)
  // of the folder named "test"
  if ( !usRC )
  {
    usRC = EqfChangeFolProps(hSession, "test", 'e',
                             "English(U.S)",
                             "MEM1,MEM2", "DICT1,DICT2");
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

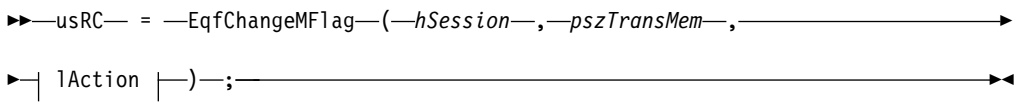
In this sample some properties of the folder "test" are changed: the target language is changed to "English(U.S.)", the memories "Mem1" and "Mem2" are added to the list of folder search memories and the dictionaries "DICT!" and "DICT2" are used as folder dictionaries.

EqfChangeMFlag

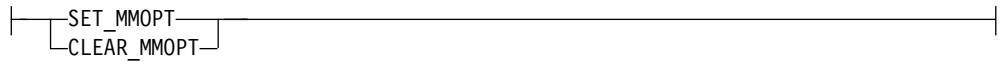
Purpose

Segments that were translated by machine are prefixed with an [m]. OpenTM2 provides a command to have these m prefixes removed from machine-translated segments in a Translation Memory. Alternatively, this function lets you add m flags to segments that did not have such a flag before.

Format



lAction:



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszTransMem	The name of the Translation Memory that you want to work with.
LONG	lAction	Specifies whether you want to remove (CLEAR_MMOPT) or set (SET_MMOPT) the m flags in the specified Translation Memory.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  // Remove m flags in Translation Memory TestTM.
  if ( !usRC )
  {
    usRC = EqfCreateITM(hSession, "TestTM", CLEAR_MMOPT);
   } //endif
```

```
  // terminate the session
  EqfEndSession( hSession );
}
```

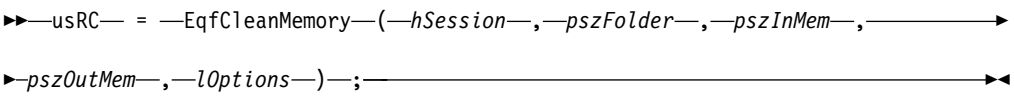
EqfCleanMemory

Purpose

The API call *EqfCleanMemory* removes all segments which are not relevant for a given translation package from an external memory. The “cleaned” memory can be created in internal or external format.

This function performs the cleanup in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolder	The name of a OpenTM2 folder (already imported into TM and the documents have to be analyzed).
PSZ	pszInMem	The fully qualified file name of the input memory in Ansi or UTF-16 encoding.
PSZ	pszOutMem	The name of an new internal memory or the fully qualified name of an external.

Type	Parameter	Description
LONG	lOptions	The option to be used for the cleanup of a memory: CLEANMEM_INTERNAL_MEMORY_OPT to create an internal memory CLEANMEM_EXTERNAL_MEMORY_OPT to create an external memory (default) OVERWRITE_OPT to overwrite any existing output memory CLEANMEM_COMPLETE_IN_ONE_CALL_OPT If set the API call does not return after each processing step but stays in the API call until the function has been completed CLEANMEM_BESTMATCH_OPT if set only the best match is written to the output memory, if not set the best three matches are written to the output memory CLEANMEM_MERGE_OPT when specified the cleaned memory matches are merged into an existing memory rather than creating a new one CLEANMEM_KEEP_DUPS_OPT when specified duplicate exact matches are left in the memory (without this option only the first exact match is left in the memory). Fuzzy matches are left in the memory as long there is no exact match for the same segment (withhout this option only the best fuzzy match is left in the memory)

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The memory cleanup has not been completed yet. Call again.
other	Error code (EQF message number). Use “EqfGetLastError” on page 96 to retrieve the complete error information.

Code sample

```
HSESSION hSession;
USHORT usRC;

usRC = EqfStartSession( &hSession );
usRC = EqfCleanMemory( "TestFolder",
"C:\EXPMEMORY\SAMPLE2.EXP",
"C:\\EXPMEMORY\SAMPLEOUT.EXP",
CLEANMEM_EXTERNAL_MEMORY_OPT | CLEANMEM_COMPLETE_IN_ONE_CALL_OPT | OVERWRITE_OPT );

usRC = EqfEndSession( hSession );
```

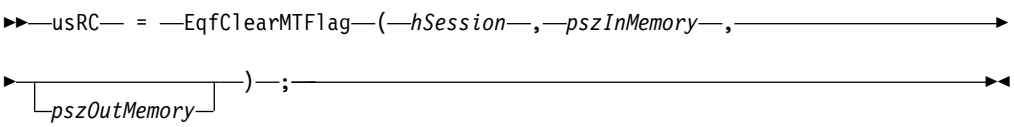
EqfClearMTFlag

Purpose

The API call *EqfClearMTFlag* clears the MT-flag (machine translation flag) of an external translation memory in the *.EXP format.

This API function processes a memory in the *.EXP format (encoding UTF-16, ANSI or ASCII), and clears any machine translation flag (MT flags) of the memory proposals. If an output memory is specified, the processed memory is written to the specified output file, otherwise the input memory is overwritten with the modified memory.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszInMemory	The fully qualified file name of the input memory.
PSZ	pszOutMemory	The fully qualified file name of the output memory. If not specified, the output translation memory overwrites the input translation memory.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The MT flags in the memory have been cleared successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

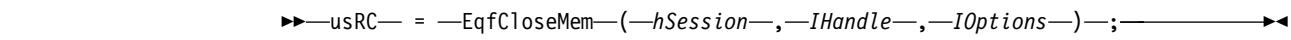
```
{
  USHORT usRC = 0; HSESSION hSession = 0L; // start the OpenTM2 API session usRC =
  EqfStartSession( &hSession ); // Clear all MT flags in the external Translation Memory
  MTMEM1.EXP // and write the resultinh memory entries to the memory MTCLEARED.EXP if ( !usRC
  ) { usRC = EqfClearMTFlag( hSession, "C:\\MTMEM1.EXP", "C:\\MTCLEARED.EXP" ); } /* endif */
  // terminate the session EqfEndSession( hSession );
}
```

EqfCloseMem

Purpose

EqfCloseMem closes a previously opened Translation Memory.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
LONG	IHandle	Translation Memory handle from a Translation Memory previously opened using EqfOpenMem (see "EqfOpenMem" on page 126).
LONG	lOptions	The options for the closing (currently none).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  LONG lHandle = 0;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // open the memory TestMem
  if ( !usRC )
  {
    usRC = EqfOpenMem( hSession, "TestMem", &lHandle, 0 );
  } /* endif */

  ...

  // close the memory
  if ( !usRC )
  {
    usRC = EqfCloseMem( hSession, lHandle, 0 );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

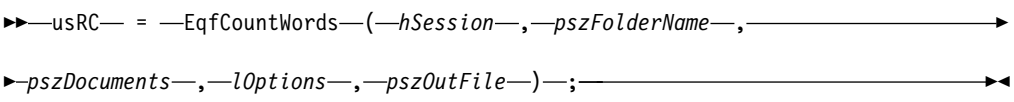
EqfCountWords

Purpose

EqfCountWords counts the words of one or more documents.

This function performs the counting in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents which are to be counted.
PSZ	pszDocuments	The pointer to a list of documents or NULL if no documents are specified. If no documents are specified, the words of all documents in the folder are counted.

Type	Parameter	Description
LONG	lOptions	The options to be used for the counting: SOURCE_OPT (source word count) TARGET_OPT (translated/untranslated word count) TMMATCH_OPT (memory match count) DUPLICATE_OPT (count duplicate words) DUPMEMMATCH_OPT (count duplicate words and include memory match information) For the TMMATCH_OPT the following option can be specified: SEPERATEREPLMATCH_OPT to count replace matches seperately. These constants are mutually exclusive, the can by combined with the format of the output file: XML_OUTPUT_OPT (output as XML file) or TEXT_OUTPUT_OPT (output in text format) or HTML_OUTPUT_OPT (output in HTML format) and the OVERWRITE_OPT (to overwrite existing output files) using "|" (bitwise OR operator). If no output format is specified TEXT_OUTPUT_OPT is used as default.
PSZ	pszOutFile	The fully qualified name of the output file. If the file already exists, specify the OVERWRITE_OPT option (otherwise this call fails).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	Word counting has not completed yet. Call *EqfCountWords* again.
other	

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );
```

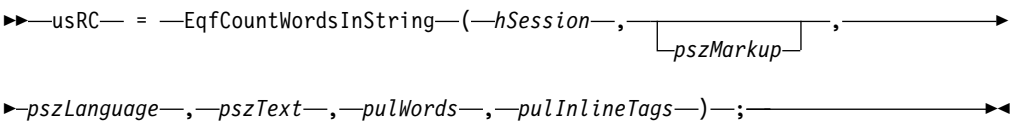
```
  // Count the source (=original) words of all documents in folder
  // SAMPLE1 and store the counting results in file C:\COUNT.OUT
  if ( !usRC )
  {
    do
    {
      usRC = EqfCountWords( hSession, "SAMPLE1", NULL, SOURCE_OPT,
                            "C:\\COUNT.OUT" );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfCountWordsInString

Purpose

The API call *EqfCountWordsInString* counts the number of words in a given string.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMarkup	The name of the markup table to be used for the recognition of in-line tags. If this parameter is NULL, no in-line tag recognition will be performed.
PSZ	pszLanguage	OpenTM2 name for the language of the given text.
PSZ	pszText	A null-terminated string containing the text to be counted. The encoding is UTF-16.
PULONG	pulWords	Points to an unsigned long value receiving the number of words in the text.
PULONG	pulInlineTags	Points to an unsigned long value receiving the number of inline tags in the text.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The words in the given text string have been counted.

Value	Description
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

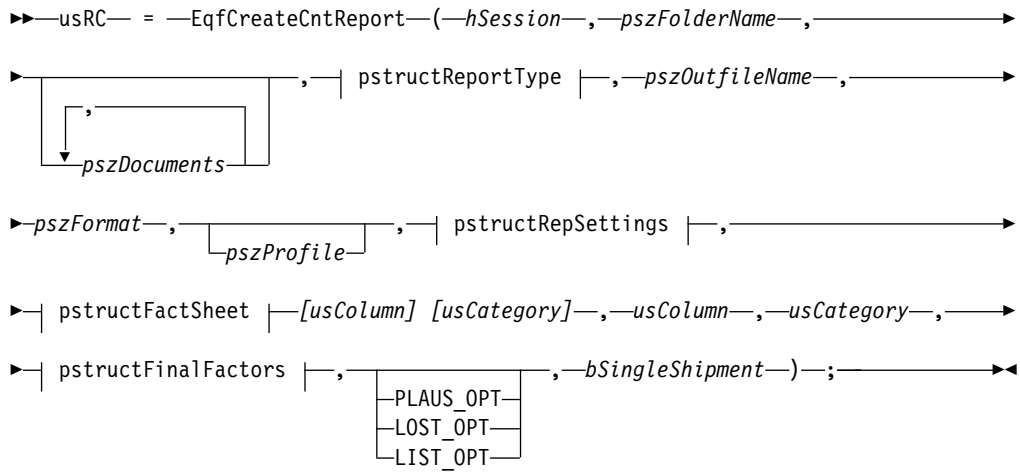
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  if ( !usRC )
  {
    ULONG ulWords = 0;
    ULONG ulTags = 0;

    // Count the words in the text string "This is a small test"
    // the result is stored in the variables ulWords and ulTags
    usRC = EqfCountWordsInString( hSession, "EQFANSI", "English(U.S.)", "This is a small test", &u
  } /* endif */
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfCreateCntReport

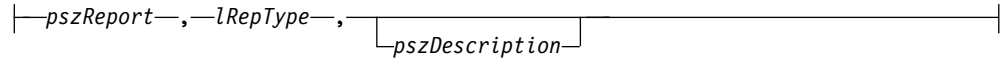
Purpose

EqfCreateCntReport creates Calculating, Preanalysis, Redundancy, Redundant segment list reports.

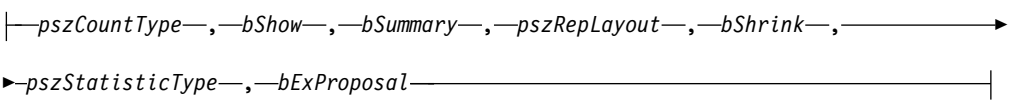
Format



structReportType:

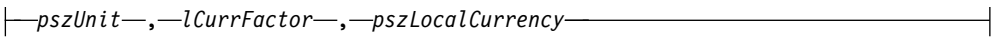


structRepSettings:



structFactSheet:

structFinalFactors:



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
PSZ	pszDocuments	List of documents, or `NULL` if all documents of the folder should be used.
PREPORT TYPE	pstructReportType	See “Parameters for structReportType” on page 52 for details.
PSZ	pszOutfileName	The name of the file where the report is to be stored (along with the drive and directory information).
PSZ	pszFormat	Format of the Output file ("ASCII"," HTML", or "XML").
PSZ	pszProfile	The name of the profile to be loaded, or `NULL`.
PREPORT SETTINGS	pstructRepSettings	See “Parameters for structRepSettings” on page 52 for details.
PFACTSHEET	pstructFactSheet [usColumn][usCategory]	Array of **structFactSheet**. See “Parameters for structFactSheet” on page 53 for details.
USHORT	usColumn	The first array index represents the column number according to the listed columns in the dialog “Create Counting Report”, tab “Fact Sheet”.
USHORT	usCategory	The second array index represents the category number according to the listed categories in the dialog “Create Counting Report”, tab “Fact Sheet”.
PFINAL FACTORS	pstructFinalFactors	See “Parameters for structFinalFactors” on page 53 for details.
LONG	lOptSecurity	The options to be used for security: • PLAUS_OPT (Plausibility check) • LOST_OPT (Lost Data: Force new shipment) • LIST_OPT (List of Documents) You can combine the options using OR.

Type	Parameter	Description
BOOL	bSingleShipment	• TRUE = Single Shipments • FALSE = All Shipments

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT   usRC = 0;
  HSESSION hSession = 0L;
  int      i,j;
  #define  COLUMN  10;
  #define  CATGORY 3;

  REPORTTYPE  ReportType = {NULL, 0L, NULL};
  REPSETTINGS ReportSettings = {NULL, 0, 0, NULL, 0, NULL, 0};
  FACTSHEET   FactSheet[COLUMN][CATEGORY];
  FINALFACT   FinalFactors = {0L, 0L NULL};

  //fill ReportType structure
  ReportType.pszReport = "Calculating Report");
  ReportType.lRepType=BASE_TYP | FACT_TYP | SUM_TYP;
  ReportType.pszDescription[0]='\0';

  //fill ReportSettings strucure
  RepSettings.pszCountType = "Source Words";
  RepSettings.bShow=TRUE;
  RepSettings.bSummary=TRUE;
  RepSettings.pszRepLayout = "Standard";
  RepSettings.bShrink=FALSE;
  RepSettings.pszStatisticType = NULL;
  RepSettings.bExProposal=FALSE;

  //fill FactSheet structure
  for(i=0;i++,i<COLUMN)
  {
    for(j=0,j++,j<CATEGORY)
    {
      FactSheet[i][j].lComplexity = (float)1.0;
      FactSheet[i][j].lPayFactor = (float)1.0;
    }
  }

  // fill FinalFactors structure
  FinalFactors.lUnit = 1;
  FinalFactors.lCurrFactor = (float)1.0;
  FinalFactors.pszLocalCurrency = "EUR";

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  if ( !usRC )
  {
```

```
    usRC = EqfCreateCntReport(hSession, 'e', "TEST", "test.doc,
                              test2.doc", &ReportType,
                              "E:\\Project\\CalcReport", "HTML",
                              NULL,
                             &RepSettings,(void *)FactSheet,
                              COLUMN, CATEGORY, &FinalFactors,
                              PLAUS_OPT, TRUE);
  } //endif
  // terminate the session
  EqfEndSession( hSession );
}
```

Parameters for structReportType

```
typedef struct _REPORTTYPE
{
  PSZ  pszReport;
  LONG lRepType;
  PSZ  pszDescription;
} REPORTTYPE, *PREPORTTYPE;
```

Type	Parameter	Description
PSZ	pszReport	Specifies one of the following reports: • "Calculating Report" • "Pre-Analysis Report" • "Redundancy Report" • "Redundant Segment List"
LONG	lRepType	One, or a combination, of the following report types: • BASE_TYP • FACT_TYP • SUM_TYP Allowed combinations are: • Base • Summary • Fact Sheet • Base & Summary • Summary & Fact Sheet • Base, Summary & Fact Sheet
PSZ	pszDescription	The report description, or NULL.

Parameters for structRepSettings

```
typedef struct _REPORTSETTINGS
{
  PSZ  pszCountType;
  BOOL bShow;
  BOOL bSummary;
  PSZ  pszRepLayout;
  BOOL bShrink;
  PSZ  pszStatisticType;
  BOOL bExProposal;
} REPORTSETTINGS, *PREPORTSETTINGS;
```

Type	Parameter	Description
PSZ	pszCountType	Specifies what to count: • "Source Words" • "Target Words" • "Segments" • "Modified Words"
BOOL	bShow	• TRUE = Show categories • FALSE = Hide categories
BOOL	bSummary	Build summary of categories
PSZ	pszRepLayout	Specifiy one of the following layouts: • "Standard" • "Standard and Group-Summary" • "Shrunk to Groups"
BOOL	bShrink	Automatic Shrink
PSZ	pszStatisticType	For pszReport = "Calculating Report" specify one of the following keywords: • "Standard" • "Advanced" or NULL for all other reports or no statistics.
BOOL	bExProposal	Use Existing Proposals.

Parameters for structFactSheet

```
typedef struct _FACTSHEET
{
  LONG lComplexity;
  LONG lPayFactor;
} FACTSHEET,*PFACTSHEET;
```

Type	Parameter	Description
float	lComplexity	Specifies the Complexity Factor.
float	lPayFactor	Specifies the Pay Factor.

Parameters for structFinalFactors

```
typedef struct _FINALFACTORS
{
  LONG lUnit;
  LONG lCurrFactor;
  PSZ  pszLocalCurrency;
} FINALFACTORS,*PFINALFACTORS;
```

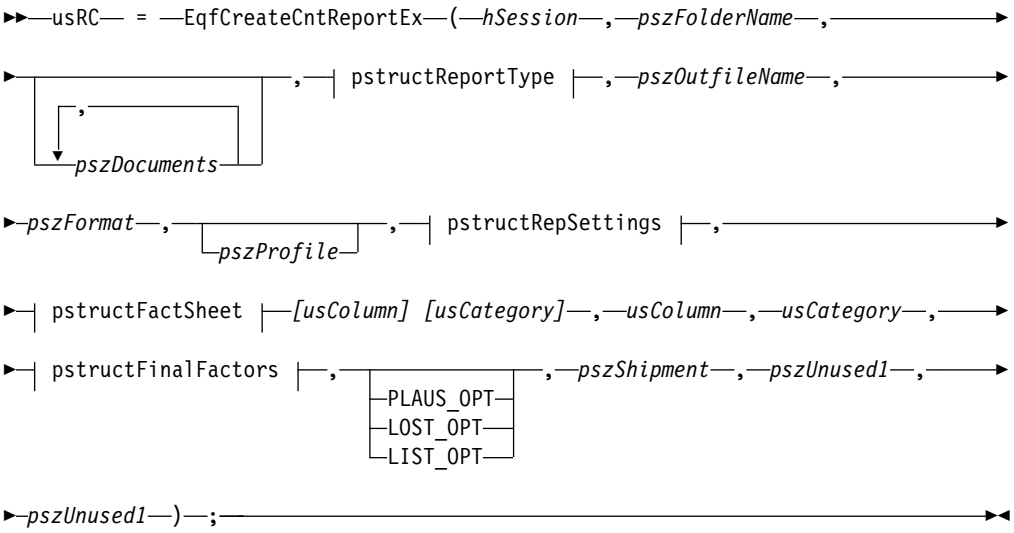
Type	Parameter	Description
LONG	lUnit	Values (in words): • 1 • 10 • 250
float	lCurrFactor	Specifies the local currency factor.
PSZ	pszLocalCurrency	Specifies the local currency. The local currencies correspond to the values of the dialog “Create Counting Report”, tab “Fact Sheet”.

EqfCreateCntReportEx

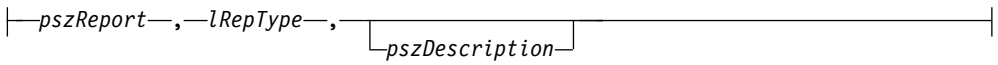
Purpose

EqfCreateCntReportEx creates Calculating, Preanalysis, Redundancy, Redundant segment list reports.

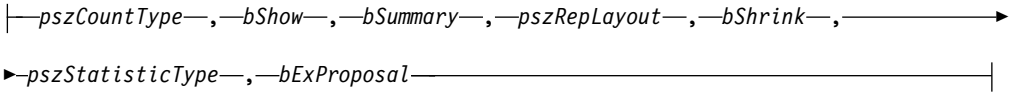
Format



structReportType:

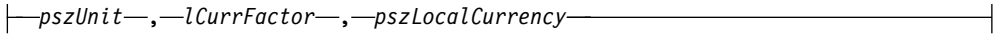


structRepSettings:



structFactSheet:

structFinalFactors:



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.

Type	Parameter	Description
PSZ	pszDocuments	List of documents, or NULL if all documents of the folder should be used.
PREPORT TYPE	pstructReportType	See “Parameters for structReportType” on page 56 for details.
PSZ	pszOutfileName	The name of the file where the report is to be stored (along with the drive and directory information).
PSZ	pszFormat	Format of the Output file ("ASCII"," HTML", or "XML").
PSZ	pszProfile	The name of the profile to be loaded, or NULL.
PREPORT SETTINGS	pstructRepSettings	See “Parameters for structRepSettings” on page 57 for details.
PFACTSHEET	pstructFactSheet [usColumn][usCategory]	Array of **structFactSheet**. See “Parameters for structFactSheet” on page 58 for details.
USHORT	usColumn	The first array index represents the column number according to the listed columns in the dialog “Create Counting Report”, tab “Fact Sheet”.
USHORT	usCategory	The second array index represents the category number according to the listed categories in the dialog “Create Counting Report”, tab “Fact Sheet”.
PFINAL FACTORS	pstructFinalFactors	See “Parameters for structFinalFactors” on page 58 for details.
LONG	lOptSecurity	The options to be used for security: • PLAUS_OPT (Plausibility check) • LOST_OPT (Lost Data: Force new shipment) • LIST_OPT (List of Documents) You can combine the options using OR.
PSZ	pszShipment	Shipment number, or “Single shipments” used for single shipments, or “All shipments” used for all shipments.
PSZ	pszUnused1	For future enhancements, currently not in use and should be NULL.
PSZ	pszUnused2	For future enhancements, currently not in use and should be NULL.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT   usRC = 0;
  HSESSION hSession = 0L;
  int      i,j;
  #define  COLUMN  10;
  #define  CATGORY 3;

  REPORTTYPE  ReportType = {NULL, 0L, NULL};
  REPSETTINGS ReportSettings = {NULL, 0, 0, NULL, 0, NULL, 0};
  FACTSHEET   FactSheet[COLUMN][CATEGORY];
  FINALFACT   FinalFactors = {0L, 0L NULL};

  //fill ReportType structure
  ReportType.pszReport = "Calculating Report");
  ReportType.lRepType=BASE_TYP | FACT_TYP | SUM_TYP;
  ReportType.pszDescription[0]='\0';

  //fill ReportSettings strucure
  RepSettings.pszCountType = "Source Words";
  RepSettings.bShow=TRUE;
  RepSettings.bSummary=TRUE;
  RepSettings.pszRepLayout = "Standard";
  RepSettings.bShrink=FALSE;
  RepSettings.pszStatisticType = NULL;
  RepSettings.bExProposal=FALSE;

  //fill FactSheet structure
  for(i=0;i++,i<COLUMN)
  {
    for(j=0,j++,j<CATEGORY)
    {
      FactSheet[i][j].lComplexity = (float)1.0;
      FactSheet[i][j].lPayFactor = (float)1.0;
    }
  }

  // fill FinalFactors structure
  FinalFactors.lUnit = 1;
  FinalFactors.lCurrFactor = (float)1.0;
  FinalFactors.pszLocalCurrency = "EUR";

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  if ( !usRC )
  {
    usRC = EqfCreateCntReportEx(hSession, 'e', "TEST", "test.doc,
                                test2.doc", &ReportType,
                                "E:\\Project\\CalcReport", "HTML",
                                NULL,
                               &RepSettings,(void *)FactSheet,
                                COLUMN, CATEGORY, &FinalFactors,
                                PLAUS_OPT, "1", NULL, NULL );
  } //endif
  // terminate the session
  EqfEndSession( hSession );
}
```

Parameters for structReportType

```
typedef struct _REPORTTYPE { PSZ pszReport; LONG lRepType; PSZ pszDescription; }
        REPORTTYPE, *PREPORTTYPE;
```

Type	Parameter	Description
PSZ	pszReport	Specifies one of the following reports: • "Calculating Report" • "Pre-Analysis Report" • "Redundancy Report" • "Redundant Segment List"
LONG	lRepType	One, or a combination, of the following report types: • BASE_TYP • FACT_TYP • SUM_TYP Allowed combinations are: • Base • Summary • Fact Sheet • Base & Summary • Summary & Fact Sheet • Base, Summary & Fact Sheet
PSZ	pszDescription	The report description, or `NULL`.

Parameters for structRepSettings

```
typedef struct _REPORTSETTINGS { PSZ pszCountType; BOOL bShow; BOOL bSummary; PSZ
        pszRepLayout; BOOL bShrink; PSZ pszStatisticType; BOOL bExProposal; } REPORTSETTINGS,
        *PREPORTSETTINGS;
```

Type	Parameter	Description
PSZ	pszCountType	Specifies what to count: • "Source Words" • "Target Words" • "Segments" • "Modified Words"
BOOL	bShow	• TRUE = Show categories • FALSE = Hide categories
BOOL	bSummary	Build summary of categories
PSZ	pszRepLayout	Specifiy one of the following layouts: • "Standard" • "Standard and Group-Summary" • "Shrunk to Groups"
BOOL	bShrink	Automatic Shrink
PSZ	pszStatisticType	For pszReport = "Calculating Report" specify one of the following keywords: • "Standard" • "Advanced" or `NULL` for all other reports or no statistics.
BOOL	bExProposal	Use Existing Proposals.

Parameters for structFactSheet

```
typedef struct _FACTSHEET { LONG lComplexity; LONG lPayFactor; }
        FACTSHEET,*PFACTSHEET;
```

Type	Parameter	Description
float	lComplexity	Specifies the Complexity Factor.
float	lPayFactor	Specifies the Pay Factor.

Parameters for structFinalFactors

