Appendix B : Application Programming Interface Reference

Please note that this appendix is only available in English.

Introduction

▪TRADOS Translator's Workbench is shipped with templates that enable data exchange between Translation Memory databases and your word processor. However, text to translate not only occurs in word processors but also in such applications as spreadsheet or presentation programs, for example. Many of these applications are equipped with an interface that allows communication with other programs through what is referred to as an Application Programming Interface (API). Translator's Workbench 2 is also equipped with an API. This makes it possible to access Workbench functionality from those applications. In general, API-enabled programs offer a development environment like macro editors in which the code for the user-defined functions can be written. This manual contains a description of the Translator's Workbench functions that are available through its API.

Writing Conventions

Command lines are given before the instruction they refer to. Sets of values are listed in curly brackets. Each item of the set is separated by a semicolon.

API Description

The API's class module is TW4Win. It has three subclasses on separate levels. The first-level class is **TranslationMemory**. The second-level class depending from it is **TranslationUnit** which in turn contains the third-level class **Term**. The class module and the **TranslationMemory** class are described in the API tutorial below. The **TranslationUnit** and **Term** classes will be covered in separate sections.

The diagram below provides an overview of all classes, methods and properties available in the Translator's Workbench API. It also shows the dependency relation between methods, properties and their respective classes. Classes are represented by boxes, methods by ovals, and properties by parallelograms.

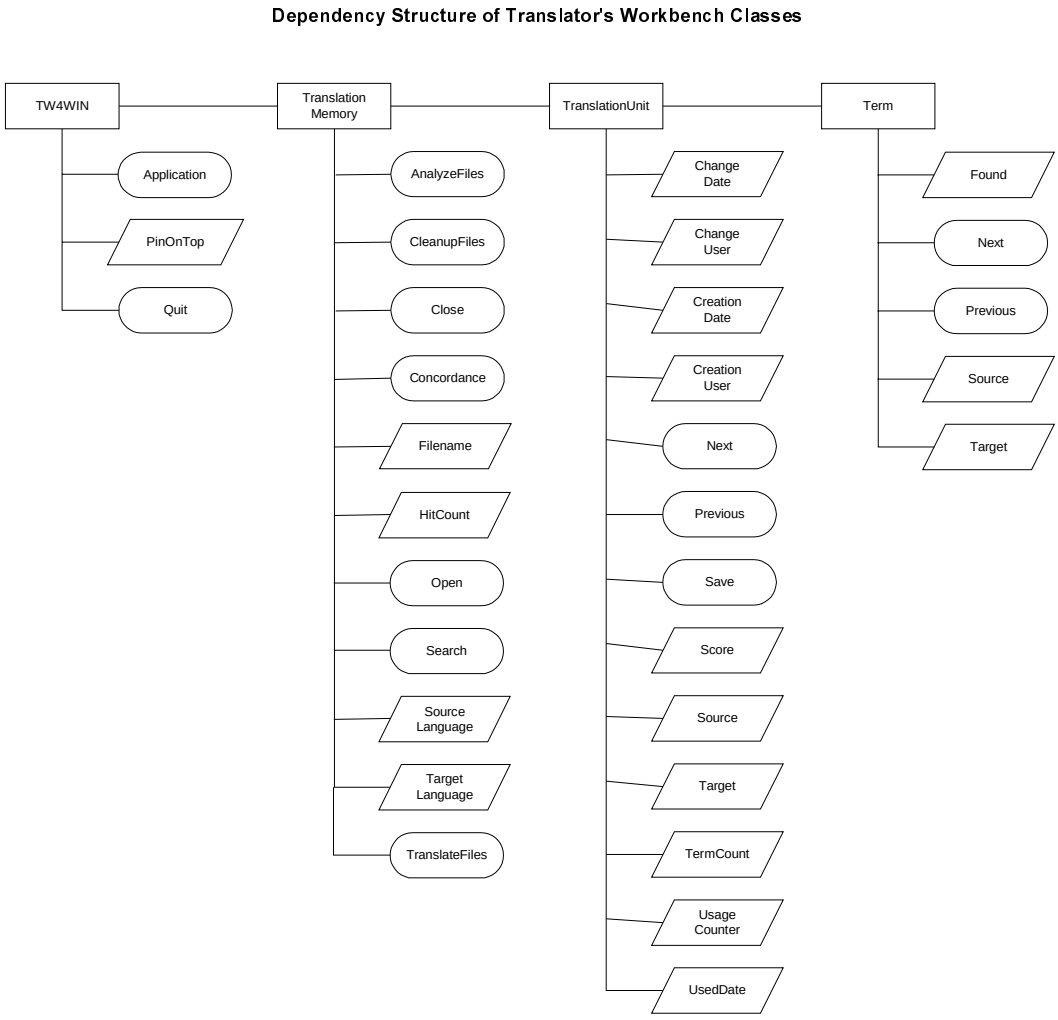

Figure B–1: Dependency structure of Translator's Workbench API Classes

API Tutorial

The methods in the diagram above correspond to commands that allow access to Translation Memory at various levels. We will describe and explain them below in the form of a tutorial. Short examples show how to implement the methods. This will allow you to follow the instructions step by step. At the same time a sample program is created that will help you understand the core functionality of the Translator's Workbench API. The examples below have

been written in Visual Basic for Applications as found in Microsoft Office 97. Proficient C++ programmers should be able to easily transfer them to their programming language.

Preparing Word97® For Writing Programs With Visual Basic For Applications®

1. Start Word97. To open the editor for Visual Basic for Applications press <Alt> + <F11> or select the Macro command in the Tools menu. From the sub-window select Visual Basic Editor.

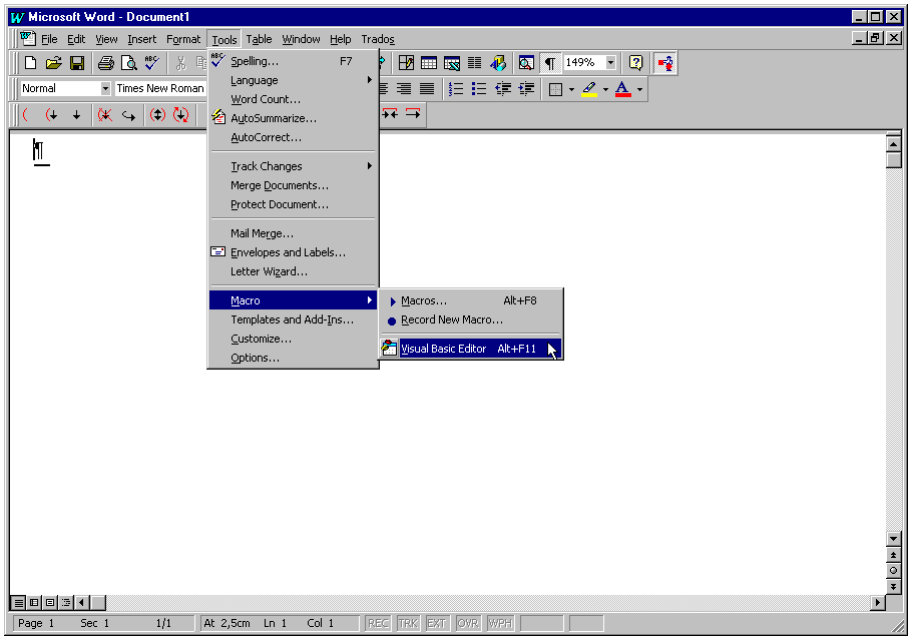

Figure B–2: Starting the Visual Basic Editor

2. The editor window opens. The next step is to inform the Visual Basic environment that you want to use the Translator's Workbench API and where the class module is located on your system. Therefore we have to load the Translator's Workbench type library. Open the Tools menu and select the Reference command. The Reference window opens. Click on Browse to locate the Workbench type library. Let's assume that the Workbench is installed in C:\Trados\TW4Win. Go to this directory, select the file TW4Win.tlb and click on Open.

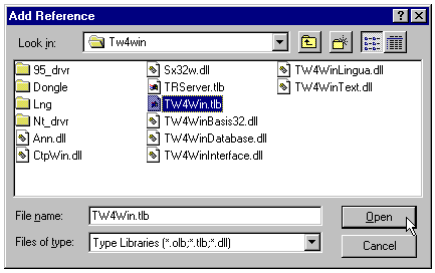

Figure B–3: Selecting the type library

3. The entry TW4WIN is added to your Reference window. Use the up or down arrow to move the entry at the desired position. Click OK to close the Reference window.

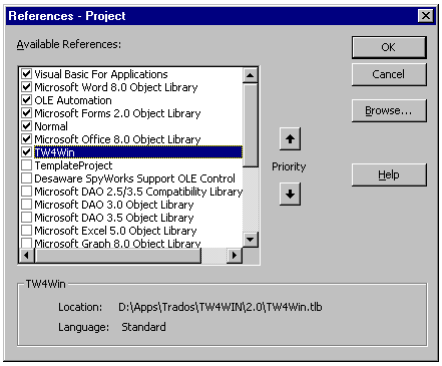

Figure B–4: Reference List with Workbench library

4. It is recommended that you use the Object Browser to have an overview of Workbench API functions. Open it by pressing <F2> or select the Object Browser command from the View menu.

