



# Software License Agreement

TE Edit control

Vesion 32

1990-2025

ALL RIGHTS RESERVED BY

SUB SYSTEMS, INC.

**3200 Maysilee Street**

**Austin, TX 78728**

**512-733-2525**

The Software is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The Software is licensed, not sold. This LICENSE AGREEMENT grants you the following rights:

A) This product is licensed per developer basis. Each developer working with this package needs to purchase a separate license.

B) When used this product within a desktop application, you are granted the right to modify and link the editor routine into your application with these conditions: the target application is 'larger' than the editor; the target application is not a standalone word-processor; the target application uses the editor for one operating system platform only; the target application is not a programmer's utility 'like' a text editor; the target application is not a programmer's product to convert to or from the RTF, HTML, PDF and DOCX formats; and the source code (or part) of the editor is not distributed in any form.

C. The DESKTOP LICENSE allows for the desktop application development. Each desktop license allows one developer to use this product on up to two development computers. A developer must purchase additional licenses to use the product on more than two development computers.

D. The SERVER LICENSE allows for the server application development. The server licenses must be purchased separately when using this product in a server application. Additionally, the product is licensed per developer basis. Only an UNLIMITED SERVER LICENSE allows for royalty-free distribution of your server applications using this product.

E. ENTERPRISE LICENSE: The large corporations with revenue more than \$50 million and large government entities must purchase an Enterprise License. An Enterprise license is also applicable if any target customer of your product using the Software have revenue more than \$500 million. Please contact us at [info@subsystems.com](mailto:info@subsystems.com) for a quote for an Enterprise License.

F. Your license rights under this LICENSE AGREEMENT are non-exclusive. All rights not expressly granted herein are reserved by Licensor.

G. You may not sell, transfer or convey the software license to any third party without Licensor's prior express written consent.

H. The license remains valid for 12 months after the issue date. The subsequent year license renewal cost is discounted by 20 percent from the license acquisition cost. The license includes standard technical support, patches and new releases.

I. You may not disable, deactivate or remove any license enforcement mechanism used by the software.

This software is designed keeping the safety and the reliability concerns as the main considerations. Every effort has been made to make the product reliable and error free. However, Sub Systems, Inc. makes no warranties against any damage, direct or indirect, resulting from the use of the software or the manual and can not be held responsible for the same. The product is provided 'as is' without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of suitability for a particular purpose. The buyer assumes the entire risk of any damage caused by this software. In no event shall Sub Systems, Inc. be liable for damage of any kind, loss of data, loss of profits, interruption of business or other financial losses arising directly or indirectly from the use of this product. Any liability of Sub Systems will be exclusively limited to refund of purchase price.

Sub Systems, Inc. offers a 30-day money back guarantee with the product. The money back guarantee is not available when the product is purchased with dll source.



## General Overview

The TER editor routine allows a developer to incorporate text editing features into an MS Windows application. This product is designed to be simple to use. The TER editor offers the following features:

**Character Formatting and Text Color:** The editor allows for multiple fonts and point sizes within a document. The character styles include **bold**, **underline**, *italic*, superscript, subscript, and strikeout. The text can be painted in multiple colors.

**Paragraph Formatting Features:** Left indentation, right indentation, hanging indentation, centering, justification and double spacing.

**Tab Support:** Left, right, center, and decimal tab positions.

**Imbedded Picture:** The picture can be imported from a disk file or from the clipboard. The editor supports the bitmap, device independent bitmap and metafile picture formats.

**Word Wrapping:** The word wrapping can be enabled or disabled.

**Support for Multiple Section, Multiple Column documents.** The editor also offers the Page Mode feature to edit side-by-side multiple column text.

**Block Highlighting:** The text can be highlighted using character highlighting or line highlighting functions.

**Cut/Paste:** The cut/paste to clipboard is performed using these formats:

Text Format

Rich Text Format

**Printing and Mail/Merge:** The editor can print the selected text or the entire document to the selected printer. An API function allows your application to print a text buffer without invoking the text window. This process can also replace the field names with data strings.

**Application Programming Interface:** The APIs allow you to insert or retrieve text and format attributes anywhere in the text window. You can also interface to the editor window using message communication.

In most instances, only a single API function is needed to invoke the editor.

**Input and Output Source:** The editor can accept data in a buffer or from a disk file. Likewise, the output can be obtained in a buffer or disk file.

**Input and Output text format:** The editor supports the following file formats:

Text format

Rich Text Format

DOCX format

Native TER format

HTML Format (requires HTML-Add)

DOC format import (requires DOC Add-on)

**File and buffer size:** The editor supports unlimited size text files using Windows' virtual memory capability.

**OLE Support:** The editor allows you to embed and link the OLE objects. The editor also supports embedding the files using the Drag/Drop method directly from the Windows' File Manager.

**Text and Picture Frames:** The editor in the Page Mode allows for frames which can be moved and positioned anywhere on a page.

**Source Code:** The product is distributed as a DLL with the complete source code. The source code is simple to follow should you ever need to modify it.

**Other Features:**

Search/replace

Page break and automatic repagination

Page header/footer

Column break

Table support

Optional tool bar

Optional ruler

Optional menu interface

Optional scroll bars

Optional status ribbon

Control over editor window style

Protected and hidden text

Print preview with zoom

Hyperlink link support

Page numbering

Paragraph Spacing

Picture Alignment  
Embed other controls



## Getting Started

This chapter describes the contents of the software diskettes and provides a step by step process of incorporating the TER routine into your application.

### In This Chapter

[Upgrade Note](#)

[Files](#)

[Creating an Editor Window](#)

[License Key](#)



## Upgrade Note

The latest TE Edit Control v25 can co-exists with older versions of TE Edit Control in the same machine. This is made possible by using new files names, new control and class names, new guides, etc. The following table describes the changes and the applications affected.

Item	Old name	New Name	Application Affected
Dll file name	ter15.dll, ter16.dll, ter32.dll ter17.dll, ter18.dll, ter19.dll  ter20.dll  ter21.dll  ter22.dll  ter23.dll  ter24.dll  ter25.dll  ter26.dll		All. Use the new dll file name to incorporate into your application installation scripts.
Dll file name	None	txml2.dll	Used for the import of docx formatted files.  <i>Microsoft MSXML v2.0 or higher package must be installed for the docx file import feature to be</i>



*available.*

Dll file name	None	ssgp.dll	GDI Plus interface dll. The TFLAG8_ENABLE_GDIPLUS flag must also be turned on to make use of gdi-plus drawing.  Also, Microsoft's GDIPLUS package must be installed to use this feature.
---------------	------	----------	--

ActiveX file name	toc15.ocx, toc16.ocx, toc17.ocx toc18.ocx, toc19.ocx  toc20.ocx toc21.ocx toc22.ocx toc23.ocx toc24.ocx toc25.ocx  toc26.ocx	toc30.ocx	Visual Basic, Visual Fox Pro, PowerBuilder, or any application using TE as an ActiveX control. Please ensures that your application registers the new ocx.
-------------------	---	-----------	--

*When upgrading a form containing TOC object, please follow these steps:*

- 1. Delete the existing TOC Control on the form.*
- 2. Remove the old TOC componentl from your application space, and include the new TOC component.*
- 3. Drop the new TOC control into your form to replace the old control. Use the same control name, so the control reference in your application remain valid.*

Class Name	Ter13Class, Ter14Class Ter15Class  Ter16Class Ter17Class Ter18Class Ter19Class Ter20Class Ter21Class Ter22Class Ter23Class Ter24Class	Ter32Class	An application creating the TE control window using the CreateWindow Windows SDK call. It is a good practice to use the TER_CLASS constant instead of hard-coding the class name.
------------	---	------------	---

## Ter25Class

Control Name	TOC15.TOC14Ctrl.1 TOC16.TOC15Ctrl.1 TOC17.TOC17.Ctrl.1 TOC18.TOC18.Ctrl.1 TOC19.TOC19.Ctrl.1 or TOC20.TOC20.Ctrl.1 TOC21.TOC21.Ctrl.1 TOC22.TOC22.Ctrl.1 TOC23.TOC23.Ctrl.1 TOC24.TOC24.Ctrl.1 TOC25.TOC25.Ctrl.1 TOC26.TOC26.Ctrl.1	Toc30.Toc30Ctrl.1	An application creating a TE control using OLE control creation method.
--------------	---	-------------------	---

### Other Changes:

Please note that starting from version 19 and later, the value of the MAX\_WIDTH constant has been increased from 1000 to 2000. This may affect any function which returns a string type value using one of the function parameters. One example would be the TerGetLine function.

If you are upgrading from version 18 or an earlier version, please check your code to ensure that you are not using a hard-coded value of 300 or 1000 to declare or initialize a variable passed to such functions. Preferably the constant MAX\_WIDTH should be used to define and initialize such string variables. Example:

```
dim text as string*MAX_WIDTH
```

```
text=space$(MAX_WIDTH) ' allocate space for the variable
```

If your programming language does not support the use of the constants, then please use the hard-coded value of 2000.



## Files

The package contains the DLL files, the source code files, the include files, resource files, and make files that are necessary to incorporate the TER routine into your application. In addition, the package also includes a set of files to construct a demo program. The demo program shows by example the process of linking the editor to your program.

### **DLL Demo Files:**

(Demo files are contained in the demo\_src.zip file)

DEMO.C	Source code for the demo program
DEMO.H	Include file for the demo program
DEMO.RC	Resource source file for the demo program
DEMO.DEF	Definition file for linking the demo program
DEMO.EXE	Executable demo program
DEMO_DLG.H	Dialog Identifiers for the demo program
DEMO_DLG.DLG	Dialog templates for the demo program
DEMO_DLG.RES	Compiled dialogs for the demo program

### Custom Control Demo Files:

CTRL.C	Source code for the custom control demo program
CTRL.H	Include file for the demo program
CTRL.RC	Resource source file for the demo program
CTRL.DEF	Definition file for linking the demo program
CTRL.EXE	Executable demo program
CTRL_DLG.H	Dialog Identifiers for the demo program
CTRL_DLG.DLG	Dialog templates for the demo program
CTRL_DLG.RES	Compiled dialogs for the demo program

TER Source Files (included in the C\_SOURCE.ZIP file):

TER.C	Contain Application Interface routines and the main window process function.
TER1.C	Main source file. Includes painting functions, line edit and scrolling functions and word wrapping functions.
TER2.C	Contains the overflow from the TER1.C module.
TER3.C	Frames and other editing functions.
TER_INIT.C	Initialization and exit functions
TER_IO.C	File I/O and printing functions
TER_RTF.C	Rich text format data input functions.
TER_RTF1.C	Rich text format data output functions.
TER_BLK.C	Block edit and clipboard functions
TER_MISC.C	Search/Replace and other auxiliary functions.
TER_FMT.C	Contains programming specific to the character formatting, font control and line centering features.
TER_FMT1.C	Overflow routines from TER_FMT.C module.
TER_FRM.C	Frame functions.
TER_CTL.C	Dialog editor interface routines.
TER_PAGE.C	Page mode screen handling routines.
TER_BAR.C	Tool bar management functions.
TER_OLE.C	OLE routines

(These source files with .C extension contain the Include statements for TER.H,WINDOWS.H and other runtime function headers)

TER.H	The <i>include</i> file to include into your application module that calls the TER routine. It contains the definition of the argument parameter structure and the prototypes for the API functions.
TER_MSG.H	Contains the text of messages used by the DLL.
TER1.H	The <i>include</i> file used by the TER modules. It contains global variables and some definition statements. All variable declaration statements are prefixed by 'PREFIX' word. The

PREFIX word is defined in all source files except TER.C as 'extern'. This strategy avoids the need for duplicate declarations for the main file and other auxiliary files that refer to the global variables as 'extern'. This file contains the Include statement for TER\_CMD.H and TER\_DLG.H files.

TER_CMD.H	Include file. Contains definition statements for editor command ids.
TER_DLG.H	Include file. Contains definition statements for dialog box control ids. Also serves as the include file for Windows' dialog editor.
TER_PROT.H	Contains the prototypes for the internal TER functions.
TER.RC	Main resource file for the TER routine containing accelerator and menu resource statements. This file contains the Include statements for TER_CMD.H, TER_DLG.H and TER.DLG files.
TER.DLG	The file created by the dialog editor, which contains the source for the dialog boxes. This file is included into TER.RC file.
TER.RES	The compiled resource file required by the Windows' dialog editor. The include file that is required by the dialog editor with TER.RES is TER_DLG.H.
TER.DEF	The definition file to create the TER.DLL file.
txml2.dll	The dll used for importing the docx formatted files.
ter32.dll	The main DLL file
ter32.LIB	Import library for the ter30 DLL
ter30BC.LIB	Import library for 32 bit Borland C/C++ or Builder environment
Help System Files:	
TER.RTF	The help text in RTF format.
TER.HPJ	The help definition file. This file contains the Include statement for TER_CMD.H file.
TER_HLP.ZIP	Contains TER.HLP file which is a user's help file for editor.