```
typedef struct _FINALFACTORS { LONG lUnit; LONG lCurrFactor; PSZ pszLocalCurrency;
        } FINALFACTORS,*PFINALFACTORS;
```

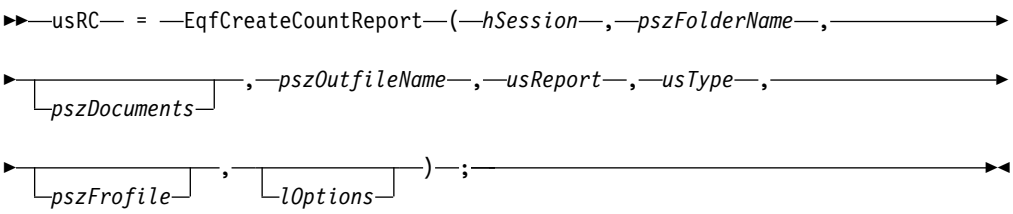
Type	Parameter	Description
LONG	lUnit	Values (in words): • 1 • 10 • 250
float	lCurrFactor	Specifies the local currency factor.
PSZ	pszLocalCurrency	Specifies the local currency. The local currencies correspond to the values of the dialog “Create Counting Report”, tab “Fact Sheet”.

EqfCreateCountReport

Purpose

EqfCreateCountReport creates Calculating, Preanalysis, Redundancy, Redundant segment list reports **using counting profiles**.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
PSZ	pszDocuments	List of documents, or NULL if all documents of the folder should be used.
PSZ	pszOutfileName	The name of the file where the report is to be stored (along with the drive and directory information).

Type	Parameter	Description
USHORT	usReport	Type of report: • HISTORY_REP (History Report) • COUNTING_REP (Counting Report) • CALCULATING_REP (Calculation Report) • PREANALYSIS_REP (PreAnalysis Report) • REDUNDANCY_REP (Redundncy Report) • REDUNDANCYSEGMENT_REP (Redundancy Segment List)
USHORT	usType	Type of report: for HISTORY_REP • BRIEF_SORTBYDATE_REPTYPE • BRIEF_SORTBYDOC_REPTYPE • DETAIL_REPTYPE for HISTORY_REP • WITHTOTALS_REPTYPE • WITHOUTTOTALS_REPTYPE for CALCULATING_REP, PREANALYSIS_REP, and REDUNDANCY_REP • BASE_REPTYPE • BASE_SUMMARY_REPTYPE • BASE_SUMMARY_FACTSHEET_REPTYPE • SUMMARY_FACTSHEET_REPTYPE • FACTSHEET_REPTYPE
PSZ	pszProfile	The name of the profile to be loaded.
LONG	lOptions	Options for the counting report: OVERWRITE_OPT or 0

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT   usRC = 0;
  HSESSION hSession = 0L;
 // start the Eqf calling interface session
 usRC 0 EqfStartSession(&hSession)

  if ( !usRC )
  {
    usRC = EqfCreateCountReport(hSession, 'e', "TEST", "test.doc,
                                test2.doc",
                                "E:\\Project\\CalcReport",
```

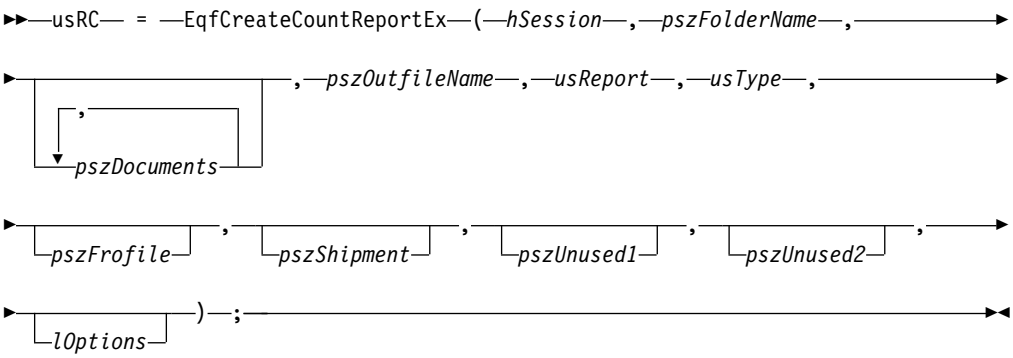
```
                                COUNTING_REP, BASESUMMARY_REPTYPE
                                "PUB0205", OVERWRITE_OPT);
  } //endif
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfCreateCountReportEx

Purpose

EqfCreateCountReportEx creates Calculating, Preanalysis, Redundancy, Redundant segment list reports **using counting profiles**. Reports can be created for single shipments, for all shipments, or for a specific shipment.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
PSZ	pszDocuments	List of documents, or `NULL` if all documents of the folder should be used.
PSZ	pszOutfileName	The name of the file where the report is to be stored (along with the drive and directory information).
USHORT	usReport	Type of report: • HISTORY_REP (History Report) • COUNTING_REP (Counting Report) • CALCULATING_REP (Calculation Report) • PREANALYSIS_REP (PreAnalysis Report) • REDUNDANCY_REP (Redundancy Report) • REDUNDANCYSEGMENT_REP (Redundand Segment List)

Type	Parameter	Description
USHORT	usType	Type of report: For HISTORY_REP • BRIEF_SORTBYDATE_REPTYPE • BRIEF_SORTBYDOC_REPTYPE • DETAIL_REPTYPE For HISTORY_REP • WITHTOTALS_REPTYPE • WITHOUTTOTALS_REPTYPE For CALCULATING_REP, PREANALYSIS_REP, and REDUNDANCY_REP • BASE_REPTYPE • BASE_SUMMARY_REPTYPE • BASE_SUMMARY_FACTSHEET_REPTYPE • SUMMARY_FACTSHEET_REPTYPE • FACTSHEET_REPTYPE
PSZ	pszProfile	The name of the profile to be loaded, or NULL.
PSZ	pszShipment	Shipment number or “Single shipments” for single shipments or “All shipments” for all shipments.
PSZ	pszUnused1	For future enhancements, currently not in use and should be NULL.
PSZ	pszUnused2	For future enhancements, currently not in use and should be NULL.
LONG	lOptions	Options for the counting report: • HTML_OUTPUT_OPT to create an **HTML** report or • XML_OUTPUT_OPT to create an **XML** report or • TEXT_OUTPUT_OPT to create a **plain text** report and • OVERWRITE_OPT to overwrite any existing report.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT   usRC = 0;
  HSESSION hSession = 0L;
 // start the Eqf calling interface session
```

```
 usRC 0 EqfStartSession(&hSession)

  if ( !usRC )
  {
    usRC = EqfCreateCountReportEx(hSession, 'e', "TEST", "test.doc,
                                  test2.doc",
                                  "E:\\Project\\CalcReport",
                                  COUNTING_REP, BASESUMMARY_REPTYPE
                                  "PUB20184", "1", NULL, NULL, OVERWRITE_OPT);
  } //endif
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfCreateControlledFolder

Purpose

EqfCreateControlledFolder creates a new controlled folder by using the specified values. Configure the target drive for the folder using the "Configure Drives" window of OpenTM2.

Format

Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.

Type	Parameter	Description
PSZ	pszFolderName	The name of the folder to be created.
PSZ	pszDescription	The folder description, or NULL.
CHAR	chTargetDrive	The target drive for the new folder. If omitted, the primary EQF drive is used. The drive must be the primary EQF drive or one of the secondary EQF drives defined in the "Configure Drives" window.
PSZ	pszTransMem	The name of the Translation Memory to be used for the documents in the new folder.
PSZ	pszMarkup	The name of the markup table, for example EQFMRI.
PSZ	pszEditor	The name of the editor. If not specified, the editor STANDARD is used.
PSZ	pszDictionaries	The list of dictionaries to be used during translation. You can specify up to 10 dictionaries.
PSZ	pszSourceLanguage	The source language for the documents in this folder. Specify the language exactly as it appears in the "Language List" window, for example English(U.S.).
PSZ	pszTargetLanguage	The target language for the documents in this folder. Specify the language exactly as it appears in the "Language List" window, for example English(U.S.). The target language must different from the source language.
PSZ	pszConversion	The export conversion type, or NULL for no conversion.
PSZ	pszReadOnlyMems	The list of Translation Memories to search through during translation, or NULL. You can specify up to 4 Translation Memories.
PSZ	pszPassword	The password to protect the folder against changes. The password can be up to six characters long.
PSZ	pszProjCoordName	The name of the project coordinator, or NULL.
PSZ	pszProjCoordMail	The e-mail address of the project coordinator, or NULL.
PSZ	pszTranslatorName	The name of the translator responsible for this folder, or NULL.
PSZ	pszTranslatorMail	The e-mail address of the translator, or NULL.
PSZ	pszProductName	The product name this folder is assigned to, or NULL.
PSZ	pszProductFamily	The product family this folder is assigned to, or NULL.
PSZ	pszSimilarProduct	The name of a similar product this folder is assigned to, NULL.
PSZ	pszProductDict	The product-specific dictionary to be used during translation, or NULL.
PSZ	pszProductMem	The product-specific memory to be used during translation, or NULL.

Type	Parameter	Description
PSZ	pszPreviousVersion	The previous version number of the product specified above, or NULL.
PSZ	pszVersion	The actual version number of the product specified above, or NULL.
PSZ	pszShipmentNumber	The number of the shipment, or NULL.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

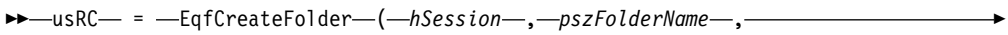
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  // Create a new controlled folder name 'Test' on the
  // primary Eqf drive
  if ( !usRC )
  {
    usRC = EqfCreateControlledFolder(hSession, "Test",
                              "Description of folder Test",
                              '\0', // use primary Eqf drive
                              "MEM1", "EQFASCII",
                              "STANDARD", "DICT1,ENGLGERM",
                              "English(U.S.)","German(national)",
                              NULL, NULL, "passwd",
                              "ProjCoordName", "ProjCoordMail",
                              "TranslatorName","TranslatorMail", NULL,
                              "Family", NULL, "Dict", "MemoryName",
                              "1.0", "2.0", "1");
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

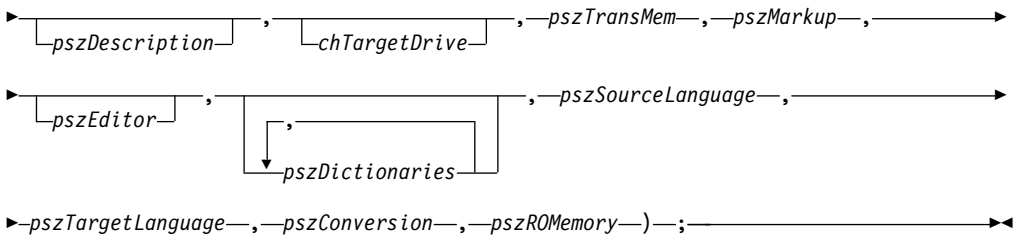
EqfCreateFolder

Purpose

EqfCreateFolder creates a new folder by using the specified values. Configure the target drive for the folder using the "Configure Drives" window of OpenTM2.

Format





Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder to be created.
PSZ	pszDescription	The folder description, or `NULL`.
CHAR	chTargetDrive	The target drive for the new folder. If omitted, the primary EQF drive is used. The drive must be the primary EQF drive or one of the secondary EQF drives defined in the "Configure Drives" window.
PSZ	pszTransMem	The name of the Translation Memory to be used for the documents in the new folder.
PSZ	pszMarkup	The name of the markup table, for example EQFMRI.
PSZ	pszEditor	The name of the editor. If not specified, the editor STANDARD is used.
PSZ	pszDictionaries	The list of dictionaries to be used during translation. You can specify up to 10 dictionaries.
PSZ	pszSourceLanguage	The source language for the documents in this folder. Specify the language exactly as it appears in the "Language List" window, for example `English(U.S.)`.
PSZ	pszTargetLanguage	The target language for the documents in this folder. Specify the language exactly as it appears in the "Language List" window, for example `English(U.S.)`.
PSZ	pszConversion	Conversion to be used for the folder or NULL
PSZ	pszROMemory	List of read-only memories to be searched or NULL

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Create a new folder name SAMPLE1 on the primary Eqf drive
  if ( !usRC )
  {
    usRC = EqfCreateFolder( hSession, "SAMPLE1",
                            "Description of folder SAMPLE1",
                            '\0',     // use primary Eqf drive
                            "MEM01", "EQFASCII", "STANDARD",
                            "DICT1,ENGLGERM", "English(U.S.)",
                            "German(National)" );

  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

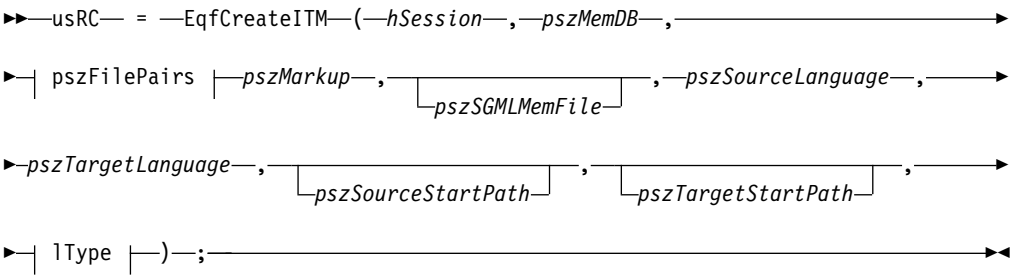
EqfCreateITM

Purpose

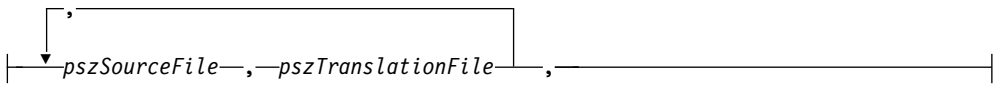
EqfCreateITM creates an Initial Translation Memory (ITM) database from an existing Translation Memory. It can create an internal Translation Memory and an external Translation Memory. The internal Translation Memory must not be filled.

Important hint: If you want to generate a source English-English memory (i.e. a memory where the source sentence and the target sentence are identical), please always use EQFArchiveTM function with the option SOURCESOURCEMEM_OPT.

Format



pszFilePairs:



lType:

Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemDB	The name of a previously created OpenTM2 Translation Memory (without the file extension). This Translation Memory can still be empty. It can be filled with original segments and their corresponding translations.
PSZ	pszFilePairs	List of file names to use when creating the ITM, in the form `(original1, translation1, original2, translation2)`.
PSZ	pszMarkup	The name of the markup table, for example `EQFMRI`.
PSZ	pszSGMLMemFile	The name you want to give to the external ITM, and the path where it is to be located. The ITM is in SGML format and can subsequently be imported into OpenTM2 after you have checked it.
PSZ	pszSourceLanguage	The source language for the documents in this folder. Specify the language exactly as it appears in the "Language List" window, for example `English(U.S.)`.
PSZ	pszTargetLanguage	The target language for the documents in this folder. Specify the language exactly as it appears in the "Language List" window, for example `English(U.S.)`.
PSZ	pszSourceStartPath	The path information that you do *not* want to become part of the document name when the original document is stored in the Initial Translation Memory. For example, if your source file is stored in `e:\tm\project\english`, and you do not want `e:\tm\project` to become part of the name under which it is stored, specify `e:\tm\project`. The path you specify here can differ from the *pszTargetStartPath*. However, if you specify a source start path, you must also specify a *pszTargetStartPath*.

Type	Parameter	Description
PSZ	pszTargetStartPath	The path information that you do **not** want to become part of the document name when the target document is stored in the Initial Translation Memory. For example, if your source file is stored in `e:\tm\project\english` and you do not want `e:\tm\project` to become part of the name under which it is stored, specify `e:\tm\project`. The path you specify here can differ from the *pszSourceStartPath*. However, if you specify a source start path, you must also specify a *pszSourceStartPath*.
LONG	lType	One or more of the following: • NOANA_TYP Do not analyze the selected files because they have already been analyzed by OpenTM2. • NOTM_TYP Do not fill the internal Translation Memory (*pszMemDB*). Fill the external Translation Memory. It is in SGML format and you can check it afterwards. • PREPARE_TYP The source documents are related to their corresponding translations. The file pairs are prefixed with **p**. You can combine the options using OR.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  // Create a new Initial Translation Memory TestITM
  if ( !usRC )
  {
    usRC = EqfCreateITM(hSession, "TestITM",
                        "E:\TM\PROJECT\ENFGLISH\original1,
                         E:\TM\PROJECT\GERMAN\translation1",
                        "EQFASCII", NULL,
```

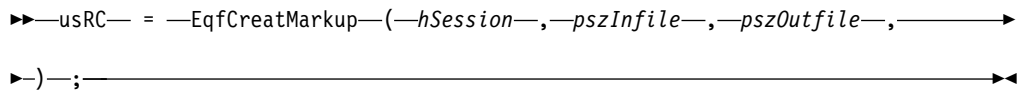
```
                        "English(U.S.)", "German(national)",
                        "E:\TM\PROJECT", "E:\TM\PROJECT", 0);
   } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfCreateMarkup

Purpose

The API call *EqfCreateMarkup* creates an internal markup table (*.TBL) from an external markup table (*.TBX).

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszInfile	The fully qualified name of input file (*.TBX format).
PSZ	pszOutfile	The fully qualified name of output file (*.TBL format).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The markup table has been converted successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

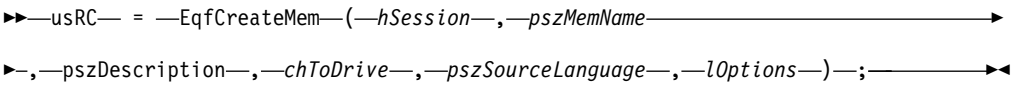
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  if ( !usRC )
  {
    // Convert the external markup table MYMARKUP.TBX into the internal
    // format and store the result under MYMARKUP.TBL
    usRC = EqfCreateMarkup( hSession, "C:\\MYMARKUP.TBX", "C:\\OTM\\TABLE\MYMARKUP.TBL" );
  } /* endif */
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfCreateMem

Purpose

EqfCreateMem creates a new shared or local Translation Memory.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemName	The name of the Translation Memory to be created.
PSZ	pszDescription	The description of the Translation Memory.
CHAR	chToDrive	The target drive for the new Translation Memory. If omitted, the primary EQF drive is used. The drive must be the primary EQF drive or one of the secondary EQF drives defined in the "Configure Drives" window.
LONG	lOptions	The type of the new Translation Memory: LOCAL_OPT, which is the default SHARED_OPT
PSZ	pszSourceLanguage	The source language to be used for the Translation Memory

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Create the new local Translation Memory MEMDB2 on the
  // primary Eqf system drive
  if ( !usRC )
  {
       usRC = EqfCreateMem( hSession, "MEMDB2",
```

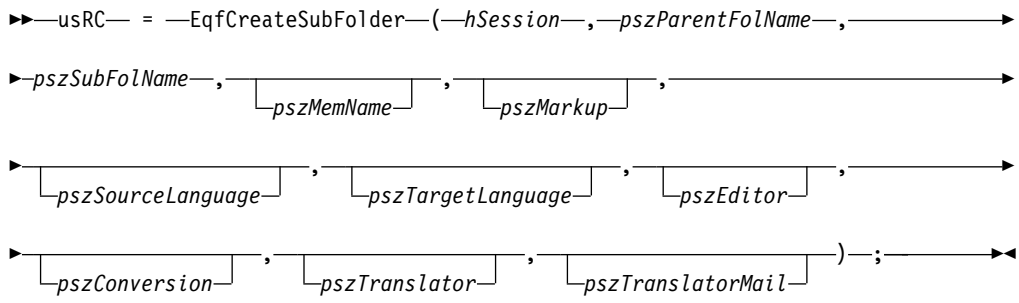
```
                      "TM created via Func I/F",
                      '\0', "English(U.S.)", LOCAL_OPT );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfCreateSubFolder

Purpose

EqfCreateSubFolder creates a subfolder from a parent folder by using the specified values. The parent folder itself can be a subfolder.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszParentFolName	The name of the parent folder, or the name of a subfolder that acts as a parent folder.
PSZ	pszSubFolName	The name of the subfolder to be created.
PSZ	pszMemName	The name of the Translation Memory to be used for the documents in the new folder. If you want the same as in the parent folder, specify `NULL`.
PSZ	pszMarkup	The name of the markup table, for example EQFMRI. If you want the same as in the parent folder, specify `NULL`.
PSZ	pszSourceLanguage	The source language for the documents in this folder. Specify the language exactly as it appears in the “Language List” window, for example `English(U.S.)`. If you want the same as in the parent folder, specify `NULL`.
PSZ	pszTargetLanguage	The target language for the documents in this folder. Specify the language exactly as it appears in the “Language List” window, for example `English(U.S.)`. The target language must different from the source language. If you want the same as in the parent folder, specify `NULL`.

Type	Parameter	Description
PSZ	pszEditor	The name of the editor. If not specified, the editor STANDARD is used.
PSZ	pszConversion	The export conversion type. If you want the same as in the parent folder, specify NULL.
PSZ	pszTranslator	The name of the translator. If you want the same as in the parent folder, specify NULL.
PSZ	pszTranslatorMail	The e-mail address of the translator. If you want the same as in the parent folder, specify NULL.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

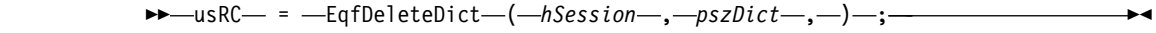
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  // Create a subfolder "SUBSUBTEST" of the parent folder "SUBTEST",
  // which itself is a subfolder of parent folder "TEST".

if ( !usRC )
 {
          usRC = EqfCreateSubFolder(hSession,
                                    "TEST\\SUBTEST", "SUBSUBTEST",
                                    "MEM1", "EQFASCII",
                                    "English(U.S.)",
                                    "German(national)", NULL, NULL,
                                     "Translator",
                                     "Translator@xyz.com");
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfDeleteDict

Purpose

The API call *EqfDeleteDict* deletes the given dictionary.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszDict	The name of the dictionary to be deleted.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The dictionary has been deleted successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

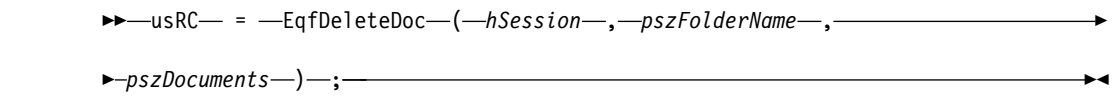
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  if ( !usRC )
  {
    // Delete the dictionary MySuperfluousDict
    usRC = EqfDeleteDic( hSession, "MySuperfluousDict" );
  } /* endif */
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfDeleteDoc

Purpose

EqfDeleteDoc deletes the specified documents.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents to be deleted.
PSZ	pszDocuments	The name of the documents to be deleted.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

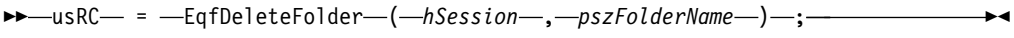
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Delete document DOC1.TXT in folder SAMPLE1
  if ( !usRC )
  {
         usRC = EqfDeleteDoc( hSession, "SAMPLE1", "DOC1.TXT" );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfDeleteFolder

Purpose

EqfDeleteFolder deletes the specified folder and all the documents that it contains.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder to be deleted.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

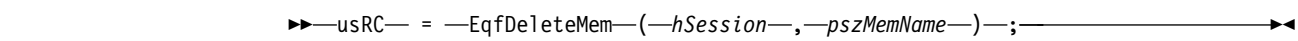
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Delete the folder SAMPLE1
  if ( !usRC )
  {
        usRC = EqfDeleteFolder( hSession, "SAMPLE1" );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfDeleteMem

Purpose

EqfDeleteMem deletes a Translation Memory.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemName	The name of the Translation Memory to be deleted.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Delete the Translation Memory MEMDB2
  if ( !usRC )
  {
```

```
    usRC = EqfDeleteMem( hSession, "MEMDB2" );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfDeleteMTLog

Purpose

The API call *EqfDeleteMTLog* deletes the MT-log (machine translation LOG) of a given folder.

Format

Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder to be processed.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The MT-log has been deleted successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

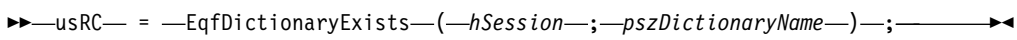
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  if ( !usRC )
  {
    // Delete the MT-Log of the folder "MyTestFolder"
    usRC = EqfDeleteMTLog( hSession, "MyTestFolder" );
  } /* endif */
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfDictionaryExists

Purpose

EqfDictionaryExists checks if the given dictionary exists in OpenTM2.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszDictionaryName	The name of the dictionary for which the existence is to be checked

Return code

USHORT

Value	Description
0 (NO_ERROR)	The specified dictionary exists in OpenTM2
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

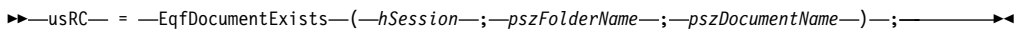
```
{
  USHORT usRC = 0; HSESSION hSession = 0L;
  // start the Eqf calling interface
  session usRC = EqfStartSession( &hSession );
  usRC = EqfDictionaryExists( hSession, "MyDictionary" );
  // terminate the session EqfEndSession( hSession ); }
```

EqfDocumentExists

Purpose

EqfDocumentExists checks if the given document exists in OpenTM2.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the document
PSZ	pszDocumentName	The name of the document for which the existence is to be checked

Return code

USHORT

Value	Description
0 (NO_ERROR)	The specified dictionary exists in OpenTM2
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the Eqf calling interface
  session usRC = EqfStartSession( &hSession );
  usRC = EqfDocumentExists( hSession, "MyFolder", "MyDocument" );
  // terminate the session EqfEndSession( hSession );
}
```

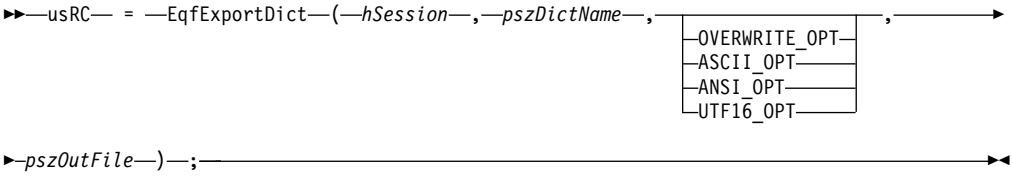
EqfExportDict

Purpose

EqfExportDict exports a dictionary in SGML format to the specified file. It fails if the output file exists already unless the OVERWRITE_OPT has been set. Default encoding of output SGML dictionary is Unicode (UTF16). Specify the option ASCII_OPT or ANSI_OPT if the export dictionary should have the corresponding format.

This function performs the export in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszDictName	The name of the dictionary to be exported.
LONG	lOptions	The option to be used for the export: • OVERWRITE_OPT • ASCII_OPT • ANSI_OPT • UTF16_OPT

Type	Parameter	Description
PSZ	pszOutFile	The fully qualified name of the output file. If the output file exists already, specify the OVERWRITE_OPT option.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The dictionary export has not completed yet. Call *EqfExportDict* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Export the dictionary ENGLGERM to SGML file C:\DICT1.SGM
  // and overwrite any existing SGML file with this name
  if ( !usRC )
  {
    do
    {
      usRC = EqfExportDict( hSession, "ENGLGERM", OVERWRITE_OPT | UTF16_OPT
                            "C:\\DICT1.SGM" );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

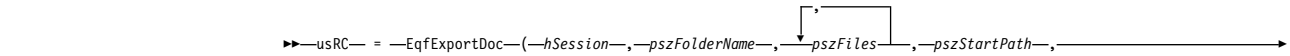
EqfExportDoc

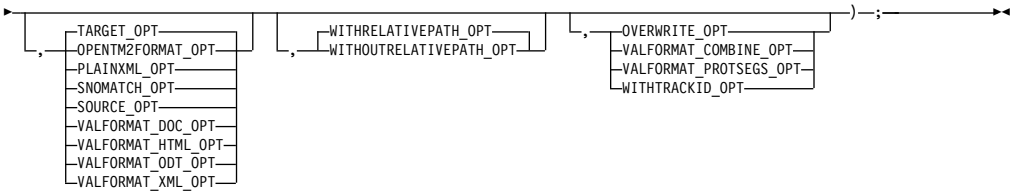
Purpose

EqfExportDoc exports one or more documents.