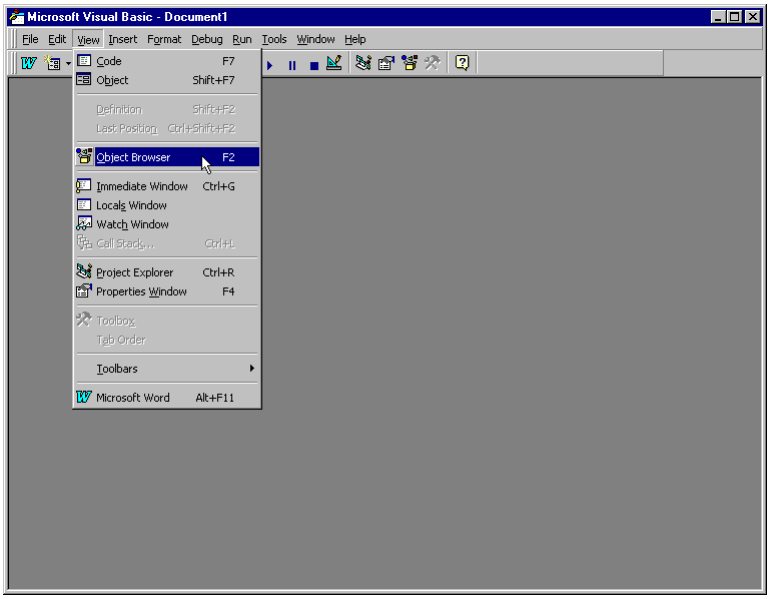

Figure B–5: Opening the Object Browser

5. The Object Browser opens. In the upper selection box pick the value TW4WIN. Now you can see the Translator's Workbench classes in the browser. Select the class you are interested in and the methods and properties are displayed. Click on an item in the right panel and you will see its type, parameters and a short description.

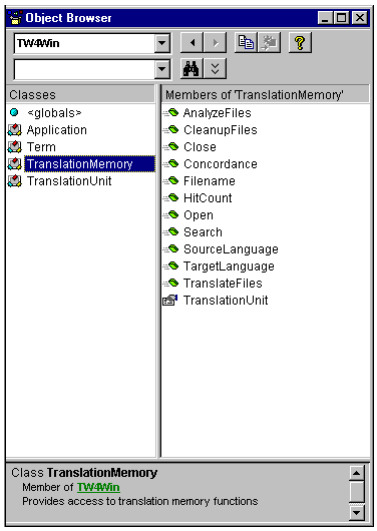

Figure B–6: The Object Browser window

6. Now you are ready to write your first program with the Translator's Workbench API functions.

Accessing the Application

To be able to access a Translation Memory an object has to be created first of all. This object will then later refer to a running Translator's Workbench instance. This is achieved by declaring a variable of the "Object" type. The next step is to start a Translator's Workbench and instantiate the object. The name of the class module is TW4Win and the method that launches a Workbench is **Application**.

```
Public Sub Main()
'Declare a variable named Workbench which refers to an object
Dim Workbench As Object
'Start a new Translator's Workbench
Set Workbench = CreateObject("TW4Win.Application")
```

After executing these instructions, a new Translator's Workbench will be started up. Such Workbench settings as penalties, window dimensions, etc. are stored in the registry in the key **My Computer\HKEY_CURRENT_USER\Software\TRADOS\TW4Win** and its subkeys **Last Directory**, **Last Session**, **Options** and **Tools**. At normal (that is, non-API controlled) start-up, Translator's Workbench reads the complete contents of these keys and initialises all settings with the values it finds there. As a result, when you start Translator's Workbench from your desktop, the last opened Translation Memory will be re-opened automatically. However, when launching Translator's Workbench through its API, this step is *omitted*. So, after the execution of the above instruction *no* Translation Memory will be opened.

Loading a Translation Memory

Since a TM is not opened when the Workbench is started through the API, this must be done explicitly using the **Open** method. This method has three parameters. The first obligatory parameter contains the full path and name of the Translation Memory. The second also obligatory parameter contains a UserID. This may be the ID of a "real" user as well as the ID of a "virtual" one. For instance, you may want to use a virtual user ID to indicate that Translator's Workbench is accessed through the API. The third parameter is usually optional and contains the password for the user specified. It is obligatory, however, if access rights have been specified in

the Translation Memory setup. For details on access rights, see the "Network Operation" chapter of the Translator's Workbench User's Guide.

```
'Open the TM demo.tmw in the path C:\Trados\Tw4Win with the userID FLAGMAN
'who has the password "ahead"
Workbench.TranslationMemory.Open "C:\Trados\Tw4Win\demo.tmw", "flagman",
      "ahead"
```

If the TM can be opened successfully, the following properties are instantiated: **Filename**, **SourceLanguage**, and **TargetLanguage**. These properties can be used for testing whether the Translation Memory was opened successfully or not.

Searching the Translation Memory

To find a sentence in Translation Memory the **Search** method is used. This method looks through the TM to find one or more matching sentences in the source segment of a translation unit. Such settings as the minimum match value or the maximum number of hits are taken into account during this search (see the "Translation Memory Options" chapter for details). If the search is successful the corresponding unit will be opened and the result will be displayed in the Translation Memory window. At the same time all properties related to the Translation Memory and current translation unit are instantiated. The **HitCount** property can be used to determine if, and how many, matching sentences have been found. If the search is successful, this property will contain the number of matches from the TM. Otherwise it will have the value 0.

```
'Perform a search for a string
Workbench.TranslationMemory.Search ("What exactly is a Translation Memory
      (TM)?")
```

Saving a Translation

The **Save** method is used to save a translation in Translation Memory. If a previous search was without result a new translation unit will be created. If a previous search was successful the previous translation will be overwritten with the new one. After this the translation unit will be closed so that the system is ready for the next search.

```
'Assign a string as new translation to the open segment
Workbench.TranslationMemory.TranslationUnit.Save ("Was genau ist ein
      Translation Memory (TM)?")
```

Concordance Search

The API can also perform a Concordance search, that means, it can search for all translation units that contain a specific word or phrase. The method for this is called **Concordance**. It works like the **Search** method, that is, a search string has to be specified as the parameter of this method. The search results will be displayed in a Concordance window. This window will close once Translator's Workbench is exited.

```
'Search for all sentences that contain the term "Translation Memory"
Workbench.TranslationMemory.Concordance ("Translation Memory")
```

Calling Batch Functions

Translator's Workbench has three so-called "batch functions" which are **Analyze**, **Translate** and **Cleanup** (see the "Document Analysis, Translation, and Cleanup" chapter for details). These functions can be executed through the API. The corresponding methods are **AnalyzeFiles**, **TranslateFiles**, and **CleanupFiles**. Each of the methods uses a job file as parameter. The job file contains the settings for the batch functions. For further details see "The Job File" on page B–25.

```
'Analyze documents using the settings specified in the job file
Workbench.TranslationMemory.AnalyzeFiles ("C:\Trados\Tw4Win\job.txt")

'Translate documents using the settings specified in the job file
Workbench.TranslationMemory.TranslateFiles ("C:\Trados\Tw4Win\job.txt")

'Clean up documents using the settings specified in the job file
Workbench.TranslationMemory.CleanupFiles ("C:\Trados\Tw4Win\job.txt")
```

Closing a Translation Memory

At the end of each program using the API, the current Translation Memory should be closed with the **Close** method. Any open remaining translation unit will be closed without saving. After the execution of this method it is possible to open another Translation Memory.

```
'Close the TM
Workbench.TranslationMemory.Close
```

Exiting Translator's Workbench

Likewise, Translator's Workbench should be closed properly with the **Quit** method. When exiting Translator's Workbench, the current user settings are written to the registry key **My Computer\HKEY_CURRENT_USER\Software\TRADOS\TW4Win**. Translator's Workbench will also close if the program from which it was called is terminated, but in this situation it is not assured that all settings are written correctly to the registry. Therefore it is strongly recommended that Translator's Workbench be exited with the **Quit** method. Any remaining open Concordance windows will also close once this method is called.

```
'Exit Workbench
Workbench.Quit
```

The "TranslationMemory" Properties

As already mentioned above, the **Filename** property contains the path and name of the current Translation Memory database. **HitCount** contains the number of matching translation units after a search. It is 0 if no match has been found. **SourceLanguage** and **TargetLanguage** contain the ID of the input locale of each language. In addition to these items, you can use the **PinOnTop** property which directly depends on the **TW4Win** class. It determines whether the Workbench program window is the topmost window or not. The program sample below shows how these different information items can be displayed through the API.

```
'Show the path and the name of the current TM in a message box
MsgBox "The current Translation Memory is: " &
       Workbench.TranslationMemory.FileName, vbInformation

'Read the locale ID of source and target language into variables
SrcLng = Workbench.TranslationMemory.SourceLanguage
TrgLng = Workbench.TranslationMemory.TargetLanguage

'Display the source and target language in message boxes
MsgBox "The current source language is: " & Languages(SrcLng).Name,
       vbInformation
MsgBox "The current target language is: " & Languages(TrgLng).Name,
       vbInformation

'Make Workbench the topmost window
Workbench.PinOnTop = True
```

The "TranslationUnit" Class

The **TranslationUnit** class enables you to read or change the contents of a translation unit.