#### **Visual Basic Interface Files :**

TER.BAS:	Included in the VBDEMO.ZIP file. It contains the TER API function declarations, input structure type declarations, TER global constant declarations, and auxiliary functions. This file
----------	---

must be included by any VB application interfacing to the editor.

#### **Visual Basic DLL Interface Demo:**

(These files are included in the VBDEMO.ZIP file)

DMO_VB.BAS:	Contains the global variables for the internal use of the demo program.
DMO_VB.FRM:	Main form. It contains the menu and initialization functions.
DMO_VB1.FRM:	Accepts the parameters to open a new TER window.
DMO_VB2.FRM:	Selects a TER window to close.
DMO_VB3.FRM:	Select a TER window to reset.
DMO_VB4.FRM:	Select a TER window to monitor.
DMO_VB5.FRM:	Allows the user to edit the operational variable of a currently open TER window.
DMO_VB6.FRM:	Allows the user to insert a text string with specified formatting attributes.
DMO_VB7.FRM:	Print a text buffer or file.
DMO_VB.MAK:	Project file for the this demo program.

#### **Borland Delphi Interface and Demo files:**

(These files are included in the DELPHI.ZIP file)

TER.PAS	Function and constant declaration file.
DMO_DLP.DPR	Demo project file.
DMO_DLP1.DFM	Main demo form file.
DMO_DLP2.DFM	Secondary demo form file.
DMO_DLP1.PAS	Main demo unit.
DMO_DLP2.PAS	Secondary demo unit.

Visual C++ Interface Files:

TER_MFC.H	Header file for the CTer class.
-----------	---------------------------------

TER_MFC.CPP	Implementation file for the CTer Class
TER_VIEW.H	Header for the CTerView class.
TER_VIEW.CPP	Implementation file for the CTerView class.
TER_VIEW.RCH	Read only symbols to be included in Visual C++ Application Studio.
MFC.ZIP	Contains a demo application using the CTerView class.

#### **Make Files:**

MAKE-MC.BAT	Compiles and links the TER routines using
MAKE-MC	Microsoft 'C'.
MAKE-BC.BAT	Compiles and links the TER routines using
MAKE-BC	Borland 'C'.



## **Creating an Editor Window**

TE Edit Control can be created in a number of different ways.

Please make sure that ter32.dll and txml2.dll files is installed in a directory available at run-time. When using this product as an ActiveX control, you would also need to copy the toc30.ocx file to the system directory and then register the toc30.ocx files using the regsvr32 Windows program (or using another option available in your programming environment).

When the control is used without using toc30.ocx, no MFC components are needed to run the ter32.dll file.

The product [Licenes Key](#) must be provided before or after creating a TE control for unrestricted used.

Once a TE control instance is created, you can use any of the functions or methods listed in the [Application Interface Functions](#) topic.

Here are various methods of creating a TE Edit Control window in different programming environments:

### **Visual Basic**

You would register the toc30.ocx control file and drop the control into your application form. Please refer to the [Visual Basic Interface](#) and

[ActiveX Control](#) topics for detail information on this subject.

## Visual C++ and C

You can use the [ActiveX](#) interface to drop the control into your application window.

You can also create a TE window using the [CreateTerWindow](#) or [TerCreateWindowAlt](#) functions.

The package contains the ter32.LIB (ter30BC.LIB for Borland C/C++ Builder) file which must be linked to your application. Modify the link statement of your application to include the ter32.LIB file into the list of libraries.

Your application modules which use TER API functions, should also include the TER.H and TER\_CMD.H files.

## MFC Interface

Please refer to the [Visual C++ Interface](#) topic for the description of using the MFC wrapper classes.

**Borland Delphi Interface** Please refer to the [Borland Delphi Interface](#) topic for using the editor as an ActiveX control or a direct DLL control.

**PowerBuilder** Please refer to the [PowerBuilder Interface](#) topic for help in creating a TE control instance in your PB application.

**Visual FoxPro Interface** Please refer to the [Visual FoxPro Interface](#) topic for help in creating a TE control instance in your VFP application.

## Using CreateWindow SDK function.

The window class should be specified as "Ter32Class".

The style parameter to the CreateWindow API should be OR'd with one or many of the following TE style constants as needed by your application.

TER_WORD_WRAP	Enable word wrapping
TER_PRINT_VIEW	Wrap the lines on the screen as they would on the selected printer.
TER_PAGE_MODE	Edit one page at a time (useful for multicolumn documents)
TER_FITTED_VIEW	Special case of page mode in which the text wraps to the window width and the soft page breaks are not displayed.



TER_HSCROLL	Enable the horizontal scroll bar.
TER_VSCROLL	Enable the vertical scroll bar.
TER_SHOW_STATUS	Show the status ribbon
TER_SHOW_RULER	Show horizontal ruler with tab stops
TER_SHOW_TOOLBAR	Show the tool bar
TER_BORDER_MARGIN	Create a space margin around the text window
TER_OUTPUT_RTF	Store the text in the RTF format
TER_READ_ONLY	Do not allow edits.
TER_USE_EXT_STYLES	Indicates that additional style constants are passed using the lpParam parameter of the CreateWindow SDK functions.

*Here is the list of additional constants passed using lpParam parameter:*

TEREX_SHOW_VRULER	Show vertical ruler.
-------------------	----------------------

Include the TER.H file to define these constants.

Once the editor window is created, you can insert and retrieve the editor data using the 'SetTerBuffer' and 'GetTerBuffer' functions. You can also use other API functions (see Application Interface Functions) to manipulate the text data in the TER control.



## License Key

*Your license key is e-mailed to you after your order is processed.* You would set the license key using the TerSetLicenseKey function. This should be preferably done before creating any TE control to avoid pop-up nag screens.

```
TerSetLicenseKey("xxxxx-yyyyy-zzzzz")
```

Replace the 'xxxxx-yyyyy-zzzzz' by your license key.

You do not need to call this function when using TE Edit Control in an evaluation mode.



## Application Interface functions

These API functions allow you to open, close and manipulate data in a TER window. Your application must include the TER.H file, which defines these functions.

Note: Your application can also communicate with the TER window by using message communication. Your application can utilize any of the messages processed by the TER window process (see the *TerWndProc* function in the TER.C file).

The following is a description of the TER API functions in an alphabetic order:

### In This Chapter

- [Import and Export](#)
- [Text Insert, Append and Delete](#)
- [Print and Print-preview](#)
- [Character Formatting](#)
- [Paragraph Formatting](#)
- [Section Formatting](#)
- [Document](#)
- [Text Selection](#)
- [Cursor and Text Position](#)
- [Table](#)
- [Hyperlink](#)
- [Mail-merge](#)
- [Picture and OLE objects](#)
- [Page Header/Footer](#)
- [Frame and Drawing Objects](#)
- [Footnote, Endnote, Bookmark, Tag](#)
- [Stylesheet](#)
- [Control Creation](#)
- [Control Flags](#)
- [List Numbering](#)
- [Toolbar](#)
- [Undo](#)
- [Input Field](#)
- [Page](#)
- [Search, Replace, Locate](#)
- [Track Changes](#)
- [Comments](#)
- [Menu Command](#)
- [Screen Drawing](#)
- [PDF Output](#)
- [SpellChecking](#)
- [Html Add-on Interface](#)
- [Hyphenation Support](#)
- [Miscellaneous](#)



## Import and Export

This chapter includes the data import and export functions.

When the editor is used as an ActiveX control, you can also use the 'data' property to insert or retrieve the data from the control.

control.Data = string or

string = control.Data

Example:

control.Data = "This is a test data"

### In This Chapter

[GetTerBuffer](#)  
[HandleToIntArray](#)  
[HandleToStr](#)  
[HandleToStrInChunks](#)  
[ReadTerFile](#)  
[SaveTerFile](#)  
[SetTerBuffer](#)  
[StrToHandle](#)  
[StrToHandleInChunks](#)  
[TerDocName](#)  
[TerFileToMem](#)  
[TerGetReadOnly](#)  
[TerIsModified](#)  
[TerSetModify](#)  
[TerSetOutputFormat](#)  
[TerSetReadOnly](#)  
[TerQueryExit](#)



## GetTerBuffer

### Retrieve Window Text:

```
HANDLE GetTerBuffer(hWnd, size)
```

```
HWND hWnd;           // Handle of the window to be accessed
```

```
long far*size;        // The GetTerBuffer function returns the size of the global  
                      // buffer using this variable
```

**Description:** You can use this function to retrieve the window text and format data in a global buffer. This function can be called any time for a TER window. The format of the data within the buffer is controlled by the 'SaveFormat' field in the arg\_list parameter structure (see Getting Started). Typically, you would call this function before closing a window.

**Return Value:** This function returns the global handle to the buffer containing the text and format data. The size of the global buffer is returned using the second argument. Lock (GlobalLock) the returned handle to access the text and format data. Your application owns this memory handle. When the data is no longer required, use the GlobalFree function to free the global handle.

A NULL value of the handle indicates an error.

**See Also**  
[SetTerBuffer](#)



## HandleToIntArray

**Copy the contents a global memory handle to an integer array.**

```
int HandleToStr(UniText, length, hMem)
```

```
LPWORD UniText           // a VB integer array
```

```
long length;              // length of the string
```

```
HGLOBAL hMem;             // Global memory handle
```

**Description:** This function can be used to copy the contents of a global memory handle to a given visual basic interger array. Example:

```
Dim size As Long
Dim hMem As Long
Dim UniText(MAX_WIDTH) As Integer
Dim count As Long

hMem = TOCl.TerGetTextSelU(size)
count = TOCl.HandleToIntArray(UniText(0), size, hMem)
```

The UniText array now contains unicode character values.

The input global memory handle is freed up after copying its contents to the string.

**Return Value:** This function returns TRUE if successful.



## HandleToStr

### Convert a global memory handle to a Visual Basic string.

BOOL HandleToStr(string, length, hMem)

char huge \*string;                      //pointer to a visual basic string

long length;                            // length of the string

HGLOBAL hMem;                         // Global memory handle

**Description:** This function can be used to copy the contents of a global memory handle to a given visual basic string. The calling routine must expand the string to appropriate length before calling this function. Example:

string=space(length)

HandleToStr(string,length,hMem)

The input global memory handle is freed up after copying its contents to the string.

**Return Value:** This function returns TRUE if successful.

#### See Also

[GetTerBuffer](#)  
[StrToHandle](#)



## HandleToStrInChunks

### Extract a VB string from the global memory handle.

BOOL HandleToStrInChunks(string, length, hMem, ChunkPos, ChunkLen)

char huge \*string;                      //pointer to a visual basic string

long length;                            // length of the global memory handle buffer

HGLOBAL hMem;                         // Global memory handle

long ChunkPos;                         // beginning position of the string to extract. This value must be between 0 to (length-1).

long ChunkLen;                         // Length of the string to extract.

**Description:** This function can be used to copy the contents of a part of the global memory handle to a given visual basic string. The calling routine must expand the string to appropriate length before calling this function. The following example extracts two 100 byte chunks from the beginning of the global memory handle.

ChunkLen = 100

ChunkPos = 0

```

string=space(ChunkLen)
HandleToStrInChunks(string,hMemLen,hMem,ChunkPos,ChunkLen)
ChunkPos = ChunkPos + ChunkLen
HandleToStrInChunks(string,hMemLen,hMem,ChunkPos,ChunkLen)

```

The input global memory handle is freed up when the last chunk is extracted.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[GetTerBuffer](#)  
[StrToHandleInChunks](#)



## ReadTerFile

### Load a file into the editor window

BOOL ReadTerFile(hWnd, FileName)

BOOL ReadTerFileU(hWnd, FileNameW)

HWND hWnd; // The handle of the window to be accessed

LPBYTE FileName; // Name of the disk file to read. This parameter is used by the ReadTerFile method only.

LPWORD FileNameW; // Name of the unicode disk file to read. This parameter is used by the ReadTerFileU method only.