This function performs the export in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format





Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents to be exported.
PSZ	pszFiles	The name, including the target path, of the documents to be exported.
PSZ	pszStartPath	The start path if the documents are to be exported with relative path information. If a start path is specified, the files in the *pszFiles* list only contain the relative path.
LONG	lOptions	The options to be used for the document export: OPENTM2FORMAT_OPT PLAINXML_OPT SNOMATCH_OPT SOURCE_OPT TARGET_OPT, which is the default VALFORMAT_DOC_OPT to create MS WORD DOC outputs VALFORMAT_HTML_OPT to create HTML outputs VALFORMAT_ODT_OPT to create Open Office Writer outputs VALFORMAT_XML_OPT to create XML outputs WITHOUTRELATIVEPATH WITHRELATIVEPATH_OPT OVERWRITE_OPT to replace existing documents. VALFORMAT_COMBINE_OPT to combine validation format exports into one document. VALFORMAT_PROTSEGS_OPT to export with protected segments. WITHTRACKID_OPT to export documents with a tracking-ID per segment. These options correspond to those in the OpenTM2 “Export Documents” window.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The document export has not completed yet. Call *EqfExportDoc* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Export the translations of documents DOC1.TXT and DOC2.TXT of
  // folder SAMPLE1
  if ( !usRC )
  {
    do
    {
      usRC = EqfExportDoc( hSession, "SAMPLE1", NULL,
                           "C:\\DOC1.TXT,C:\\DOC2.TXT",
                           TARGET_OPT );

    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

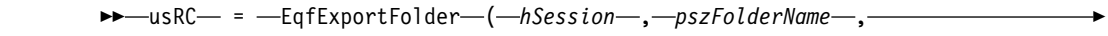
EqfExportFolder

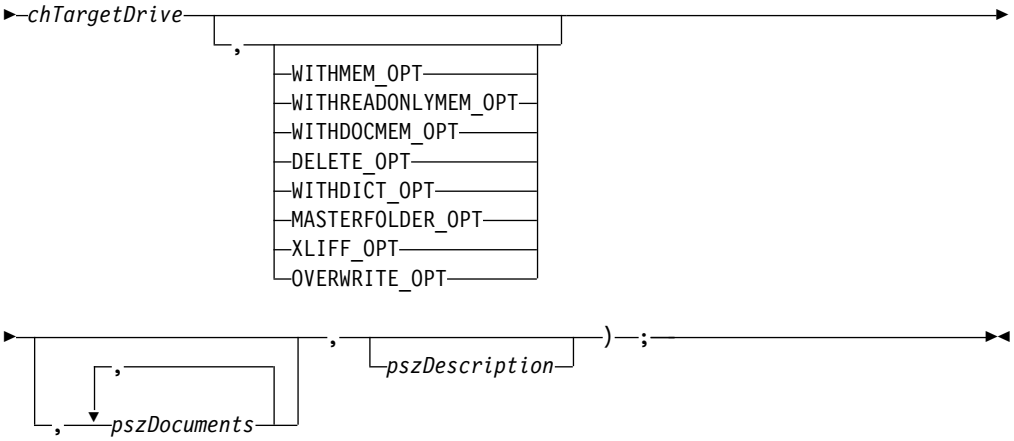
Purpose

EqfExportFolder exports a folder to a specific target drive. The path is always `\otm\export`.

This function performs the export in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format





Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder to be exported.
PSZ	pszDescription	The folder description, or NULL.
CHAR	chTargetDrive	The drive to which the folder is exported.
LONG	lOptions	The options to be used for the export: WITHMEM_OPT WITHREADONLYMEM_OPT WITHDOCMEM_OPT DELETE_OPT WITHDICT_OPT MASTERFOLDER_OPT XLIFF_OPT OVERWRITE_OPT These options correspond to those in the "Export Folder" window (see). You can combine the constants using OR.
PSZ	pszDocuments	The name of one or more documents.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The folder export has not completed yet. Call *EqfExportFolder* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Export the folder SAMPLE1 to drive C: with the folder
  // Translation Memory and all folder dictionaries, overwrite
  // any previously exported folder on drive C:
  if ( !usRC )
  {
    do
    {
            usRC = EqfExportFolder( hSession, "SAMPLE1", 'C',
                        WITHMEM_OPT | WITHDICT_OPT | OVERWRITE_OPT,
                        NULL, NULL );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

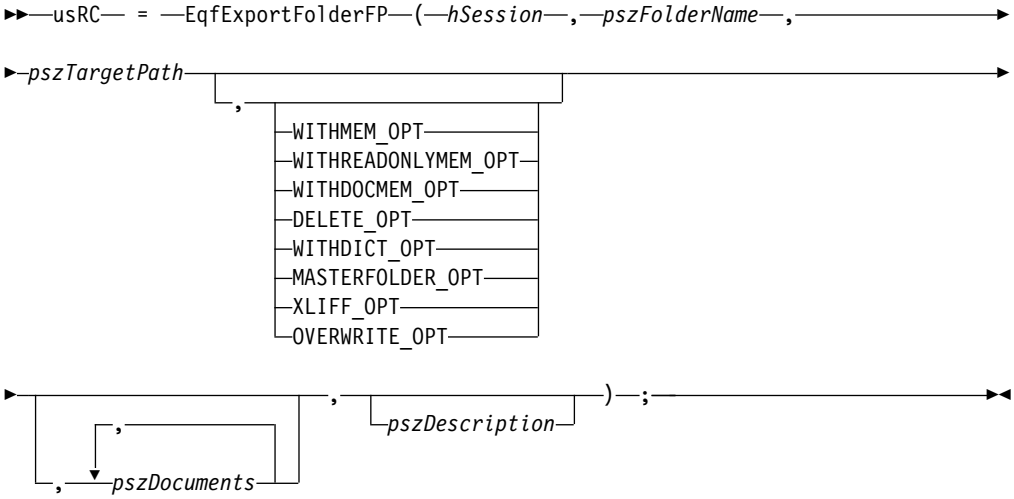
EqfExportFolderFP

Purpose

EqfExportFolderFP exports a folder to a specific path.

This function performs the export in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.

Type	Parameter	Description
PSZ	pszFolderName	The name of the folder to be exported.
PSZ	pszDescription	The folder description, or NULL.
PSZ	pszTargetPath	The path to which the folder is exported.
LONG	lOptions	The options to be used for the export: WITHMEM_OPT WITHREADONLYMEM_OPT WITHDOCMEM_OPT DELETE_OPT WITHDICT_OPT MASTERFOLDER_OPT XLIFF_OPT OVERWRITE_OPT These options correspond to those in the "Export Folder" window (see). You can combine the constants using OR.
PSZ	pszDocuments	The name of one or more documents.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The folder export has not completed yet. Call *EqfExportFolderFP* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Export the folder SAMPLE1 to path C:\PROJECT with the
  // folder Translation Memory and all folder dictionaries,
  // overwrite any previously exported folder in path C:\PROJECT
  if ( !usRC )
  {
    do
    {
            usRC = EqfExportFolderFP( hSession, "SAMPLE1",
                        'C:\PROJECT',
                        WITHMEM_OPT | WITHDICT_OPT | OVERWRITE_OPT,
                        NULL, NULL );
    } while ( usRC == CONTINUE_RC );
  } /* endif */
```

```
  // terminate the session
  EqfEndSession( hSession );
}
```

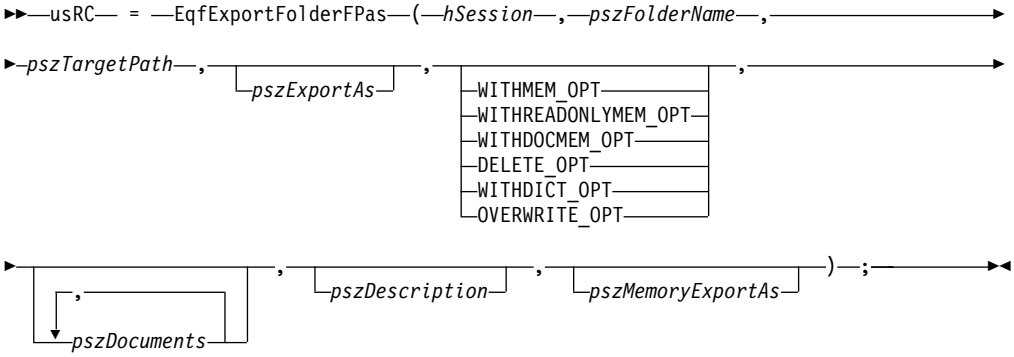
EqfExportFolderFPas

Purpose

EqfExportFolderFPas exports a folder to a specific path with the option to specify a new filename to the exported folder.

This function performs the export in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder to be exported.
PSZ	pszTargetPath	The path to which the folder is exported.
PSZ	pszExportAs	The filename of the exported folder, or NULL.
LONG	lOptions	The options to be used for the export: WITHMEM_OPT WITHREADONLYMEM_OPT WITHDOCMEM_OPT DELETE_OPT WITHDICT_OPT OVERWRITE_OPT These options correspond to those in the "Export Folder" window (see). You can combine the constants using OR.
PSZ	pszDocuments	The name of one or more documents.
PSZ	pszDescription	The folder description, or NULL.

Type	Parameter	Description
PSZ	pszMemoryExportAs	The filename of the exported memory in the folder, or NULL.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The folder export has not completed yet. Call *EqfExportFolderFP* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Export the folder SAMPLE1 to path C:\PROJECT with the
  // folder Translation Memory and all folder dictionaries,
  // overwrite any previously exported folder in path C:\PROJECT
  // the folder memory is renamed to "MEM1"
  if ( !usRC )
  {
    do
    {
            usRC = EqfExportFolderFPas( hSession, "SAMPLE1",
                        'C:\PROJECT', "MyFoll",
                        WITHMEM_OPT | WITHDICT_OPT | OVERWRITE_OPT,
                        NULL, NULL, "MEM1" );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

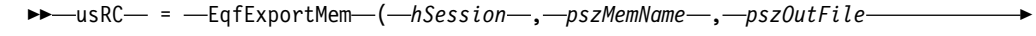
EqfExportMem

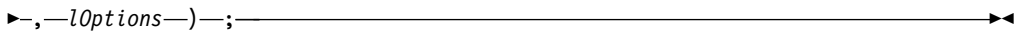
Purpose

EqfExportMem exports a Translation Memory in external format.

This function performs the export in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format





Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemName	The name of the Translation Memory to be exported.
PSZ	pszInFile	The fully qualified name of the file receiving the exported Translation Memory.
LONG	lOptions	The option to be used for the Translation Memory export: • OVERWRITE_OPT to replace an existing Translation Memory. • ANSI_OPT (Export in Ansi) • ASCII_OPT (Export in ASCII) • UTF16_OPT (Export in Unicode UTF-16) • TMX_UTF16_OPT (Export in TMX format, use UTF-16 encoding) • TMX_UTF8_OPT (Export in TMX format, use UTF-8 encoding) • TMX_NOCRLF_OPT to remove line breaks from the segment text, can be used together with the TMX_UTF16_OPT or the TMX_UTF8_OPT option only

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
559 (ERROR_MEM_ DATACORRUPT)	The export completed successfully but some characters habe been corrupted (i.e. these characters cannot be re-converted to Unicode without loss of data)
CONTINUE_RC	The Translation Memory export has not completed yet. Call *EqfExportMem* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Export the Translation Memory MEMDB1 to the external file MEM1.EXP
  if ( !usRC )
```

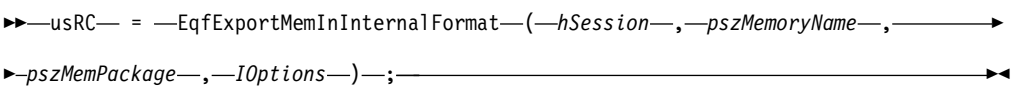
```
  {
    do
    {
            usRC = EqfExportMem( hSession, "MEMDB1", "C:\\MEM1.EXP", 0L );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfExportMemInInternalFormat

Purpose

EqfExportMemInInternalFormat exports the OpenTM2 internal files of a Translation Memory to a Zip package.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemoryName	The name of the Translation Memory.
PSZ	pszMemPackage	The fully qualified name of the new ZIP package which will be filled with the OpenTM2 internal Translation Memory files.
LONG	lOptions	The options for the import (currently none).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // export the memory TestMem to ZIP package c:\data\TestMem.ZIP
  if ( !usRC )
```

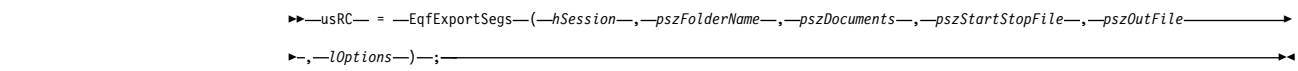
```
  {
    usRC = EqfExportMemInternalFormat( hSession, "TestMem",
    "C:\data\TestMem.ZIP", 0 );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfExportSegs

Purpose

EqfExportSegs lets you export segments within specific tag groups.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
PSZ	pszDocuments	List of documents or NULL
PSZ	pszStartStopFile	File name of the text file containing the list of start stop tags.The list of start/stop tags is a plain text file. Each text line of the file contains a start and stop tag separated by a comma. The start and stop tag can be enclosed in double-quotes. `Sample:` <title>,</title> " "," "
PSZ	pszOutFile	The name of the output file receiving the segments in the memory export format. The file is in Unicode (UTF-16) encoding.
LONG	lOptions	Options for the EqfExportSegs function: • OVERWRITE_OPT to overwite any existing output file • COMPLETE_IN_ONE_CALL to perform the export in one single call. Without using this option the function has to be called repetitively until the function return code is not CONTINUE_RC

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The function processed a small unit of work and is ready to process the next unit.
other	Error code (EQF message number). Use EqfGetLastError to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);
  // Export the segments in the title of the documents of folder test
   if ( !usRC )
{     usRC = EqfExportSergs(hSession, "test", NULL, "C:\MyDocs\taglist.txt",
                                "C:\MyDocs\output.exp", COMPLETE_IN_ONE_CALL );
} //endif

// terminate the session
EqfEndSession( hSession );
}
```

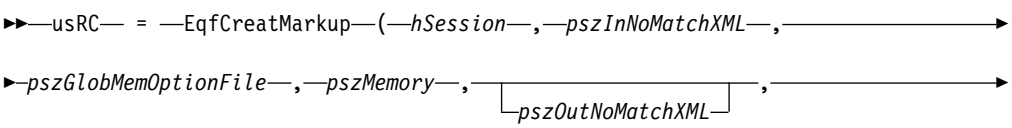
EqfFilterNoMatchFile

Purpose

The API call *EqfFilterNoMatchFile* Checks matches from a NOMATCH file against a memory and applies any Global Memory option file.

This API function looks up all matches contained in a NOMATCH file (in XML format) in the given memory, and applies the specified Global Memory option file on the memory proposals. The function creates a memory match word count, and writes any matches not found in the input memory to a new NOMATCH file. The new NOMATCH file can be in the XML format and/or the *.EXP format. The processing is done in small units, and the API call is to be called repetitively as long as the return code CONTINUE_RC is returned. To do the processing in one block, specify the option COMPLETE_IN_ONE_CALL_OPT. The word count report can be created in the XML format (use the option XML_OUTPUT_OPT) or in plain text format (use the option TEXT_OUTPUT_OPT). The word count report creation in plain text format is the default.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszInNoMatchXML	The fully qualified file name of the input NOMATCH file in XML format.
PSZ	pszGlobMemOptionFile	The fully qualified file name of the Global Memory option file.
PSZ	pszMemory	The name of the internal memory being used for the look-up.
PSZ	pszOutNoMatchXML	The fully qualified file name of the new NOMATCH file in the XML format (can be NULL when not used).
PSZ	pszOutNoMatchEXP	The fully qualified file name of the new NOMATCH file in the EXP format (can be NULL when not used).
PSZ	pszWordCountReport	The fully qualified file name of the created memory match word count report (can be NULL when not used).
LONG	lOptions	**The options for the processing**: • COMPLETE_IN_ONE_CALL_OPT to do the processing in one call (rather than doing the processing in small units). • TEXT_OUTPUT_OPT to create the word count report in plain text format (=default). • XML_OUTPUT_OPT to create the word count report in XML format.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The no match file has been filtered successfully.
10003 (CONTINUE_RC)	The processing has not completed yet. Call EqfFilterNoMatchFile again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  if ( !usRC )
  {
```

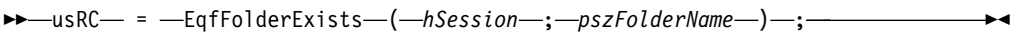
```
    // Filter the no match file NOMATCH.XML using the memory LookupMemory
    // and the global memory option file GlobMemOption.XML and write the
    // filtered no match file to NEWNOMATCH.XML. In addition create the
    // wordcount file WordCounts.XML in the XML format
    usRC = EqfFilterNoMatchFile( hSession, "C:\\NOMATCH.XML", "C:\\GlobMemOption.XML",
    "LookupMemory", "C:\\NEWNOMATCH.XML", NULL, "C:\\WordCounts.XML",
    COMPLETE_IN_ONE_CALL_OPT | XML_OUTPUT_OPT );
  } /* endif */
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfFolderExists

Purpose

EqfFolderExists checks if the given folder exists in OpenTM2.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder for which the existence is to be checked

Return code

USHORT

Value	Description
0 (NO_ERROR)	The specified dictionary exists in OpenTM2
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

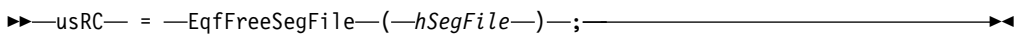
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );    usRC = EqfFolderExists( hSession, "MyFolder" );    // term
}
```

EqfFreeSegFile

Purpose

Releases the memory occupied by a file loaded into memory using EqfLoadSegFile.

Format



Parameters

Type	Parameter	Description
HPARSESEGFILE	hSegFile	Handle of loaded segmented file

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT   usRC = 0;
  HPARSSEGFILE *hSegFile = NULL;
  HSESSION hSession = 0L;
  PARSSEGMENTW Segment;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  if ( !usRC )
  {
    usRC = EqfBuildSegDocName( hSession, "SAMPLE1", "Document1",
                               1, szFileName );
  } //endif

  if ( !usRC )
  {
    usRC = EqfLoadSegFile( hSession, szFileName, &hSegFile );
    if ( !usRC )
    {
      usRC = EqfGetSegW( hSegFile, 1, &Segment );
      if ( !usRC )
      {
        wcslwr( Segment.szData );
        usRC = EqfUpdateSegW( hSegFile, 1, &Segment );
        if ( !usRC )
        {
          usRC = EqfWriteSegFile( hSegFile, szFileName );
        } //endif
      } //endif
      EqfFreeSegFile(hSegFile );
    } //endif
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

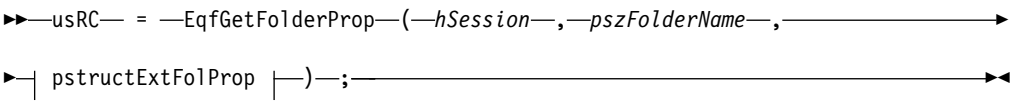
EqfGetFolderProp

Purpose

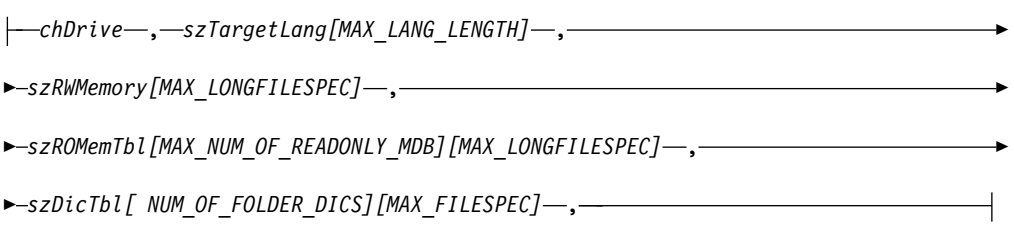
EqfGetFolderProp retrieves the following properties of the specified folder or subfolder:

- Target drive
- Target language
- Name of the read-write memory
- List of read-only memories
- List of dictionaries.

Format



structExtFolProp:



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
PEXTFOLPROP	pstructExtFolProp	See "Parameters for structExtFolProp" on page 95 for details.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  EXTFOLPROP FolderProps;
```

```
  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  // Retrieve properties of Folder Test.
  if ( !usRC )
  {
    usRC = EqfGetFolderProp(hSession,"Test",&FolderProps);
   } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

Parameters for structExtFolProp

```
typedef struct _EXTFOLPROP { CHAR chDrive; CHAR szTargetLang[MAX_LANG_LENGTH]; CHAR
        szRWMemory[MAX_LONGFILESPEC]; CHAR szROMemTbl[MAX_NUM_OF_READONLY_MDB][MAX_LONGFILESPEC];
        CHAR szDicTbl[ NUM_OF_FOLDER_DICS][MAX_FILESPEC]; } EXTFOLPROP, *PEXTFOLPROP;
```

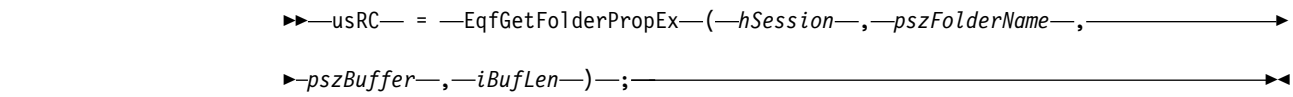
Type	Parameter	Description
CHAR	chDrive	Returns the target drive of the folder.
CHAR	szTargetLang [MAX_LANG_LENGTH]	Returns the target language.
CHAR	szRWMemory [MAX_LONGFILESPEC]	Returns the read-write memory.
CHAR	szROMemTbl [MAX_NUM_OF_READONLY_MDB] [MAX_LONGFILESPEC]	Returns a list of read-only memories.
CHAR	szDicTbl [NUM_OF_FOLDER_DICS] [MAX_FILESPEC]	Returns the list of dictionaries.

EqfGetFolderPropEx

Purpose

EqfGetFolderPropEx retrieves the requested value from the properties of the specified folder or subfolder.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.

Type	Parameter	Description
PSZ	pszKey	Name of the requested property value: • ANALYSISPROFILE to retrieve the analysis profile name • COUNTINGPROFILE to retrieve the counting profile name • DESCRIPTION to retrieve the folder description • DICTIONARIES to retrieve the list of dictionaries • DRIVE to retrieve the folder drive • MEMORY to retrieve the folder memory • ROMEMORIES to retrieve the list of rea-only memories • SHIPMENT to retrieve the folder shipment number • SOURCELANGUAGE to retrieve the folder source language • TARGETLANGUAGE to retrieve the folder target language
PSZ	pszBuffer	Points to a buffer receiving the requested value.
int	iBufLen	Length of the buffer in number of bytes.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

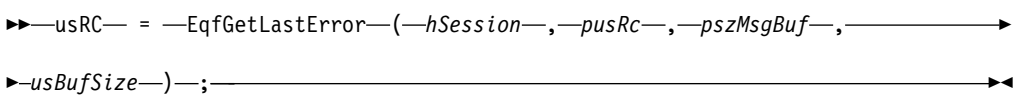
```
{
  USHORT usRC = 0; HSESSION hSession = 0L; // start the Eqf calling interface
  session usRC = EqfStartSession(&hSession); // Retrieve the source language of folder Test.
  if ( !usRC )
  {
    CHAR szBuffer[100]; usRC =
    EqfGetFolderPropEx(hSession,"Test","SOURCELANGUAGE", szBuffer, sizeof(szBuffer) );
  } //endif
  // terminate the session EqfEndSession( hSession );
}
```

EqfGetLastError

Purpose

EqfGetLastError receives the text of the last error message.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PUSHORT	pusRc	The OpenTM2 return code (message number).
PSZ	pszMsgBuf	An allocated area to receive the message text.
USHORT	usBufSize	The size of the preallocated buffer.

Return code

USHORT

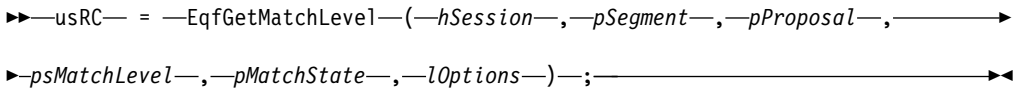
Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number).

EqfGetMatchLevel

Purpose

The API call *EqfGetMatchLevel* computes the match level of the given proposal for the supplied segment. The segment data and the proposal is passed to the function using a EQFSEGINFO structure.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PEQFSEGINFO	pSegment	Pointer to an EQFSEGINFO structure containing the segment data. Note: The target part has not to be filled.
PEQFSEGINFO	pProposal	Pointer to an EQFSEGINFO structure containing the proposal data.
PSHORT	psMatchLevel	Pointer to a SHORT variable receiving the match level. The returned match level is in the range from 0 to 100.

Type	Parameter	Description
PSHORT	pMatchState	Pointer to a SHORT variable receiving the match state. The returned match state can be: • REPLACE_MATCHSTATE for a replace match • FUZZYREPLACE_MATCHSTATE for a fuzzy replace matche • FUZZY_MATCHSTATE for a fuzzy matche • NONE_MATCH if theproposal is no match at all • EXACT_MATCHSTATE for an exact matche • EXACTEXACT_MATCHSTATE for an exact match coming from the same document and same segment.
LONG	lOptions	The options to be used for the function: • NO_GENERIC_INLINETAG_REPL_OPT if set the function “generic inline tag replacement” is not used • USE_GENERIC_INLINETAG_REPL_OPT if set the function “generic inline tag replacement” is always used If none of these values is specified, the settings from the “System preferences” are used.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use “EqfGetLastError” on page 96 to retrieve the complete error information.

EQFSEGINFO structure

Type	Field	Description
WCHAR [2048]	szSource	The proposal source in UTF-16 encoding
WCHAR [2048]	szTarget	The proposal target in UTF-16 encoding
LONG	lSegNumber	The segment number
CHAR [256]	szDocument	The name of the document
CHAR [20]	szSourceLanguage	The source language of the proposal
CHAR [20]	szTargetLanguage	The target language of the proposal
CHAR [13]	szMarkup	The name of the markup table

Code sample

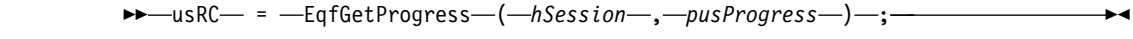
```
// the segment data from the document
EQFSEGINFO SegmentData =
{
  L"The <strong>IBM Websphere Translation Server</strong> performs automatic translations.",
  L"",
  1,
  "document.idd",
  "English(U.S.)",
  "German(DPAnat)",
  "IBMIDDOC"
};

// data for a fuzzy match
EQFSEGINFO FuzzyMatch =
{
  L"The <strong>IBM Websphere Translation Server</strong> does automatic translations.",
  L"Der <strong>IBM Websphere Translation Server</strong> macht automatische Uebersetzungen.",
  7,
  "anotherdoc.idd",
  "English(U.S.)",
  "German(DPAnat)",
  "IBMIDDOC"
};

{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // check the match level of the match in FuzzyMatch
  if ( !usRC )
  {
    SHORT sMatchLevel = 0;
    EQFMATCHSTATE MatchState;

    usRC = EqfGetMatchLevel( hSession, &SegmentData , &FuzzyMatch, &sMatchLevel, &MatchState, 0 );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfGetProgress

Purpose

Get the progress of the currently performed function. The progress values returned are in the range from 0 to 100. This API call can only be used for nonDDE API processes which are called repeatedly until the function has been completed (e.g. *EqfImportFolder*).