TranslationUnit Methods

One **TranslationUnit** method has already been used in the tutorial, namely the **Save** method. It stores a translation together with the source sentence in Translation Memory. If a search results in more than one matching translation unit the user can access the other matches through the **Next** and **Previous** method. When one of these methods is called the next or previous matching translation unit is displayed in the Translation Memory window of the Workbench. At the same time all translation unit properties are instantiated with the values of the corresponding TU.

```
'Go to the next matching TU
Workbench.TranslationMemory.TranslationUnit.Next
'Go back to the previous TU
Workbench.TranslationMemory.TranslationUnit.Previous
```

TranslationUnit Properties

The properties of a translation unit are too numerous to be mentioned here. Most of them correspond to system fields that are specified when creating a new Translation Memory. If a system field does not exist or is not filled, the corresponding property will remain empty. For a comprehensive overview of system fields, see "Creating a new Translation Memory" in the User's Guide. Apart from system fields, the **TranslationUnit** class contains four properties that are worth taking a brief look at. The **Score** property contains the match value of a found translation unit. The number of recognised terms can be retrieved via the **TermCount** property. **Source** and **Target** contain the contents of the source and target segment of the current translation unit. The sample program below illustrates the use of these properties.

```
'Declare a variable named TU which refers to an object
Dim TU As Object
'Assign the object TranslationUnit to the variable
'This will shorten the statements needed to access a TU
Set TU = Workbench.TranslationMemory.TranslationUnit

'Display the current match value in a message box
MsgBox "The current match value is: " & TU.Score, vbInformation

'Display the number of found terms in message box
MsgBox "The number of found terms is: " & TU.TermCount, vbInformation

'Display the source and target sentences in message boxes
MsgBox "The current source sentence is: " & TU.Source, vbInformation
MsgBox "The current target sentence is: " & TU.Target, vbInformation
```

The "Term" Class

The **Term** class provides access to the results of terminology recognition. If terminology recognition is not active its methods will have no effect and the properties will remain empty. **My Computer\HKEY_CURRENT_USER\Software\TRADOS\TW4Win\Options\TermRecognition** is the registry key that contains the status of this option. If the key has the value 0, term recognition is inactive; if it is 1, term recognition is active.

Term Methods

The two methods in the **Term** class are **Next** and **Previous**. They are used for browsing from one found term to the next and vice versa. Not surprisingly, **Next** scrolls to the next found term. If the sentence boundary has been reached, **Next** will again scroll to the first term. Likewise, the **Previous** method is used to scroll backwards through the known terms of a sentence.

Term Properties

Workbench uses fuzzy-matching algorithms for terminology recognition. The **Found** property contains the term from the source sentence for which Translator's Workbench has retrieved a match from the terminology database. The source term from the termbase is stored in the **Source** property. Since fuzzy-matching is used the contents of **Found** and **Source** may be different. The translation of the term is stored in the **Target** property. From here it can be copied to the translated sentence, for instance. The sample script below illustrates the use of the **Term** class.

```
'Declare a variable named Terms which refers to an object
Dim Terms As Object
'Assign the object Term to the variable
'This will shorten the statements to acces the Term class
Set Terms = Workbench.TranslationMemory.TranslationUnit.Term

'Assign the number of found terms in the open sentence to the variable
      TermNum
TermNum = Workbench.TranslationMemory.TranslationUnit.TermCount

'Loop through all found terms
For i = 1 To TermNum

    'Display the term information in message boxes
    MsgBox "The term in the source sentence is: " & Terms.Found
    MsgBox "The source term in the termbase is: " & Terms.Source
    MsgBox "The target term in the termbase is: " & Terms.Target

    'Go to the next term
    Terms.Next

Next i
```

API Instructions: Reference List

All classes, methods and properties that are available in the Translator's Workbench (TW4Win) object browser are listed below in alphabetical order.

Concept	Description
AnalyzeFiles	This method is the equivalent to the Analyze command from the Tools menu. It analyzes documents as specified in the [Analyze] section of the job file.
Application	TW4Win.Application starts Translator's Workbench. At startup the settings in the "My Computer\HKEY_CURRENT_USER\Software\TRADOS\TW4Win" branch are read from the registry. This branch contains general Workbench user settings (but not the Translation Memory setup).
ChangeDate	This property returns the date of the last change to the current translation unit as specified in the system field with the same name.
ChangeUser	This property returns the ID of the user who last changed the current translation unit.
CleanupFiles	This method is the equivalent to the Clean Up command from the Tools menu. It cleans up translated files as specified in the [Cleanup] section of the job file.
Close	This method is the equivalent to the Close command from the File menu. It closes the current Translation Memory.
Concordance	This method is the equivalent to the Concordance command from the Tools menu. It starts a Concordance search for the text specified in the parameter. The search is performed using the current settings for Concordance searching as specified in the Concordance tab of the Translation Memory Options dialog.
CreationDate	This property returns the date on which the current translation unit was created.

Concept	Description
CreationUser	This property returns the ID of the user who created the current translation unit.
Filename	This property returns the filename of the current Translation Memory.
Found	This property returns the current source term as found in the terminology database. This property only contains one term at a time. If multiple terms were found in a segment the Next or Previous methods can be used to access them.
HitCount	This property returns the number of matching translation units found in the last search.
Next (Translation Unit class)	This method scrolls to the next matching translation unit. When the last translation unit is reached and the method is called again it still returns the last matching translation unit.
Next (Term class)	This method is the equivalent to the Get Next Term command from the Trados menu. It scrolls in a ring to the next found term. That means if the last term is reached and the command is used again this method will return to the first term.
Open	This method is the equivalent to the Open command from the File menu. It opens the Translation Memory as specified in the Filename parameter.
PinOnTop	This property is the equivalent to the Pin on Top command from the View menu. If set to "True" Translator's Workbench is the topmost window. If set to "False" it may be hidden behind other windows.
Previous (Translation Unit class)	This method scrolls to the previous matching translation unit. When the first translation unit is reached and the method is called again it still returns the first matching translation unit.
Previous (Term class)	This method is the equivalent to the Get Previous Term command from the Trados menu. It scrolls in a ring to the previous found term. That means if the first term is reached and the command is used again this method will return to the last term.
Quit	This method is the equivalent to Exit command from the File menu.
Save	This method is the equivalent to the Set command from the Trados menu. It writes a new translation to the current translation unit.
Score	This property returns the match value of the current translation unit.
Search	This method is the equivalent to the OpenGet command. It transfers the string given in the parameter to the Translation Memory and searches for matching translation units.
Source (Translation Unit class)	This property returns the source sentence of the current translation unit.
Source (Term class)	This property returns the current term as found in the current segment. That means the term is returned as it occurs in the document to translate. Since terminology recognition uses fuzzy searching the source term may be different from the term found in the terminology database (returned by the Found property).
SourceLanguage	This property returns the ID number of the source language locale. It can be used to determine the source language of the current Translation Memory.
Target (Translation Unit class)	This property returns the target sentence from the current translation unit.
Target (Term class)	This method is the equivalent to the Get Current Term command from the Trados menu. It returns the translation of a found term.
TargetLanguage	This property returns the number of the target language locale. It can be used to determine the target language of the current Translation Memory.
Term	This class provides access to the terminology recognition results. Its methods are **Next** and **Previous**. It has the properties **Found**, **Source** and **Target**.
TermCount	This property returns the number of terms in the current sentence as found by terminology recognition.

Concept	Description
TranslateFiles	This method is the equivalent to the Translate command from the Tools menu. It translates documents as specified in the [Translate] section of the job file.
TranslationMemory	This class provides access to the Translation Memory methods and properties. It contains the methods **AnalyzeFiles**, **CleanupFiles**, **Close**, **Concordance**, **Open**, **Search** and **TranslateFiles**. It has the properties **Filename**, **HitCount**, **SourceLanguage** and **TargetLanguage**. It contains the class **TranslationUnit**.
TranslationUnit	This class provides access to an individual translation unit. It has the methods **Next**, **Previous** and **Save**. Its properties are **ChangeDate**, **ChangeUser**, **CreationDate**, **CreationUser**, **Score**, **Source**, **Target**, **TermCount**, **UsageCounter** and **UsedDate**. It contains the class **Term**.
TW4Win	TW4Win is the class module for the Translator's Workbench classes. Its methods are **Application** and **Quit**. Its property is **PinOnTop**. It contains the class **TranslationMemory**.
UsageCounter	This property returns the number of times a translation unit has been used.
UsedDate	This property returns the date of the last use of the translation unit.

The following section contains detailed descriptions of all Translator's Workbench API classes, methods and properties.

AnalyzeFiles

Description: The AnalyzeFiles method is the equivalent to the Analyze command from the Tools menu. It analyzes documents as specified in the [Analyze] section of the job file.

Syntax: `Object.TranslationMemory.AnalyzeFiles(JobFile)`

Type: Method

Element of TW4Win.Translation Memory

On Error: The method causes an error event if the job file is not found or if an error occurs during the processing of the job file.