**Description:** This function instructs the editor to load the specified file for editing. Any existing text in the window is discarded.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[SaveTerFile](#)  
[GetTerBuffer](#)  
[SetTerBuffer](#)



## SaveTerFile

### Save the editor text to disk

BOOL SaveTerFile(hWnd, FileName)

BOOL SaveTerFileU(hWnd, FileNameW)

HWND hWnd; // Handle of the window to be accessed

LPBYTE FileName; // Name of the disk file to save text. This parameter is used by the SaveTerFile method only.

LPWORD FileNameW; // Name of the unicode disk file to save text. This parameter is used by the SaveTerFileU method only.

**Description:** This function instructs the editor to save the text to the specified disk file.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[ReadTerFile](#)

[GetTerBuffer](#)

[SetTerBuffer](#)



## SetTerBuffer

### Set Window Text:

BOOL SetTerBuffer(hWnd, hBuffer, size, title, release)

HWND hWnd; // The handle of the window to be accessed

HANDLE hBuffer; // The global handle to the buffer containing the new text and format data.

long size; // The size of the hBuffer buffer

LPBYTE title; // new title for the window. Specify a NULL value if you do not wish to change the window title

BOOL release; // Release the buffer after applying

**Description:** You can use this function to set new data in an existing TER window. *The existing text in the window is discarded.* The data in the buffer can be provided in one of these formats:

Text Format

Rich Text Format

TER Native Format

If the 'release' flag is set, the hBuffer handle becomes the property of the TER window. Your application must not try to lock or free this buffer.

**Return Value:** This function returns a TRUE value if successful. Otherwise it returns a FALSE value.

**See Also:**  
[GetTerBuffer](#)  
[InsertTerText](#)  
[InsertRtfBuf](#)  
[SaveTerFile](#)  
[ReadTerFile](#)



## StrToHandle

**Convert a Visual Basic string to a global memory handle.**

HANDLE StrToHandle(string, length)

HWND hWnd; // Handle of the window to be accessed

BYTE huge \*string; //pointer to a visual basic string

long length; // length of the string

**Description:** This function copies the content of a visual basic string to a global memory block.

This function is primarily used to copy the visual basic text data to a global memory block to insert in a TER window (see SetTerBuffer).

**Return Value:** This function returns the global memory handle if successful. Otherwise it returns NULL.

**See Also:**  
[HandleToStr](#)  
[SetTerBuffer](#)



## StrToHandleInChunks

**Build a global memory buffer from the VB string chunks.**

HANDLE StrToHandleInChunks(string, length,hMem,ChunkPos,ChunkLen)

BYTE huge \*string; //pointer to a visual basic string



long length;	// Final length of the global memory handle
HANDLE hMem;	// The global memory handle being built. Set this value to 0 on the first call to this function. On the subsequent calls, set this value to the return value from the previous call to this function.
long ChunkPos;	// Position of the current chunk in the memory buffer. This value must be between 0 and (length-1).
long ChunkLen;	// Length of the chunk being added.

### Example:

The following code adds two 100 byte strings to build a memory buffer.

```

BufferLen=200
ChunkLen = 100
ChunkPos =0
hMem=0
hMem=StrToHandleInChunks(string,200,hMem,ChunkPos,ChunkLen)
ChunkPos = ChunkPos + ChunkLen
hMem=StrToHandleInChunks(string,200,hMem,ChunkPos,ChunkLen)

```

**Return Value:** This function returns the global memory handle if successful. Otherwise it returns NULL.

### See Also:

[HandleToStrInChunks](#)  
[SetTerBuffer](#)



## TerDocName

**Set or retrieve the current document name.**

BOOL TerDocName(HWND, get, name)

HWND hWnd;	// The handle of the window to be accessed
BOOL get;	// Set to TRUE to retrieve the current document name, or set to FALSE to set the new document name.
LPBYTE name;	// The pointer to receive the current document name, or

a pointer to the new document name.

**Return Value:** This function returns TRUE if successful.

**VBScript and JavaScript Compatible methods:** The TerDocName method would not work with VBScript and JavaScript since this method uses a reference parameter for document name. Here are two methods can be used with script languages instead:

String TerGetDocName() // retrieve current document name

BOOL TerSetDocName(String NewDocName) // set new document name



## TerFileToMem

**A utility function to read a file into a global memory block.**

HGLOBAL TerFileToMem(FileName, size)

DWORD far \*size; // Location to receive the size of the memory block

**Description:** This function simply reads a file into a global memory buffer. Please note that this function does not load the file into the editor's buffer. To load a file into the editor, please use the function: ReadTerFile.

**Return Value:** This function returns the global memory handle containing the file. A NULL value indicates an error condition.



## TerGetReadOnly

**Retrieve the current Read Only status.**

BOOL TerGetReadOnly(hWnd)

HWND hWnd; // The handle of the window to be accessed

**Return Value:** The function returns TRUE if read-only mode is turned on.

**See Also:**

[TerSetReadOnly](#)



## TerIsModified

**Check if the editor data needs saving.**

BOOL TerIsModified(hWnd)

HWND hWnd;                               // Handle of the window to be accessed

**Return Value:** This function returns TRUE if the text is modified and is not yet saved.

**See Also:**  
[TerSetModify](#)



## TerSetModify

**Set or reset the modification flag.**

BOOL TerSetModify(hWnd,modify)

HWND hWnd;                               // The handle of the window to be accessed

BOOL modify;                             // TRUE to set the modification flag, FALSE to reset it

**Description:** This flag is used to set or reset the modification flag. The modification flag is used to prompt the user to save the data before exiting the editor.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerIsModified](#)



## TerSetOutputFormat

**Set the format of the output data.**

BOOL TerSetOutputFormat(hWnd,format)

HWND hWnd;                               // The handle of the window to be accessed

int format;                               // The output format can be set to one of the following:

SAVE_TER:	Save in the TER native format. This format is deprecated. Please use the RTF format instead.
-----------	--

SAVE_TEXT:	Save in the text format
SAVE_TEXT_LINES:	Save in the text format with line breaks.
SAVE_RTF:	Save in the Rich Text Format
SAVE_DOCX	Save in the DOCX format
SAVE_HTML:	Save in the HTML format. This option is available only when HTML Add-on is installed.
SAVE_UTEXT:	Save in the Unicode text format. (not available in the 16 bit product)
SAVE_DEFAULT:	Save in the format of the original document

**Description:** This function is used to change the data format of the document when saved. The original output format is specified by the calling application when the editor window is created.

**Return Value:** This function returns TRUE if successful.



## TerSetReadOnly

**Set Read Only status.**

BOOL TerSetReadOnly(hWnd, ReadOnly)

HWND hWnd;                      // Window handle to access

BOOL ReadOnly;                      // (TRUE/FALSE) New status of the Read Only flag

**Description:** This function is used to set or reset the Read Only status.

**Return Value:** The function returns the previous value of the Read Only status.

**See Also:**  
[TerGetReadOnly](#)



## TerQueryExit

**Check if it is OK to close a TE window.**

BOOL TerQueryExit(hWnd)

HWND hWnd;                                      // The handle of the text window to be accessed

**Return Value:** This function returns TRUE if it is OK to close a TE window. Otherwise it returns a FALSE value.



## Text Insert, Append and Delete

### In This Chapter

[GetTerLine](#)  
[InsertRtfBuf](#)  
[InsertTerText](#)  
[SetTerLine](#)  
[TerAddAutoCompWord](#)  
[TerAppendText](#)  
[TerAppendTextEx](#)  
[TerDeleteBlock](#)  
[TerGetLine](#)  
[TerGetLineInfo](#)  
[TerGetLineParam](#)  
[TerGetLineWidth](#)  
[TerGetRtfSel](#)  
[TerInsertLine](#)  
[TerInsertRtfFile](#)  
[TerInsertText](#)  
[TerSetLine](#)



## GetTerLine

**Retrieve a text line.**

int GetTerLine(hWnd, LineNo, text, font)

HWND hWnd;                                      // Handle of the window to be accessed

long LineNo;                                    // line number (0 to TotalLines - 1) to retrieve.

LPBYTE text;                                   // pointer of the location to retrieve the text for the line.

LPBYTE font;                                   // pointer of the location to retrieve the font ids for each

character in the line.

**Description:** This function is being phased out. Please use the TerGetLine function instead.



## InsertRtfBuf

**Insert a text buffer in the RTF format at the specified cursor location.**

BOOL InsertRtfBuf(hWnd, buffer, length, line, column, repaint)

BOOL InsertRtfBufU(hWnd, UnicodeBuffer, length, line, column, repaint)

HWND hWnd; // Handle of the window to be accessed

LPBYTE buffer; // pointer to the buffer containing RTF data. This parameter is used by the InsertRtfBuf function only.

LPWORD UnicodeBuffer; // pointer to the unicode buffer containing RTF data. This parameter is used by the InsertRtfBufU function only.

long length; // length of the RTF buffer

long line; // line location (base 0) where the text will be inserted. You can also set the 'line' parameter to -1 to insert the rtf document at the current cursor position. Or, set it to -2 to insert the rtf file at the end of the current document.

int column; // column location (base 0) where the text will be inserted. This parameter is ignored when the 'line' parameter is set to -1 or -2.

BOOL repaint; // TRUE to refresh the window after this operation

**Description:** This function is used to insert a buffer containing the text (in RTF format) into the specified TER window. The text is inserted at the specified line and column position.

To specify the location in terms of the line and column numbers, specify the line number in the 'line' argument and column number in the 'column' argument. To specify the absolute location, set the 'column' argument to -1, and set the 'line' argument to the absolute text location. To insert the text at the current cursor location, set the 'line' and 'col' arguments to -1. To append the text at the end of the document, set the 'line' to -2.

This function is available in the 'word wrap' mode only.

**Return Value:** This function returns TRUE if successful.

**See Also:**

[InsertTerText](#)  
[GetTerBuffer](#)  
[TerInsertRtfFile](#)



## InsertTerText

**Insert a text buffer in the ASCII format at the current cursor location.**

BOOL InsertTerText(hWnd, buffer, repaint)

HWND hWnd; // Handle of the window to be accessed

BYTE huge \*buffer; // pointer to the buffer containing ASCII text data

BOOL repaint; // TRUE to refresh the window after this operation

**Description:** This function is used to insert a buffer containing the text (in ASCII format) into the specified TER window. The text is inserted at the current cursor position.

The text lines within the buffer must be delimited using CR/LF pair.

The buffer must be terminated using a NULL character.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[InsertRtfBuf](#)  
[TerAppendText](#)  
[SetTerBuffer](#)  
[SetTerCursorPos](#)



## SetTerLine

**Set the text and font ids for a line**

BOOL SetTerLine(hWnd, LineNo, text, font)

HWND hWnd; // The handle of the window to be accessed

long LineNo; // The text line number (0 to TotalLines - 1) to set

LPBYTE text; // Contains the pointer to a text string to apply. This string must be NULL terminated.

LPBYTE font; // Contain the pointer to a font id array to apply. You can

set this parameter to NULL if the font ids are not available.

**Description:** This function is being phased out. Please use the TerSetLine function instead.



## TerAddAutoCompWord

**Set an auto-completion word/phrase pair.**

BOOL TerAddAutoCompWord(hWnd, ACWord, ACPhrase)

HWND hWnd;	// The handle of the window to be accessed
LPBYTE ACWord;	// Contains the pointer to an auto-completion key word.
LPBYTE ACPhrase;	// Contain the pointer to the auto-completion phrase for the key word.

**Description:** This function adds the key word and the expansion phrase to the auto-completion list. When the user type a key word, the control automatically replaces it with the corresponding phrase.

**Result:** This function returns True when successful.



## TerAppendText

**Append specified text at the end of the buffer.**

BOOL TerAppendText(hWnd, text, FontId, ParaId, repaint)

HWND hWnd;	// Handle of the window to be accessed
BYTE huge *text;	// pointer to the text to be appended. The text must be NULL terminated.
int FontId;	// font id to use for the new text. Use -1 for the default value.
int ParaId;	// paragraph id to use for the new text. Use -1 for the default value.



BOOL repaint;                               //Repaint the window after this operation

**Description:** This function adds the specified text at the end of the buffer. The current cursor position does not change after the insertion.