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PUSHORT	pusProgress	Address of a variable receiving the current progress value

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

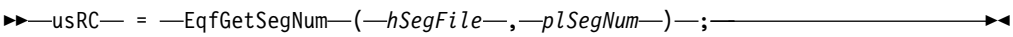
```
{
  USHORT  usRC = 0;
  HSESSION hSession = 0L;
 HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  if ( !usRC )
  {
    do
    {
            usRC = EqfImportFolder( hSession, "SAMPLE1", 'C', '\0', WITHMEM_OPT );
            if ( usRC == CONTINUE_RC )
            {
                EqfGetProgress ( hSession, &usProgress );
            } //endif
  } while ( usRC == CONTINUE_RC );
  } // endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfGetSegNum

Purpose

Get the number of segments contained in a segmented file loaded into memory using *EqfLoadSegFile*.

Format



Parameters

Type	Parameter	Description
HPARSESEGFILE	hSegFile	Handle of loaded segment file

Type	Parameter	Description
PLONG	pISegNum	Pointer to a buffer receiving the number of segments in the loaded file

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

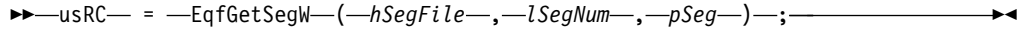
```
{
  USHORT   usRC = 0;
  HPARSSEGFILE *hSegFile = NULL;
  HSESSION hSession = 0L;
  LONG     lNumberOfSegments = 0;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  if ( !usRC )
  {
    usRC = EqfBuildSegDocName( hSession, "SAMPLE1", "Document1",
                               1, szFileName );
  } //endif

  if ( !usRC )
  {
    usRC = EqfLoadSegFile( hSession, szFileName, &hSegFile );
    if ( !usRC )
    {
      usRC = EqfGetSegNum( hSegFile, &lNumberOfSegments);
      EqfFreeSegFile(hSegFile );
    } //endif
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfGetSegW

Purpose

Get the data of a specific segment from a segmented file loaded into memory using *EqfLoadSegFile*.

Format



Parameters

Type	Parameter	Description
HPARSESEGFILE	hSegFile	Handle of loaded segment file

Type	Parameter	Description
LONG	ISegNum	Number of segment being rereived
PPARSESEGMENTW	pSeg	Pointer to structure receiving the segment data

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

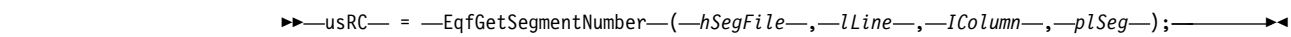
```
{
  USHORT   usRC = 0;
  HPARSSEGFILE *hSegFile = NULL;
  HSESSION hSession = 0L;
  PARSSEGMENTW Segment;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  if ( !usRC )
  {
    usRC = EqfBuildSegDocName( hSession, "SAMPLE1", "Document1",
                               1, szFileName );
  } //endif

  if ( !usRC )
  {
    usRC = EqfLoadSegFile( hSession, szFileName, &hSegFile );
    if ( !usRC )
    {
      usRC = EqfGetSegW( hSegFile, 1, &Segment );
      if ( !usRC )
      {
        wcslwr( Segment.szData );
        usRC = EqfUpdateSegW( hSegFile, 1, &Segment );
        if ( !usRC )
        {
          usRC = EqfWriteSegFile( hSegFile, szFileName );
        } //endif
      } //endif
      EqfFreeSegFile(hSegFile );
    } //endif
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfGetSegmentNumber

Purpose

EqfGetSegmentNumber computes the number of the segment to which the character at the given line and column position belongs to.

Format



Parameters

Type	Parameter	Description
HPARSESEGFILE	hSegFile	The handle of a segmented file as returned by function *EqfLoadSegFile.*
LONG	ILine	Number of the line for which the segment number is requested
LONG	IColumn	Column position of the segment within the line
PLONG	plSeg	Pointer to a LONG buffer which reveives the segment number matching the line and column number

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
10009 (NOMATCHINGSEGMENT_RC)	There is no segment with the given position (either the line number or the column number is out of range)
10008 (INVALIDFILEHANDLE_RC)	The file handle hSegFile is invalid
other	Error code (EQF message number). Use function EqfGetLastError to retrieve complete error information.

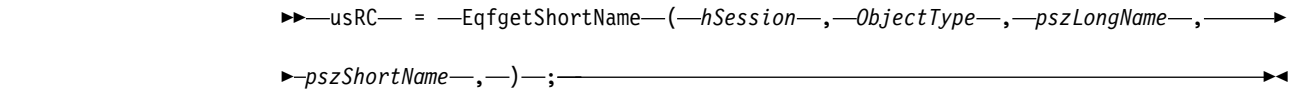
EqfGetShortName

Purpose

The API call *EqfGetShortName* is used to get the internally used short name for a folder, dictionary, Translation Memory, or document.

Attention: this API function will only work for the older OpenTM2 plugins. Newer plugins will (hopefully) not use short names anymore.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
USHORT	ObjectType	**Type of the object being processed**: • FOLDER_OBJ object is a folder. • MEMORY_OBJ object is a Translation Memory. • DICT_OBJ object is a dictionary. • DOCUMENT_OBJ object is a document.
PSZ	pszLongName	Long name of the object for documents also the folder name has to be specified in the form foldername:documentname.
PSZ	pszShortName	Pointer to a buffer for the returned short name.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The MT flags in the memory have been cleared successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

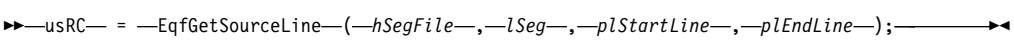
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  if ( !usRC )
  {
    char szShortName[31]; // buffer for folder short name

    // Get the short name of folder "MyFolder" and write the short name
    // to the variable szShortName
    usRC = EqfGetShortName( hSession, FOLDER_OBJ, "MyFolder", szShortName );
  } /* endif */
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfGetSourceLine

Purpose

EqfGetSourceLine computes the start line and the end line of the given segment based on the linefeeds contained in the document.

Format



Parameters

Type	Parameter	Description
HPARSESEGFILE	hSegFile	The handle of a segmented file as returned by function *EqfLoadSegFile*.
LONG	lSeg	Number of segment for which the source line information is requested
PLONG	plStartLine	Pointer to a LONG buffer which receives the starting line number of the segment
PLONG	plEndLine	Pointer to a LONG buffer which receives the end line number of the segment

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
10006 (SEGMENTISJOINED_RC)	The given segment is joined to a previous segment is not visible in the document
10007 (INVALIDSEGMENT_RC)	The given segment number is invalid or out of range
10008 (INVALIDFILEHANDLE_RC)	The file handle hSegFile is invalid
other	Error code (EQF message number). Use function EqfGetLastError to retrieve complete error information

EqfGetSysLanguage

Purpose

EqfGetSys Language allows to retrieve the currently active default target language of OpenTM2.

Format

```
►►—usRC— = —EqfGetSysLanguage—(—hSession—,—pszSysLanguage—————————————►◄
```

Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszSystemLanguage	Buffer provided to contain the system language string at output. The length of the buffer has to be at least 20 characters.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetSysLanguage* to retrieve the complete error information.

Code sample

```
{
                    USHORT usRC = 0;
                    HSESSION hSession = 0L;
                    CHAR   chSystemLanguage[20];

                    // start the Eqf calling interface session
                    usRC = EqfStartSession( hSession );

                    // get the system language
                    if ( !usRC )
                    {
                        usRC = EqfGetSysLanguage( hSession, chSystemLanguage );
                    } /* endif */

                    // terminate the session
                    EqfEndSession( hSession );
                }
```

EqfGetVersion

Purpose

EqfGetVersion retrieves the version info of OpenTM2.

Format

Parameters

—none —

Return code

ULONG

Value	Description
ulVersion	The version of OpenTM2 in a byte array, see the code sample for details how to access the version info.

Code sample

```
                    #include <stdlib.h>
                    #include "eqf_api.h"

                    int main( int argc, char *argv[], char *envp[] )
                    {
```

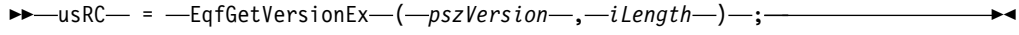
```
  BYTE abVersion[4];
  ULONG ulVersion = EQFGETVERSION();

  memcpy( abVersion, &ulVersion, sizeof(ULONG) );

  printf( "TM Version    %d\n", (short)abVersion[0] );
  printf( "TM Release    %d\n", (short)abVersion[1] );
  printf( "TM Subrelease %d\n", (short)abVersion[2] );
  printf( "TM Build      %d\n", (short)abVersion[3] );
} /* end of main */
```

EqfGetVersionEx

Purpose

The API call *EqfGetVersionEx* is used to get the OpenTM2 version information.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszVersion	Pointer to a buffer for the version string.
int	iLength	Size of the buffer for the version string.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The version string was returned successfully.

Code sample

```
{
  char szVersion[128];

  // get the current version of OpenTM2 into buffer szVersion
  EqfGetVersionEx( pszVersion, sizeof(szVersion) );
}
```

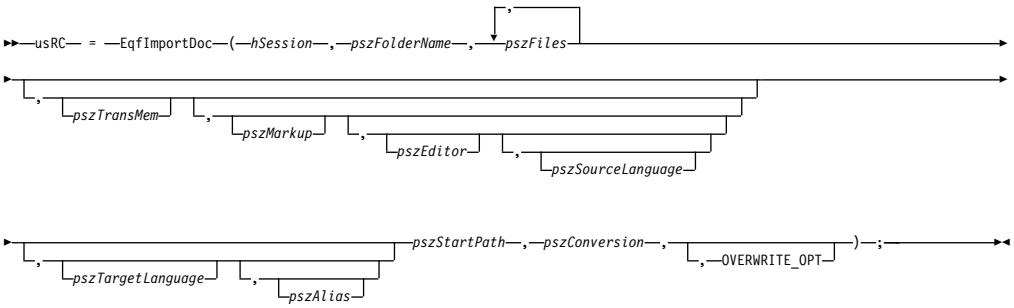
EqfImportDoc

Purpose

EqfImportDoc imports one or more documents and sets the document properties to the specified values. The specified values apply to all documents to be imported. If a document needs different settings, for example a different markup, import it separately.

This function performs the import in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder receiving the documents.
PSZ	pszStartPath	The start path if the documents are to be imported with relative path information. If a start path is specified, the files in the *pszFiles* list only contain the relative path.
PSZ	pszFiles	The fully qualified name of the documents to be imported.
PSZ	pszTransMem	The name of the Translation Memory to be used for the document, if different from that of the folder.
PSZ	pszMarkup	The name of the markup table to be used for the document, if different from that of the folder.
PSZ	pszEditor	The name of the editor to be used for the document, if different from that of the folder.
PSZ	pszSourceLanguage	The name of the source language to be used for the document, if different from that of the folder.
PSZ	pszTargetLanguage	The name of the target language to be used for the document, if different from that of the folder.
PSZ	pszAlias	The alias name for the document.
LONG	lOptions	The option to be used for the document import: OVERWRITE_OPT to replace existing documents.
PSZ	pszConversion	Conversion to be used for document or NULL

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The document import has not completed yet. Call *EqfImportDoc* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Import the documents DOC1.TXT and DOC2.TXT into folder SAMPLE1
  // and overwrite any existing documents, the format of the documents
  // is to EQFASCII for all other settings the folder settings will be
  // used
  if ( !usRC )
  {
    do
    {
      usRC = EqfImportDoc( hSession, "SAMPLE1", NULL,
                           "C:\\DOC1.TXT,C:\\DOC2.TXT",
                           NULL, "EQFASCII", NULL, NULL, NULL, NULL,
                           OVERWRITE_OPT );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

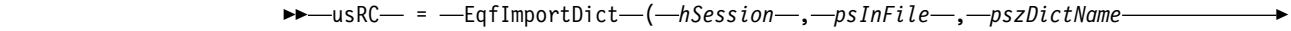
EqfImportDict

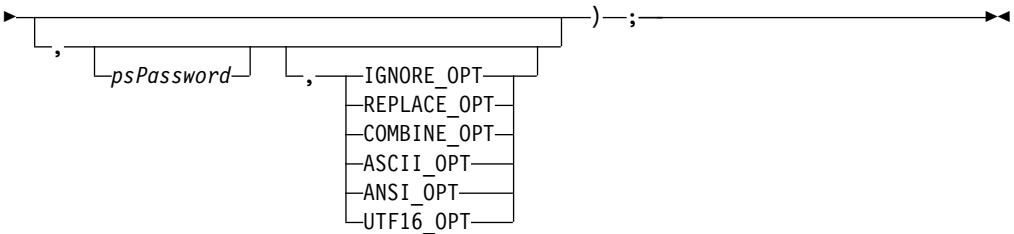
Purpose

EqfImportDict imports a dictionary in SGML dictionary.

This function performs the import in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format





Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszInFile	The fully qualified name of the SGML file to be imported.
PSZ	psDictName	The name of the dictionary to be exported.
PSZ	pszPassword	The dictionary password. Only required if the dictionary exists already and is protected.
LONG	lOptions	The options to be used for the merge of entries during the import: • IGNORE_OPT • REPLACE_OPT • COMBINE_OPT • ASCII_OPT • ANSI_OPT • UTF16_OPT

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The dictionary import has not completed yet. Call *EqfImportDict* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Import the SGML file C:\DICT1.SGM into dictionary ENGLGERM
  // and replace existing entries with the imported data
 if ( !usRC )
  {
    do
    {
```

```
      usRC = EqfImportDict( hSession, "C:\\DICT1.SGN",
          "ENGLGERM", NULL, REPLACE_OPT )
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

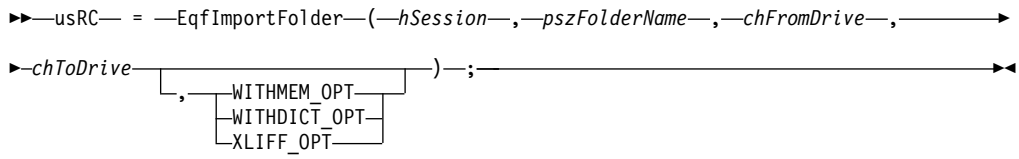
EqfImportFolder

Purpose

EqfImportFolder imports a folder from a specific drive to the specified OpenTM2 drive. The path from which the folder is imported is always `\otm\export`.

This function performs the import in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder to be imported.
CHAR	chFromDrive	The drive from which the folder is exported.
CHAR	chToDrive	The target drive for the imported folder. If omitted, the primary EQF drive is used. The drive must be the primary EQF drive or one of the secondary EQF drives defined in the “Configure Drives” window.
LONG	lOptions	The options to be used for the export: WITHMEM_OPT WITHDICT_OPT XLIFF_OPT These options correspond to those in the “Import Folder” window (see). You can combine the constants using OR.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.

Value	Description
CONTINUE_RC	The folder import has not completed yet. Call *EqfImportFolder* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Import the folder SAMPLE1 from drive C: to the primary Eqf
  // system drive, import the folder with Translation Memory databases
  // and dictionaries
  if ( !usRC )
  {
    do
    {
      usRC = EqfImportFolder( hSession, "SAMPLE1", 'C',
                              '\0',     // use primary Eqf drive
                              WITHDICT_OPT | WITHMEM_OPT );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

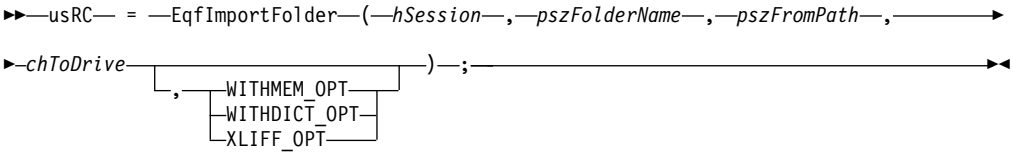
EqfImportFolderFP

Purpose

EqfImportFolderFP imports a folder from a specific path to the specified OpenTM2 drive.

This function performs the import in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder to be imported.
PSZ	pszFromPath	The path from which the folder is exported.

Type	Parameter	Description
CHAR	chToDrive	The target drive for the imported folder. If omitted, the primary EQF drive is used. The drive must be the primary EQF drive or one of the secondary EQF drives defined in the "Configure Drives" window.
LONG	lOptions	The options to be used for the export: WITHMEM_OPT WITHDICT_OPT XLIFF_OPT These options correspond to those in the "Import Folder" window (see). You can combine the constants using OR.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The folder import has not completed yet. Call *EqfImportFolderFP* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Import the folder SAMPLE1 from path C:\PROJECT to the primary
  // Eqf system drive, import the folder with Translation Memory
  // databases and dictionaries
  if ( !usRC )
  {
    do
    {
      usRC = EqfImportFolderFP( hSession, "SAMPLE1", 'C:\PROJECT',
                                '\0',       // use primary Eqf drive
                                WITHDICT_OPT | WITHMEM_OPT );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

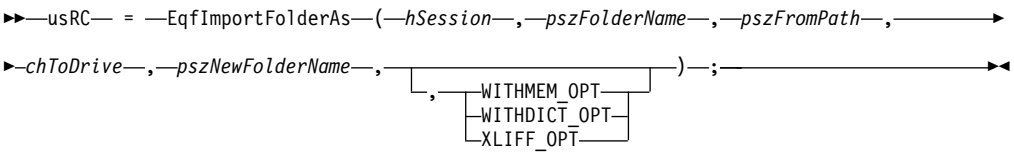
EqfImportFolderAs

Purpose

EqfImportFolderAs imports a folder from a specific path to the specified OpenTM2 drive, and the folder name can be changed druing the import.

This function performs the import in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder to be imported.
PSZ	pszFromPath	The path from which the folder is imported from.
CHAR	chToDrive	The target drive for the imported folder. If omitted, the primary OTM drive is used. The drive must be the primary OTM drive or one of the secondary OTM drives defined in the "Configure Drives" window.
PSZ	pszNewFolderName	The **new** name of the folder.
LONG	lOptions	The options to be used for the export: WITHMEM_OPT WITHDICT_OPT XLIFF_OPT These options correspond to those in the "Import Folder" window (see). You can combine the constants using OR.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The folder import has not completed yet. Call *EqfImportFolderFP* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0; HSESSION hSession = 0L;
  // start the Eqf calling interface
  session usRC = EqfStartSession( &hSession );
  // Import the folder SAMPLE1 from path C:\PROJECT to the primary
  // Eqf system drive, import the folder with Translation Memory
  // databases and dictionaries
  if ( !usRC )
  {
    do
      {
        usRC = EqfImportFolderFP( hSession, "SAMPLE1", "C:\PROJECT",
               '\0',              // use primary OTM drive
               "myNewFolderName", // give a new folder name
               WITHDICT_OPT | WITHMEM_OPT );
      } while ( usRC == CONTINUE_RC );
  } /* endif */
  // terminate the session
  EqfEndSession( hSession );
}
```

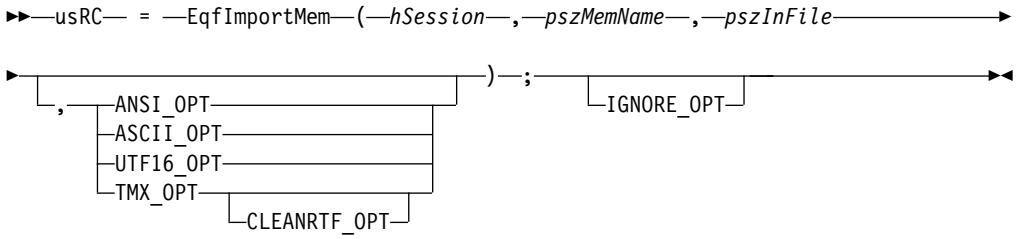
EqfImportMem

Purpose

EqfImportMem imports a Translation Memory into OpenTM2.

This function performs the import in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemName	The name of the Translation Memory to be imported. If a Translation Memory with this name already exists, the imported data is merged into the existing Translation Memory.
PSZ	pszInFile	The fully qualified name of the file containing the Translation Memory data.

Type	Parameter	Description
LONG	lOptions	The options to be used for the Translation Memory import: ANSI_OPT (Export/Import in Ansi) ASCII_OPT (Export/Import in ASCII) UTF16_OPT (Export/Import in Unicode UTF-16) TMX_OPT (Import in TMX format) CLEANRTF_OPT can be used together with the TMX_OPT to remove RTF tags from the imported data IGNORE_OPT (Ignore invalid segments and continue with the import)

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The Translation Memory import has not completed yet. Call *EqfImportMem* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // Import the external Translation Memory MEM1.EXP into Translation
  // Memory MEMDB1
  if ( !usRC )
  {
    do
    {
      usRC = EqfImportMem( hSession, "MEMDB1", "C:\\MEM1.EXP", 0L );
    } while ( usRC == CONTINUE_RC );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfImportMemEx

Purpose

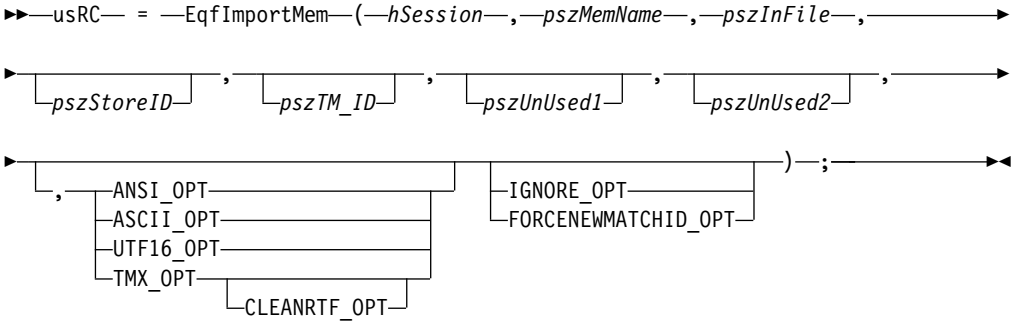
EqfImportMemEx imports a Translation Memory into OpenTM2.

This API-call imports a translation memory the same way as with **EqfImportMem**. In addition, a **match segment ID** is created, if the memory proposal does not

contain a match segment ID yet, and one (or both) of the new parameters **pszStoreID** and **pszTM_ID** has/have been specified. Using the new option **FORCENEWMATCHID_OPT** any existing match segment ID is ignored and a new match segment ID is always created.

This API-call performs the import in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemName	The name of the Translation Memory to be imported. If a Translation Memory with this name already exists, the imported data is merged into the existing Translation Memory.
PSZ	pszInFile	The fully qualified name of the file containing the Translation Memory data.
PSZ	pszStoreID	Identifier of the origin of the translation memory.
PSZ	pszTM_ID	Identifier for the memory within the StoreID.
PSZ	pszUnUsed1	Unused parameter, free for future enhancements.
PSZ	pszUnUsed2	Unused parameter, free for future enhancements.

Type	Parameter	Description
LONG	lOptions	The options to be used for the Translation Memory import: • Import Mode: – ANSI_OPT (The translation memory is ANSI encoded). – ASCII_OPT (The translation memory is ASCII encoded). – TMX_OPT (The translation memory is a TMX memory). – UTF16_OPT (The translation memory is UTF-16 encoded). – XLIFF_MT_OPT (The translation memory is a XLIFF memory). • Markup Table Handling: – CANCEL_UNKNOWN_MARKUP_OPT (Cancel import if unknown markup detected). – GENRIC_UNKNOWN_MARKUP_OPT (Put a generic markup table to unknown markup). – SKIP_UNKNOWN_MARKUP_OPT (Skip segments with unknown markup). • Other: – CLEANRTF_OPT can be used together with the **TMX_OPT** to remove **RTF tags** from the imported data. – FORCENEWMATCHID_OPT (Ignore existing match segment ID's. New match segment IDs are always created). – IGNORE_OPT (Ignore invalid segments and continue with the import).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The translation memory has been imported successfully.
10003 (CONTINUE_RC)	The Translation Memory import has not completed yet. Call *EqfImportMemEX* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
USHORT usRC = 0;
HSESSION hSession = 0L;
// start the OpenTM2 API session
usRC = EqfStartSession( &hSession );
// Import the external Translation Memory MEM1.EXP into Translation
// Memory MEMDB1 and create a match segment ID using the provided
```

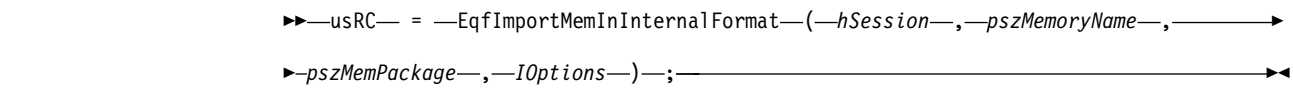
```
// StoreID "TMB" and the TM_ID "ACP005AV2"
if ( !usRC )
{
  do
  {
    usRC = EqfImportMemEx( hSession, "MEMDB1", "C:\\MEM1.EXP", "TMB",
                           "ACP005AV2", NULL, NULL, 0L );
  } while ( usRC == CONTINUE_RC );
} /* endif */
// terminate the session
EqfEndSession( hSession );
}
```

EqfImportMemInInternalFormat

Purpose

EqfImportMemInInternalFormat imports a Translation Memory into OpenTM2 using the internal memory files from a Zip package.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemoryName	The name of the Translation Memory.
PSZ	pszMemPackage	The fully qualified name of the ZIP package containing the internal Translation Memory files.
LONG	lOptions	The options for the import (currently none).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );
```

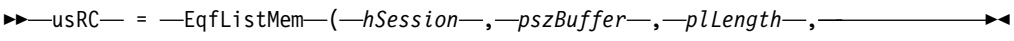
```
  // import the memory TestMem from ZIP package c:\data\TestMem.ZIP
  if ( !usRC )
  {
    usRC = EqfImportMemInternalFormat( hSession, "TestMem",
           "C:\data\TestMem.ZIP", 0 );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession
  );
}
```

EqfListMem

Purpose

EqfListMem generates a list of all names of all available Translation Memory databases.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszBuffer	Pointer to a buffer receiving the comma separated list of Translation Memory names, or NULL if the required length of the buffer is requested.
PLONG	plLength	Pointer to a variable containing the size of the pszBufferArea, on return this variable is filled with the length of the Translation Memory name list or, if pszBuffer is NULL, with the required size for the buffer.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  PSZ pszBuffer = NULL;
  LONG lSize = 0;
```

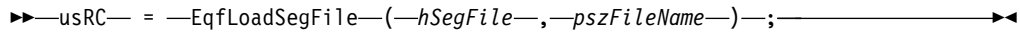
```
  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // get the required length for the memory name buffer
  if ( !usRC )
  {
    usRC = EqfListMem( hSession, pszBuffer, &lSize );
  } /* endif */

  // allocate a buffer for the memory names
  if ( !usRC )
  {
    pszBuffer = new char[lSize];
  } /* endif */

  // get the list of the memory names
  if ( !usRC )
  {
    usRC = EqfListMem( hSession, pszBuffer, &lSize );
  } /* endif */

  if ( pszBuffer != NULL ) delete pszBuffer;

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfLoadSegFile

Purpose

Loads a segmented OpenTM2 document file into memory. The segments of the loaded file can be accessed using the *EqfGetSegW* API.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemName	Fully qualified file name
HPARSESEGFILE *	hSegFile	Points to the buffer receiving the handle of the loaded segmented file