Parameter `JobFile`

Type String (obligatory)

Explanation The string must contain the full path and name of the job file.

Example:
```
'Analyzes files according to the commands in jobs.txt
Workbench.TranslationMemory.AnalyzeFiles("d:\temp\jobs.txt")
```

Application

Description: TW4Win.Application starts Translator's Workbench. At startup the settings in the "My Computer\HKEY_CURRENT_USER\Software\TRADOS\ TW4Win" branch are read from the registry. This branch contains general Workbench user settings (but not the Translation Memory setup).

Syntax: `TW4Win.Application`

Type: Method

Element of TW4Win

Result: The function assigns the connection to Translator's Workbench to an object.

On Error: If a method fails it produces an error event. Some of these events are specific to Workbench. In Visual Basic for Applications these events are influencing the Error Object "Err". The properties of Err.Number and the Err.Description are changed. It is recommended that the information in the Err-Object is used for error messages.

Example:
```
Dim Workbench As Object
Set Workbench = CreateObject("TW4Win.Application")
```

ChangeDate

Description: The ChangeDate property returns the date of the last change to the current translation unit as specified in the system field with the same name.

Syntax: `Object.TranslationMemory.TranslationUnit.ChangeDate`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit

On Error: If the system field does not exist, the property returns the empty string.

Return Value `Date`

Type String

Explanation The format of the returned string is the same as specified in the **Short Date Style** box of the **Date** tab in **Regional Settings** (Windows Control Panel).

Example:
```
'Read the date of the last change of the TU into a variable
ChD$ = Workbench.TranslationMemory.TranslationUnit.ChangeDate
```

ChangeUser

Description: This property returns the ID of the user who last changed the current translation unit.

Syntax: `Object.TranslationMemory.TranslationUnit.ChangeUser`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit

On Error: If the system field does not exist, the property returns an empty string.

Return Value `UserID`

Type String

Explanation The property returns the ID of the user who last changed the translation unit.

Example:
```
'Read the ID of the user who last changed the TU
ChU$ = Workbench.TranslationMemory.TranslationUnit.ChangeUser
```

CleanupFiles

Description: This method is the equivalent to the Clean Up command from the Tools menu. It cleans up translated files as specified in the [Cleanup] section of the job file.

Syntax: `Object.TranslationMemory.CleanupFiles(JobFile)`

Type: Method

Element of TW4Win.Translation Memory

On Error: This method causes an error event it the job file is not found or if an error occurs

during the processing of the job file.

Parameter `JobFile`

Type String (obligatory)

Explanation This string must contain the full path and name of the job file.

Example:
```
'clean up translated files as specified in the job file
Workbench.TranslationMemory.CleanupFiles("d:\temp\job.txt")
```

Close

Description: This method is the equivalent to the Close command from the File menu. It closes the current Translation Memory.

Syntax: `Object.TranslationMemory.Close`

Type: Method

Element of TW4Win.Translation Memory

Result: After the execution of this method the last opened Translation Memory is closed. The application Workbench is not terminated. If the TM is closed the property Filename returns an empty string.

On Error: This method causes an error event if the Translation Memory cannot be closed.

Example:
```
'Closes the TM that Workbench refers to
Workbench.TranslationMemory.Close
```

Concordance

Description: This method is the equivalent to the Concordance command from the Tools menu. It starts a Concordance search for the text specified in the parameter. The search is performed with the settings specified in the Concordance tab of the Translation Memory Options dialog.

Syntax: `Object.TranslationMemory.Concordance(Text)`

Type: Method

Element of TW4Win.Translation Memory

Result: If the Concordance search is successful an Concordance window is opened in which the matching translation units are displayed. If no matching translation units are found the method has no result.

On Error: This function causes an exception if no Translation Memory is open.

Parameter `Text`

Type String (obligatory)

Explanation The content of the parameter specifies the string to search for.

Example:
```
'Performs a Concordance search for "terminology recognition"
Object.TranslationMemory.Concordance("terminology recognition")
```

CreationDate

Description: This property returns the date on which the current translation unit was created.

Syntax: `Object.TranslationMemory.TranslationUnit.CreationDate`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit

On Error: If the system field does not exist, the property returns an empty string.

Return Value `Date`

Type String

Explanation The format of the returned string is the same as specified in the **Short Date Style** box of the **Date** tab in **Regional Settings** (Windows Control Panel).

Example:
```
'Read the date when the TU was created into a variable
CrD$ = Workbench.TranslationMemory.TranslationUnit.CreationDate
```

CreationUser

Description: This property returns the ID of the user who created the current translation unit.

Syntax: `Object.TranslationMemory.TranslationUnit.CreationUser`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit

On Error: If the system field does not exist, the property returns an empty string.

Return Value `UserID`

Type String

Explanation This property returns the ID of the user who created the translation unit.

Example:
```
'reads the ID of the user who created the TU into a variable
CrU$ = Workbench.TranslationMemory.TranslationUnit.ChangeUser
```

Filename

Description: This property returns the filename of the current Translation Memory.

Syntax: `Object.TranslationMemory.Filename`

Type: Property

Element of TW4Win.Translation Memory

Result: This property returns the path and the name of the current Translation Memory as a string. The string is the same as the file name used in the Open method. If no Translation Memory is open the property returns an empty string.

Return Value `Name`

Type String

Explanation The file name is only given with the extension `.tmw` if this was also specified in the Open method.

Example:
```
'assign path and filename of current TM to variable TMname$
TMname$ = Workbench.TranslationMemory.Filename
```

Found

Description: This property returns a term in the source language as found in the terminology database. The property only returns one term at a time. If multiple terms were found in a segment the Next or Previous methods can be used to access the other terms.

Syntax: `Object.TranslationMemory.TranslationUnit.Term.Found`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit.Term

On Error: If terminology recognition is inactive or MultiTerm is not running or no matching term is found the property returns an empty string.

Return Value `Term`

Type String

Explanation The return value is the term as found in the terminology database. Since terminology recognition operates with fuzzy-matching algorithms the result may be different from the term in the document.

Example:
```
'assign the found term to a variable
Term$ = Workbench.TranslationMemory.TranslationUnit.Term.Found
```

HitCount

Description: This property returns the number of matching translation units found in the last search.

Syntax: `Object.TranslationMemory.HitCount`

Type: Property

Element of TW4Win.Translation Memory

Result: The number of matches may not exceed the maximum number of hits defined in Translation Memory Options. If no match was found the property returns 0.

Return Value `Number`

Type Long

Explanation The return value is the number of found translation units after a search.

Example:
```
'assign number of found matches to variable Hits.
Hits = Workbench.TranslationMemory.HitCount
```

Next (Translation Unit Class)

Description: This method scrolls to the next matching translation unit. When the last translation unit is reached and the method is called again it still returns the last matching translation unit.

Syntax: `Object.TranslationMemory.TranslationUnit.Next`

Type: Method

Element of TW4Win.TranslationMemory.TranslationUnit

Return Value `Flag`

Type Boolean

Explanation If the last match has been found and the method is called again it returns False. Otherwise it returns True.

Example:
```
'scroll to the next match
Workbench.TranslationMemory.TranslationUnit.Next
```

Next (Term Class)

Description: This method is the equivalent to the Get Next Term command from the Trados menu. It scrolls in a ring to the next found term. That means if the last term is reached and the command is used again it returns to the first term.

Syntax: `Object.TranslationMemory.TranslationUnit.Term.Next`

Type: Method

Element of TW4Win.TranslationMemory.TranslationUnit.Term

Example:
```
'scroll to the next term
Workbench.TranslationMemory.TranslationUnit.Term.Next
```

Open

Description: This method is the equivalent to the Open command from the File menu. It opens the Translation Memory specified in the Filename parameter.

Syntax: `Object.TranslationMemory.Open Filename, UserID, [Password], [Mode]`

Type: Method

Element of TW4Win.Translation Memory

Result: The Translation Memory is opened in the access mode defined in the Access Rights tab in the Translation Memory setup. If no access rights are defined the Translation Memory is opened in exclusive mode.

On Error: This method causes an error event if the Translation Memory cannot be opened. If an invalid password is given the method causes an error event. If passwords are defined in the Translation Memory setup but no password is given the method causes an error event.

Parameter `Filename`

Type String (obligatory)

Explanation The filename must contain the path and the name of the Translation Memory. The filename is not case sensitive. The extension .tmw may be omitted.

Parameter `UserID`

Type String (obligatory)

Explanation The UserID must contain the ID of the user. It is not case sensitive. If the UserID does not exist it is created.

Parameter `Password`

Type String (optional)

Explanation If passwords have been defined in the Translation Memory setup this parameter must be given. The password is case sensitive.

Parameter `Mode`

Type String (optional)

Explanation The mode may be one of the following values: - r (Read); - w (Read/Write); - m (Maintenance); - x (Exclusive).

Example:
```
'Open the TM demo.tmw with userID FLAGMAN and password AHEAD
Workbench.TranslationMemory.Open "d:\temp\demo.tmw", "FLAGMAN",
"AHEAD"
```

PinOnTop

Description: This property is the equivalent to the Pin on Top command from the View menu. If set to "True" the Translator's Workbench is the topmost window. If set to "False" it may be covered by other windows.