This is a very efficient function which can be used to rapidly build a document. This function does not attempt to wrap the text as they are being added. Your application should call the TerRewrap or TerRepaginate functions after making a series of calls to this function.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerAppendTextEx](#)  
[TerInsertText](#)  
[TerCreateFont](#)  
[TerCreateParald](#)



## TerAppendTextEx

**Append specified text at the end of the buffer (enhanced version).**

BOOL TerAppendTextEx(hWnd, text, FontId, Parald, CellId, reserved, repaint)

HWND hWnd;                               // Handle of the window to be accessed

BYTE huge \*text;                       // pointer to the text to be appended. The text must be NULL terminated.

int FontId;                               // font id to use for the new text. Use -1 for the default value.

int Parald;                               // paragraph id to use for the new text. Use -1 for the default value.

int CellId;                               // Cell id to use for this new text. Use -1 for the default value. When creating a table structure using this function, you must specify the current cell id for this parameter.

int ParaFID;                             // Paragraph frame id. Use -1 for the default value.

BOOL repaint;                            //Repaint the window after this operation

**Description:** Please refer to the TerAppendText function for more information.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerAppendText](#)  
[TerInsertText](#)  
[TerInsertLine](#)  
[TerCreateFont](#)  
[TerCreateParald](#)



## TerDeleteBlock

**Delete a highlighted text block.**

BOOL TerDeleteBlock(hWnd, repaint)

HWND hWnd; // The handle of the window to be accessed

BOOL repaint; // repaint the screen after this operation.

**Return Value:** This function returns TRUE when successful.



## TerGetLine

**Retrieve a text line.**

int TerGetLine(hWnd, LineNo, text, font)

int TerGetLineEx(hWnd, LineNo, text, lead, font)

int TerGetLineU(hWnd, LineNo, UnicodeText, font)

HWND hWnd; // The handle of the window to be accessed

long LineNo; // line number (0 to TotalLines - 1) to retrieve.

LPBYTE text; // pointer of the location to retrieve the text for the line.

LPWORD UnicodeText; // pointer of the location to retrieve the Unicode text for the line.

LPBYTE lead; // pointer to the lead bytes

LPWORD font; // pointer of the location to retrieve the font ids for each character in the line. Each font is is a two bytes long.

**Description:** This routine is used to retrieve the text and the font ids for a specified line. The text line is NULL terminated. The 'font' pointer receives an array of font ids. You can get further information about each font id by calling the 'GetFontInfo' function.

The 'text', 'lead' or the 'font' pointer can be set to NULL, if you do not wish for that information.

The TerGetLineEx function support an additional 'lead' pointer. This argument can be used to retrieve the lead bytes for the text in an MBCS (Multibyte Character Set) system. In this case, the 'text' argument will return the tail bytes and the 'lead' argument will return the lead bytes. You can use the TerMakeMbc function to build an MBCS string using the lead and the tail bytes.

**Return Value:** This function returns the length of the retrieved line. This function return -1 if an error is encountered or when the 'LineNo' exceeds the total lines in the windows.

**Comment:** The VB users might find the following variation of this function easier to use:

*int TerGetLineU2(LineNo, UnicodeText, font)*

In the above method, the UnicodeText and font parameters are passed as simple VB strings.

#### **C Example:**

```
BYTE text[MAX_WIDTH];
WORD font[MAX_WIDTH];
int len;
' Get the text and font ids
TerGetLine(hWnd,LineNo,text,font);
' Get only the text
TerGetLine(hWnd,LineNo,text,NULL);
```

#### **VB Example:**

(1)

```
Dim text As String
Dim font As String
Dim result As Long

result = Toc1.TerGetLineU2(0, text, font)
```

The font id for a particular column position can be retrieved as following

```
FontId=AscW(Mid(font, col, 1)) ' font id for a column
```

(2)

```

    dim text as string*MAX_WIDTH
    dim font(MAX_WIDTH) as Integer      ' array to receive the
fontids
    dim len as integer
    text=space$(MAX_WIDTH)             ' allocate space for up to 300
character

    ' Get the text and font
    len = TerGetLine(Ter1.hWnd, LineNo, ByVal text, font(0))

    ' Please note that the font array is passed as the
    reference to the first element of the font
    'Get only text - pass NULL for the font parameter
    len = TerGetLine(Ter1.hWnd,LineNo,ByVal text, ByVal 0&)
    ('ByVal' is prefixed because 'text' is defined as
    'As Any' in the VB declaration)

```

### Pascal Example:

```

var
text: array [0..MAX_WIDTH] of char
font: array [0..MAX_WIDTH] of Word
LineLen: Integer
begin
    LineLen:=TerGetLine(ter1.Handle,LineNo,text,font[0]);
    {Please note that the font array is passed as
    the reference to the first element of the array.}
end

```

**See Also:**  
[TerSetLine](#)  
[TerGetLineInfo](#)  
[TerGetParaInfo](#)  
[GetFontInfo](#)



## TerGetLineInfo

### Get the current line attributes.

BOOL TerGetLineInfo(hWnd, LineNo, Parald, CellId, ParaFID, x, y, height, lflags, InfoFlags);

HWND hWnd;	// The handle of the window to be accessed
long LineNo;	// Line number to retrieve the information. Set to -1 to use the current line number.
LPINT ParaId;	// Pointer to the paragraph id
LPINT CellId;	// Pointer to the table cell id
LPINT ParaFID;	// Pointer to the paragraph frame id
LPINT x;	// pointer to the x position (in printer units) of the line.
LPINT y;	// Pointer to the y position (in printer units) of the line relative to the top of the page.
LPINT height;	// Pointer to the height (in printer units) of the line.
LPDWORD lflags;	// Line flags as defined by the LFLAG_ constants in the TER.H file
UINT far *InfoFlags;	// Miscellaneous flags as defined by the INFO_ constants in the TER.H file.

**Description:** The x, y, and the height values are available in the Page Mode only.

**Return Value:** This function returns a TRUE value when successful. Otherwise it returns FALSE.

**See Also:**  
[GetTerLine](#)  
[TerGetLineWidth](#)



## TerGetLineParam

**Get a parameter associated with a text line number.**

int TerGetLineParam(HWND hWnd, LineNo, type)

HWND hWnd;	// The handle of the window to be accessed
int LineNo;	// Line number (0 to TotalLines -1) to retrieve parameters. Set to -1 to specify the current line.
int type;	// The parameter to retrieve:
LP_LINE_NUM	Current line number.

LP_LINE_LEN	Return the line length.
LP_CELL_ID	Cell id associated with the line. A non-zero value indicates that the line is included in a table.
LP_PARA_FRAME_ID	Paragraph frame id associated with the line. A non-zero value indicates that the line is included in a positionable object.
LP_LIST_ID	List id associated with the line. A non-zero value indicates that the line is included in a list.
LP_LIST_OR_ID	List override id associated with the line. A non-zero value indicates that the line is included in a list.
LP_LIST_LEVEL	List nesting level.
LP_LIST_FONT	Font id associated with the bullet text on this line.

**Return Value:** The function returns the value for the requested parameter. It returns LP\_ERROR to indicate an error condition.



## TerGetLineWidth

**Get the line width.**

```
int TerGetLineWidth(hWnd, LineNo)
```

```
HWND hWnd;           // The handle of the window to be accessed
```

```
long LineNo;         // line number (0 to TotalLines - 1) to get width.
```

**Return Value:** This function returns the line width in twips. A return value of -1 indicates an error condition.

**See Also**  
[TerGetLineInfo](#)



## TerGetRtfSel

**Retrieve the selected text in the RTF format.**

HANDLE TerGetRtfSel(hWnd, size)

HWND hWnd; // The handle of the window to be accessed

long far \*size; // The size of the output global buffer

**Return Value:** This function returns the global handle to the buffer containing the selected text in the RTF format. The size of the global buffer is returned using the second argument. Lock (GlobalLock) the returned handle to access the RTF text. Your application owns this memory handle. When the data is no longer required, use the GlobalFree function to free the global handle.

A NULL value of the handle indicates an error.

**See Also:**  
[TerGetTextSel](#)  
[GetTerBuffer](#)  
[SetTerBuffer](#)  
[ReadTerFile](#)  
[SaveTerFile](#)  
[CloseTer](#)  
[HandleToStr](#)



## TerInsertLine

**Insert a line of text before the current line.**

BOOL TerInsertLine(hWnd, text, FontId, ParaId, CellId, reserved, repaint)

HWND hWnd; // The handle of the window to be accessed

BYTE huge \*text; // pointer to the text to be appended. The text must be NULL terminated. The length of the text must not exceed MAX\_WIDTH - 1 or 299.

int FontId; // font id to use for the new text. Use -1 for the default value.

int ParaId; // paragraph id to use for the new text. Use -1 for the default value.

int CellId; // Cell id to use for this new text. Use -1 for the default value. When creating a table structure using this function, you must specify the current cell id for this parameter.

```
BOOL repaint;           //Repaint the window after this operation
```

**Return Value:** This function returns TRUE when successful.

- TerAppendText
- TerInsertText
- TerCreateFont
- TerCreateParald



**Insert an RTF file at the specified cursor location.**

```
HWND hWnd;           // Handle of the window to be accessed
```

long line;                      // line location (base 0) where the text will be inserted.  
You can also set the 'line' parameter to minus 1 to insert  
the rtf document at the current cursor position. Or, set it  
to minus 2 to insert the rtf file at the end of the current  
document.

```
BOOL repaint;           // TRUE to refresh the window after this operation
```

This function is available in the 'word wrap' mode only.

## InsertRtfBuf SetTerBuffer





## TerInsertText

**Insert text at the cursor position.**

BOOL TerInsertText(hWnd, text, FontId, ParaId, repaint)

BOOL TerInsertTextU(hWnd, UText, FontId, ParaId, repaint)

HWND hWnd; // Handle of the window to be accessed

BYTE huge \*text; // pointer to the text to be inserted. The text must be NULL terminated. This parameter is applicable to the TerInsertText function only.

WORD huge \*UText // pointer to the unicode text to be inserted. The text must be NULL terminated. This parameter is applicable to the TerInsertTextU function only.

int FontId; // font id to use for the new text. Use -1 for the default value.

int ParaId; // paragraph id to use for the new text. Use -1 for the default value.

BOOL repaint; //Repaint the window after this operation

**Description:** This function inserts the specified text at the current cursor position. After the operation the cursor is positioned after the inserted text.

**Return Value:** This function returns TRUE when successful.

**Comment:** A VB application might find easier to use the following variable of these methods available through ActiveX interface:

*BOOL TerInsertTextU2(UText, FontId, ParaId, repaint)*

This method allows you to pass unicode text as simple VB string using the UText parameter. The remaining parameters and the result values are the same as described above.

### See Also:

[TerAppendText](#)  
[InsertTerText](#)  
[TerCreateFont](#)  
[TerCreateParaId](#)



## TerSetLine

### Set the text and font ids for a line

BOOL TerSetLine(hWnd, LineNo, text, font)

BOOL TerSetLineEx(hWnd, LineNo, text, lead, font)

HWND hWnd;	// The handle of the window to be accessed
long LineNo;	// The text line number (0 to TotalLines - 1) to set
LPBYTE text;	// Contains the pointer to a text string to apply. This string must be NULL terminated.
LPBYTE lead;	// Contains the pointer to the lead bytes.
LPWORD font;	// Contain the pointer to a font id array to apply. Each font id is two bytes long. You can set this parameter to NULL if the font ids are not available.

**Description:** This function is used to set new text for a line. The number of bytes in the 'font' character array must be identical to the number of bytes in the text string. The font array must contain valid font ids (0 to TotalFonts - 1). You can get information about an editor font id by using the GetFontInfo function.

The TerSetLineEx function support an additional 'lead' pointer. This argument can be used to provide the lead bytes for the text in an MBCS (Multibyte Character Set) system. In this case, the 'text' argument will provide the tail bytes and the 'lead' argument will provide the lead bytes. You can use the TerSplitMbcS function to split an MBCS string into lead and tail bytes to supply to this function.

**Return Value:** This function returns a TRUE value when successful.

#### See Also:

[TerGetLine](#)  
[GetFontInfo](#)



## Print and Print-preview

### In This Chapter

[TerClosePriner](#)  
[TerGetPrinter](#)  
[TerIsPrinting](#)  
[TerMergePrint](#)  
[TerMergePrintRep](#)  
[TerMergePrintVB](#)  
[TerOverridePageSize](#)  
[TerPrint](#)  
[TerPrintPreview](#)  
[TerSelectPrint](#)  
[TerSetDefPrinter](#)  
[TerSetPreview](#)  
[TerSetPrinter](#)

[TerSetPrintPreview](#)  
[TerSpoolBegin](#)  
[TerSpoolEnd](#)



## TerClosePriner

**Close the printer driver.**

BOOL TerClosePrinter();

HWND hWnd;                               // Handle of the window to be accessed

**Description:** The editor automatically closes all shared printer drivers when the last edit window is closed, unless the TFLAG2\_KEEP\_PRINTER\_OPEN flag is set. If your application has set this flag, then your application should close the printer drivers manually by calling TerClosePrinter **after** the last edit window is closed.