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

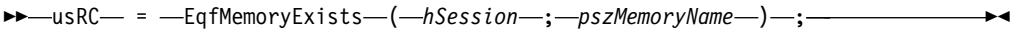
```
{
  USHORT   usRC = 0;
  HPARSSEGFILE *hSegFile = NULL;
  HSESSION hSession = 0L;
  PARSSEGMENTW Segment;

  // start the Eqf calling interface session
  usRC = EqfStartSession(&hSession);

  if ( !usRC )
  {
    usRC = EqfBuildSegDocName( hSession, "SAMPLE1", "Document1",
                               1, szFileName );
  } //endif

  if ( !usRC )
  {
    usRC = EqfLoadSegFile( hSession, szFileName, &hSegFile );
    if ( !usRC )
    {
      usRC = EqfGetSegW( hSegFile, 1, &Segment );
      if ( !usRC )
      {
        wcslwr( Segment.szData );
        usRC = EqfUpdateSegW( hSegFile, 1, &Segment );
        if ( !usRC )
        {
          usRC = EqfWriteSegFile( hSegFile, szFileName );
        } //endif
      } //endif
      EqfFreeSegFile(hSegFile );
    } //endif
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfMemoryExists

Purpose

EqfMemoryExists checks if the given translation memory exists in OpenTM2.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemoryName	The name of the translation memory for which the existence is to be checked

Return code

USHORT

Value	Description
0 (NO_ERROR)	The specified dictionary exists in OpenTM2
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

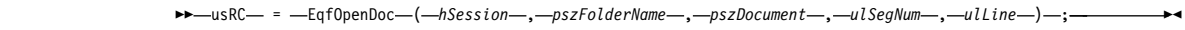
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );    usRC = EqfMemoryExists( hSession, "MyMemory" );    // te
}
```

EqfOpenDoc

Purpose

opens a document at the given segment or line in the TranslationEnvironment.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents to be opened.
PSZ	pszDocument	The name of the document being opened.
ULONG	ulSegNum	The segment number to go to (0 if not used)
ULONG	ulLine	The line to go to (ulSegNum must be 0 if a line number is specified)

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

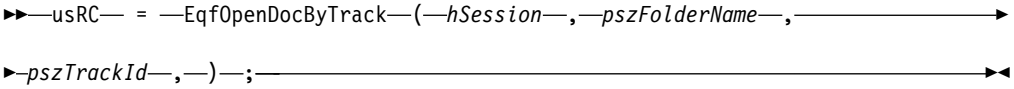
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );
// Open the document DOC1.TXT in folder SAMPLE1 at line
42   if ( !usRC )
{
  usRC = EqfOpenDoc( hSession, "SAMPLE1",
"DOC1.TXT", 0, 42 );   }
 /* endif */
// terminate the session
EqfEndSession( hSession );
}
```

EqfOpenDocByTrack

Purpose

The API call *EqfOpenDocByTrack* opens a document in the OpenTM2 editor and positions to a specific segment based on the specified TVT tracking Id.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder.
PSZ	pszTrackId	The tracking-ID of a segment within a specific document in the folder.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The document has been opened successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  if ( !usRC )
```

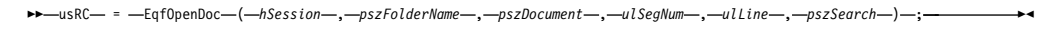
```
  {
    wchar_t *pszTrackID = L"1A:FF3";

    // open a document in folder "MyFolder" and position to a specific
    // segment based on the provided TVT track ID
    usRC = EqfOpenDocByTrack( hSession, "MyFolder", pszTrackID );
  } /* endif */
  // terminate the session
  EqfEndSession( hSession );
}
```

EqfOpenDocEx

Purpose

opens a document at the given segment or line or the first location of a specific search string in the Translation Environment.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents to be opened.
PSZ	pszDocument	The name of the document being opened.
ULONG	ulSegNum	The segment number to go to (0 if not used)
ULONG	ulLine	The line to go to (ulSegNum must be 0 if a line number is specified)
PSZ_W	pszSearch	Points to search string in UTF-16 encoding, if specified (and ulSegNum and ulLine are zero) the specified search string is searched in the opened document and the segment containing the first occurence of the search string is activated

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the Eqf calling interface
  session usRC = EqfStartSession( &hSession );
```

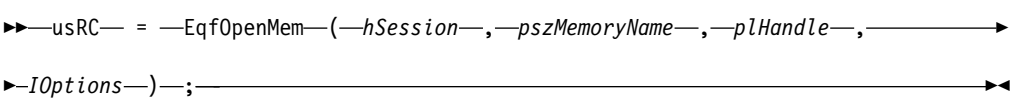
```
  // Open the document DOC1.TXT in folder SAMPLE1 at the first occurence of string "error"
  if ( !usRC )
  {
    usRC = EqfOpenDocEx(hSession, "SAMPLE1", "DOC1.TXT", 0, 0, L"error" );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfOpenMem

Purpose

EqfOpenMem opens a Translation Memory for searching or updating proposals.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemoryName	The name of the Translation Memory.
PLONG	plHandle	Pointer to a long value receiving the handle of the opened Translation Memory or -1 in case of failures.
LONG	lOptions	The options for the opening (currently none).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // open the memory TestMem
  if ( !usRC )
  {
    LONG lHandle = 0;
```

```
    usRC = EqfOpenMem( hSession, "TestMem", &lHandle, 0 );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

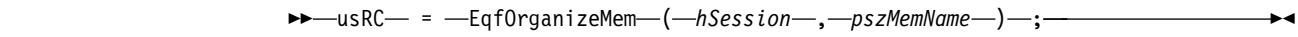
EqfOrganizeMem

Purpose

EqfOrganizeMem organizes the specified Translation Memory.

This function performs the organization in small units. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszMemName	The name of the Translation Memory to be organized.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The organization of the Translation Memory has not completed yet. Call *EqfOrganizeMem* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // organize the Translation Memory MEMDB1
  if ( !usRC )
  {
    do
    {
      usRC = EqfOrganizeMem( hSession, "MEMDB1" );
    } while ( usRC == CONTINUE_RC );
  } /* endif */
```

```
  // terminate the session
  EqfEndSession( hSession );
}
```

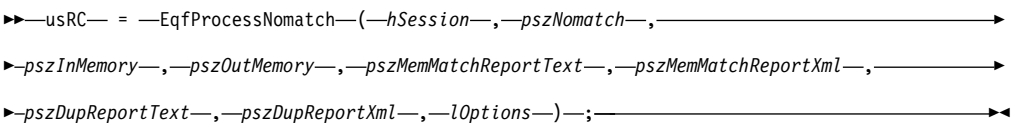
EqfProcessNomatch

Purpose

The API call *EqfProcessNomatch* reads one or more SNOMATCH files (created using the analysis option "Create file containing untranslated segments") and looks up the segments contained in the SNOMATCH files in the input memory. Each matching proposal (exact and fuzzy match) is written to the output memory. The API call creates a memory match word count and a duplicate word count for the segments in the SNOMATCH files. The word count reports can be created in text and XML form.

This function performs the processing in small units unless told to complete in one call using the COMPLETE_IN_ONE_CALL_OPT flag. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszNomatch	The specification for the SNOMATCH files to use for the processing. This parameter is evaluated in the following way: **the specified value contains wildcard characters** the specified value is used as fully qualified search pattern for the SNOMATCH FILES to be used e.g. "C:\OTM\TEST.F00\SNOMATCH\A*.*" **the specified value contains path delimiters** the specified value is used as fully qualified name of the SNOMATCH file to process, if the specified value points to a directory all files in the directory are processed e.g. "C:\OTM\TEST.F00\SNOMATCH\File1.txt", "C:\OTM\TEST.F00\SNOMATCH" **the value contains no path delimiters** the specified value is used as name of a TM folder, all SNOMATCH files contained in the SNOMATCH directory of this folder are processed e.g. "TEST"

Type	Parameter	Description
PSZ	pszInMemory	The name of the input memory (TM internal)
PSZ	pszOutMemory	The name of an existing or new internal memory receiving the relevant proposals from the input memory
PSZ	pszMemMatchReportText	The fully qualified name for the memory match word count report in text format, specify NULL if no report of this type should be created
PSZ	pszMemMatchReportXml	The fully qualified name for the memory match word count report in XML format, specify NULL if no report of this type should be created
PSZ	pszDupReportText	The fully qualified name for the duplicate word count report in text format, specify NULL if no report of this type should be created
PSZ	pszDupReportXml	The fully qualified name for the duplicate word count report in XML format, specify NULL if no report of this type should be created
LONG	lOptions	The option(s) to be used for the processing: **COMPLETE_IN_ONE_CALL_OPT** If set the API call does not return after each processing step but stays in the API call until the function has been completed **RESPECTCRLF_OPT** If set memory proposals having different linebreaks are not used as exact match The options can be combined by using the logical OR operator

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The SNOMATCH processing is not complete yet. Call *EqfProcessNomatch* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
HSESSION hSession;
USHORT usRC;

usRC = EqfStartSession( &hSession );
usRC = EqfProcessNomatch( hSession, "TestFolder",
          "PrevMemory", "NewMemory", NULL,
          "C:\Reports\MemMatch.XML", NULL,
          "C:\Reports\Dups.XML, COMPLETE_IN_ONE_CALL_OPT);
usRC = EqfEndSession( hSession );
```

The API *EqfProcessNomatch* is called to process all SNOMATCH files of folder **"TestFolder"**, the segments are looked up ion the memory **"PrevMemory"** and any relevant matches found are written to the memory **"NewMemory"**, the memory match count in XML format will be stored under **"C:\Reports\MemMatch.XML"** and the XML duplicate word count will be stored under **"C:\Reports\Dups.XML"**, the text versions of the reports are not being used. The API call will complete in one call.

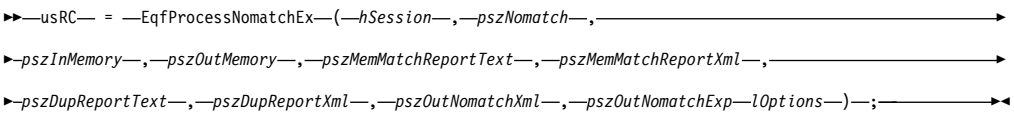
EqfProcessNomatchEx

Purpose

The API call *EqfProcessNomatchEx* reads one or more **SNOMATCH** files (created using the analysis option "Create file containing untranslated segments") and looks up the segments contained in the **SNOMATCH** files in the input memory. Each matching proposal (exact and fuzzy match) is written to the output memory. The API call creates a memory match word count, a duplicate word count for the segments in the **SNOMATCH** files, and files containing all segments which have no memory match . The word count reports can be created in text and XML form.

This function performs the processing in small units unless told to complete in one call using the COMPLETE_IN_ONE_CALL_OPT flag. Call it repetitively until it returns a return code other than CONTINUE_RC.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.

Type	Parameter	Description
PSZ	pszNomatch	The specification for the SNOMATCH files to use for the processing. This parameter is evaluated in the following way: **the specified value contains wildcard characters** the specified value is used as fully qualified search pattern for the SNOMATCH FILES to be used e.g. "C:\OTM\TEST.F00\SNOMATCH\A*.*" **the specified value contains path delimiters** the specified value is used as fully qualified name of the SNOMATCH file to process, if the specified value points to a directory all files in the directory are processed e.g. "C:\OTM\TEST.F00\SNOMATCH\File1.txt", "C:\OTM\TEST.F00\SNOMATCH" **the value contains no path delimiters** the specified value is used as name of a TM folder, all SNOMATCH files contained in the SNOMATCH directory of this folder are processed e.g. "TEST"
PSZ	pszInMemory	The name of the input memory (TM internal)
PSZ	pszOutMemory	The name of an existing or new internal memory receiving the relevant proposals from the input memory
PSZ	pszMemMatchReportText	The fully qualified name for the memory match word count report in text format, specify NULL if no report of this type should be created
PSZ	pszMemMatchReportXml	The fully qualified name for the memory match word count report in XML format, specify NULL if no report of this type should be created
PSZ	pszDupReportText	The fully qualified name for the duplicate word count report in text format, specify NULL if no report of this type should be created
PSZ	pszDupReportXml	The fully qualified name for the duplicate word count report in XML format, specify NULL if no report of this type should be created
PSZ	pszOutNomatchXml	The fully qualified name for the list of segments which have no memory match (in the nFluent XML format), specify NULL if no list of this type should be created
PSZ	pszDupReportXml	The fully qualified name for the list of segments which have no memory match (in the EXP format), specify NULL if no list of this type should be created

Type	Parameter	Description
LONG	lOptions	The option(s) to be used for the processing: **COMPLETE_IN_ONE_CALL_OPT** If set the API call does not return after each processing step but stays in the API call until the function has been completed **RESPECTCRLF_OPT** If set memory proposals having different linebreaks are not used as exact match The options can be combined by using the logical OR operator

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
CONTINUE_RC	The SNOMATCH processing is not complete yet. Call *EqfProcessNomatch* again.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
HSESSION hSession;
USHORT usRC;

usRC = EqfStartSession( &hSession );
usRC = EqfProcessNomatchEx( hSession, "TestFolder",
          "PrevMemory", "NewMemory", NULL,
          "C:\Reports\MemMatch.XML", NULL,
          "C:\Reports\Dups.XML, COMPLETE_IN_ONE_CALL_OPT);
usRC = EqfEndSession( hSession );
```

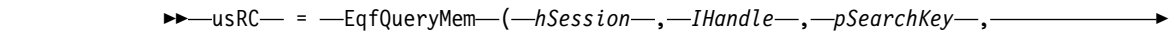
The API *EqfProcessNomatchEx* is called to process all SNOMATCH files of folder **"TestFolder"**, the segments are looked up in the memory **"PrevMemory"** and any relevant matches found are written to the memory **"NewMemory"**, the memory match count in XML format will be stored under **"C:\Reports\MemMatch.XML"** and the XML duplicate word count will be stored under **"C:\Reports\Dups.XML"**, the text versions of the reports are not being used. The API call will complete in one call.

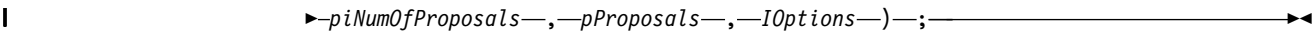
EqfQueryMem

Purpose

EqfQueryMem looks for matching Translation Memory proposals.

Format





Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
LONG	lHandle	Translation Memory handle from a Translation Memory previously opened using EqfOpenMem (see "EqfOpenMem" on page 126).
PMEMPROPOSAL	pSearchKey	Pointer to a MEMPROPOSAL structure filled with the search criteria. At least the source text, the source language, the target language, and the markup table have to be filled.
int *	piNumOfProposals	Pointer to a variable containing the number of requested Translation Memory proposals. On return, this variable is updated with the number of found Translation Memory proposals.
PMEMPROPOSAL	pProposals	Pointer to a array of MEMPROPOSAL structures. The array has to be large enough to receive the number of requested Translation Memory proposals. This array is filled with the Translation Memory proposals matching the search criteria.
LONG	lOptions	The options for the lookup (currently none).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
USHORT usRC = 0;
HSESSION hSession = 0L;
LONG lHandle = 0;

// start the Eqf calling interface session
usRC = EqfStartSession( &hSession );

// open the memory TestMem
if ( !usRC )
{
  usRC = EqfOpenMem( hSession, "TestMem", &lHandle, 0 );
} /* endif */
```

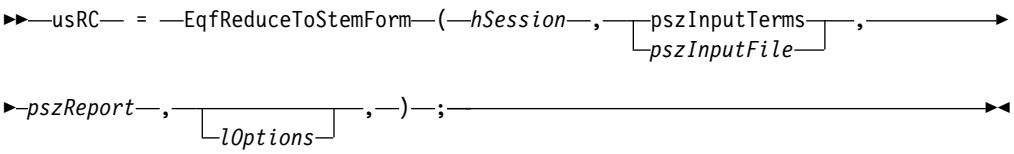
```
// search some memory proposals
if ( !usRC )
{
  PMEMPROPOSAL pSearchKey = new MEMPROPOSAL;
  PMEMPROPOSAL pProposals = new MEMPROPOSAL[5];
  int iProposals = 5;

  // fill-in search criteria
  wcscpy( pSearchKey->szSource, L"This is a segment." );
  strcpy( pSearchKey->szSourceLanguage, "English(U.S.)" );
  strcpy( pSearchKey->szTargetLanguage, "German(Reform)" );
  strcpy( pSearchKey->szMarkup, "OTMANSI" );

  usRC = EqfQueryMem( hSession, lHandle, pSearchKey, &iProposals,
  pPropsals, 0 );

  delete pSearchKey,
  delete pProposals;
} /* endif */
...

// close the memory
if ( !usRC )
{
  usRC = EqfCloseMem( hSession, lHandle, 0 );
} /* endif */

// terminate the session
EqfEndSession( hSession );
}
```

EqfReduceToStemForm

Purpose

The API call *EqfReduceToStemForm* reduces a list of words or words contained in a text file to their stem forms.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszLanguage	The name of the language being used for the spell checking.
PSZ	pszInputTerms	A comma separated list of terms or NULL if an input file is being used.
PSZ	pszInputFile	The fully qualified name of a plain text file containing the terms, one term per line or NULL if pszInputTerms is being used.

Type	Parameter	Description
PSZ	pszReport	The name of the report file receiving the results of the operation.
LONG	lOption	**Options for the output of the report**: • TEXT_OUTPUT_OPT for plain text output (CSV) or • XML_OUTPUT_OPT (= default) for XML output.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The terms have been reduced to their stem form successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  // start the OpenTM2 API session
  usRC = EqfStartSession( &hSession );
  if ( !usRC )
}
```

EqfRemoveDocs

Purpose

EqfRemoveDocs removes documents from a folder. The names of the removed documents are specified in a text file, one document per line.

Format

```
►►—usRC— = —EqfRemoveDocs—(—hSession—,—pszFolderName—,—pszListFile—)—;————►◄
```

Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents to be removed
PSZ	pszListFile	The name of the list file containing the names of the documents being removed

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use EqfGetLastError to retrieve the complete error information.

Code sample

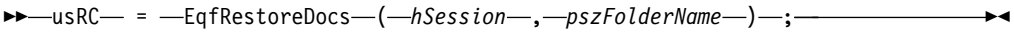
```
{
  USHORT usRC = 0; HSESSION hSession = 0L;
  // start the Eqf calling interface
  session usRC = EqfStartSession( &hSession );
  // Remove the documents listed in file
  REMOVELIST.TXT from folder SAMPLE1
  if ( !usRC )
    { usRC = EqfRemoveDocs( hSession, "SAMPLE1", "C:\REMOVELIST.TXT" );
    } /* endif */
  // terminate the session EqfEndSession( hSession );
}
```

EqfRestoreDocs

Purpose

EqfRestoreDocs restored documents which have been removed using the EqfRemoveDocs API call.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszFolderName	The name of the folder containing the documents to be restored

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use EqfGetLastError to retrieve the complete error information.

Code sample

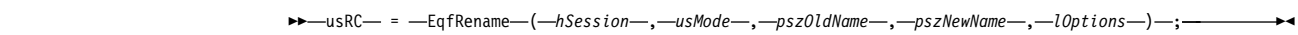
```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );    // Restore the removed documents of folder SAMPLE1   if
```

EqfRename

Purpose

EqfRename renames a folder, a dictionary or a Translation Memory.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
USHORT	usMode	Descibes the type of object being renamed, valid are RENAME_FOLDER, RENAME_MEMORY or RENAME_DICTIONARY
PSZ	pszOldName	The name of the existing folder, dictionary or Translation Memory.
PSZ	pszNewName	The new name for the folder, dictionary or Translation Memory.
LONG	lOptions	Additional options for the rename function: • ADJUSTREFERENCES_OPT to adjust all references to the rename object (valid only for the rename of a Translation Memory)

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use EqfGetLastError to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

   // rename the Translation Memory MyMemory to MyNewMemory and adjust all references
```

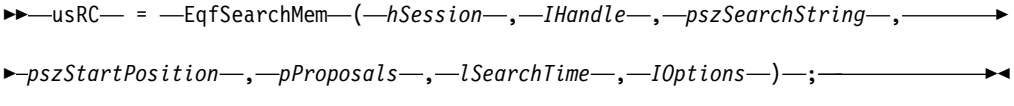
```
  if ( !usRC )
  {
       usRC = EqfRename( hSession, RENAME_MEMORY, "MyMemory",
                         "MyNewMemory", ADJUSTREFERENCES_OPT );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfSearchMem

Purpose

EqfSearchMem does a concordance search in a Translation Memory.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
LONG	IHandle	Translation Memory handle from a Translation Memory previously opened using EqfOpenMem (see “EqfOpenMem” on page 126).
wchar_t *	pszSearchString	Pointer to the search string (in UTF-16 encoding).
PSZ	pszStartPosition	Pointer to a buffer (min size = 20 characters) containing the start position. On completion, this buffer is filled with the next search position. To start at the search at the begin of the Translation Memory, leave this buffer empty.
PMEMPROPOSAL	pProposal	Pointer to MEMPROPOSAL structure receiving the matching proposal.
LONG	lSearchTime	Number of miliseconds to search for an entry. When this time is exceeded, the function returns with an return code of TIMEOUT_RC. To search for a indefinite time, specify the value 0.

Type	Parameter	Description
LONG	lOptions	Options for the import, valid options are: • SEARCHINSOURCE_OPT to search in the source text. • SEARCHINTARGET_OPT to search in the target text. • SEARCH_CASEINSENSITIVE to search case insensitive. • SEARCH_WHITESPACETOLERANT to handle all types of whitespace (blank, tab, linefeed) the same, and to treat multiple whitespace characters as a single space character. The options can be combined using the logical or operator (|).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
10010 (ENDREACHED_RC)	The end of the Translation Memory has been reached.
10011 (TIMEOUT_RC)	A time out occurred (exceeded given search time).
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  LONG lHandle = 0;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // open the memory TestMem
  if ( !usRC )
  {
    usRC = EqfOpenMem( hSession, "TestMem", &lHandle, 0 );
  } /* endif */

  // search the memory for the text "IBM"
  if ( !usRC )
  {
    PMEMPROPOSAL pProposal = new MEMPROPOSAL;
    char szSearchPos[20] = "";

    do
      {
      usRC = EqfSearchMem( hSession, lHandle, L"IBM", szSearchPos, pProposal,
      0, SEARCHSOURCE_OPT );

      // do something with the found proposal
      ...
```

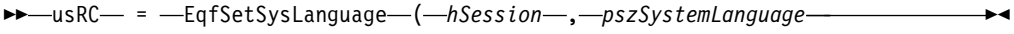
```
    } while (usRC == 0 );
    delete pProposal;
  } /* endif */
  ...

  // close the memory
  if ( !usRC )
  {
    usRC = EqfCloseMem( hSession, lHandle, 0 );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfSetSysLanguage

Purpose

EqfSetSysLanguage sets the default target language for the OpenTM2 system environment. All OpenTM2 internal character conversions (Unicode to ASCII/ANSI, ASCII/ANSI to Unicode) and linguistic functions will use the provided default target language if no other language settings are available. This happens e.g. during Translation Memory import/export in ASCII. It is a good coding practice to retrieve the default target language first (EqfGetSysLanguage), set the requested default target language, do your processing and reset the default target language to the previously stored value. Using the EqfSetSysLanguage has the same effect as modifying the Default Target Language on the System Preferenece Dialog via the GUI.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
PSZ	pszSystemLanguage	Buffer provided to contain the system language string. The length of the buffer has to be at least 20 characters.

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfSetSysLanguage* to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
```

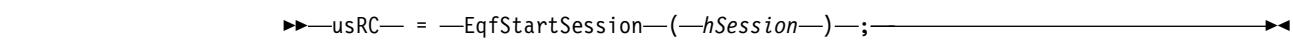
```
  // start the Eqf calling interface session
  usRC = EqfStartSession( hSession );


  // Set the default target language to be Japanese
  if ( !usRC )
  {
    usRC = EqfSetSysLanaguage( hSession, "Japanese" );

  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfStartSession

Purpose

EqfStartSession prepares the internal data areas for other non-DDE batch function calls. Call it before any other batch function. After you are finished, call the *EqfEndSession* function to clean up all resources.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The variable receiving the EQF session handle.

Return code

USHORT

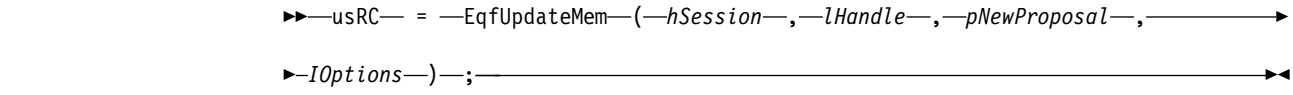
Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). You cannot use *EqfGetLastError* to retrieve complete error information if a call to *EqfStartSession* failed.

EqfUpdateMem

Purpose

EqfUpdateMem adds a new Translation Memory proposal to the Translation Memory, or updates an existing one having the same key.

Format



Parameters

Type	Parameter	Description
HSESSION	hSession	The EQF session handle, as returned by *EqfStartSession*.
LONG	lHandle	The Translation Memory handle from a Translation Memory previously opened using EqfOpenMem (see "EqfOpenMem" on page 126).
PMEMPROPOSAL	pNewProposal	Pointer to a MEMPROPOSAL structure filled with the proposal data which will be added to the Translation Memory, at least the source text, the target text, the source language, the target language and the markup table have to be filled.
LONG	lOptions	The options for the update process (currently none).

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* (see page "EqfGetLastError" on page 96) to retrieve the complete error information.

Code sample

```
{
  USHORT usRC = 0;
  HSESSION hSession = 0L;
  LONG lHandle = 0;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  // open the memory TestMem
  if ( !usRC )
  {
    usRC = EqfOpenMem( hSession, "TestMem", &lHandle, 0 );
  } /* endif */

  // add a new memory proposal
  if ( !usRC )
  {
    PMEMPROPOSAL pProposal = new MEMPROPOSAL;

    // fill-in proposal data
    memset( pProposal, 0, sizeof(MEMPROPOSAL) );
    wcscpy( pProposal->szSource, L"This is a sentence." );
    wcscpy( pProposal->szTarget, L"Dies ist ein Satz." );
    strcpy( pProposal->szSourceLanguage, "English(U.S.)" );
    strcpy( pProposal->szTargetLanguage, "German(Reform)" );
    strcpy( pProposal->szMarkup, "OTMANSI" );

    usRC = EqfUpdateMem( hSession, lHandle, pPropsal, 0 );
```

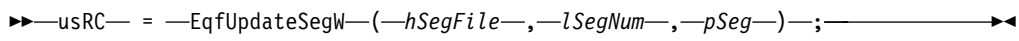
```
    delete pProposal;
  } /* endif */
  ...

  // close the memory
  if ( !usRC )
  {
    usRC = EqfCloseMem( hSession, lHandle, 0 );
  } /* endif */

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfUpdateSegW

Purpose

Update the segment data of a specific segment in a segmented file loaded into memory using *EqfLoadSegFile*.

Format



Parameters

Type	Parameter	Description
HPARSESEGFILE	hSegFile	Handle of loaded segmented file
LONG	lSegNum	Number of segment being updated
PPARSESEGMENTW	pSeg	Pointer to structure containing the updated segment data

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT    usRC = 0;
  HPARSSEGFILE *hSegFile = NULL;
 HSESSION   hSession = 0L;
 PARSSEGMENTW Segment;

  // start the Eqf calling interface session
  usRC = EqfStartSession( &hSession );

  if ( !usRC )
  {
       usRC = EqfBuildSegDocName( hSession, "SAMPLE1", "Document1",
                                  1, szFileName );
```

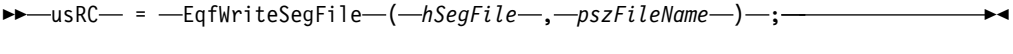
```
  } /* endif */

  if ( !usRC )
  {
   usRC = EqfLoadSegFile( hSession, szFileName, &hSegFile );
   if ( !usRC )
    {
      usRC = EqfGetSegW( hSegFile, 1, &Segment );
      if ( !usRC )
      {
        wcslwr( Segment.szData );
        usRC = EqfUpdateSegW( hSegFile, 1, &Segment );
        if ( !usRC )
        {
          usRC = EqfWriteSegFile( hSegFile, szFileName );
        } //endif
      } //endif
      EqfFreeSegFile(hSegFile );
    } //endif
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

EqfWriteSegFile

Purpose

Writes a segmented file loaded into memory using *EqfLoadSegFile* back to disk.