Syntax: `Object.PinOnTop = {True; False}`

Type: Property

Element of TW4Win

Return Value `Flag`

Type Boolean

Explanation If the Workbench window is the topmost window it returns the status "True". If it is not the topmost window it returns the status "False".

Example:
```
'Make Workbench the topmost window
Workbench.PinOnTop = True
```

Previous (Translation Unit Class)

Description: This method scrolls to the previous matching translation unit. When the first translation unit is reached but the method is called again it shows the first matching translation unit.

Syntax: `Object.TranslationMemory.TranslationUnit.Previous`

Type: Method

Element of TW4Win.TranslationMemory.TranslationUnit

Return Value `Flag`

Type Boolean

Explanation If the first match has been found but the method is called it returns False. Otherwise the method returns True.

Example:
```
'scroll to the previous match
Workbench.TranslationMemory.TranslationUnit.Previous
```

Previous (Term Class)

Description: This method is the equivalent to the Get Previous Term command from the Trados menu. It scrolls in a ring to the previous found term. That means if the first term is reached and the command is used again the last term is found.

Syntax: `Object.TranslationMemory.TranslationUnit.Term.Previous`

Type: Method

Element of TW4Win.TranslationMemory.TranslationUnit.Term

Example:
```
'scroll to the previous term
Workbench.TranslationMemory.TranslationUnit.Term.Previous
```

Quit

Description: This method is the equivalent to Exit command from the File menu. It closes a running Translator's Workbench

Syntax: `Object.Quit`

Type: Method

Element of TW4Win

Result: This method closes the instance of Workbench the object refers to and releases the connection between the object and Workbench. The execution of this method is always successful. Upon exiting the Workbench its settings are written into the registry in the key "My Computer\HKEY_CURRENT_USER\Software\TRADOS\TW4Win".

Example:
```
'Exit Workbench
Workbench.Quit
```

Save

Description: This method is the equivalent to the Set command from the Trados menu. It writes a new translation to the current translation unit.

Syntax: `Object.TranslationMemory.TranslationUnit.Save(Translation)`

Type: Method

Element of TW4Win.TranslationMemory.TranslationUnit

Result: If the previous search did not result in a match a new translation unit is created. It contains in the source segment the string from the last search and in the target segment the string given in the parameter. System fields are created and initialized. If the previous search brought up a match the contents of the target segment is overwritten with the string given in the parameter. System fields are updated according to the settings in the Translation Memory Options.

On Error: If no Translation Memory or translation unit is open or the user has no write permission the method causes an error event.

Parameter `Translation`

Type String (obligatory)

Explanation The content of the string is written into the Translation Memory as new translation.

Example:
```
'save new translation in current TU
Workbench.TranslationMemory.TranslationUnit.Save ("Was genau
ist ein Translation Memory (TM)?")
```

Score

Description: This property returns the match value of the current translation unit.

Syntax: `Object.TranslationMemory.TranslationUnit.Score`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit

Return Value `MatchValue`

Type	Long
Explanation	The return value is the match value of the current translation unit. If no matching translation unit was found the property returns 0. In any other case it will not fall below the minimum match value from the Translation Memory settings.
Example:	`'assign current match value to a variable` `mv = Workbench.TranslationMemory.TranslationUnit.Score`

Search

Description:	This method is the equivalent to the OpenGet command. It transfers the string given in the parameter to the Translation Memory and searches for matching translation units.
Syntax:	`Object.TranslationMemory.Search(SourceSenctence)`
Type:	Method
Element of	TW4Win.Translation Memory
Result:	If the search was successful all values accessible through TW4Win.TranslationMemory.TranslationUnit properties will be initialized. If no matching translation unit was found these values remain empty. A new translation can be assigned with the Save method.
On Error:	If no Translation Memory is open the method causes an error event.
Parameter	`SourceSentence`
Type	String (obligatory)
Explanation	The string contains the sentence that is searched in the Translation Memory.

Example:	`'Perform a search for the string between the double quotes`
	`Workbench.TranslationMemory.Search ("What exactly is a`
	`Translation Memory (TM)?")`

Source (Translation Unit Class)

Description:	This property returns the source sentence of the current translation unit.
Syntax:	`Object.TranslationMemory.TranslationUnit.Source`
Type:	Property
Element of	TW4Win.TranslationMemory.TranslationUnit
On Error:	If no match was found the property returns the empty string.
Return Value	`SourceText`
Type	String
Explanation	The return value is the content of the source segment of the current translation unit.
Example:	`'assign the source sentence to a variable` `Src$ = Workbench.TranslationMemory.TranslationUnit.Source`

Source (Term Class)

Description: This property returns the found term from the current segment. That means the term is displayed as it occurred in the document to translate. Since terminology recognition uses fuzzy searching the source term may be different from the found term.

Syntax: `Object.TranslationMemory.TranslationUnit.Term.Source`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit.Term

On Error: If no term was found the property returns the empty string.

Return Value `Term`

Type String

Explanation The return value is the source term from the terminology database as a string.

Example:
```
'assign the term from the segment to a variable
STerm$=Workbench.TranslationMemory.TranslationUnit.Term.Source
```

SourceLanguage

Description: This property returns the ID number of the source language locale. It can be used for determining the source language of the current Translation Memory.

Syntax: `Object.TranslationMemory.SourceLanguage`

Type: Property

Element of TW4Win.Translation Memory

Return Value `LocaleID`

Type Long

Explanation The return value is a number of the locale that is used for the source language. This value may be used for comparing the source language of the document with the settings in the Translation Memory.

Example:
```
'write number of source language locale into a variable
SrcLng = Workbench.TranslationMemory.SourceLanguage
```

Target (Translation Unit Class)

Description: This property returns the translation from the current translation unit.

Syntax: `Object.TranslationMemory.TranslationUnit.Target`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit

On Error: If no match was found the function returns the empty string.

Return Value `TargetText`

Type String

Explanation This property returns the content of the target segment of the current translation unit. If no match was found the property returns the empty string.

Example:
```
'assign the translation to a variable
Trg$ = Workbench.TranslationMemory.TranslationUnit.Target
```

Target (Term Class)

Description: This method is the equivalent to the Get Current Term command from the Trados menu. It returns the translation of a found term.

Syntax: `Object.TranslationMemory.TranslationUnit.Term.Target`

Type: Method

Element of TW4Win.TranslationMemory.TranslationUnit.Term

Return Value `Term`

Type String

Explanation The return value is the content of the index field which is set as target language in MultiTerm. If no term was found the function returns the empty string.

Example:
```
'assign the translation of a term to a variable
TTerm$=Workbench.TranslationMemory.TranslationUnit.Term.Target
```

TargetLanguage

Description: This property returns the number of the target language locale. It can be used for determining the target language of the current Translation Memory.

Syntax: `Object.TranslationMemory.TargetLanguage`

Type: Property

Element of TW4Win.Translation Memory

Return Value `LocaleID`

Type Long

Explanation This property returns the number of the locale that is used for the target language. This value may be used for comparing the target language of the document with the settings in the Translation Memory.

Example:
```
'write the number of the target language locale into a variable
TrgLng = Workbench.TranslationMemory.TargetLanguage
```

Term

Description: The class provides access to the terminology recognition. Its methods are **Next** and **Previous**. It has the properties **Found**, **Source** and **Target**.

Syntax: `TW4Win.TranslationMemory.TranslationUnit.Term`

Type: Class

Element of TW4Win.TranslationMemory.TranslationUnit

TermCount

Description: This property returns the number of terms in the current sentence that were found by terminology recognition.

Syntax: `Object.TranslationMemory.TranslationUnit.TermCount`

Type:	Property
Element of	TW4Win.TranslationMemory.TranslationUnit
Return Value	`NumberOfMatches`
Type	Long
Explanation	The property returns the number of found terms in a segment. If no terms were found or terminology recognition is deactivated the property returns 0.
Example:	`'assign the number of found terms to a variable` `Terms = Workbench.TranslationMemory.TranslationUnit.TermCount`

TranslateFiles

Description:	This method is the equivalent to the Translate command from the Tools menu. It translates documents as specified in the [Translate] section of the job file.
Syntax:	`Object.TranslationMemory.TranslateFiles(JobFile)`
Type:	Method
Element of	TW4Win.Translation Memory
On Error:	This method causes an error event if the job file is not found or if an error occurs during the processing of the job file.
Parameter	`JobFile`
Type	String (obligatory)
Explanation	The string must contain the full path and name of the job file.
Example:	`'Translates files according to the commands in jobs.txt` `Workbench.TranslationMemory.TranslateFiles("d:\temp\jobs.txt")`

TranslationMemory

Description:	The class provides access to the Translation Memory methods and properties. It contains the methods **AnalyzeFiles**, **CleanupFiles**, **Close**, **Concordance**, **Open**, **Search** and **TranslateFiles**. It has the properties **Filename**, **HitCount**, **SourceLanguage** and **TargetLanguage**. It contains the class **TranslationUnit**.
Syntax:	`TW4Win.TranslationMemory`
Type:	Class
Element of	TW4Win

TranslationUnit

Description:	The class provides access to an individual translation unit. It has the methods **Next**, **Previous** and **Save**. Its properties are **ChangeDate**, **ChangeUser**, **CreationDate**, **CreationUser**, **Score**, **Source**, **Target**, **TermCount**, **UsageCounter** and **UsedDate**. It contains the class **Term**.
Syntax:	`TW4Win.TranslationMemory.TranslationUnit`
Type:	Class
Element of	TW4Win.Translation Memory

TW4Win

Description: TW4Win is the class module for the Translator's Workbench classes. Its methods are **Application** and **Quit**. Its property is **PinOnTop**. It contains the class **TranslationMemory**.