**Return Value:** This function returns TRUE if successful.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

**See Also:** TFLAG2\_Keep\_Printer\_Open

**See also:**

[TerSetFlags2](#)



## TerGetPrinter

**Retrieve the current print name and driver information.**

BOOL TerGetPrinter(hWnd, name, driver, port)

BOOL TerGetHypertext2(hWnd, LineNo, ColNo, text, code, select)

HWND hWnd;                               // The handle of the window to be accessed

LPBYTE name;                            // Pointer to receive the printer name

LPBYTE driver;                          // Pointer to receive the printer driver

LPBYTE port;                            // Pointer to receive the printer port

**Return Value:** This function returns TRUE if successful.



## TerIsPrinting

**Check if the editor is printing a document.**

BOOL TerIsPrinting(hWnd)

HWND hWnd; // Handle of the window to be accessed

**Return Value:** This function returns TRUE if the editor is printing or previewing a document.

**See Also:**

[TerPrint](#)



## TerMergePrint

**Print a document without invoking the editing session.**

BOOL TerMergePrint(prt)

HWND hWnd; // Handle of the window to be accessed

struct StrPrint far \*prt; // Print request parameter structure.

**Description:** This function is used to print a buffer or a file to a specified printer or window device context and at a specified location on the page. If requested, this function can replace the field names with field data. The print parameter structure (StrPrint) is used to specify the printing parameters.

```
struct StrPrint {
    char    InputType;
    char    file[129];
    HANDLE  hBuffer;
    long    BufferLen;
    char    delim;
    char    padding1;
    HDC     hDC;
    LPRECT  rect;
    long    StartPos;
    BOOL    OnePage;
    long    NextPos;
```

```

LPBYTE    MergeFields;
LPBYTE    MergeData;
BOOL      PrintHiddenText;
HANDLE    hInst;
HWND      hParentWnd;
int       NextY;
HDC       hMetaDC;
}

```

### Member Variables:

InputType:	This flag specifies the input type. If you wish to specify a disk file name, set the 'InputType' to 'F'. Conversely, if you wish to pass the text in a buffer, set this field to 'B'.
file:	If the 'InputType' is 'F', specify the input file path name in this field.
hBuffer:	If the 'InputType' is 'B', specify the global memory handle for the buffer containing the text.
BufferLen:	If the 'InputType' is 'B', this field specifies the length of the buffer (hBuffer).
delim:	If the 'InputType' is 'B', this field specifies the special character used to delimit the lines. This character can be a carriage-return (ASCII 13), or any other character of your choice. This field is not used when the buffer contains RTF data.
hDC:	<p>The device context of the printer or the window to print the text. The device context <b>MUST</b> be mapped to use the pixel units.</p> <p>If this field is set to NULL, the editor opens a new device context to the current printer. If the 'hMetaDC' field is specified (non-NULL), then hDC may not be NULL.</p>
rect:	This field contains a far pointer to a printing rectangle. The rectangle co-ordinates are specified in the millimeter (mm) units. For a higher precision, you can specify the rectangle coordinates in twips units by specifying the negative values for the left, right, top and bottom variables in the rect structure.
StartPos:	The positive value for this field specifies the character position to begin the printing. The negative value specifies the page number to begin printing. Set to 0 to

begin the printing from the first page.

OnePage:	Set this variable to TRUE to print one page only. When this variable is set to FALSE, the editor prints the entire document by spooling each page.
NextPos:	(OUTPUT) The editor returns the character position of the next page to be printed. When the entire document is printed, the editor sets this field to 0.
MergeFields:	This field specifies the pointer to a list of mail merge field names. Each field name must be separated by a ' ' character. The list must be terminated by a NULL character. If you do not wish to merge field data, set this field to NULL.
MergeData:	This field specifies the pointer to a list of mail merge data strings. Each data string must be separated by a ' ' character. The number of data elements in the 'MergeData' array MUST be the same as the number of elements in the 'MergeFields' array. The list must be terminated by a NULL character. If you do not wish to merge field data, set this field to NULL.
PrintHiddenText:	Set this field to TRUE to print any hidden text.
hInst:	Pass the instance handle of your application using this field.
hParentWnd:	Pass the window handle of your application using this field.
NextY:	(OUTPUT) The field returns the Y pixel position on the device context after printing.
hMetaDC:	<p>This field is used to specify the metafile device context when printing to a metafile or when you need to specify different device contexts for attribute and data. In all other situations it should be set to NULL.</p> <p>When a 'hMetaDC' is specified, then the editor uses the device context used in the 'hDC' field for only retrieving the device attributes. All text output is directed to the 'hMetaDC' device context.</p> <p>When 'hMetaDC' specifies a metafile device context, the left and top coordinates in the 'rect' field must be set to zero.</p>

**Return Value:** This function returns a TRUE value when successful.

**Comment:** The Visual Basic application should use the TerMergePrintVB function instead of TerMergePrint function.

You can set a print callback function (TerRegisterPrintCallback) to manipulate the document before the printing begins. This technique can also be used to print an HTML file using HTML Add-on.

**Please note that this function is not available as an ActiveX control method. It must be used as a DLL function.**

#### See Also:

[TerPrint](#)  
[TerMergeFields](#)  
[TerMergePrintVB](#)  
[TerMergePrintRep](#)  
[TerSetDefPrinter](#)  
[TerRegisterPrintCallback](#)  
[TerSetInitTypeface](#)



## TerMergePrintRep

**Print a document without invoking an editor window (alternate function).**

```
long TerMergePrintRep(hDC, file, hBuf, BufLen, rect, MergeFields, MergeData, StartPos, NextY)
```

```
long TerMergePrintRep2(hDC, file, hBuf, BufLen, rect, MergeFields, MergeData, StartPos, NextY, hAttribDC)
```

HDC hDC;	// target device context
LPBYTE file;	// file name if the data is specified in a disk file, otherwise set to NULL.
HGLOBAL hBuf;	// Buffer handle if the data is supplied in a global buffer
long BufLen;	// buffer length if the data is supplied in a global buffer.
RECT far *rect;	// rectangle within which the printing is to take place.
MergeFields;	// Mail merge field names (NULL for default)
MergeData;	// Mail merge data string (NULL for default)
long StartPos;	// starting character position.
LPINT NextY;	// (output) This location receives the Y position after printing the entire buffer.
HDC hAttribDC;	// Attribute DC, if different from target DC, otherwise NULL.

**Description:** This function provides a functionality similar to the TerMergePrint function. Please refer to the TerMergeField function for further description about the parameters. This function prints only one page at a time.

**Return Value:** This function returns the character position of the next page. It returns 0 if the last page is printed. It returns -1 if an error occurs.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

**See Also:**  
[TerMergePrint](#)



## TerMergePrintVB

**Print a document without invoking the editing session in a Visual Basic application.**

BOOL TerMergePrintVB(prt, PrtRect, MergeFields, MergeData)

struct StrPrint far *prt;	// Print request parameter structure.
LPRECT PrintRect;	// Rectangle to print inside. Pass a NULL value to print on the entire page.
LPBYTE MergeFields;	// Merge fields. Pass a NULL value when merge fields are not used in the document.
LPBYTE MergeData;	// Merge data values. Pass a NULL value when merge fields are not used in the document.

**Description:** This function is to be used by a Visual Basic application. This function provides the same functionality as the TerMergePrint function. But since Visual Basic does not allow passing the pointers (PrtRect, MergeFields, and MergeData) inside a structure (StrPrint), this function allows you to pass these values as argument to the function.

Please refer to the TerMergePrint function for the description of the 'prt', 'MergeFields', and 'MergeData' arguments.

This function is prototypes as following in the VB environment:

```
TerMergePrintVB(prt As StrPrint, ByRef PrintRect As Any, ByVal MergeFields As Any,
ByVal MergeData As Any) As Integer
```

The last three arguments are defined as 'As Any' to allow you to pass a NULL value for these arguments. To pass a NULL value to any of these arguments, pass the value as 'ByVal 0&'.

Example:

```
result = TerMergePrintVB(prt, ByVal 0&, ByVal FieldNames, ByVal FieldData)
```

```
result = TerMergePrintVB(prt, PrtRect, ByVal FieldNames, ByVal FieldData)
```



```
result = TerMergePrintVB(prt, PrtRect, ByVal 0&, ByVal 0&)
```

Please also refer to the dmo\_vb project as an example (print option) of using this function.

**Return Value:** This function returns a TRUE value when successful.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

**See Also:**  
[TerMergePrint](#)



## TerOverridePageSize

**Override the document page size.**

```
BOOL TerOverridePageSize(width, height)
```

```
int width;                // The page width in twips unit (for the portrait orientation)
```

```
int height;               // The page height in twips unit (for the portrait orientation)
```

**Description:** This function is used to specify a page size which may not be supported by the selected printer. The editor formats the document to the given page size, but it does **not** enforce the page size to the printer. This function is called before an editor window is opened. Set the width and height to 0 to reset to the regular page size for the subsequent editor windows.

**Return Value:** This function returns the TRUE when successful.

**See Also:**  
[TerSetPaper](#)



## TerPrint

**Print the current document.**

```
BOOL TerPrint(hWnd,dialog)
```

```
BOOL TerPrintEx(hWnd, dialog, FirstPage, LastPage)
```

```
BOOL TerPrint2(hWnd,dialog, FirstPage, LastPage, Copies, Colate)
```

```
HWND hWnd;                // Handle of the window to be accessed
```

```
BOOL dialog;              // Set to TRUE to show a dialog box to the user.
```

int FirstPage;	// First page (zero based) to print. Set to -1 to disable page range printing. Set to -2 to print the current page. This parameter is ignored if the 'dialog' parameter is set true. Set to -3 to print the selected text if text is selected, or the entire document if text is not selected.
int LastPage;	// Last page (zero based) to print. This parameter is ignored if the 'FirstPage' parameter is set to -1, -2, -3, or the 'dialog' parameter is set to TRUE.
int Copies;	// Number of copies to print Set to 0 to assume the 'copies' value from the printer setup information.
int Colate;	// When printing multiple copies, set this parameter to TRUE to print all pages of the first set, before printing the next copy.

**Description:** This function can be used to print the current document. When the 'dialog' parameter is TRUE, the editor displays a dialog box. The dialog box allows the user to print the entire document or the selected text.

The text is printed to the currently selected printer using the current settings for margins. The editor creates a new printer device context for printing.

**Return Value:** The editor returns a TRUE value when the print job is successfully completed.

**See Also:**  
[TerMergePrint](#)  
[TerIsPrinting](#)



## TerPrintPreview

**Preview the specified page in the current document.**

BOOL TerPrintPreview(hWnd, hDC, rect, page, scale)

HWND hWnd;	// Handle of the window to be accessed
HDC hDC;	// The window device context on which to display the preview output.
RECT far * rect;	// The rectangle on the device context where the preview output is to be displayed. The rectangle must be specified in the device units.

*If you programming language does not allow you to pass a rectangle pointer, then use the TerSetParamRect function to set the rectangle parameters before calling this function.*

int page;	// Page number to preview (0 to TotalPages-1). Specify a -1 value to preview the current page.
BOOL scale;	// Set to TRUE if you wish this API to perform scaling to fit the page within the rectangle. Set to FALSE if your application performs the required scaling.

**Description:** This function is used to draw the image of a page at the specified location on the specified device context.

**Return Value:** The editor returns a TRUE value when successful

**See Also:**  
[TerPrint](#)  
[TerSetPrintPreview](#)  
[TerSetPreview](#)  
[TerSetParamRect](#)



## TerSelectPrint

**Print the selected text.**

BOOL TerSelectPrint(hWnd)

HWND hWnd;	// The handle of the window to be accessed.
------------	---

**Description:** This function extracts the selected text and uses the TerMergePrint function to print it to the current printer. Since this function creates a temporary buffer for the selected text, it is not efficient for huge text selection.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerPrint](#)  
[TerMergePrint](#)



## TerSetDefPrinter

**Set the default printer for the editor.**

BOOL TerSetDefPrinter(name,driver,port)

LPBYTE name;	// The new default printer name. Set to "" to restore Windows original default printer.
--------------	---

LPBYTE driver;	// Printer driver name.
----------------	-------------------------

LPBYTE port;                               // Printer port

**Description:** Use this function to override the Windows default printer. The new default printer is effective for the controls created after this function is called. This function can also be called before calling the TerMergePrint function.