Format



Parameters

Type	Parameter	Description
HPARSESEGFILE	hSegFile	Handle of loaded segmented file
PSZ	pszFileName	Fully qualified file name

Return code

USHORT

Value	Description
0 (NO_ERROR)	The function completed successfully.
other	Error code (EQF message number). Use *EqfGetLastError* to retrieve the complete error information.

Code sample

```
{
  USHORT   usRC = 0;
  HPARSSEGFILE *hSegFile = NULL;
  HSESSION hSession = 0L;
  PARSSEGMENTW Segment;

  // start the Eqf calling interface session
```

```
  usRC = EqfStartSession(&hSession);

  if ( !usRC )
  {
    usRC = EqfBuildSegDocName( hSession, "SAMPLE1", "Document1",
                               1, szFileName );
  } //endif

  if ( !usRC )
  {
    usRC = EqfLoadSegFile( hSession, szFileName, &hSegFile );
    if ( !usRC )
    {
      usRC = EqfGetSegW( hSegFile, 1, &Segment );
      if ( !usRC )
      {
        wcslwr( Segment.szData );
        usRC = EqfUpdateSegW( hSegFile, 1, &Segment );
        if ( !usRC )
        {
          usRC = EqfWriteSegFile( hSegFile, szFileName );
        } //endif
      } //endif
      EqfFreeSegFile(hSegFile );
    } //endif
  } //endif

  // terminate the session
  EqfEndSession( hSession );
}
```

Chapter 3. Working with external markup tables

This chapter provides the information required to work with external markup tables. It describes the format of external markup tables so that you can modify them or create new ones. The user exit mechanism of markup tables and its entry points are described to allow for customized processing of documents at different stages. Finally, a parser application programming interface provides some of OpenTM2's internal functions to expand the possibilities of user exits.

The contents of external markup tables are described in terms of the SGML syntax. You should be familiar with SGML to modify or create markup tables. For a complete description of SGML refer to *ISO 8879, Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)*.

Creating new markup tables

You can create your own markup table by exporting an existing markup table in external SGML format, modifying it with any text editor, and importing it back into OpenTM2 under a different name. Markup tables need to be available in an SGML-based format to be imported into OpenTM2. Notice that an exported markup table contains only the nondefault entries.

To become familiar with the content of markup tables you might want to export a markup table and study it before you create a new markup table. See for details.

When you have exported one of the markup tables provided by OpenTM2 you might see a second tag in the second line `<SEGMENTEXIT>`*userexit*`</SEGMENTEXIT>`. *userexit* is the name of the dynamic-link library (DLL) containing the user exit code. This tag is only required if a user exit is to be used. For more information, refer to "Creating user exits for markup tables" on page 154.

Layout and content of a markup table

The general layout and content of a markup table are as follows:

- A markup table must begin with a `<TAGTABLE>` tag and end with a `</TAGTABLE>` tag.
- Following the `<TAGTABLE>` tag are *header tags* that are descriptive or of general purpose for the markup table. These header tags do not declare individual markup data. You can use them to give the markup table a name and a description, to specify a character set for conversion, or to specify substitution characters. Header tags in a markup table are optional. See Table 1 on page 149 for a list of allowed header tags and a detailed description.

  An example of a header tag in a markup table is `<DESCRNAME>descriptive name</DESCRNAME>`, which lets you specify a name for the markup table that is different from its file name.
- Next, a list of *markup tag definitions* follows. These definitions are the core of a markup table. Each definition describes a specific formatting tag, for example, a header tag, or a soft line feed. The definition always includes the name of the markup tag, and either its length or the delimiting characters. A markup tag definition can include further information, for example, whether the text associated with a markup tag needs to be translated. See Table 2 on page 150 for a list of allowed tags to define a markup tag in detail.

A single markup tag definition always starts with the start tag <TAG> and ends with the corresponding end tag </TAG>. An example of a markup tag definition is:

```
<TAG>
 <STRING>[soft line feed]</STRING>
 <LENGTH>16</LENGTH>
 <TYPE>STNEUTRAL</TYPE>
 <SEGINFO>SEGNEUTRAL</SEGINFO>
</TAG>
```

which defines the markup of a soft line feed. The keyword [soft line feed] is defined as <STRING>[soft line feed]</STRING> and has a length of 16 characters. <TYPE>STNEUTRAL</TYPE> specifies that this markup tag has no influence on segmenting, and <SEGINFO>SEGNEUTRAL</SEGINFO> specifies that this markup tag does not influence the segmenting status.

- Markup tags often have *attributes* that specify additional characteristics. For example, a markup tag for tables and figures in a document might use a width attribute to specify the width of the element. You need to define all attributes of a markup language in your markup table as well. The definition of attributes is similar to the definition of markup tags, except that each attribute definition is enclosed between the <ATTRIBUTE> and </ATTRIBUTE> tags. See Table 2 on page 150 for a list of allowed tags to define an attribute in detail.

  An example of an attribute definition is:

  ```
  <ATTRIBUTE>
  <STRING>WIDTH=%</STRING>
  <ENDDELIM>' .\r\n'</ENDDELIM>
  </ATTRIBUTE>
  ```

  which defines the markup of a WIDTH attribute. Here, you will notice that the keyword WIDTH is supposed to be delimited by one of four delimiting characters, as opposed to the previous example, where an explicit length is specified.

In summary, a markup table has the following layout:

```
<TAGTABLE>
Header tags, as required
<TAG>
markup tag definition
</TAG>
⋮
<TAG>
markup tag definition
</TAG>
<ATTRIBUTE>
attribute definition (optional)
</ATTRIBUTE>
⋮
<ATTRIBUTE>
attribute definition (optional)
</ATTRIBUTE>
</TAGTABLE>
```

Notice that all entries use the SGML syntax. All SGML tags must be enclosed in "<" and ">". There are always a start tag and an end tag.

Your markup table can contain up to 1000 entries.

An SGML markup tag or attribute must be at least specified with STRING and ENDDELIM, or STRING and LENGTH.

After you have edited the markup table, you can import it into OpenTM2. If you import it into an existing markup table, this table is overwritten.

Substitution characters in a markup table

Your markup tag and attribute definitions in a markup table might require that you specify variable parts. An example is the definition of the WIDTH attribute in the previous section (<STRING>WIDTH=%</STRING>). Because a document can contain any value for the WIDTH attribute, the percentage sign % is used as a substitution character.

You can use the following two substitution characters in a markup table:

- The percentage character (%) substitutes any number of characters.
- The question mark (?) substitutes a single character.

The substitution characters do not distinguish between numeric and alphabetic characters.

Note that these substitution characters can be redefined in the markup table header.

SGML tags for markup table header

The following table contains the definition of the SGML tags that you can use in a markup table header.

Table 1. SGML tags for markup table header

SGML tag	Definition
DESCRIPTION	Specifies a markup table description, which is shown in the "Markup Table Properties" window and the "Markup Table List" window.
DESCRNAME	Specifies a descriptive name for this markup table. For example, the specification of <DESCRNAME>ASCII</DESCRNAME> in the markup table EQFASCII would give it the name ASCII. If nothing is specified, the file name of the markup table is used.
CHARSET	Specifies the character set to be used for import and export of documents that use this markup table. The documents will be converted using the selected character set without the need to do the conversion in a user exit. Specify one of the following character sets: **ASCII** **ANSI** **UTF8** **UNICODE**
SINGLESUBST	Specifies the substitution character to use for single character substitution. The default character is ?.
MULTSUBST	Specifies the substitution character to use for multiple character substitution. The default character is %.
USEUNICODE	Specifies whether segmented source and target files in subdirectories SSOURCE and STARGET are stored in Unicode UTF-16 format. Specify one of the following: **YES** **NO** This is the default.

Table 1. SGML tags for markup table header (continued)

SGML tag	Definition
REFLOW	Specifies whether CRLF are allowed to be changed during translation or not. EQFMRI is an example of a markup where RELOW is specified and set to NO. Specify one of the following: **YES** This is the default. **NO**
SEGMENTEXIT	Contains the name of the user exit, if the markup table uses one.

SGML tags for markup tags and markup attributes

The following table contains the definition of the SGML tags that you can use to define markup tags and markup attributes in a markup table.

Table 2. SGML tags for markup tags and markup attributes

SGML tag	Definition
STRING	Specifies the name of the markup tag or markup attribute. The specification of STRING is required for an entry in the markup table.
ENDDELIM	Specifies one character as end delimiter of the markup tag or markup attribute, if it has any. You can enter more than one end delimiter. OpenTM2 checks for all possible string combinations to determine the end of the tag or attribute. A string as end delimiter is not possible. When a tag or attribute has an end delimiter, the specification of its length is omitted or can be set to 0. If a tag or attribute has no end delimiter, its length must be specified. The specification of ENDDELIM is required for an entry in the markup table, if LENGTH is not defined.
LENGTH	Defines the length of a markup tag or markup attribute. It must be specified only if the length of the tag or attribute cannot be determined by a delimiter specified by ENDDELIM.
COLPOSITION	Specifies the column position where the markup tag starts. If a markup tag has no special start position and can occur anywhere in a line, COLPOSITION is omitted or can be set to 0. The default is 0.
TYPE	Defines the type of the markup tag. If TYPE is not specified, STDEL is taken as the default. The following types are possible: **STDEL** Indicates the start of a new text segment. **ENDDEL** Indicates the end of a text segment. **SELFC** The markup tag is self-contained, that is, it is a text segment by itself. **STNEUTRAL** The markup tag is a start tag, which has no influence on segmenting. **ENDNEUTRAL** The markup tag is an end tag, which has no influence on segmenting.

Table 2. SGML tags for markup tags and markup attributes (continued)

SGML tag	Definition
SEGINFO	Determines whether the text following the markup tag is to be segmented. If SEGINFO is not specified, SEGNEUTRAL is taken as the default. **SEGOFF** Sets segmenting off, that is, no segmentation is done until the next markup tag is found that sets segmenting on again. If two tags follow each other that set segmenting off, it needs two tags that set segmenting on to start segmentation again. **SEGON** Sets segmenting on again. **SEGNEUTRAL** Does not influence the segmenting status. **SEGRESET** Resets the segmenting status to on, even if the segmenting level requires more than one SEGON tag to set segmentation on. **PROTECTON** All following text, including segmentation control flags, is protected until a markup tag with **PROTECTOFF** is encountered. **PROTECTOFF** Turns off text protection. The following text is handled using normal segmentation rules.
ASSTEXT	Defines types of text following the markup tag. If ASSTEXT is not specified, NOEXPL is taken as the default. **TSNL** Text follows on the same or the next line and will be associated with the markup tag. **TSL** Text follows on the same line and will be associated with the makeup tag. **NOEXPL** No special processing for associated text is required.
ADDINFO	Specifies whether specific text is to be ignored when segments are aligned during the creation of an Initial Translation Memory: **4** Marks the start of an area to be ignored. **6** Marks the start of an area to be partly ignored. This applies to tags containing a % sign, for example `HEADER]%`. **8** Marks the end of an area to be ignored. **10** Marks the end of an area to be partly ignored. This applies to tags containing a % sign, for example `HEADER %`.
CLASSID	Specifies how the contents of STRING is handled. The only class is **CLS_HEAD**. This means that the text specified for STRING becomes an entry of the table of contents that you can display during the translation of a document using the **Special go to...** dialog.
ATTRINFO	Specifies whether a markup tag has attached attributes (YES/NO). NO is the default. If YES is specified, the ATTRIBUTE SGML tag must be used to specify the attributes.

Table 2. SGML tags for markup tags and markup attributes (continued)

SGML tag	Definition
TRANSLATEINFO	Specifies whether the segment associated with the markup tag or markup attribute must be translated or not (YES/NO). If TRANSLATEINFO is not specified, NO is taken as the default.

Examples of markup data and corresponding markup tags

If a document contains, for example, `[soft line feed]` as markup data, it is usually meant as a so-called inline tag, which means that it is contained in the segment. It has no influence on the segmentation of the document. The corresponding markup tag definition in a markup table looks as follows:

```
<TAG>
 <STRING>[soft line feed]</STRING>
 <LENGTH>16</LENGTH>
 <TYPE>STNEUTRAL</TYPE>
 <SEGINFO>SEGNEUTRAL</SEGINFO>
</TAG>
```

`<STRING>`... defines the markup string, and `<LENGTH>`... specifies its length. Because the length is specified, no `ENDDELIM` tag is required. `<TYPE>STNEUTRAL<`... defines that this markup string has no influence on segmentation. All other markup table SGML tags will be set to the default and therefore need not be specified.

Assumed that such markup tag causes segmentation, we define this as follows:

```
<TAG>
 <STRING>[soft line feed]</STRING>
 <LENGTH>16</LENGTH>
 <TYPE>STDEL</TYPE>
 <SEGINFO>SEGNEUTRAL</SEGINFO>
</TAG>
```

The following table lists some imaginary markup data with a description.

Markup data	Definition
[bold]*text*[/bold]	The text following this tag (until the end tag) is printed bold; this tag is part of the segment and has no influence on segmenting.
[Heading *x*]*text*	This tag describes a heading; the heading text must follow on the same line; *x* is the level of heading and goes from 1 to 9; this tag ends the previous segment and starts a new segment.
[page: even]	A page break; the following text starts on an even page; this tag always starts on the first column and has no text following in the same line; a blank must separate the attribute *even* from the tag.
[page: odd]	A page break; the following text starts on an odd page; this tag always starts on the first column and has no text following in the same line; a blank must separate the attribute *odd* from the tag.
[paragraph]	A paragraph; this tag ends the previous segment and starts a new segment; the tag occurs at the end of the previous paragraph.
%	Stands for any number of characters. For example, in `b%`, `%` stands for the characters `old`.
[break]	Starts a new segment. You use this tag to split an existing segment into two or more segments.
[*%]	* indicates the start of a comment and `%` stands for the comment text.

This markup data would lead to the following markup table definitions. The defaults will not be shown.

Markup definition	Explanation
<TAG> <STRING>[bold]</STRING> <LENGTH>6</LENGTH> <TYPE>STNEUTRAL</TYPE> </TAG> or <TAG> <STRING>[bold</STRING> <ENDDELIM>]</ENDDELIM> <TYPE>STNEUTRAL</TYPE> </TAG> or <TAG> <STRING>[b%</STRING> <ENDDELIM>]</ENDDELIM> <TYPE>STNEUTRAL</TYPE> </TAG>	The markup tag should be part of the segment, therefore STNEUTRAL is used. All examples have the same result, you can specify this markup tag by its length or end delimiter. You can also substitute part of the inline tag by %.
<TAG> <STRING>[Heading ?</STRING> <ENDDELIM>]</ENDDELIM> <SEGINFO>SEGRESET</SEGINFO> <ASSTEXT>TSL</ASSTEXT> <TRANSLATEINFO>YES</TRANSLATEINFO> </TAG>	Single substitution is used for the heading level; the end of the tag is]; the heading requires the reset of segmenting with SEGRESET; the text associated with the tag occurs on the same line; the text associated with the tag is translatable.
<TAG> <STRING>[page:</STRING> <ENDDELIM> </ENDDELIM> <ATTRINFO>YES</ATTRINFO> <COLPOSITION>1</COLPOSITION> </TAG>	The markup tag ends with a blank; attributes may follow; the tag always starts at the first column in a line.
<TAG> <STRING>[paragraph</STRING> <ENDDELIM>]</ENDDELIM> <TYPE>ENDDEL</TYPE> </TAG> or <TAG> <STRING>[paragraph]</STRING> <LENGTH>11</LENGTH> <TYPE>ENDDEL</TYPE> </TAG>	The tag ends with] or is defined by its length; the tag should end the previous segment, therefore ENDDEL is used.
<ATTRIBUTE> <STRING>even</STRING> <ENDDELIM>]</ENDDELIM> </ATTRIBUTE>	This is an attribute; it ends with].
<ATTRIBUTE> <STRING>odd</STRING> <ENDDELIM>]</ENDDELIM> </ATTRIBUTE>	This is an attribute; it ends with].

Markup definition	Explanation
<TAG> <STRING>[break]</STRING> <LENGTH>7</LENGTH> <TYPE>STDEL</TYPE> </TAG>	Indicates that a new segment starts.
<TAG> <STRING>*%</STRING> <ENDDELIM>\r\n/ENDDELIM> <COLPOSITION>1</COLPOSITION> </TAG>	Indicates a comment that ends at the end of the line. COLPOSITION defines that the asterisk is only recognized as the start of a comment if it appears in the first column of a line.

Creating user exits for markup tables

There are document formats that require a user exit for their markup table:

- Binary documents, for example Microsoft Word for Windows documents
- Documents that require code page conversion, for example ANSI documents
- Documents that have a fixed record layout
- Documents that contain nontranslatable text parts, for example, RTF documents
- Binary documents like Lotus Notes database files and template files that require context-dependent processing.

OpenTM2 provides two markup tables that are already combined with a user exit:

- The user exit part of the EQFHTML4 markup table converts the code page and preprocesses JavaScripts to limit segments to 2048 characters. The markup table part controls text segmentation and the recognition of inline tags.
- The user exit part of the EQFANSI markup table converts the code page, and the markup table part inserts segment breaks after empty lines.

In addition, OpenTM2 provides a user exit that you can use with the appropriate markup table. This user exit is a dynamic-link library (DLL) with predefined entry points. The code for the exit can be written in any programming language that supports PASCAL calling conventions. The include file `EQF_API.H` contains the definitions required for a user exit written in C.

The user exit is activated using the `<SEGMENTEXIT>` tag of the markup table (see also Segment exit).

General user exit entry points

The user exit entry points (their names start with EQF) are called at different stages during the analysis, translation, and export of a document.

- During the analysis (see Figure 1 on page 155):
  - "EQFPRESEG2" on page 155 is called *before* the text is segmented. It can be used to preprocess a document and decide whether text segmentation is done by OpenTM2 after EQFPRESEG2.
  - "EQFPOSTSEGW" on page 157 is called *after* the text is segmented. It can be used to postprocess a document.
  - "EQFPOSTTMW" on page 158 is called *after* Translation Memory matches are processed and terms lists are created. It can be used to modify segments.

Figure 1. Analysis of a document using the user exit

- During the translation:
  - "EQFCHECKSEGW" on page 159 is called after a segment is translated but before it is saved in the Translation Memory. It can be used to modify a segment.
  - "EQFSHOW" on page 160 is called when the user selects the "Show translation" menu item.
- During the export (see Figure 2):
  - "EQFPREUNSEGW" on page 167 is called *before* OpenTM2 removes the segmentation from a document. It can be used for the same purpose, or whatever is required at this step.
  - "EQFPOSTUNSEG2" on page 168 is called *after* OpenTM2 (or EQFPREUNSEG2) removed the segmentation. It can be used, for example, to establish the external document format.
  - Alternatively, "EQFPOSTUNSEGW" on page 168 can be called *after* OpenTM2 (or EQFPREUNSEG2) removed the segmentation. If EQFPOSTUNSEGW entry point exists, OpenTM2 uses EQFPOSTUNSEGW, without regard of the existence of EQFPOSTUNSEG2. EQFPOSTUNSEGW requires that the input text is always UTF16. If EQFPOSTUNSEGW entry point exists, OpenTM2s' "Undo text segmentation" step outputs an UTF16 file.

Figure 2. Export of a document using the user exit

The following sections describe the individual entry points in detail. Note that entry points from earlier versions of OpenTM2 (without the trailing letter W) are supported, and the calling syntax remains unchanged. However, you should use the entry points as listed in this section. See for details.

EQFPRESEG2

Purpose

EQFPRESEG2 is called during the analysis of a document before the text is segmented. It preprocesses the document, for example converts code pages, and decides whether text segmentation is done by OpenTM2 or *EQFPRESEG2* itself. If an error occurs, it can stop the analysis.

Format

```
►►—EQFPRESEG2—(—MarkupTable—,—Editor—,—Path—,—SourceFile—,————————►
►—Buffer—,—OutputFlag—,—SliderWindowHandle—,—ReturnFlag—)————————►◄
```

Parameters

MarkupTable
The pointer to the name of a markup table.

Editor
The pointer to the name of the editor.

Path
The pointer to the program path.

SourceFile
The pointer to the name of the source file (with full path).

Buffer
The pointer to the buffer containing the name of the temporary output file.

OutputFlag
The output flag indicating whether the text is to be segmented by EQFPRESEG2 instead of OpenTM2.

SliderWindowHandle
The handle of the slider window.

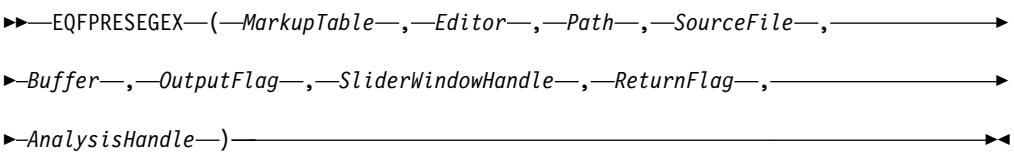
ReturnFlag
The pointer to the return flag. If this flag changes to TRUE, the user exit must return immediately.

EQFPRESEGEX

Purpose

EQFPRESEGEX is called during the analysis of a document before the text is segmented. It preprocesses the document, for example converts code pages, and decides whether text segmentation is done by OpenTM2 or EQFPRESEGEX itself. If an error occurs, it can stop the analysis. The EQFPRESEGEX entry point is identical to "EQFPRESEG2" on page 155 except for the additional parameter Analsysis handle.

Format



Parameters

MarkupTable
The pointer to the name of a markup table.

Editor
The pointer to the name of the editor.

Path
The pointer to the program path.

SourceFile
The pointer to the name of the source file (with full path).

Buffer
The pointer to the buffer containing the name of the temporary output file.

OutputFlag
The output flag indicating whether the text is to be segmented by EQFPRESEGEX instead of OpenTM2.

SliderWindowHandle
The handle of the slider window.

ReturnFlag
The pointer to the return flag. If this flag changes to TRUE, the user exit must return immediately.

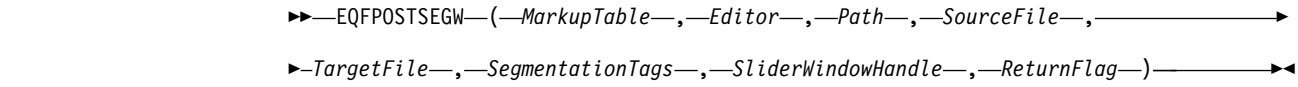
AnalysisHandle
The analysis handle. This handle is required for the API calls "EQFSETTAOPTIONS" on page 170 and "EQFGETTAOPTIONS" on page 169.

EQFPOSTSEGW

Purpose

EQFPOSTSEGW is called during the analysis of a document after the text is segmented. It postprocesses the document, for example adjusts segment boundaries. If an error occurs, it can stop the analysis.

Format



Parameters

MarkupTable
The pointer to the name of a markup table.

Editor
The pointer to the name of the editor.

Path
The pointer to the program path.

SourceFile
The pointer to the name of the source file (with full path).

TargetFile
The pointer to the name of the target file.

SegmentationTags
The pointer to the tags inserted during text segmentation.

SliderWindowHandle
The handle of the slider window.

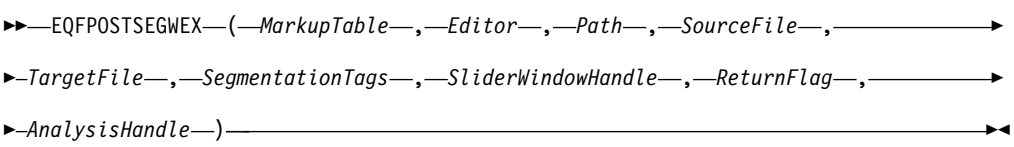
ReturnFlag
The pointer to the return flag. If this flag changes to TRUE, the user exit must return immediately.

EQFPOSTSEGWEX

Purpose

EQFPOSTSEGWEX is called during the analysis of a document after the text is segmented. It postprocesses the document, for example adjusts segment boundaries. If an error occurs, it can stop the analysis. The EQFPOSTSEGWEX entry point is identical to "EQFPOSTSEGW" except for the additional parameter Analysis handle.

Format



Parameters

MarkupTable
The pointer to the name of a markup table.

Editor
The pointer to the name of the editor.

Path
The pointer to the program path.

SourceFile
The pointer to the name of the source file (with full path).

TargetFile
The pointer to the name of the target file.

SegmentationTags
The pointer to the tags inserted during text segmentation.

SliderWindowHandle
The handle of the slider window.

ReturnFlag
The pointer to the return flag. If this flag changes to TRUE, the user exit must return immediately.

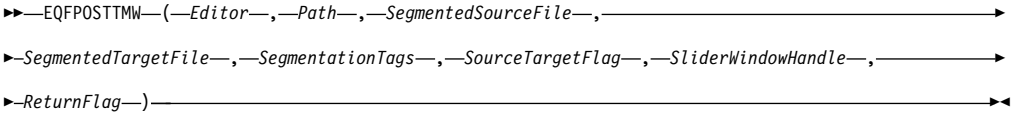
AnalysisHandle
The analysis handle. This handle is required for the API calls "EQFSETTAOPTIONS" on page 170 and "EQFGETTAOPTIONS" on page 169.

EQFPOSTTMW

Purpose

EQFPOSTTMW is called during the analysis of a document after Translation Memory matches have been inserted and terms lists have been created. It is used to modify the segments. If an error occurs, it can stop the analysis.

Format



Parameters

Editor
The pointer to the name of the editor.

Path
The pointer to the program path.

SegmentedSourceFile
The pointer to the name of the segmented source file.

SegmentedTargetFile
The pointer to the name of the segmented target file.

SegmentationTags
The pointer to the tags inserted during text segmentation.

SourceTargetFlag
The flag indicating if the segmented source differs from the segmented target.

SliderWindowHandle
The handle of the slider window.

ReturnFlag
The pointer to the return flag. If this flag changes to TRUE, the user exit must return immediately.

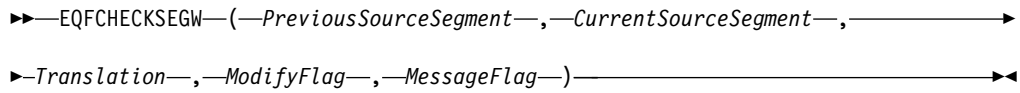
EQFCHECKSEGW

Purpose

EQFCHECKSEGW is called during the translation of a document after a segment has been translated but not saved yet in the Translation Memory. It can modify the segment, for example change lowercase characters to uppercase, and prevent the segment from being saved, for example if specific length limits have been exceeded.

EQFCHECKSEGW is also called when exact matches are automatically substituted during the analysis of a document.

Format



Parameters

PreviousSourceSegment
The pointer to the text of the previous source segment.

CurrentSourceSegment
The pointer to the text of the current source segment.

Translation
The pointer to the translation of the current segment.

ModifyFlag
The pointer to the flag that is set when the user exit has modified the translated segment.

MessageFlag
The flag indicating whether a message box is shown.

Return code

The return code indicates if the segment can be saved.