Syntax: `TW4Win`

Type: Class

UsageCounter

Description: This property returns the number of times a translation unit has been used.

Syntax: `Object.TranslationMemory.TranslationUnit.UsageCounter`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit

On Error: If the usage counter is not defined, the translation unit does not exist or has not been re-used, the property returns 0.

Return Value `Counter`

Type Long

Explanation The return value is number of times a translation unit has been re-used.

Example:
```
'assign the content of the usage counter to a variable
UsC = Workbench.TranslationMemory.TranslationUnit.UsageCounter
```

UsedDate

Description: This property returns the date of the last use of the translation unit.

Syntax: `Object.TranslationMemory.TranslationUnit.UsedDate`

Type: Property

Element of TW4Win.TranslationMemory.TranslationUnit

On Error: If the system field does not exist, the property returns the empty string.

Return Value `Date`

Type String

Explanation The property returns the date when the TU was last used as a string in the format dd.mm.yy, hh:mm

Example:
```
'read the date of the last use of the TU into a variable
UsD$ = Workbench.TranslationMemory.TranslationUnit.ChangeDate
```

The Job File

The Workbench offers three so-called “batch functions”:

1. Analyze
2. Translate
3. Cleanup

A detailed description of these functions can be found in the Workbench User’s Guide. At the time of printing the corresponding chapter is Chapter 9 “Document Analysis, Translation and Cleanup” One of the main advantages of the API is the ability to start these three commands from another program. The corresponding methods are

1. AnalyzeFiles
2. TranslateFiles
3. CleanupFiles

which we will call “batch methods”. As explained in the User’s Guide there are parameters that govern the execution of these commands and thereby influence the result of the process. Consequently it must be possible to control the execution of the batch methods. This is achieved by the so-called “job file” which is a control file that determines what documents should be processed and what parameters are used. In this section it will be explained how the job file must be written and what commands are available to select files and set parameters.

General Remarks

The job file must be a text only file. Every command must be given in a separate line. Leading white spaces at the beginning of a line are ignored. Comment lines must begin with an apostrophe (').

Jobs

The job file can consist of one or more jobs. A job is a set of instructions that either analyze or translate or cleanup one or more files. Each job needs a separate section. If one of the batch methods is used it searches for the corresponding section in the job file that is given as parameter of the method. It is possible to create a job file that contains the settings for all batch methods in one file or to create separate job files for each method. The following figure shows the relation between the batch methods and the corresponding section in the job file.

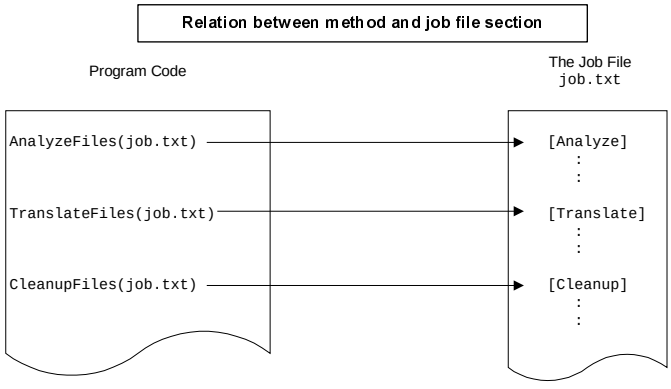

Figure B–7: Relation between methods and job file

At the same time it is possible to use a separate job file for each method or different job files for one method if you want to process multiple files with variable settings.

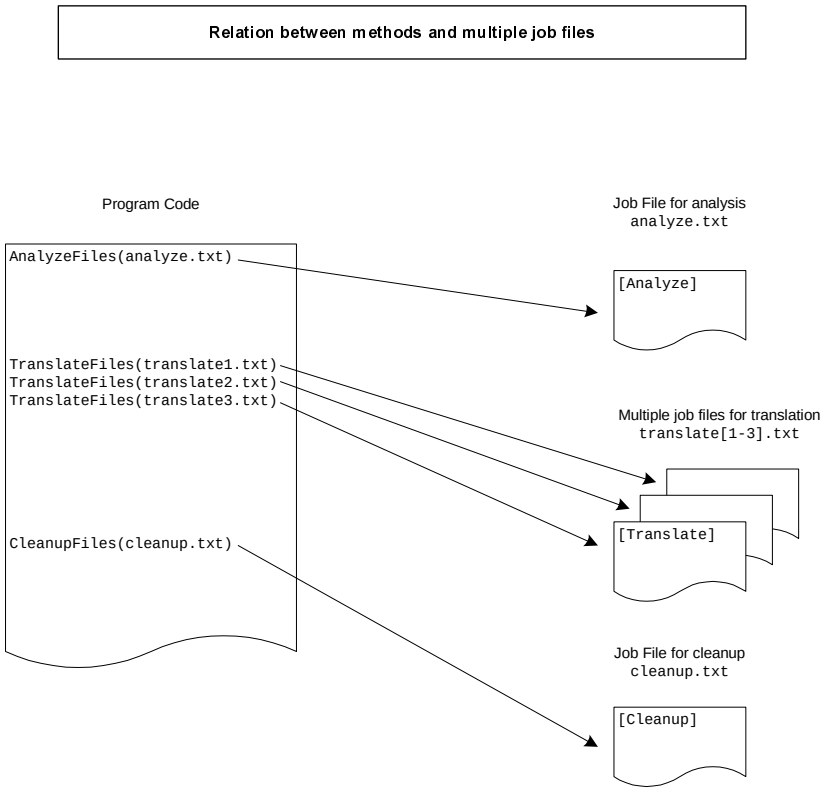

Figure B–8: Relation between methods and multiple job files

Job: Analyze

The Analyze section in the job file consists of sub-sections which are called “tasks”. The number of tasks must be given with the **Tasks** command. Each task is numbered consecutively and the subsections for the tasks are marked with the section marker **Task n**. In the analyze section the following tasks are available:

- Analyze
- ExportUnknown
- ExportFrequent
- CreateProjectTM

The **Analyze** task is obligatory and it must be the first task. The other tasks are optional and may be carried out in any order. Within the Analyze section the user must specify the number of tasks. Since the Analyze task is required at least one task must be given. If the user wants to record the analysis result in a file this can be achieved with the **LogFile** command. The beginning of a typical analysis section would look like this:

```
’This section defines the settings for analysis
[Analyze]
’The whole analysis process consits of 4 tasks
Tasks=4
’The analysis results are recorded in a log file
LogFile=D:\Trados\TW4Win\samples\test.log
```

```
'This section defines the settings for the first task
[Task1]
'The first task is to analyze a set of files
Task=Analyze
```

Analyze Task

As pointed out in the preceding section the **Analyze** task is obligatory and must be the first task. If desired the user can advise the Workbench to use the Translation Memory from the previous analysis if multiple files are treated. This can be achieved with the **UseTMFromPreviousAnalysis** command. Moreover it must be declared how many files are to be analyzed and what the paths and names of the files are. The appropriate commands are **Files** and **File n**. **Files** sets the total number of files to process. The files must be specified with the **File n** command where every file has its own number. Here the full path and file name must be set. It is possible to access files remotely over a network. In this case the path can contain a mounted network drive or a universal naming convention name (UNC path). A typical **Analyze** task would look like this:

```
'This section defines the settings for the first task
[Task1]
'The first task is to analyze a set of files
Task=Analyze
'Do not use the TM from the previous analysis
UseTMFromPreviousAnalysis=0
'Analyze two files
Files=2
'Here are the file names
File1=C:\Trados\TW4Win\samples\demo97.rtf
'The second file is on a machine named SPOCK
File2=\\SPOCK\Trados\TW4Win\samples\test97.rtf
```

ExportUnknown Task

The Translator's Workbench offers the possibility to export all sentences that are unknown to the Translation Memory into a file. This feature can be accessed through the **ExportUnknown** Task. It must be coded as a sub-section of the Analyze section. After the task command has been executed the user can define the match value which determines whether a segment is exported or not with the **MaxMatch** command. The system compares every sentence from the document to the Translation Memory. All sentences that have a match value identical or below the given match value are written to the export file. The path and the name of this file are given with the **File** command. If the format of the export file should be another than Workbench 2 exchange format the user must specify this with the **FileType** command. This is necessary if the unknown text should be translated with a machine translation system like Logos or Systran, e.g. A sample for a task to export unknown segments is given below.

```
'This section defines the settings for the second task
[Task2]
'The task is to export all unknown sentences
Task=ExportUnknown
'Consider all sentences as unknown w/ a match value of 95% or lower
MaxMatch=95
'File to which all unknown sentences are exported
File=C:\Trados\TW4Win\samples\unknown.txt
'Create a file in Workbench 2 exchange format
FileType=1
```