**Return Value:** This function always returns TRUE.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

**Example:** Set the default printer to HP Laserjet on the second printer port:

```
TerSetDefPrinter("HP LaserJet 4", "HPPCL5MS.DRV", "LPT2:");
```



## TerSetPreview

**Set preview parameters.**

```
BOOL TerSetPreview(hWnd, NumPages, ZoomPct, ShowToolbar)
```

HWND hWnd;                               // Window handle to access

int NumPages;                            // number of preview pages to display. This value must be a 1 or 2.

int ZoomPct;                             // Zoom percentage (0 to 200). A value of 0 displays full pages fitted in the window area.

BOOL ShowToolbar;                        // Show the print preview tool bar. This setting takes effect after the current print preview session.

**Return Value:** The function returns TRUE when successful.



## TerSetPrinter

**Set new printer selection information.**

```
BOOL TerSetPrinter(hWnd, device, driver, port, hInitDevMode)
```

HWND hWnd;                               // Window handle to access

LPBYTE device;                            // New printer name. Set to NULL to use the default

printer name.

LPBYTE driver;	// New printer driver name. Set to NULL to use the default printer driver.
LPBYTE port;	// New printer port. Set to NULL to use the default printer port
HGLOBAL hInitDevMode;	// Initial printer setup information structure. Set to NULL to use the default printer initialization information. In some programming environments, set this parameter to zero when using this function as exported through toc30.ocx

**Description:** This function is used to set the new printer selection information. In many cases you need only supply a valid printer name leaving all other parameters NULL.

**Return Value:** The function returns a TRUE value when successful..



## TerSetPrintPreview

**Allocate or deallocate print preview resources.**

BOOL TerSetPrintPreview(hWnd, begin)

HWND hWnd;	// Window handle to access
BOOL begin;	// TRUE to allocate print preview resources, and FALSE to deallocate them.

**Description:** This function can be called before and after calling the TerPrintPreview function to increase the print preview performance.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerPrintPreview](#)



## TerSpoolBegin

**Begin a multi-document print job.**

BOOL TerSpoolBegin(name)

LPBYTE name; // The name of the print job. The print-job appears with this name in the printer queue.

**Description:** This method together with the TerSpoolEnd method is used to combine multiple calls to the [TerMergePrint](#) method into a single print job. When calling the [TerMergePrint](#) method, the member variable hDC within the StrPrint structure must be set to NULL.

**Return Value:** This function returns TRUE when successful.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

**Example:**

```
TerSpoolBegin("MyPrintJob") // begin the print job

TerMergePrint(...) // print the first document
TerMergePrint(...) // print the second document
TerMergePrint(...) // print the third document

TerSpoolEnd() // end the print job
```



## TerSpoolEnd

**End a multi-document print job.**

BOOL TerSpoolEnd()

**Description:** This method together with the TerSpoolBegin method is used to combine multiple calls to the [TerMergePrint](#) method into a single print job. When calling the [TerMergePrint](#) method, the member variable hDC within the StrPrint structure must be set to NULL.

**Return Value:** This function returns TRUE when successful.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

**Example:**

```
TerSpoolBegin("MyPrintJob") // begin the print job

TerMergePrint(...) // print the first document
TerMergePrint(...) // print the second document
TerMergePrint(...) // print the third document
```

```
TerSpoolEnd()           // end the print job
```



## Character Formatting

### In This Chapter

- [GetFontInfo](#)
- [SetTerBkColor](#)
- [SetTerCharStyle](#)
- [SetTerColor](#)
- [SetTerDefaultFont](#)
- [SetTerPointSize](#)
- [SetTerFont](#)
- [TerCreateFont](#)
- [TerGetCurFont](#)
- [TerGetEffectiveFont](#)
- [TerGetFontAux1Id](#)
- [TerGetFontLang](#)
- [TerGetFontFieldId](#)
- [TerGetFontParam](#)
- [TerGetFontSpace](#)
- [TerGetFontStyleId](#)
- [TerGetTextColor](#)
- [TerLocateFontId](#)
- [TerLocateStyle](#)
- [TerLocateStyleChar](#)
- [TerRestrictFont](#)
- [TerSelectCharStyle](#)
- [TerSetCharAuxId](#)
- [TerSetCharLang](#)
- [TerSetCharScaleX](#)
- [TerSetCharSet](#)
- [TerSetCharSpace](#)
- [TerSetCustomFontList](#)
- [TerSetDefLang](#)
- [TerSetDefTextColor](#)
- [TerSetEffectiveFont](#)
- [TerSetFontId](#)
- [TerSetFontSpace](#)
- [TerSetFontStyleId](#)
- [TerSetInitTypeface](#)
- [TerSetNextFontAux1Id](#)
- [TerSetTcField](#)
- [TerSetTextCase](#)
- [TerSetUlineColor](#)
- [TerSetUserFont](#)



## GetFontInfo

**Retrieve information about an editor font id.**

**BOOL GetFontInfo(hWnd, FontId, typeface, PointSize, styles)**

BOOL GetFontInfo2(hWnd, FontId, typeface, TwipsSize, styles)

HWND hWnd; // Handle of the window to be accessed

int FontId; // The editor font id to retrieve. A valid font id would a value between 0 and TotalFonts - 1.

You can also use this function to get the font information for a stylesheet item by specifying the style id as a negative value for this parameter. Also, use the following constant to specify normal or the current style item.

SID\_NORMAL: Normal style item

SID\_CUR: Style item being current edited

LPBYTE typeface; // This character pointer receives the typeface for the specified font id. The typeface string can be up to 32 characters.

LPINT PointSize; // This integer pointer receives the point size for the specified font id. This argument is used by the GetFontInfo function only.

LPINT TwipsSize; // This integer pointer receives the size of the specified font in the twips unit. This is useful when a font uses fractional point size (1 point equal 20 twips). This argument is used by the GetFontInfo2 function only.

LPUINT styles: // This integer pointer receives the styles flags for the specified font id. The style flag consists of the following bits:

BOLD: Bold

ULINE: Underline

ULINED: Double Underline



ITALIC:	Italic
STRIKE:	Strikethrough
DOUBLE_STRIKE	Double line strike.
SUPSCR:	Superscript
SUBSCR:	Subscript
HIDDEN:	Hidden Text
PROTECT:	Protected Text
CAPS	All caps
SCAPS	Small Caps

**Description:** Use this function to retrieve information about an editor font id.

The script environment such as Java script do not support a pass-by-reference parameter. In such environment you would first call the GetFontInfo method. Then call the TerGetRefParamStr and TerGetRefParam to retrieve the string and numeric values of the parameters returned by the previous call to the GetFontInfo method.

**Return Value:** This function returns TRUE if successful. The function returns FALSE if an error is encountered or when the FontId specifies an invalid font id.

#### See Also

[TerGetLine](#)  
[GetTerFields](#)  
[TerGetFontStyleId](#)



## SetTerBkColor

**Set the background color for the text.**

**BOOL SetTerBkColor(hWnd,color,repaint)**

HWND hWnd;                               // The handle of the window to be accessed

COLORREF color;                        // new background color

BOOL repaint;                            // TRUE to repaint the screen after this operation.

**Description:** If a text block is selected before this operation, the new background color is applicable for every character in the block. If a block is not highlighted, this function selects the new color for the next character input.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[SetTerColor](#)  
[TerGetTextColor](#)



## SetTerCharStyle

### Set or reset the character styles

BOOL SetTerCharStyle(hWnd, styles, OnOff, repaint)

HWND hWnd;                      // The handle of the window to be accessed

UINT styles;                    // Character style to set or reset:

BOLD:                          **Bold**

ULINE:                         **Underline**

ULINED:                       Double Underline

ULINE\_DOTTED                 Dotted underline

ITALIC:                        *Italic*

STRIKE:                        Strikethrough

DOUBLE\_STRIKE               Double strikethrough

SUPSCR:                        Superscript

SUBSCR                         Subscript

HIDDEN:                        Hidden Text

PROTECT:                       Protected Text

HLINK:                         Hyperlink

CAPS:                          All caps

SCAPS:                         Small Caps

To specify more than one styles, use the 'logical OR' (|) operator.

BOOL OnOff;                      //TRUE to set the specified styles, FALSE to reset the specified styles.

BOOL repaint;                    //TRUE to refresh the window after this operation

**Description:** This function is used to set or reset the give character styles. If a text block is highlighted, this operation is applicable to all characters in the block. Otherwise only the current character is affected.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[SelectTerText](#)  
[TerLocateStyle](#)  
[TerSetUlineColor](#)



## SetTerColor

### Set text color

BOOL SetTerColor(hWnd, COLORREF color, repaint)

HWND hWnd;                      // The handle of the window to be accessed

COLORREF color;                // new color to apply

BOOL repaint;                    //TRUE to refresh the window after this operation

**Description:** This function is used to apply the new color to the text. If a text block is highlighted, this operation is applicable to all characters in the block. Otherwise the new color is selected for the next character input.

**Return Value:** This function returns TRUE if successful.

### Example:

SelectTerText(hWnd,0,0,0,40,FALSE); // select the first 40 characters of the first line.

SetTerColor(hWnd,0x00FF00,TRUE); // apply green color to the selected text

**See Also:**  
[SetTerBkColor](#)  
[TerGetTextColor](#)  
[SelectTerText](#)



## SetTerDefaultFont

### Set the default font and pointsize

BOOL SetTerDefaultFont(hWnd, typeface, PointSize, style, color, repaint)

HWND hWnd;	// The handle of the window to be accessed
LPBYTE typeface;	// typeface of the default font
int PointSize;	// Point size for the default font. You can also specify the font size in twips by specifying a negative value (1 point equal 20 twips)
UINT style;	// style of the default font. Refer to the SetTerCharStyle function for a list of style constants. Use 0 for default.
COLORREF color;	// default text color. Use 0 for default.
BOOL repaint;	//TRUE to refresh the window after this operation

**Description:** This function is used by an application to change the initial or default font for the editor. Any text that used the previous default font now gets printed with the new font, point size, style and color.

**Return Value:** This function returns TRUE if successful.

### Example:

```
// use Arial typeface and 12 pointsize for default.
SetTerDefaultFont(hWnd,"Arial",12, 0,0, TRUE);

// use Arial typeface and 12 pointsize
SetTerDefaultFont(hWnd,"Arial",-240, 0, 0, TRUE);
```

**See Also:**  
[SetTerFont](#)



## SetTerPointSize

### Set font pointsize for the text

BOOL SetTerPointSize(hWnd, pointsize, repaint)

HWND hWnd;	// Handle of the window to be accessed
------------	--

int pointsize;                      // size of the new font in points (72 points = 1 inch). You can specify the font size in twips unit by using a negative value (1 point equal 20 twips).

BOOL repaint;                      //TRUE to refresh the window after this operation

**Description:** This function is used to apply the new font size to the text. If a text block is highlighted, this operation is applicable to all characters in the block. Otherwise only the current character is affected.

**Return Value:** This function returns TRUE if successful.

**Example:**

```
// select the first 40 characters of the first line.
SelectTerText(hWnd,0,0,0,40,FALSE);

// Set the point size of 14 to the selected text
SetTerPointSize(hWnd,14,TRUE);

// Set the font size to 280 twips (14 points).
SetTerPointSize(hWnd,-280,TRUE);
```

**See Also:**  
[SelectTerText](#)  
[SetTerFont](#)  
[SetTerCharStyle](#)



## SetTerFont

**Set font typeface for the text**

BOOL SetTerFont(hWnd, typeface, repaint)

HWND hWnd;                      // The handle of the window to be accessed

LPBYTE typeface;                      // typeface of the new font

BOOL repaint;                      //TRUE to refresh the window after this operation

**Description:** This function is used to apply the new font typeface to the text. If a text block is highlighted, this operation is applicable to all characters in the block. Otherwise only the current character is affected.