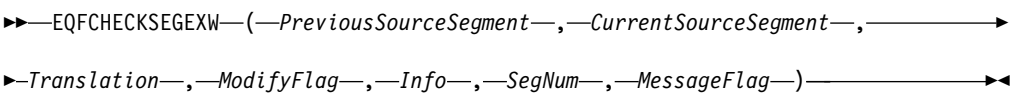
EQFCHECKSEGEXW

Purpose

EQFCHECKSEGEXW is called during the translation of a document after a segment has been translated but not saved yet in the Translation Memory. It can modify the segment, for example change lowercase characters to uppercase, and prevent the segment from being saved, for example if specific length limits have been exceeded. It has the same functionality as the entry point EQFCHECKSEGW and has two additional parameteres to allow the usage of the EQFGETPREVSEG(W) and EQGGETNEXTSEG(W) API functions.

EQFCHECKSEGEXW is also called when exact matches are automatically substituted during the analysis of a document

Format



Parameters

PreviousSourceSegment
The pointer to the text of the previous source segment.

CurrentSourceSegment
The pointer to the text of the current source segment.

Translation
The pointer to the translation of the current segment.

ModifyFlag
The pointer to the flag that is set when the user exit has modified the translated segment.

Info
A long info value which has to be passed to EQFGETPREVSEG(W) and EQGGETNEXTSEG(W) API functions

SegNum
An unsigned long value representing the current segment number. The segment number should be stored in a local unsigned long variable. A pointer to this variable has to be to be passed to EQFGETPREVSEG(W) and EQGGETNEXTSEG(W) API functions

MessageFlag
The flag indicating whether a message box is shown.

Return code

The return code indicates if the segment can be saved.

EQFSHOW

Purpose

EQFSHOW is called during the translation of a document when the user selects the "Show Translation" menu item. It is up to the user exit to prepare and display the document in a window. The user exit can use the API calls “EQFGETNEXTSEG” on page 162,

"EQFGETNEXTSEGW" on page 163, "EQFGETPREVSEG" on page 163, "EQFGETPREVSEGW" on page 164, "EQFGETCURSEG," "EQFGETCURSEGW" on page 162 and "EQFGETINFO" on page 166 to retrieve the document segments and to get other document information.

Format

Parameters

lInfo
A handle to the target document. This handle has to be specified in the API calls for accessing the segment text.

hwndParent
The handle of the window which should be specified as parent window for the window displaying the document.

Return code

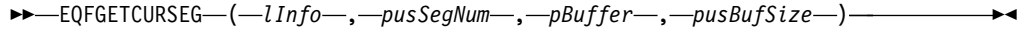
The user exit should return TRUE if the document could be displayed and FALSE in case of errors.

EQFGETCURSEG

Purpose

EQFGETCURSEG returns a specific segment from the document identified by the lInfo handle. The text of the segment is stored in the buffer pointed to by pBuffer as a zero terminated string. The variable pointed to by pusSegNum contains the number of the requested segment.

Format



Parameters

lInfo
The document handle which has been passed to the user exit as the first parameter of the EQFSHOW entry point.

pusSegNum
The pointer to a ULONG variable containing the segment number.

pBuffer
The pointer to a buffer for the segment text.

pusBufSize
The pointer to a USHORT variable containing the size of the buffer pointed to by pBuffer.

Return code

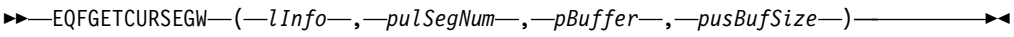
The function returns zero if successful otherwise an error code is returned.

EQFGETCURSEGW

Purpose

EQFGETCURSEGW returns a specific segment from the document identified by the lInfo handle. The text of the segment is stored in the buffer pointed to by pBuffer in UTF16-encoding and is terminated by 0x0000. The variable pointed to by pulSegNum contains the number of the requested segment.

Format



Parameters

lInfo
The document handle which has been passed to the user exit as the first parameter of the EQFSHOW entry point.

pulSegNum
The pointer to a ULONG variable containing the segment number.

pBuffer
The pointer to a buffer for the segment text in UTF-16 encoding.

pusBufSize
The pointer to a USHORT variable containing the size of the buffer pointed to by pBuffer in number of UTF-16 characters.

Return code

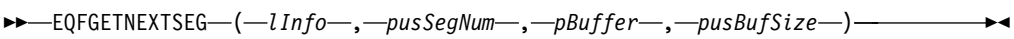
The function returns zero if successful otherwise an error code is returned.

EQFGETNEXTSEG

Purpose

EQFGETNEXTSEG returns the next segment from the document identified by the lInfo handle. The text of the segment is stored in the buffer pointed to by pBuffer as a zero-terminated string. The API call increments the segment number automatically.

Format



Parameters

lInfo
The document handle which has been passed to the user exit as the first parameter of the EQFSHOW entry point.

pusSegNum
The pointer to a USHORT variable containing the segment number. This variable should be set to 1 before the first call. The segment number is automatically incremented.

pBuffer
The pointer to a buffer for the segment text.

pusBufSize
The pointer to a USHORT variable containing the size of the buffer pointed to by pBuffer. Attention: this size value is set to the actual length of the returned segment data on exit.

Return code

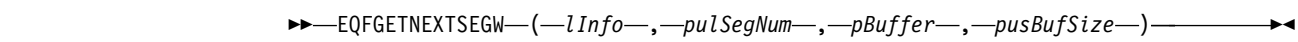
The function returns zero if successful otherwise an error code is returned. The error code 510 is also issued when the buffer size is not large enough to receive the segment data.

EQFGETNEXTSEGW

Purpose

EQFGETNEXTSEGW returns the next segment from the document identified by the lInfo handle. The text of the segment is stored in the buffer pointed to by pBuffer in UTF-16 encoding and is terminated by 0x0000. The API call increments the segment number automatically.

Format



Parameters

lInfo
The document handle which has been passed to the user exit as the first parameter of the EQFSHOW entry point.

pulSegNum
The pointer to a ULONG variable containing the segment number. This variable should be set to 1 before the first call. The segment number is automatically incremented.

pBuffer
The pointer to a buffer for the segment text in UTF-16 encoding.

pusBufSize
The pointer to a USHORT variable containing the size of the buffer in number of UTF-16 characters. Attention: this size value is set to the actual length of the returned segment data on exit.

Return code

The function returns zero if successful otherwise an error code is returned. The error code 510 is also issued when the buffer size is not large enough to receive the segment data.

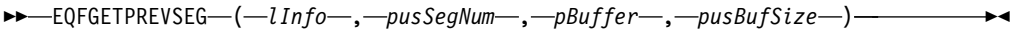
EQFGETPREVSEG

Purpose

EQFGETPREVSEG returns the previous segment from the document identified by the lInfo handle. The text of the segment is stored in the buffer pointed to by

pBuffer as a zero-terminated string. The API call decrements the segment number automatically.

Format



Parameters

lInfo
The document handle which has been passed to the user exit as the first parameter of the EQFSHOW entry point.

pulSegNum
The pointer to a USHORT variable containing the segment number. The segment number is automatically decremented.

pBuffer
The pointer to a buffer for the segment text.

pusBufSize
The pointer to a USHORT variable containing the size of the buffer pointed to by pBuffer. Attention: this size value is set to the actual length of the returned segment data on exit.

Return code

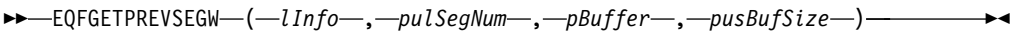
The function returns zero if successful otherwise an error code is returned. The error code 510 is also issued when the buffer size is not large enough to receive the segment data.

EQFGETPREVSEGW

Purpose

EQFGETPREVSEGW returns the previous segment from the document identified by the lInfo handle. The text of the segment is stored in the buffer pointed to by pBuffer in UTF16-encoding and is terminated by 0x0000. The API call decrements the segment number automatically.

Format



Parameters

lInfo
The document handle which has been passed to the user exit as the first parameter of the EQFSHOW entry point.

pulSegNum
The pointer to a ULONG variable containing the segment number. The segment number is automatically decremented.

pBuffer
The pointer to a USHORT variable containing the size of the buffer pointed to by pBuffer in number of UTF-16 characters. Attention: this size value is set to the actual length of the returned segment data on exit.

pusBufSize
The pointer to a USHORT variable containing the size of the buffer pointed to by pBuffer.

Return code

The function returns zero if successful otherwise an error code is returned. The error code 510 is also issued when the buffer size is not large enough to receive the segment data.

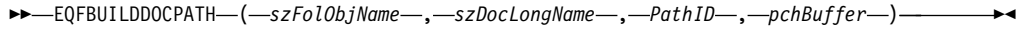
EQFBUILDDOCPATH

Purpose

EQFBUILDDOCPATH creates the fully qualified file name for a OpenTM2 document using the folder object name and the document long name.

This function can be used to access documents stored in OpenTM2 folders.

Format



Parameters

szFolObjName
The folder object name as returned using EQFGETINFO with the GETINFO_FOLDEROBJECT ID.

szDocLongName
The document long name.

PathID
The ID of the requested document path, valid IDs are:

```
PATHID_SOURCE to build the path to the source document
PATHID_SEGSOURCE to build the path to the segmented source document
PATHID_SEGTARGET to build the path to the segmented target document
PATHID_TARGET to build the path to the target document
```

pchBuffer
The pointer to a buffer receiving the fully qualified document path, the size of this buffer has to be at least 60 bytes.

Return code

0 function completed successfully

ERROR_INVALID_PARAMETER
wrong or missing parameter

ERROR_PATH_NOT_FOUND
the folder did not exist

ERROR_FILE_NOT_FOUND
the document does not exist

Examples

The folder "AnotherTestFolder" contains the document "myTest.HTML". The folder is located on drive "E:" and has a short name of

"ANOTH000.F00". The document short name is "MYTESTHT.000". The primary drive of the OpenTM2 installation is "C:".

EQFBUILDDOCPATH("C:\OTM\ANOTH000.F00", "myTest.HTML", PATHID_SOURCE, szBuffer) would return " E:\OTM\ANOTH000.F00\ SOURCE\ MYTESTHT.000" in szBuffer.

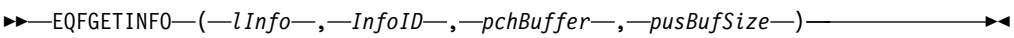
EQFGETINFO

Purpose

EQFGETINFO returns specific on the document currently being processed in the EQFSHOW function of the user exit.

This function is used by the user exit to get more information concerning the document and its location.

Format



Parameters

lInfo
The info handle passed to the user exit in the EQFSHOW call.

InfoID
The ID of the requested information, valid IDs are:

```
GETINFO_MARKUP to retrieve the markup table of the document
GETINFO_FOLDEROBJECT to retrieve the object name of the folder containing the document
GETINFO_FOLDERLONGNAME to retrieve the long name (in ASCII) of the folder containing the documen
GETINFO_DOCFULLPATH to retrieve the fully qualified path of the document segmented target file
GETINFO_DOCLONGNAME to retrieve the document long name
```

pchBuffer
The pointer to a buffer receiving the requested information, if this parameter is NULL the size of the requested information is returned using the pusBufSize parameter.

pusBufSize
The pointer to a USHORT value containing the buffer size, on return this value contains the size of the returned information.

Return code

0 function completed successfully

ERROR_INVALID_PARAMETER
unknown InfoID or missing parameter

ERROR_INVALID_HANDLE
invalid lInfo handle

ERROR_NOT_ENOUGH_MEMORY
not enough memory / memory allocation failed

ERROR_INSUFFICIENT_BUFFER
buffer is not large enough for the returned information, *pusBufSize contains required buffer size

Examples

Assuming the document "myTest.HTML" located in folder "AnotherTestFolder" is opened using EQFSHOW. The folder is located on drive "E:" and has a short name of "ANOTH000.F00". The document short name is "MYTESTHT.000". The primary drive of the OpenTM2 installation is "C:"

usBufSize = sizeof(szBuffer); EQFGETINFO(lInfo, GETINFO_MARKUP, szBuffer, &usBufSize) would return "IBMHTM32" in szBuffer

usBufSize = sizeof(szBuffer); EQFGETINFO(lInfo, GETINFO_FOLDEROBJECT, szBuffer, &usBufSize) would return "C:\OTM\ANOTH000.F00" in szBuffer

usBufSize = sizeof(szBuffer); EQFGETINFO(lInfo, GETINFO_FOLDERLONGNAME, szBuffer, &usBufSize) would return "AnotherTestFolder" in szBuffer

usBufSize = sizeof(szBuffer); EQFGETINFO(lInfo, GETINFO_DOCFULLPATH, szBuffer, &usBufSize) would return "E:\OTM\ANOTH000.F00\STARGET\MYTESTHT.000" in szBuffer

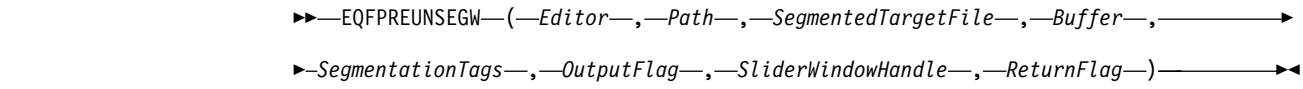
usBufSize = sizeof(szBuffer); EQFGETINFO(lInfo, GETINFO_DOCLONGNAME, szBuffer, &usBufSize) would return "MyTest.HTML" in szBuffer

EQFPREUNSEGW

Purpose

EQFPREUNSEGW is called during the export of a document before the segmentation tags inserted by OpenTM2 are removed. It decides whether the segmentation tags are removed by OpenTM2 or *EQFPREUNSEGW* itself. However, it is normally used to remove the segmentation tags. If an error occurs, it can stop the export.

Format



Parameters

Editor
The pointer to the name of the editor.

Path
The pointer to the program path.

SegmentedTargetFile
The pointer to the name of the segmented target file (with full path).

Buffer
The pointer to the buffer containing the name of the temporary output file.

SegmentationTags
The pointer to the tags inserted during text segmentation.

OutputFlag
The output flag indicating whether the segmentation tags are removed by EQFPREUNSEGW instead of OpenTM2.

SliderWindowHandle
The handle of the slider window.

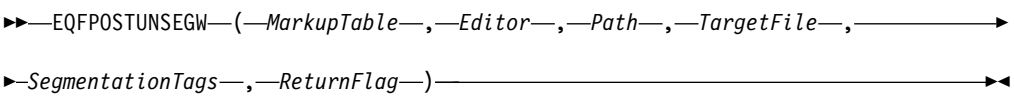
ReturnFlag
The pointer to the return flag. If this flag changes to TRUE, the user exit must return immediately.

EQFPOSTUNSEGW

Purpose

EQFPOSTUNSEGW is called during the export of a document after the segmentation tags have been removed from the text. The text must be in UTF16. It is normally used to establish the external document format. If an error occurs, it can stop the export.

Format



Parameters

MarkupTable
The pointer to the name of a markup table.

Editor
The pointer to the name of the editor.

Path
The pointer to the program path (with full path).

TargetFile
The pointer to the name of the target file (with full path).

SegmentationTags
The pointer to the tags inserted during text segmentation.

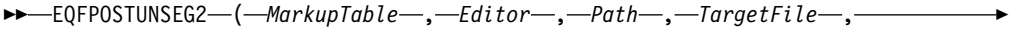
ReturnFlag
The pointer to the return flag. If this flag changes to TRUE, the user exit must return immediately.

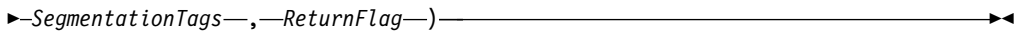
EQFPOSTUNSEG2

Purpose

EQFPOSTUNSEG2 is called during the export of a document after the segmentation tags have been removed from the text. It is normally used to establish the external document format. If an error occurs, it can stop the export.

Format





Parameters

MarkupTable
The pointer to the name of a markup table.

Editor
The pointer to the name of the editor.

Path
The pointer to the program path (with full path).

TargetFile
The pointer to the name of the target file (with full path).

SegmentationTags
The pointer to the tags inserted during text segmentation.

ReturnFlag
The pointer to the return flag. If this flag changes to TRUE, the user exit must return immediately.

API calls for user exits

This group contains the API calls which can be called by the markup table user exits to access and modify OpenTM2 settings. Currently these are

- "EQFGETTAOPTIONS" to get the active analysis settings. This API call can be called by the user exit during the "EQFPRESEGEX" on page 156, and "EQFPOSTSEGWEX" on page 157 processing.
- "EQFSETTAOPTIONS" on page 170 to modify the analysis settings. This API call can be called by the user exit during the "EQFPRESEGEX" on page 156, and "EQFPOSTSEGWEX" on page 157 processing.

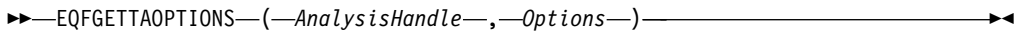
The following sections describe the individual API calls in detail.

EQFGETTAOPTIONS

Purpose

EQFGETTAOPTIONS can be used by the markup table user exit to retrieve the currently active analysis settings. The settings are returned in an "EQFTAOPTIONS" on page 170 structure. The analysis handle used by this call is passed to the user exit by the user exit entry points "EQFPRESEGEX" on page 156, and "EQFPOSTSEGWEX" on page 157.

Format



Parameters

AnalysisHandle
The analysis handle passed to the user exit by the entry points "EQFPRESEGEX" on page 156, and "EQFPOSTSEGWEX" on page 157.

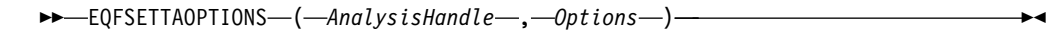
Options
The pointer to a "EQFTAOPTIONS" structure receiving the currently active analysis settings.

EQFSETTAOPTIONS

Purpose

EQFSETTAOPTIONS can be used by the markup table user exit to change the currently active analysis settings. The settings are passed to the API call in an "EQFTAOPTIONS" structure. The analysis handle used by this call is passed to the user exit by the user exit entry points "EQFPRESEGEX" on page 156, and "EQFPOSTSEGWEX" on page 157.

Format



Parameters

AnalysisHandle
The analysis handle passed to the user exit by the entry points "EQFPRESEGEX" on page 156, and "EQFPOSTSEGWEX" on page 157.

Options
The pointer to a "EQFTAOPTIONS" structure containing the analysis settings being modified.

EQFTAOPTIONS

Purpose

The structure *EQFTAOPTIONS* is used by the API calls "EQFSETTAOPTIONS" and "EQFGETTAOPTIONS" on page 169 to get or set the analysis options.

Fields

fAdjustLeadingWS
This flag represents the "Adjust leading whitespace to whitespace of source segment" flag of the GUI.

fAdjustTrailingWS
This flag represents the "Adjust trailing whitespace to whitespace of source segment" flag of the GUI.

bForFutureUse
Area for future enhancements. Currently not in use.

User exit entry points for context-dependent translations

The following user exit entry points support context-dependent translations, where translation proposals and automatic translations not only depend on text matches but also on the type of document containing the text. These entry points are designed to support the translation of Lotus Notes and Domino design elements, such as Notes database files, template files, and application templates. When OpenTM2 imports these documents (using the LOTUSNGD markup table), it

maintains context-dependent information about these design elements together with existing translations in the Translation Memory. If the user exit is used by the markup table, OpenTM2 uses the context information and the translation proposals to identify matches on the segments to be translated.

- "EQFGETCONTEXTINFO" is called once when a markup table is loaded. It returns information about the number and the names of context strings used in the Translation Memory, and it controls (based on the availability of context information) whether further context information processing is performed.
- "EQFGETSEGCONTEXT" on page 172 is called before a translated segment is saved in the Translation Memory. It gets the context strings from the user exit and passes them to the Translation Memory.
- "EQFUPDATECONTEXT" on page 172 is called subsequently for every segment during the analysis of a document and updates the user exit with the context strings from the Translation Memory for the current segment.
- "EQFCOMPARECONTEXT" on page 173 is called for every segment and compares and ranks a segment's context information against Translation Memory proposals.

OpenTM2 uses these user exit entry points to support the translation of Lotus Notes forms that contain the Form, Subform, Title, and Subtitle context strings.

EQFGETCONTEXTINFO

Purpose

EQFGETCONTEXTINFO is called once when a new markup table is loaded into the Translation Memory. It returns the number of context strings that are used by this markup and the names of these context strings (for example, `Panel ID` for MRI markup). If a markup table user exit does not support this entry point, or returns an error code, no further context information processing is performed for this markup table (neither *EQFGETSEGCONTEXT*, *EQFUPDATECONTEXT*, nor *EQFCOMPARECONTEXT* is called).

Format

Parameters

pusNumOfContextStrings
: The pointer to a USHORT variable receiving the number of context strings that are used by this markup.

pContextNames
: The pointer to a UTF16 buffer for the context names. This buffer has a size of MAX_CONTEXT_LEN(4096) characters. The context names are stored as a list of UTF-16 strings, and the list is terminated by `0x0000`.

  Currently the names will not be used. In a later version these names will be used in the translation environment to display the context of a segment.

Return code

The return code indicates whether context information could be returned.

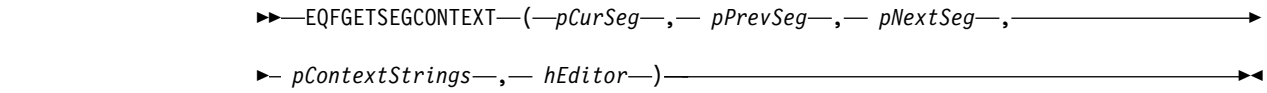
EQFGETSEGCONTEXT

Purpose

EQFGETSEGCONTEXT returns the context strings for a given segment and passes them to the Translation Memory functions before a segment is about to be saved in the Translation Memory.

This function is used by the editor during the translation. Using the supplied document handle the function can go backward or forward to other segments if necessary (for example, for an MRI markup it is necessary to go back to the segment containing the `panel ID`).

Format



Parameters

pCurSeg
: The pointer to a zero-terminated UTF-16 string containing the text of the current segment.

pPrevSeg
: The pointer to a zero-terminated UTF-16 string that contains the text of the previous segment (`NULL`, if there is none).

pNextSeg
: The pointer to a zero-terminated UTF-16 string that contains the text of the next segment (`NULL`, if there is none).

pContextStrings
: The pointer to a UTF16 buffer for the context strings. This buffer has a size of MAX_CONTEXT_LEN (4096) characters. The context strings are stored as a list of UTF-16 strings, and the list is terminated by `0x0000`.

hEditor
: The handle of type HANDLE, which is required for the EQFGetNextSeg and EQFGetPrevSeg functions.

Return code

The return code indicates whether context strings could be returned.

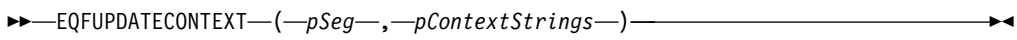
EQFUPDATECONTEXT

Purpose

EQFUPDATECONTEXT is called subsequently during the analysis of a document. If the current segment in the Translation Memory contains context information, this function updates the user exit with the context strings for this segment.

The retrieved context strings are used to identify exact context matches with the *EQFCOMPARECONTEXT* function.

Format



Parameters

pSeg
The pointer to a zero-terminated UTF-16 string containing the text of the current segment.

pContextStrings
The pointer to a UTF16 buffer containing the current context strings and receiving the updated context strings. This buffer has a size of MAX_CONTEXT_LEN(4096) characters. The context strings are stored as a list of UTF-16 strings, and the list is terminated by `0x0000`.

Return code

The return code indicates whether context strings could be updated.

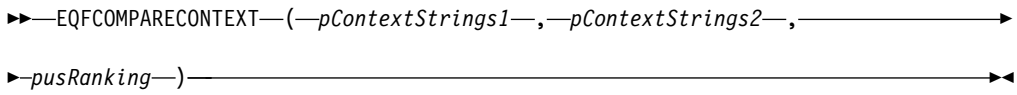
EQFCOMPARECONTEXT

Purpose

EQFCOMPARECONTEXT is called for every segment that has an exact text match and context information available. The function compares the context strings of a segment against the context strings of a Translation Memory proposal and ranks the match between 0 and 100. 0 means no context match at all, and 100 means an exact context match.

During an analysis only exact text matches *and* exact context matches of a segment lead to automatic substitutions. During a translation, the ranks are used to identify the best translation proposals.

Format



Parameters

pContextStrings1
The pointer to a buffer containing the context strings of the current segment. The context strings are stored as a list of UTF-16 strings, and the list is terminated by `0x0000`.

pContextStrings2
The pointer to a buffer containing the context strings of the proposal. The context strings are stored as a list of UTF-16 strings, and the list is terminated by `0x0000`.

pusRanking
The pointer to the variable receiving the ranking for the context strings.

Return code

The return code indicates whether context information could be compared.

Parser application programming interface

The following functions are internal OpenTM2 parsing functions that are made available to expand the possibilities of user exists. Their main purposes are:

- To access and modify segmented documents on a segment base.

  Documents can be loaded, and their segments can be retrieved and modified. Segments can be converted into an SGML tagged format. Code conversions can be done, and some document properties can be retrieved. Modified documents can be saved.
- To access and tokenize markup tables to get information about markup tags and property information.

  Markup tables can be loaded and tokenized, and the properties of markup tags can be accessed.

Because these are basically parsing functions, their names start with “Pars”. Function names ending with “W” are for Unicode documents, and for markup tables to be used with Unicode documents.

Note that these functions are not called at defined OpenTM2 processing steps (as opposed to the descriptions in “Parser application programming interface” and “User exit entry points for context-dependent translations” on page 170. However, they are well suited to be used in the code of one or more of these entry points. For example, they can be used to create or clean up markup tables. A sample parser that uses these parser API functions can be found in file `parssamp.c` in directory `\otm\nondde\`.

Further details about these functions, like the definition of data types, can be found in file `eqfpapi.h` in the same directory.

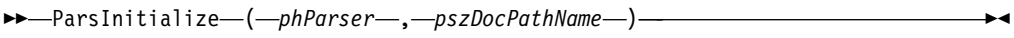
The following sections describe the parser API functions in detail. Where applicable, the parser API functions are enabled for Unicode UTF-16 support.

ParsInitialize

Purpose

ParsInitialize initializes the parser API environment and creates a parser API handle that is to be used in most of the other parser API functions.

Format



Parameters

Type	Parameter	Description
HPARSER	phParser	The pointer to the buffer for the parser API handle.
CHAR	pszDocPathName	The pointer to the zero-terminated document path name.

Return code

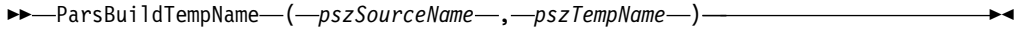
Integer of `0`, if the environment is successfully initialized, or an error code.

ParsBuildTempName

Purpose

ParsBuildTempName builds a temporary file name based on the fully qualified file name of the source document.

Format



Parameters

Type	Parameter	Description
PSZ	pszSourceName	The pointer to the zero-terminated fully qualified file name of the source document. The name serves as the model for the temporary file name.
PSZ	pszTempName	The pointer to the zero-terminated temporary file name. The buffer for the file name should have a size of 128 bytes or more.

Return code

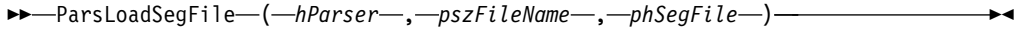
Integer of `0`, if the file name is successfully built, or an error code.

ParsLoadSegFile

Purpose

ParsLoadSegFile loads a segmented file into memory.