ExportFrequent Task

Like exporting all unknown sentences it is also possible to write all frequent segments into a file. The appropriate task is **ExportFrequent**. Per default all sentences that occur five or more times are exported. If the user wants to set a different value he must use the **Occurrences** command. Similar to the **ExportUnknown** command the name of the output file must be specified. If the user requires another output format than Workbench 2 format he must use the **FileType** command. A sample for a task to export frequent sentences is given below.

```
'This section defines the settings for the third task
[Task3]
'The task is to export the frequent segments
Task=ExportFrequent
'Export all sentences that occur twice or more
Occurrences=2
'File to which the frequent segments are exported
File=C:\Trados\TW4Win\samples\frequent.txt
'Create a file in Workbench 2 exchange format
FileType=1
```

CreateProjectTM Task

For distributing only the relevant part of a Translation Memory it is possible to create a Project TM. This is achieved with the **CreateProjectTM** command. The only instruction that is needed in addition is the specification of the path and the name of the Project Translation Memory. The output is a TM in Workbench *2* format with all index files. A sample task is given below.

```
'This section defines the settings for the fourth task
[Task4]
'The task is to create a project Translation Memory
Task=CreateProjectTM
'File name of the project TM
File=C:\Trados\TW4Win\samples\project.tmw
```

Job: Translate

The Translator's Workbench allows the automatic pre-translation of documents. Known sentences are retrieved from the Translation Memory and pasted into the document. This is particularly useful when a large portion of the text is already known to the Translation Memory. This function can be called automatically with the **TranslateFiles** method which executes the commands in the **[Translate]** section of the job file. It contains commands which are equivalent to the settings that are available in the **Translate Files** dialog.

If the user wishes to record the results of the translation process in a log file the path and name of this file must be given. Per default the Workbench only translates exact matches. If sentences with a lower match value shall be translated the minimum match value must be set with the **MinMatch** command. Next it should specified whether the system should also search for terminology with the **TranslateTerms** command. Similar to the **[Analyze]** section the user must define the number of files to process and the path and names of the files. Below a typical **[Translate]** section is given as an example:

```
'This section defines the setting for automatic pre-translation
[Translate]
'Record the results in a log file
LogFile = C:\Trados\TW4Win\xlate.log
'Translate all sentences with a match value of 95% or better
MinMatch=95
'Do not translate terms automatically
TranslateTerms=0
'Process two files
Files=2
'Here are the file names
File1=C:\Trados\TW4Win\demo97.rtf
'The second file is on a machine named SPOCK
File2=\\SPOCK\Trados\TW4Win\samples\test97.rtf
```

Job: Cleanup

The third batch function is the clean up. This process is started with the **CleanupFiles** method. The commands in the **[Cleanup]** section of the job file are corresponding to the settings that are available in the **Clean up Files** dialog.

Like in all the other sections the user may specify a log file. Per default neither the Translation Memory nor the document are updated if diverging translations are detected during the clean up process. This can be changed with the **WhenChanged** command. Finally only the list of files which should be processed must be specified. A typical **[Cleanup]** section may look like this:

```
'This section contains the settings for the clean-up
[Cleanup]
'Record the results in a file
LogFile = C:\Trados\TW4Win\cleanup.log
'Update the TM if translations have changed
WhenChanged=2
'Process two files
Files=2
'Here are the file names
File1=C:\Trados\TW4Win\demo97.rtf
'The next file is on a machine named SPOCK
File2=\\SPOCK\Trados\TW4Win\sample\test97.rtf
```

Job File Commands: Reference Lists

All command that are available for programming the Job File are listed here in alphabetic order.

Concept	Description
Analyze	In this section the user defines which files are to be analyzed and what settings control the analysis process. An analysis consits of one or more tasks. The number of tasks must be given with the command Tasks. For each task a separate section must be defined. The first task must be an Analyze task. Other tasks may be the export of unknown sentences or the creation of a project Translation Memory.
Cleanup	This section is the equivalent to the Clean-Up command from the Tools menu. It contains the settings for the clean up process and the list of files to clean.
File	The command specifies the path and the name of an output file. Per default the format of the output file is exchange format of Translator's Workbench 2. The output format can be changed with the FileType command.
File n	The command defines the position of the file in the queue. The files must be numbered consecutively and are processed in that order.
Files	The command specifies the number of files to process in a task or a section.
FileType	The command can be used for setting the format of an output file. If the command is not used all output files are written in the exchange format of Translator's Workbench 2.
LogFile	The results of analyzing, translating or cleaning up may be recorded in a log file. If this is desired the name of the file must be specified by this command.
MaxMatch	The command is the equivalent to the "% or lower Match Value" from the Analysis Results group box of the Analyze Files dialog. It defines the threshold at which an unknown sentence must be exported. If the match value of a sentence is below the threshold it will be written to the export file. If the command is not given in the job file then the default value 85 will be used.
MinMatch	The command is the equivalent to the Match Value setting in the Translate File dialog. The value defines how much the minimal match value must be so that a sentence is translated automatically. If the command is not used only exact matches are translated.
Occurrences	The command is the equivalent to the Occurrence setting in the Analysis Results group box of the Analyze Files dialog. It specifies the threshold at which frequent segments shall be written to an export file. If the command is not used all sentences occurring 5 times or more are exported.
SegmentUnknown	The command is the equivalent to the Segment Unknown Sentences check box in the Translate Files dialog. It determines whether untranslated sentences shall be segmented or not. If the command is omitted unknown sentences are not segmented.
Task	The command specifies the action to perform in the analyze section.

Concept	Description
Task n	Every task has to be marked as separate section. This command marks the beginning of new task. The tasks must be numbered consecutively in ascending order and appear in that order in the job file. A task itself consists of several commands which describe each step of the task.
Tasks	The command sets the number of tasks to process when analyzing documents against a Translation Memory.
Translate	This section is the equivalent to the Translate command from the Tools menu. It contains the settings for the batch translation process and the list of files to translate.
TranslateTerms	The command is the equivalent to the Translate Terms group box in the Translate Files dialog. It specifies whether terminology recognition shall be used at automatic translation and where to put found terms in the translated file. If the command is omitted the terminology recognition is inactive. If no MultiTerm is running then the terminology recognition is omitted regardless of the settings in the command.
UseTMFromPreviousAnalysis	This command defines whether the Workbench shall use the Translation Memory from the previous analysis or not when analyzing multiple files. If the command is not used then the TM from the previous analysis is not used.
WhenChanged	The command is the equivalent to the Update Changed Translations group box in the Translate Files dialog. It defines whether the Workbench should check if a translation has been changed since the last automatic translation and if so whether the document or the Translation Memory shall be updated. If the command is not used the check for changed translations is omitted.
WhenChanged	The command is the equivalent to the Changed Translations group box in the Cleanup Files dialog. It defines whether the Workbench should check if a translations in the document have been changed since the last translation and if so what action shall be taken. If the command is not used the check for changed translations is omitted.

The following section contains a complete description of all Job File commands:

Analyze

Description: In this section the user defines which files are to be analyzed and what settings control the analysis process. An analysis consist of one or more tasks. The number of tasks must be given with the command Tasks. For each task a separate section must be defined. The first task must be an Analyze task. Other tasks may be the export of unknown sentences or the creation of a project Translation Memory.

Syntax: `[Analyze]`

Type: Section (obligatory)

Element of Job File

Example:
```
[Analyze]
Tasks=t

[Task1]
Task = Analyze
   :
   :
```

Cleanup

Description: This section is the equivalent to the Clean-Up command from the Tools menu. It contains the settings for the clean up process and the list of files to clean.

Syntax: `[Cleanup]`

Type: Section

Element of Job File

File

Description: The command specifies the path and the name of an output file. Per default the format of the output file is exchange format of Translator's Workbench 2. The output format can be changed with the FileType command.

Syntax: `File=Filename`

Type: Command (obligatory)

Element of Job File [Analyze] Task=ExportUnknown, Job File [Analyze] Task=ExportFrequent, Job File [Analyze] Task=CreateProjectTM

On Error: If the access to the directory or the file is denied an error message is written to the log file. The processing of the job file continues.

Parameter `Filename`

Type String (obligatory)

Explanation The filename must contain the full path and the full name including the extension of the output file. If the file already exists it is overwritten without warning. UNC paths are supported.

Example:
```
'write the results of the current task to the file specified
File=d:\temp\unknown.txt
```

File n

Description: The command defines the position of the file in the queue. The files must be numbered consecutively and are processed in that order.

Syntax: `Filen=Filename`

Type: Command (obligatory)

Element of Job File [Analyze] Task=Analyze, Job File [Translate], Job File [Cleanup]

On Error: If the file does not exist or the access to it is denied an error message is written to the log file. The processing of the job file continues.

Parameter `Filename`

Type String (obligatory)

Explanation The filename must contain the full path and name of the file. UNC paths are supported.

Example:
```
'first file to process is demo.doc
File1=D:\temp\demo.doc
```

Files

Description: The command specifies the number of files to process in a task or a section.