**Return Value:** This function returns TRUE if successful.

### Example:

```
// select the first 40 characters of the first line.
SelectTerText(hWnd,0,0,0,40,FALSE);

// apply Arial font typeface to the selected text
SetTerFont(hWnd,"Arial",TRUE);
```

**See Also:**  
[SelectTerText](#)  
[SetTerPointSize](#)  
[SetTerCharStyle](#)  
[SetTerDefaultFont](#)



## TerCreateFont

### Create a font id.

```
int TerCreateFont(hWnd, Reuseld, shared, typeface, pointsize, style, color, BkColor,
FieldId, AuxId)

int TerCreateFont2(hWnd, Reuseld, shared, typeface, pointsize, style, color, BkColor,
FieldId, AuxId, CharStyld, ParaStyld, expand)

int TerCreateFont3(hWnd, Reuseld, shared, typeface, pointsize, style, color, BkColor,
FieldId, AuxId, CharStyld, ParaStyld, expand, CharSet)
```

HWND hWnd;	// The handle of the window to be accessed
int Reuseld;	// Existing font id to modify with new information. Use -1 to create a new font id.
BOOL shared;	// When TRUE, the editor matches the requested specification against the existing font ids. If a matching font id is found, it returns that id. Otherwise it creates a new id. A TRUE value for this field is mutually exclusive with a zero or positive value for the Reuseld field.
LPBYTE typeface;	// Typeface of the new font
int PointSize;	// Point size of the new font. You can also specify the font size in twips unit by using a negative value (1 point equal 20 twips).
WORD style;	// Style bits for the new font. Refer to the function SetTerCharStyle for a list of style ids. Use 0 for the default value.

COLORREF color;	// Text foreground color. use 0 for default.
COLORREF BkColor;	// Text background color. use 0xFFFFFFFF for default.
int FieldId;	// Text field id. Use 0 for the default value.
int AuxId;	// An application specified id. The editor does not use this id internally. Use 0 for default.
int CharStylId;	// Character style id. Set to 1 to use the default.
int ParaStylId;	// Paragraph style id. Set to 0 to use the default.
int exapnd;	// Character width expansion in twips unit. Use 0 for default.
UINT CharSet	// Character set for the font

**Description:** This function is used to create a new font id or to modify an exiting id with new font information. To modify an existing id, specify the old font id using the 'Reuseld' argument, otherwise set the 'Reuseld' parameter to -1. When an existing id is modified, this function automatically updates the text which uses this id with new information.

**Return Value:** When successful, this function returns the id of the new font. Otherwise it returns -1.

**See Also:**  
[TerCreateParald](#)  
[TerAppendText](#)  
[TerSetNextFontAux1Id](#)



## TerGetCurFont

**Retrieve the font id (or picture id) at the given location.**

int TerGetCurFont(hWnd, LineNo, ColNo)

HWND hWnd;	// The handle of the window to be accessed
long LineNo;	// line number for the location. Set to -1 to use the current text line and column position.
int ColNo;	// column number for the location.

**Return Value:** This function returns the font id for the character at the given location.

**Note:** For a picture character, the font ids is same as the picture id. So the font id returned by the TerGetCurFont function is actually a picture id if the LineNo/ColNo

parameters point to a picture.

**See Also:**  
[TerGetEffectiveFont](#)



## TerGetEffectiveFont

**Retrieve the font id effective at the current cursor position.**

```
int TerGetEffectiveFont(hWnd)
```

HWND hWnd; // The handle of the window to be accessed

**Return Value:** This function returns the effective font id for the next keyboard input at the current cursor position. If your application uses an external toolbar, use this font id to show the current font attributes in your toolbar.

**See Also:**  
[TerGetCurFont](#)  
[GetFontInfo](#)



## TerGetFontAux1Id

**Retrieve the Aux1Id id associated with a Font id.**

```
int TerGetFontAux1Id(hWnd, FontId)
```

HWND hWnd; // The handle of the window to be accessed

int FontId; // Font id to inquire

**Return Value:** This function returns the Aux1Id associated with a font id..

**See Also:**  
[TerSetNextFontAux1Id](#)



## TerGetFontLang

**Retrieve the language id for a font id.**

```
BOOL TerGetFontLang(hWnd, FontId)
```



HWND hWnd;	// The handle of the window to be accessed
int FontId;	// FontId (0 to TotalFonts-1). Set to 0 to specify the default font.

**Return Value:** This function returns the language id for the font id. The language ids are defined by MS RTF Spec 1.5 or later.

**See Also:**  
[TerSetCharLang](#)



## TerGetFontFieldId

**Retrieve the field id associated with the given font id.**

BOOL TerGetFontFieldId(HWND, FontId)

HWND hWnd;	// The handle of the window to be accessed
int FontId;	// FontId (0 to TotalFonts-1). Set to 0 to specify the default font.

**Return Value:** This function returns the field id associated with a font id. A value of 0 indicates that the font id is not used for a field text.

**See Also:**  
[TerGetFontSpace](#)



## TerGetFontParam

**Retrieve the font parameters.**

long TerGetFontParam(HWND, FontId, type)

HWND hWnd;	// The handle of the window to be accessed
int FontId;	// FontId (0 to TotalFonts-1). Set to 0 to specify the default font.

You can also use this function to get the font information for a stylesheet item by specifying the style id as a negative value for this parameter. Also, use the following constant to specify normal or the current style item.

SID_NORMAL	'Normal' style item.
------------	----------------------

SID\_CUR

Style item being currently edited.

int type;

// One of the following parameter types can be used:

FONTINFO_UCBASE	The unicode base for the font
FONTINFO_CHARSET	The character set for the font
FONTINFO_PICT_WIDTH	The picture width in twips assuming that the FontId represents a picture.
FONTINFO_PICT_HEIGHT	The picture height in twips assuming that the FontId represents a picture.
FONTINFO_ULINE_COLOR	The color of the underline bar.
FONTINFO_AUX_ID	The auxiliary id for the font.
FONTINFO_FLAGS	Font flags. The following font flags are available:  FFLAG_AUTO_SPELL: Font with wave underline.
FONTINFO_FRAME_TYPE	The frame type for a floating picture assuming that the FontId represents a picture..  Please refer to the <a href="#">TerSetPictFrame2</a> function for a list of PFRAME_ constants.
FONTINFO_FRAME_ID	The frame id associated with a font assuming that the FontId represents a floating picture.
FONTINFO_OFFSET	The character offset (twips) from the baseline.
FONTINFO_IS_PICT	Returns 1 if the FontId represents a picture, otherwise returns 0.
FONTINFO_IS_OCX	Returns 1 if the FontId represents an ocx object, otherwise returns 0.
FONTINFO_IS_OLE	Returns 1 if the FontId represents an ole object, otherwise returns 0.
FONTINFO_FIELD	Field id for the font.
FONTINFO_INS_REV_ID	Reviewer id for inserted text.

FONTINFO_DEL_REV_ID	Reviewer id for deleted text.
FONTINFO_SIZE	Retrieve the font size in twips.
FONTINFO_STYLE	Return the font styles. Please refer to the GetFontInfo function for the definition of style constants.
FONTINFO_TEXT_COLOR	Font text color
FONTINFO_BK_COLOR	Font text background color
FONTINFO_WLINE_COLOR	The color of the wave line under the text.

**Return Value:** This function returns the value of the requested parameter. It returns -1 to indicate an error condition.



## TerGetFontSpace

**Retrieve the character spacing for a font id.**

BOOL TerGetFontSpace(hWnd, FontId)

HWND hWnd; // The handle of the window to be accessed

int FontId; // FontId (0 to TotalFonts-1). Set to 0 to specify the default font.

**Return Value:** This function returns the font space adjustment for the font id in Twips unit.

**See Also:**  
[TerSetFontSpace](#)



## TerGetFontStyleId

**Retrieve the character style id associated with a Font id.**

int TerGetFontStyleId(hWnd, FontId)

HWND hWnd;	// The handle of the window to be accessed
int FontId;	// Font id to inquire

**Return Value:** This function returns the character style id associated with a font id.

**See Also:**  
[GetFontInfo](#)  
[TerSelectCharStyle](#)  
[TerEditStyle](#)  
[TerSetFontStyleId](#)



## TerGetTextColor

**Get the foreground and background colors for the specified font id.**

BOOL TerGetTextColor(HWND hWnd, FontId, TextColor, TextBkColor)

HWND hWnd;	// The handle of the window to be accessed				
int FontId;	// The editor font id to retrieve color. A valid font id would be a value between 0 and TotalFonts - 1.  You can also use this function to get the text color for a stylesheet item by specifying the style id as a negative value for this parameter. Also, use the following constant to specify normal or the current style item:				
	<table> <tr> <td>SID_NORMAL:</td> <td>Normal style item</td> </tr> <tr> <td>SID_CUR:</td> <td>Style item being current edited</td> </tr> </table>	SID_NORMAL:	Normal style item	SID_CUR:	Style item being current edited
SID_NORMAL:	Normal style item				
SID_CUR:	Style item being current edited				
LPDWORD TextColor;	// pointer to a double word location to receive the text foreground color.				
LPDWORD TextBkColor;	// pointer to a double word location to receive the text background color.				

**Return Value:** This function returns TRUE when successful.

Example:

// This example retrieves the text foreground and background color for font id: 0.

COLORREF TextColor, TextBkColor;

TerGetTextColor(hWnd,0,&TextColor,&TextBkColor);

**See Also:**  
[SetTerColor](#)

[SetTerBkColor](#)  
[GetFontInfo](#)  
[GetTerLine](#)  
[GetTerFields](#)



## TerLocateFontId

**Locate a font id in the document.**

BOOL TerLocateFontId(hWnd,FontId,line,col)

HWND hWnd;	// Handle of the window to be accessed
int FontId;	// Font id to locate.
LPLONG line;	// Starting line number. Set to -1 to start the search from the current cursor position.
LPINT col;	// Starting column position. Set the 'line' and 'col' arguments to 0 to begin the search from the beginning of the file.

**Return Value:** This function returns a TRUE value if the font id is found in the document. It also returns the line number and column number of the font id.

**See Also:**  
[TerLocateStyle](#)



## TerLocateStyle

**Locate text with the given character style.**

BOOL TerLocateStyle(hWnd,style,StartLine, StartCol, StringLen)

HWND hWnd;	// Handle of the window to be accessed
------------	--

WORD style;	// Style bits:
-------------	----------------

BOLD:	<b>Bold</b>
ULINE:	<b>Underline</b>
ULINED:	Double Underline

ITALIC:	<i>Italic</i>
STRIKE:	Strikethrough
SUPSCR:	Superscript
SUBSCR:	Subscript
HIDDEN:	Hidden Text
PROTECT:	Protected Text
CAPS	All Caps
SCAPS	Small Caps
PICT	Picture

Use the logical OR (|) operator to specify more than one styles. The search is successful when any of the specified styles are located.

long far *StartLine:	(INPUT/OUTPUT) Specifies the pointer to the line number to start the search. On a successful search, this field contains the line number of the located text.
int far *StartCol:	(INPUT/OUTPUT) Specifies the pointer to the column number to start the search. On a successful search, this field contains the column number of the located text.
int far *StringLen:	(OUTPUT) Specifies the pointer to the integer location that receives the length of the located text. The editor matches the text up to the end of the line.

**Description:** Use this function to locate the beginning of the text with the given character styles.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**  
[TerLocateStyleChar](#)  
[TerLocateFontId](#)  
[SetTerCharStyle](#)



## TerLocateStyleChar

**Locate the character with the given style.**

BOOL TerLocateStyleChar(hWnd, style, present, StartLine, StartCol, forward)

HWND hWnd;	// Handle of the window to be accessed
UINT style;	// Style bits. See TerLocateStyle function for the detail.
BOOL present;	// TRUE to test for the presence of the given style, or FALSE to test for the absence of the given style.
long far *StartLine;	(INPUT/OUTPUT) Specifies the pointer to the line number to start the search. On a successful search, this field contains the line number of the located text.
int far *StartCol;	(INPUT/OUTPUT) Specifies the pointer to the column number to start the search. On a successful search, this field contains the column number of the located text.
BOOL forward;	// TRUE to scan the text in the forward direction, or FALSE to scan the text in the backward direction.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**  
[TerLocateStyle](#)



## TerRestrictFont

**Restrict the font size and styles.**

BOOL TerRestrictFont(hWnd, MinSize, MaxSize, RestrictStyles, UpdateToolbar)

HWND hWnd;	// The handle of the window to be accessed.
int MinSize;	// The smallest point-size of the fonts to allow
int MaxSize;	// The largest point-size of the fonts to allow
int RestrictStyles;	// Use this parameter to specify the styles not to be allowed. The styles are specified by using the style bits. Please refer to the <a href="#">SetTerCharStyle</a> method for a list of the valid style bits.
BOOL UpdateToolbar;	// TRUE to update the tool-bar immediately to reflect the changes.