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.
CHAR	pszFileName	The pointer to the zero-terminated fully qualified file name of the document to be loaded into memory.
HPARSSEGFILE	phSegFile	The pointer to the buffer in memory that receives the segmented file.

Return code

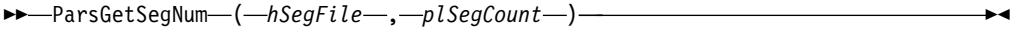
Integer of `0`, if the file is successfully loaded, or an error code.

ParsGetSegNum

Purpose

ParsGetSegNum returns the number of segments of the segmented file loaded into memory.

Format



Parameters

Type	Parameter	Description
HPARSSEGFILE	phSegFile	The handle of the segmented file in memory.
LONG	plSegCount	The pointer to the buffer that receives the number of segments.

Return code

Integer of `0`, if the number is successfully retrieved, or an error code.

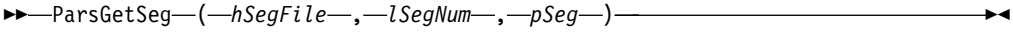
ParsGetSeg

Purpose

ParsGetSeg gets a segment from the segmented file loaded into memory.

If the segment in Unicode format, use "ParsGetSegW" on page 177.

Format



Parameters

Type	Parameter	Description
HPARSSEGFILE	hSegFile	The handle of the segmented file in memory.
LONG	lSegNum	The number of the segment to get.
PPARSSEGMENT	pSeg	The pointer to the buffer that receives the segment data.

Return code

Integer of `0`, if the segment is successfully retrieved, or an error code.

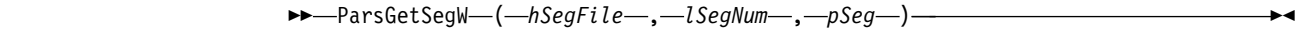
ParsGetSegW

Purpose

ParsGetSegW gets a segment from the segmented file loaded into memory.

If the segment not in Unicode format, use "ParsGetSeg" on page 176.

Format



Parameters

Type	Parameter	Description
HPARSSEGFILE	hSegFile	The handle of the segmented file in memory.
LONG	lSegNum	The number of the segment to get.
PPARSSEGMENTW	pSeg	The pointer to the buffer that receives the segment data.

Return code

Integer of `0`, if the segment is successfully retrieved, or an error code.

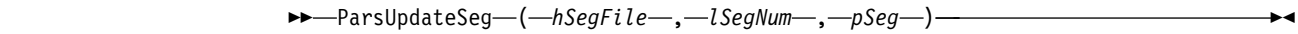
ParsUpdateSeg

Purpose

ParsUpdateSeg updates a segment of the segmented file loaded into memory.

If the segment is in Unicode format, use "ParsUpdateSegW" on page 178.

Format



Parameters

Type	Parameter	Description
HPARSSEGFILE	hSegFile	The handle of the segmented file in memory.
LONG	lSegNum	The number of the segment to update.
PPARSSEGMENT	pSeg	The pointer to the buffer that holds the new segment data.

Return code

Integer of `0`, if the segment is successfully updated, or an error code.

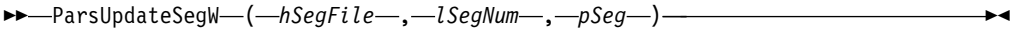
ParsUpdateSegW

Purpose

ParsUpdateSegW updates a segment of the segmented file loaded into memory.

If the segment is not in Unicode format, use "ParsUpdateSeg" on page 177.

Format



Parameters

Type	Parameter	Description
HPARSSEGFILE	hSegFile	The handle of the segmented file in memory.
LONG	lSegNum	The number of the segment to update.
PPARSSEGMENTW	pSeg	The pointer to the buffer that holds the new segment data.

Return code

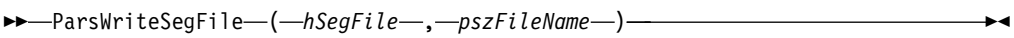
Integer of `0`, if the segment is successfully updated, or an error code.

ParsWriteSegFile

Purpose

ParsWriteSegFile writes the segmented file in memory to an external file.

Format



Parameters

Type	Parameter	Description
HPARSSEGFILE	hSegFile	The handle of the segmented file in memory.
CHAR	pszFileName	The pointer to the zero-terminated fully qualified file name of the document.

Return code

Integer of `0`, if the file is successfully written, or an error code.

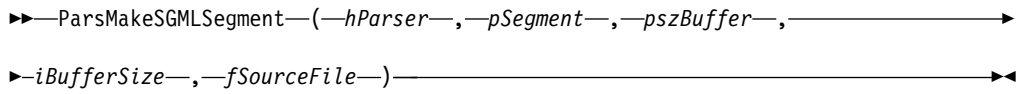
ParsMakeSGMLSegment

Purpose

ParsMakeSGMLSegment builds an SGML tagged segment as used in segmented files.

If the segment is in Unicode format, use “ParsMakeSGMLSegmentW.”

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.
PPARSSEGMENT	pSegment	The pointer to the buffer that holds the segment data.
CHAR	pszBuffer	The pointer to the buffer that receives the zero-terminated SGML-tagged segment. The buffer size for the segment should be at least twice the maximum segment size.
INT	iBufferSize	The size of *pszBuffer*.
BOOL	fSourceFile	**TRUE** Create SGML for a segmented source file. **FALSE** Create SGML for a segmented target file.

Return code

Integer of `0`, if the segment is successfully built, or an error code.

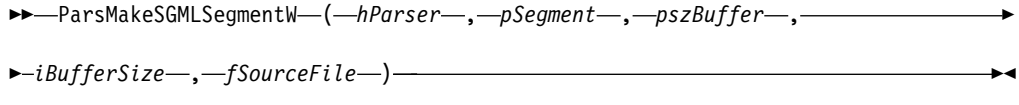
ParsMakeSGMLSegmentW

Purpose

ParsMakeSGMLSegmentW builds an SGML tagged segment as used in segmented files.

If the segment is not in Unicode format, use “ParsMakeSGMLSegment” on page 178.

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.

Type	Parameter	Description
PPARSSEGMENTW	pSegment	The pointer to the buffer that holds the segment data.
WCHAR*	pszBuffer	The pointer to the buffer that receives the zero-terminated SGML-tagged segment (in Unicode UTF-16 format). The buffer size for the segment should be at least twice the maximum segment size.
INT	iBufferSize	The size of *pszBuffer*.
BOOL	fSourceFile	**TRUE** Create SGML for a segmented source file. **FALSE** Create SGML for a segmented target file.

Return code

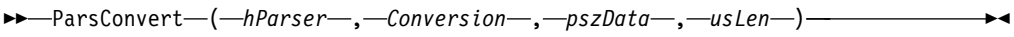
Integer of `0`, if the segment is successfully built, or an error code.

ParsConvert

Purpose

ParsConvert performs an in-place conversion from ASCII to ANSI, or vice versa.

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.
PARSCONVERSION	Conversion	The conversion mode: **ASCIItoANSI** **ANSItoASCII**
CHAR	pszData	The pointer to the zero-terminated data to be converted.
USHORT	usLen	The length of the data to convert.

Return code

Integer of `0`, if the conversion is successful, or an error code.

ParsGetDocName

Purpose

ParsGetDocName returns the long document name.

Format

Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.
CHAR	pszDocName	The pointer to the buffer that receives the zero-terminated long document name. The size of the buffer should be 256 bytes.

Return code

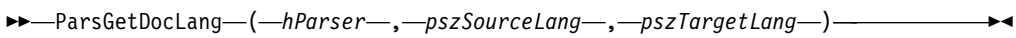
Integer of `0`, if the document name is successfully returned, or an error code.

ParsGetDocLang

Purpose

ParsGetDocLang returns the language settings of the current document.

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.
CHAR	pszSourceLang	The pointer to the buffer that receives the zero-terminated source language, or `NULL`. The buffer size should be 40 bytes or more.
CHAR	pszTargetLang	The pointer to the buffer that receives the zero-terminated target language, or `NULL`. The buffer size should be 40 bytes or more.

Return code

Integer of `0`, if the language setting are successfully returned, or an error code.

ParsSplitSeg

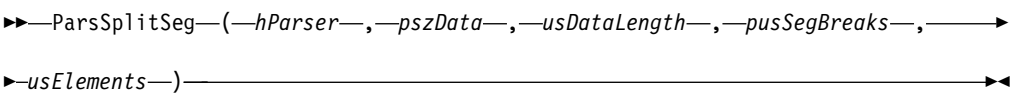
Purpose

ParsSplitSeg splits text data into segments by using OpenTM2's morphological functions. The function looks for segment breaks in the supplied data by applying the morphology for the document source language. The segment breaks are returned as a list of segment breaks. This list is a list of offsets of segment breaks within the data. The last element in this list is zero.

If the buffer for this list is too small, the function returns an error and the first element of the list contains the required size of the list (in number of list elements).

If the text data is in Unicode format, use "ParsSplitSegW."

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.
CHAR	pszData	The pointer to the zero-terminated text data that is to be split into segments.
USHORT	usDataLength	The length of the text data, as number of characters.
USHORT	pusSegBreaks	The pointer to the buffer that receives the list of segment breaks.
USHORT	usElements	The size of the buffer that receives the list of segment breaks, in number of list elements.

Return code

Integer of `0`, if the segment is successfully split, or an error code.

ParsSplitSegW

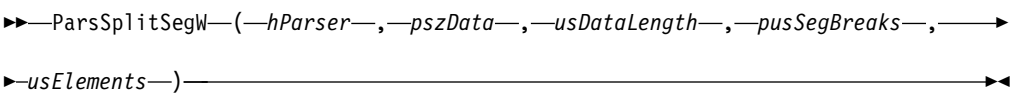
Purpose

ParsSplitSegW splits text data into segments by using OpenTM2's morphological functions. The function looks for segment breaks in the supplied data by applying the morphology for the document source language. The segment breaks are returned as a list of segment breaks. This list is a list of offsets of segment breaks within the data. The last element in this list is zero.

If the buffer for this list is too small, the function returns an error and the first element of the list contains the required size of the list (in number of list elements).

If the text data is not in Unicode format, use "ParsSplitSeg" on page 181.

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.
WCHAR*	pszData	The pointer to the zero-terminated text data (in Unicode UTF-16 format) that is to be split into segments.
USHORT	usDataLength	The length of the text data, as number of UTF-16 characters.
USHORT	pusSegBreaks	The pointer to the buffer that receives the list of segment breaks.
USHORT	usElements	The size of the buffer that receives the list of segment breaks, in number of list elements.

Return code

Integer of `0`, if the segment is successfully split, or an error code.

ParsFreeSegFile

Purpose

ParsFreeSegFile frees a segmented file from memory.

Format

Parameters

Type	Parameter	Description
HPARSSEGFILE	hSegFile	The handle of the segmented file in memory.

Return code

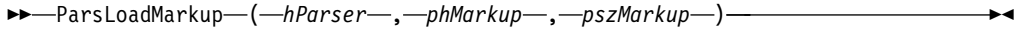
Integer of `0`, if the memory is successfully freed, or an error code.

ParsLoadMarkup

Purpose

ParsLoadMarkup loads a markup table into memory for usage with the *ParsTokenize* or *ParsTokenizeW* function. The markup table is loaded from the `\otm\table` directory.

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.
HPARSMARKUP*	phMarkup	The pointer to the buffer in memory that receives the markup handle.
CHAR	pszMarkup	The pointer to the zero-terminated markup table name (without path and extension, for example, `EQFANSI`).

Return code

Integer of `0`, if the markup table is successfully loaded, or an error code.

ParsTokenize

Purpose

ParsTokenize looks for tags in the supplied text area of the markup table loaded into memory. The result is a tag token list that can be processed by the *ParsGetNextToken* function.

If the supplied text area is in Unicode format, use "ParsTokenizeW."

Format

Parameters

Type	Parameter	Description
HPARSMARKUP	hMarkup	The markup handle, created by the *ParsLoadMarkup* function.
CHAR*	pszData	The pointer to the zero-terminated text area that is to be tokenized.

Return code

Integer of `0`, if the markup table is successfully tokenized, or an error code.

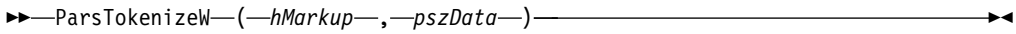
ParsTokenizeW

Purpose

ParsTokenizeW looks for tags in the supplied text area of the markup table loaded into memory. The result is a tag token list that can be processed by the *ParsGetNextToken* function.

If the supplied text area is not in Unicode format, use "ParsTokenize."

Format



Parameters

Type	Parameter	Description
HPARSMARKUP	hMarkup	The markup handle, created by the *ParsLoadMarkup* function.
WCHAR*	pszData	The pointer to the zero-terminated Unicode text area that is to be tokenized.

Return code

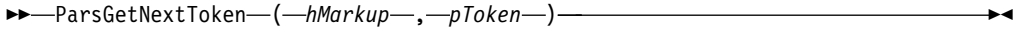
Integer of `0`, if the markup table is successfully tokenized, or an error code.

ParsGetNextToken

Purpose

ParsGetNextToken returns the next token from the token list created by the *ParsTokenize* and *ParsTokenizeW* functions. At the end of the token list a token with a token ID of `PARSTOKEN_ENDOFLIST` is returned. "The PARSTOKEN structure" describes the token structure in detail.

Format



Parameters

Type	Parameter	Description
HPARSMARKUP	hMarkup	The markup handle, created by the *ParsLoadMarkup* function.
PPARSTOKEN	pToken	The pointer to a PARSTOKEN structure (see "The PARSTOKEN structure") that receives the data of the token.

Return code

Integer of `0`, if the next token is returned, or an error code.

The PARSTOKEN structure

This structure holds the token information of a token that is returned by the *ParsGetNextToken* function.

Type	Name	Usage
INT	iTokenID	The token ID of the token returned. The token ID represents the position of the tag in the markup table. • A token ID of `PARSTOKEN_ENDOFLIST` represents the end of the tag token list. • A token ID of `PARSTOKEN_TEXT` (text token) represents text which is not recognized as a tag.
INT	iStart	The start position (in characters, not bytes) of the token in the text area (see parameter *pszData* of the *ParsTokenize* or *ParsTokenizeW* function).
INT	iLength	The length of the token (in number of characters, not bytes).
USHORT	usFixedID	A fixed token ID, or `NULL` if none is specified for the tag in the markup table.
USHORT	usAddInfo	Additional tag information, or `NULL` if none is specified for the tag in the markup table.
USHORT	usClassID	A Class ID, or `NULL` if none is specified for the tag in the markup table.

ParsFreeMarkup

Purpose

ParsFreeMarkup frees a markup table loaded with the *ParsLoadMarkup* function from memory.

Format

Parameters

Type	Parameter	Description
HPARSMARKUP	hMarkup	The markup handle, created by the *ParsLoadMarkup* function.

Return code

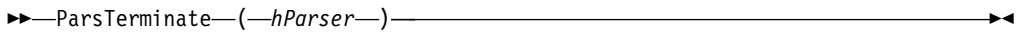
Integer of `0`, if the markup table is freed from memory, or an error code.

ParsTerminate

Purpose

ParsTerminate terminates the parser API environment.

Format



Parameters

Type	Parameter	Description
HPARSER	hParser	The parser API handle, created by the *ParsInitialize* function.

Return code

Integer of `0`, if the environment is successfully terminated, or an error code.

Part 2. Appendixes

Appendix. Notices

Trademarks

The following terms are trademarks of IBM in the United States, other countries, or both:

IBM

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Glossary of terms and abbreviations

This glossary defines and describes terms and abbreviations used in this manual.

addendum
The extension of a *language-support file* that contains individually added spellings of terms. For example, terms which have been indicated as misspelled by the spellchecker although spelled correctly.

aligning
The process of combining source segments with their corresponding target segments in an Initial Translation Memory (ITM).

analysis
A process for dividing text into *segments*. It checks the text against specific *exclusion lists* and *dictionaries*, and produces, on your request, a *new terms list* and a *found terms list*.

ANSI American National Standards Institute.

API *Application programming interface*.

application programming interface (API)
A software interface that enables applications to communicate with each other. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by an underlying operating system or service program.

automatic lookup
During translation, OpenTM2 performs an automatic lookup in the referenced *Translation Memory* and in the referenced *dictionaries*. For each segment, matching segment translations from the Translation Memory are displayed as *translation proposals* in the "Translation Memory" window, translations of its terms are displayed in the "Dictionary" window.

automatic substitution
An option in the Translate menu. It lets you start the automatic substitution process, which translates those *segments* that have been previously translated by you or another translator and are stored in the *Translation Memory*. It is particularly useful for translating updated text. However, you still must translate new text manually.

company code
Abbreviation for a particular area of usage a translation applies to. For example, certain terms are used differently depending on the companies or clients you do translations for.

controlled folder handling
Is a concept that is only available to project coordinators. It allows them to specify, and change at any time, all properties and details for a folder, including the translators for the documents to be imported into this folder. It also allows them to ship the folder once all translations are finished.

details
See *view details*.

dictionary
A database that contains terms, their translation, and other related information.

dictionary entry
All data relating to a *headword* in a *dictionary*

dictionary filter
A method to select specific entries from a *dictionary* or only parts of these entries. The filter conditions that must be met if an entry is to pass the filter can be individually defined when printing or searching a dictionary.

dictionary print format
Specifies the layout of a printed *dictionary*. OpenTM2 provides standard formats described in *format files* that can be tailored individually. The format files are on the same disk where OpenTM2 is installed under the subdirectory `eqf\prtform`.

DLL *Dynamic-link library*.

document file
A generic term used to describe all types of files containing information that is to be translated. Document files can be analyzed and opened for translation in

the *Translation Environment*. The source of the document file you translate is called the *original document*. The document file that you edit during translation is referred to as the *translation document*.

document type
Depending on the different types of *markup* used to describe the layout of document, OpenTM2 differentiates between different document types.

dynamic-link library (DLL)
A file containing executable code and data bound to a program at load time or runtime, rather than during linking. The code and data in a dynamic-link library can be shared by several applications simultaneously.

entry fields
The various fields and styles of an entry in a *dictionary*, such as meaning, usage, context, abbreviation, idioms, and grammatical information. For example, the entry field *Abbr.* would contain the abbreviation of a *headword*. The combination of all entry fields of a specific headword makes up the headword's entry in the dictionary.

entry level
The information that applies to all the *templates* of an entry. For example, the term itself, the author, and the date the entry was created.

entry section
Section in a *dictionary*. Contains all *dictionary entries* appearing one after another.

exact match
Each *segment* in the *translation document* is compared with the selected *Translation Memory*. If an identical segment is found, an *exact match* has occurred and the corresponding *translation proposal* is shown in the "Translation Memory" window. It originates from a previous translation.

exact match (1)
An *exact match* for which the following condition applies:The exact match occurs only once in the attached Translation Memory databases.

exact match (>=2)
An *exact match* for which the following condition applies:The exact match occurs at least twice in the attached Translation Memory databases.

exact-exact match
An *exact match* for which the following condition applies:The number of the active segment in the source document is identical (give or take 2) with that of the corresponding segment in the Translation Memory. In addition, the name of the document (document name = file name plus relative path (if available)) being translated is identical with that of the document stored in the Translation Memory.

exact context match
An *exact match* for which the following condition applies:The number of the active segment in the source document is not identical with that of the corresponding segment in the Translation Memory. However, the name of the document being translated is identical with that of the document stored in the Translation Memory.

exclusion list
A list containing common words such as articles, prepositions, proper nouns, and terms that occur frequently. These words are ignored when creating *new terms lists* and *found terms lists* during *analysis*, and are not shown in the "Dictionary" window during translation. Exclusion lists can be edited.

export To copy *folders*, documents, *dictionaries*, and *Translation Memory databases* to the DOS file system to make them available to another user.

folder Contains documents belonging to one project and references to the *Translation Memory databases* and *dictionaries* you want to use during translation.

format file
A file that contains the specification of a *dictionary print format*. It can be created and changed with a text editor.

found terms list
A list of all terms in the documents being analyzed that were found in the selected *dictionaries*. The list is used to update dictionaries and *exclusion lists*. Found terms lists can be edited, that is, terms

can be deleted, moved to a dictionary, or to an *exclusion list*. A found terms list can be used to fill a separate dictionary related to a document.

fuzzy match
Each *segment* in the *translation document* is compared with the selected *Translation Memory*. If an almost identical segment is found, a fuzzy match has occurred and the corresponding *translation proposal* is shown in the "Translation Memory" window with a preceding [f]. It originates from a previous translation.

fuzzy replacement match
A *replacement match* where a couple of words are not identical. It is displayed in the "Translation" window with a preceding [rf].

Example:

```
Document text: This is what happened in 1998.
TM proposal:   This happens in 1999.
```

In this example, the date in the TM proposal (1999) is automatically changed to the date in the document text (1998). However, `happened` is not replaced with `happens`.

header section
Section in a *dictionary*. Contains general dictionary information such as source language, target language, and creation date of the dictionary.

headword
Word or term placed at the beginning of an entry in a *dictionary*.

history log file
A file storing, in compressed form, records that contain the information collected during events, such as exporting or deleting a folder, and the result of this collection. There is one history log file per folder, which is stored as `HISTLOG.DAT` in the `PROPERTY` directory of the folder. New records are added at the end of the history log file.

homonym
Words that are spelled and pronounced alike but different in meaning. For example, the noun conduct and the verb conduct are homonyms.

homonym level
Part of a *dictionary entry*. Contains grammatical and syntactic information, such as part of speech, hyphenation, and abbreviation information.

HTML
Hypertext Markup Language.

Hypertext Markup Language (HTML)
A subset of the Standard Generalized Markup Language (SGML) allowing the presentation of electronically stored information within the World Wide Web (Internet).

icon A small graphical symbol. Icons can represent windows that you want to work with (such as Folder list, Document list, Dictionary list, Translation Memory list, Terminology lists) or tasks that you want to perform.

import
To copy *folders*, documents, *dictionaries*, and *Translation Memory databases* from the DOS file system to make them available to OpenTM2.

Initial Translation Memory (ITM)
A *Translation Memory* created from existing translations and their corresponding originals. Proposals originating from an ITM are shown in the "Translation Memory" window with a preceding [m] like *machine-generated matches*.

irregular match
One of the following:

- A 1:2 match, where one source segment has been connected to two target segments
- A 2:1 match, where two source segments have been connected to one target segment
- A 2:2 match, where two source segments have been connected to two target segments
- An unaligned sentence (the default color is red)
- A sentence that is ignored (the default color is grey)

ITM *Initial Translation Memory*.

JavaScript
A scripting language that resembles Java™ and was developed by Netscape for use with the Netscape browser.

language support files
Source languages supplied with OpenTM2. Language support files are required when looking up *dictionary entries* during *analysis* of document files and during *spellcheck*.

lookup
See *automatic lookup* and *search*.

machine-generated match
Originates from an *Initial Translation Memory* and is displayed in the "Translation Memory" window with a preceding [m]. Can be used in the same way as a *fuzzy match*.

maptable section
Section in a *dictionary*. Determines the structure of *dictionary entries*. Contains the total of all allowed entry fields in a dictionary.

markup
Information added to a document, for example, formatting tags, to enable a system to process it. It describes the document characteristics or specifies the actual processing to be performed.

markup language
The language specific to a word processor that describes a document layout.

markup table
Contains all tags and attributes of a particular *markup language*. Is used in OpenTM2 during *analysis* and translation.

match The fact that a source *segment* in a Translation Memory and a source segment in a document to be translated at least resemble each other (*fuzzy match* or *replacement match*). If they are completely identical, it is an *exact match* if the translation was done by a translator, or a *machine-generated match* if the translation is generated by a program.

merge Combining information of either two *dictionaries* or two *Translation Memory databases*. When merging dictionaries, OpenTM2 preserves the structure of the destination dictionary.

model dictionary
An already existing *dictionary* whose structure can be taken as a sample when creating a new dictionary.

model folder
An already existing *folder* whose *properties* can be taken as a sample when creating a new folder.

new terms list
A list of all the terms found in the documents being analyzed but not found in the selected *dictionaries* during *analysis*. New terms lists can be used to update dictionaries and *exclusion lists*. New terms lists can be edited, that is, terms can be deleted, moved to a dictionary, or to an exclusion list.

organize
Internal restructuring of frequently changed *dictionaries* and *Translation Memory databases* to shorten search times.

original document
The source of the document that you translate. You cannot edit this document but you can display it and use it for comparison or checking purposes.

postediting
Editing an already translated document. Any changes cause an automatic update of the already translated *segments* in the *Translation Memory*.

properties
A summary of the different characteristics of a *folder* or a document, such as a description, the *markup language* used in documents, and references to *Translation Memory databases* and *dictionaries*.

replacement match
An *exact match* where only a number or date differs. It is displayed in the "Translation" window with a preceding [r].

Example:

```
Document text:   This happened in 2015.
Memory proposal: This happened in 2014.
```

In this example, the date in the translation memory proposal (2014) is automatically changed to the date in the document text (2015).

reversing
Turning source segments contained in a Translation Memory into target segments and vice versa.

revision marks
Characters at the beginning and end of a

segment that can be individually defined and indicate that the enclosed segment has been translated from scratch, or by copying a *translation proposal* and changing it, or by copying a proposal without changing it.

search In the "Look up a Term" window, you can search for terms in a dictionary using predefined search criteria and user-definable *dictionary filters*. See also *automatic lookup*.

segment
A translation unit produced during *analysis*. It is usually a sentence, part of a sentence, an element of a list, or a citation.

sense level
Part of a *dictionary entry*. Contains semantic variations of a *headword* such as varying areas of meaning and usage.

SGML
Standard Generalized Markup Language.

shared translation material
A dictionary or Translation Memory file located on a shared disk. It can be concurrently accessed by all OpenTM2 users who are connected to the same LAN.

source document
See *original document*.

spellcheck
A proofreading aid to identify unrecognized or misspelled words in *translation documents*. Lists possible corrections for misspelled words.

Standard Generalized Markup Language (SGML
A set of rules that allows the format specification of a *markup language* independent of any individual processing system. The external file formats created during export are based on SGML.

stem The part of an inflected word that remains unchanged except by phonetic changes or variations throughout an inflection.

subject code
Abbreviation for a particular subject area a translation applies to.

tag Statement used to determine the format of a *document file*. Is contained in a *markup table*.

target document
See *translation document*.

target level
Contains all information applying to one translation variant of a *headword*, such as definition and usage.

template
Dictionary entry information on all levels (*entry*, *homonym*, *sense*, *and target*) relating to one specific translation of a *headword*.

terminology list
A generic term for the following types of lists: *exclusion lists*, *found terms lists*, and *new terms lists*.

translation document
The document that you translate.

Translation Environment
Environment where the actual translation is performed. It consists of a window where you can edit the document file, a window with proposals from the associated *Translation Memory*, and a window with translations for terms in the document. All *translation proposals* can be copied into the *translation document*.

Translation Memory
A database that contains previously translated *segments* added during translation and *analysis*.

Translation Memory databases
More than one *Translation Memory*.

translation proposal
The translation of a *segment* found in a *Translation Memory* during translation, where the source segment is identical (*exact match*) or almost identical (*fuzzy match*) to the current segment.

user exit
A point in a program at which a user exit routine may be given control.

A programming service provided by a software product that may be requested during the execution of an application program for the service of transferring control back to the application program upon the later occurrence of a user-specified event.

view details
Contents of the list windows displayed in the main window. You can define how detailed the contents of these lists is to be displayed. The default is to display only the names of the individual list items.

word count
Utility to count words (words to be translated, words already translated, *markup* tags) in *original documents* or *translation documents*.

workbench
The OpenTM2 main window.

Index

M

P

S

T

U

Printed in USA