Syntax: `Files=n`

Type: Command (obligatory)

Element of Job File

On Error: If the number of files is smaller than the list of files only the given number of files is processed. If the number of files is 0 or larger than the number of files in the list an exception error occurs.

Parameter `Number`

Type Long (obligatory)

Explanation The parameter specifies the number of files to process in a task or a section.

Example:
```
'process two files
Files=2
```

FileType

Description: The function can be used for setting the format of an output file. If the function is not used all output files are written in the exchange format of Translator's Workbench 2.

Syntax: `FileType=[1-4]`

Type: Command (optional)

Element of Job File [Analyze] Task=ExportUnknown, Job File [Analyze] Task=ExportFrequent

On Error: If the parameter is out of range then the value 1 is used.

Parameter `Number`

Type Long (obligatory)

Explanation The file type is defined by numbers from 1 to 4. The following table shows the numbers and the corresponding output formats:
1 = TW for Windows 2.x (*.txt)
2 = TW for Windows 1.x (*.txt)
3 = TW for DOS (*.asc)
4 = Systran (*.rtf)
5 = Logos (*.sgm)

Example:
```
'create an output file in Workbench 2 exchange format
FileType=1
```

LogFile

Description: The results of analyzing, translating or cleaning up may be recorded in a log file. If this is desired the name of the file must be specified by this command.

Syntax: `LogFile=Filename`

Type: Command (optional)

Element of Job File

On Error: If the file cannot be created or is in accessible an exception error occurs.

Parameter `Filename`

Type String (obligatory)

Explanation The filename must contain the full path and the filename of the log file. If the file already exists the results will be appended to it. UNC paths are supported.

Example:
```
'write the analysis results to test.log
LogFile=d:\temp\test.log
```

MaxMatch

Description: The command is the equivalent to the "% or lower Match Value" from the Analysis Results group box of the Analyze Files dialog. It defines the threshold at which an unknown sentence must be exported. If the match value of a sentence is below the threshold it will be written to the export file. If the command is not given in the job file then the default value 85 will be used.

Syntax: `MaxMatch=n`

Type: Command (optional)

Element of Job File [Analyze] Task=ExportUnknown

Parameter `Maximum match value`

Type Long (obligatory)

Explanation The parameter defines the threshold at which a sentence is exported as unknown. If the value is 100 or larger all sentences will be exported as unknown. If the value is 0 then all sentences with a match value between 30 and 0 will be exported. If the value is negative then no sentence will be exported. Decimal values are truncated to their integer.

Example:
```
'export all unknown sentences that have a match value below 95%
MaxMatch=95
```

MinMatch

Description: The function is the equivalent to the match value setting in the Translate File dialog. The value defines how much the minimal match value must be so that a sentence is translated automatically. If the function is not used only exact matches are translated.

Syntax: `MinMatch=n`

Type: Command (optional)

Element of Job File [Translate]

Parameter `Number`

Type Long (obligatory)

Explanation The number defines the match value that a sentence must have in order to be translated automatically. Setting it to e.g. 95 means that all sentence with a match value of 95% or higher are translated automatically. Setting the parameter to higher than 100 or below 0 will translate nothing.

Example: `'automatically translate segments with a match value > 95%`
`MinMatch=95`

Occurrences

Description: The command is the equivalent to the Occurrence setting in the Analysis Results group box of the Analyze Files dialog. It specifies the threshold at which frequent segments shall be written to an export file. If the command is not used all sentences occurring 5 times or more are exported.

Syntax: `Occurrences=n`

Type: Command (optional)

Element of Job File [Analyze] Task=ExportFrequent

Parameter `Number`

Type Long (obligatory)

Explanation The parameter defines how often a frequent segment must have occurred at last if it should be exported. If set to 2 all sentences that occurred twice or more will be written to the output file. If set to 1 all segments will be exported.

Example:
```
'export all sentences that occur twice or more times
Occurences=2
```

SegmentUnknown

Description: The function is the equivalent to the Segment Unknown Sentences check box in the Translate Files dialog. It determines whether untranslated sentences shall be segmented or not. If the function is omitted unknown sentences are not segmented.

Syntax: `SegmentUnknown={0;1}`

Type: Command (optional)

Element of Job File [Translate]

Parameter `Flag`

Type Boolean (obligatory)

Explanation If set to 0 (=false) unknown sentences are not segmented. If set to 1 (=true) unknown sentences are segmented.

Example:
```
'segment unknown sentences during batch translation
SegmentUnknown=1
```

Task

Description: The command specifies the action to perform in the analyze context.

Syntax: `Task={Analyze; ExportUnknown; ExportFrequent; CreateProjectTM}`

Type: Command (obligatory)

Element of Job File [Analyze]

Parameter `Analyze`

Type Command

Explanation This command marks the section where the analysis is performed.

Parameter `ExportUnknown`

Type Command

Explanation This command is the equivalent to the "Export Unknown Segments" button from the Analyze dialog. When this command is used all unknown segments are exported into a file.

Parameter `ExportFrequent`

Type Command

Explanation This command is the equivalent to the "Export Frequent Segments" button in the Analyze dialog. When this command is used all frequent segments are exported to a file.

Example:
```
'this is an analyze task
Task=Analyze
```

Tasks

Description: The command sets the number of tasks to process when analyzing documents against a Translation Memory.

Syntax: `Tasks=t`

Type: Command (obligatory)

Element of Job File [Analyze]

On Error: If the number is larger than the number of task sections then an exception error occurs. If the number is smaller than the number of task sections then only the number of tasks specified is executed.

Parameter `Number`

Type Long (obligatory)

Explanation The parameter sets the number of tasks that make up an analysis.

Example:
```
'the Analyze section contains 3 tasks
Tasks=3
```

Task n

Description: Every task has to be marked as separate section. This command marks the beginning of new task. The tasks must be numbered consecutively in ascending order and appear in that order in the job file. A task itself consists of several commands which describe each step of the task.

Syntax: `[Taskn]`

Type: Section (obligatory)

Element of Job File [Analyze]

On Error: If the tasks are not numbered correctly an exception error occurs.

Example:
```
'the first task is to analyze
[Task1]
Task = Analyze
     :
     :
     :
```

Translate

Description: This section is the equivalent to the Translate command from the Tools menu. It contains the settings for the batch translation process and the list of files to translate.

Syntax: `[Translate]`

Type: Section

Element of Job File

TranslateTerms

Description: The command is the equivalent to the Translate Terms group box in the Translate Files dialog. It specifies whether terminology recognition shall be used at automatic translation and where to put found terms in the translated file. If the command is omitted the terminology recognition is inactive. If no MultiTerm is running then the terminology recognition is omitted regardless of the settings in the command.

Syntax: `TranslateTerms=[0-2]`

Type: Command (optional)

Element of Job File [Translate]

Parameter `Flag`

Type Long (obligatory)

Explanation The table explains the effect of the different values:
0 = Don't translate terms. The terminology recognition is inactive.
1 = Replace terms. If a term is found the it is overwritten with its translation.
2 = Insert terms. If a term is found an annotation is inserted at the beginning of the segment. The annotations contain the term and its translation.

Example:
```
'do not translate terms during batch translation
TranslateTerms=0
```

UseTMFromPreviousAnalysis

Description: This command defines whether the Workbench shall use the Translation Memory from the previous analysis or not when analyzing multiple files. If the function is not used then the TM from the previous analysis is not used.

Syntax: `UseTMFromPreviousAnalysis={0; 1}`

Type: Command (obligatory)

Element of Job File [Analyze] Task=Analyze

Parameter `Flag`

Type Boolean (obligatory)

Explanation If the flag is 0 (= false) then the TM from the previous analysis is not used. If the flag is 1 (= true) then the TM from the previous analysis is used.

Example:
```
'do not use TM from previous analysis
UseTMFromPreviousAnalysis=0
```

WhenChanged (Translate Task)

Description: The command is the equivalent to the Update Changed Translations group box in the Translate Files dialog. It defines whether the Workbench should check if a translation has been changed since the last automatic translation and if so whether the document or the Translation Memory shall be updated. If the command is not used the check for changed translations is omitted.

Syntax: `WhenChanged=[0-2]`

Type: Command (optional)

Element of Job File [Translate]

Parameter `Flag`

Type Long (obligatory)

Explanation The following table shows the effect of the different values:
0 = Don't update. If changes are detected neither the TM nor the document is changed.
1 = Update TM. If changes are detected the translation from the document is transferred to the TM
2 = Update Document. If changes are detected the translation from the TM overwrites the document.

Example:
```
'do not update if translation has changed
WhenChanged=0
```

WhenChanged (Cleanup Task)

Description: The command is the equivalent to the Changed Translations group box in the Cleanup Files dialog. It defines whether the Workbench should check if a translations in the document have been changed since the last translation and if so what action shall be taken. If the command is not used the check for changed translations is omitted.

Syntax: `WhenChanged=[0-3]`

Type: Command (optional)

Element of Job File [Cleanup]

Parameter `Flag`

Type Long (obligatory)

Explanation The following table shows the effect of the different values:
0 = Don't update
1 = Don't clean up
2 = Update Translation Memory
3 = Update Document

Example:
```
'update TM if translation has changed
WhenChanged=2
```