**Return Value:** This function returns TRUE when successful.



## TerSelectCharStyle

**Apply a character stylesheet item.**

BOOL TerSelectCharStyle(hWnd, StyleId, repaint)

HWND hWnd; // The handle of the window to be accessed.

int StyleId; // Character style id to apply

BOOL repaint; // TRUE to refresh the screen after this operation.

**Description:** This function is used to assign the given character style to a highlighted block of text. If a text block is not highlighted, the given style id is used for the next keyboard input.

**Return Value:** This function returns TRUE when successful.

### See Also:

[TerSelectParaStyle](#)

[TerEditStyle](#)

[TerGetFontStyleId](#)



## TerSetCharAuxId

**Set the character auxiliary id.**

BOOL TerSetCharAuxId(hWnd, AuxId, repaint)

HWND hWnd; // The handle of the window to be accessed.

int AuxId; // Aux id

BOOL repaint; // Repaint the screen after this operation

**Description:** This function is used to set an auxiliary id. The editor does not use this id. If a text block is selected, the new id is applied to all characters in the block. Otherwise, the id is applied to any newly entered characters at the current cursor location.

**Return Value:** This function returns TRUE when successful.





## TerSetCharLang

**Set new language id for the selected text.**

BOOL TerSetCharLang(hWnd, lang, repaint)

HWND hWnd; // The handle of the window to be accessed.

int lang; // Language id as defined by MS RTF Spec 1.5.

BOOL repaint; // Repaint the screen after this operation

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerGetFontLang](#)  
[TerSetDefLang](#)



## TerSetCharScaleX

**Adjust horizontal character scaling.**

BOOL TerSetCharScaleX(hWnd, dialog, ScalePercent, repaint)

HWND hWnd; // The handle of the window to be accessed.

BOOL dialog; // TRUE to show the dialog box to accept the user input

int ScalePercent; // Horizontal character scaling in percentage. Set to a value greater than 100 to enlarge the character horizontally or set to a value less than 100 to shrink the character horizontally. The default value is 100.

BOOL repaint; // Repaint the screen after this operation

**Return Value:** This function returns TRUE when successful.



## TerSetCharSet

**Override the character set for font creation.**

BOOL TerSetCharSet(hWnd, NewCharSet)

HWND hWnd; // The handle of the window to be accessed.

BYTE NewCharSet; // New character set. To reset the override, set the NewCharSet to DEFAULT\_CHARSET or 1.

BOOL repaint; // Repaint the screen after this operation

**Description:** Use this function to override the character-set that the editor should use to create new fonts.

**Return Value:** This function returns TRUE when successful.



## TerSetCharSpace

**Adjust the character spacing.**

BOOL TerSetCharSpace(hWnd, dialog, delta, repaint)

HWND hWnd; // The handle of the window to be accessed.

BOOL dialog; // TRUE to show the dialog box to accept the user input

int delta; // Amount of adjustment in twips. Set to a positive value to expand the character spacing or a negative value to compress the character spacing.

BOOL repaint; // Repaint the screen after this operation

**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerSetFontSpace](#)



## TerSetCustomFontList

**Override the fonts used for the toolbar.**

BOOL TerSetCustomFontList(hWnd, FontList, repaint)

HWND hWnd; // The handle of the window to be accessed.

LPBYTE FontList;                      // A list of fonts to use. Each font typeface should be delimited by a comma characters. Example: "Arial, Times New Roman, Courier"

BOOL repaint;                      // Repaint the screen after this operation

**Description:** You can use this function to override the list of fonts shown in the toolbar combo-box.

**Return Value:** This function returns TRUE when successful.



## TerSetDefLang

**Set default language id for the document.**

BOOL TerSetDefLang(hWnd, lang)

HWND hWnd;                      // The handle of the window to be accessed.

int lang;                      // Language id as defined by MS RTF Spec 1.5.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerGetFontLang](#)  
[TerSetCharLang](#)



## TerSetDefTextColor

**Set default text foreground color.**

BOOL TerSetDefTextColor(hWnd, ForeColor, repaint)

HWND hWnd;                      // The handle of the window to be accessed.

COLORREF NewWidth;                      // new text foreground color.

BOOL repaint;                      // TRUE to repaint the screen after this operation.

**Description:** The new text color is effective only for the current session. The new text color is not written out when the document is saved.

**Please use the [GetTerFields/SetTerFields](#) functions to change the background color of the text window.**

**Return Value:** This function returns TRUE when successful..

**See Also**  
[InsertTerText](#)

## TerSetEffectiveFont

**Override the effective font for editing.**

BOOL TerSetEffectiveFont(hWnd, FontId)

HWND hWnd;                               // The handle of the window to be accessed

int FontId;                               // The font id to use for editing.

**Description:** Normally, when the user click on a text location, the editor picks a font for text entry at that location. You can override this font automatic selection by using this function. This function can also be used to set a suitable font before inserting the text using the InsertTerText function.

**Return Value:** This function returns TRUE if successful.



## TerSetFontId

**Set the object id for the next object to be inserted.**

BOOL TerSetFontId(hWnd, int NextId)

HWND hWnd;                               // The handle of the window to be accessed

int NextId;                               // The id to use by the next object to be inserted.

**Description:** Normally, the editor assigns a new id when an object (font or picture) is inserted in the text. This function can be used to utilize an existing id for the object. The object at the existing id is released before associating the new object with this id.

**Return Value:** This function returns TRUE if successful.



## TerSetFontSpace

**Adjust the character spacing for a font id.**

BOOL TerSetFontSpace(hWnd, FontId, delta, repaint)

HWND hWnd;	// The handle of the window to be accessed
int FontId;	// FontId (0 to TotalFonts-1) to modify. Set to 0 to modify the default font.
int delta;	// Amount of adjustment in twips. Set to a positive value to expand the character spacing or a negative value to compress the character spacing.
BOOL repaint;	// Repaint the screen after this operation

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerSetCharSpace](#)  
[TerSetFontSpace](#)



## TerSetFontStyleId

**Set the style ids for a font id.**

BOOL TerSetFontStyleId(hWnd, FontId, CharStyleId, ParaStyleId)

HWND hWnd;	// The handle of the window to be accessed
int FontId;	// FontId (0 to TotalFonts-1) to modify.
int CharStyleId;	// New character style id for the font. Set to -1 to leave this value unchanged.
int ParaStyleId;	// New paragraph style id for the font. Set to -1 to leave this value unchanged.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerSetFontStyleId](#)



## TerSetInitTypeface

**Set the initial font typeface.**

BOOL TerSetInitTypeface(typeface)

HWND hWnd; // The handle of the window to be accessed

LPBYTE typeface; // Initial font typeface. The editor uses this font typeface as the default font for the TerMergePrint function. It also uses this typeface if the editor window is created without specifying a font typeface, or if the window is created using the OCX control.

**Description:** To be effective, this function must be called before any editor window is created and before calling the TerMergePrint function.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

**Return Value:** This function returns TRUE when successful.



## TerSetNextFontAux1Id

**Set the next font Aux1Id to be used by the TerCreateFont function.**

BOOL TerSetNextFontAux1Id(hWnd,Aux1Id)

HWND hWnd; // The handle of the window to be accessed

int Aux1Id; // Aux1Id to set

**Description:** This function sets the value of the Aux1Id font attribute used by the TerCreateFont function. The TerCreateFont function sets this value to 0 after creating a new font id.

**Return Value:** This function returns TRUE if successful.

### See Also:

[TerCreateFont](#)

[TerGetFontAux1Id](#)



## TerSetTcField

**Set the 'tc' field to the selected text.**

BOOL TerSetTcField(hWnd, level, repaint)

HWND hWnd;	// Window handle to access
int level;	// The 'tc' field level. The level value must be between 1 and 9 inclusive.
BOOL repaint;	// TRUE to repaint the screen after this operation

**Comment:** This function can be used to set the 'tc' field to the text to be included in the table of contents. Please refer to the [TerInsetToc2](#) function for more information.

**Return Value:** This function returns a TRUE value if successful.



## TerSetTextCase

**Set the case for the selected text.**

BOOL TerSetTextCase(hWnd, upper, repaint)

HWND hWnd;	// Window handle to access
BOOL upper;	// Set to TRUE to turn the text into upper case. Set to FALSE to turn the text into lower case
BOOL repaint;	// TRUE to repaint the screen after this operation

**Return Value:** This function returns a TRUE value if successful.



## TerSetUlineColor

**Set the color to draw underline.**

BOOL TerSetUlineColor(hWnd, color, repaint)

HWND hWnd;	// The handle of the window to be accessed.
COLORREF color;	// new underline color.
BOOL repaint;	// TRUE to repaint the screen after this operation.

**Description:** This function does not automatically apply the underline attribute to the text. The SetTerCharStyle function should be used to apply the underline attribute.

**Return Value:** This function returns TRUE when successful..

**See Also**  
[SetTerCharStyle](#)



## TerSetUserFont

**Set font to use instead of the current font during RTF input or keyboard entry.**

BOOL TerSetUserFont(hWnd, typeface, TwipsSize, StyleOn, StyleOff)

HWND hWnd; // The handle of the window to be accessed

LPBYTE typeface; // typeface of the new font

int TwipsSize; // The twips size (20 twips = 1 point) to use

int StyleOn; // The style bits to turn-on. Please refer to the [SetTerCharStyle](#) method for a list of valid style bits.

int StyleOff; // The style bits to turn-off. Please refer to the [SetTerCharStyle](#) method for a list of valid style bits.

**Comment:** This method can be called from within the OverrideFont event to change the current font.

**Return Value:** This function returns TRUE if successful.

**Example:**

```
// Replace the current font selection by an Arial, 10 point, bold style. Turn-off an Italic style.
```

```
OverrideFont:
```

```
TerSetUserFont(hWnd,"Arial",10*20, BOLD, ITALIC);
```



## TerSetWaveUnderline

**Draw red wavy underline.**

BOOL TerSetWaveUnderline(hWnd, LineNo, StartCol, EndCol, set, repaint)



```

BOOL TerSetWaveUnderline2(hWnd, color, LineNo, StartCol, EndCol, set, repaint)

HWND hWnd;                // Window handle to access

COLORREF color;           // The color for the wavy underline. The default color is
                           red.

long LineNo;              // The text line number

int StartCol;             // The starting column position for the text

int EndCol;               // The ending column position for the text

BOOL set;                 // Set to TRUE to draw the underline. Set to FALSE to
                           erase the underline.

BOOL repaint;             // Set to TRUE to repaint the screen after this operation

```

**Return Value:** This function returns TRUE when successful.



## TerShrinkFontTable

**Compress the font table cache.**

```

BOOL TerShrinkFontTable()

```

Description: This function is useful if you are opening and closing a number of documents under your program's control. This function can be called after opening a new document to release the unused fonts from the font cache.

**Return Value:** This function TRUE when successful.



## Paragraph Formatting

### In This Chapter

- [ClearTab](#)
- [ClearAllTabs](#)
- [ParaHangingIndent](#)
- [ParaIndentTwips](#)
- [ParaLeftIndent](#)
- [ParaRightIndent](#)
- [ParaNormal](#)
- [SetTab](#)
- [SetTerParaFmt](#)
- [TerCreateBulletId](#)

[TerCreateListBullet](#)  
[TerCreateParald](#)  
[TerCreateTabId](#)  
[TerGetParaInfo](#)  
[TerGetParaParam](#)  
[TerGetTabStop](#)  
[TerSelectParaStyle](#)  
[TerSelectParaText](#)  
[TerSetBullet](#)  
[TerSetBulletEx](#)  
[TerSetBulletId](#)  
[TerSetDefTabWidth](#)  
[TerSetDefTabType](#)  
[TerSetParaAuxId](#)  
[TerSetParaBkColor](#)  
[TerSetParaBorderColor](#)  
[TerSetParald](#)  
[TerSetParaList](#)  
[TerSetParaShading](#)  
[TerSetParaTextFlow](#)  
[TerSetParaIndent](#)  
[TerSetParaSpacing](#)  
[TerSetPflags](#)  
[TerSetTab](#)



## ClearTab

### Clear one tab stop.

BOOL ClearTab(hWnd, TabPos, repaint)

HWND hWnd; // The handle of the window to be accessed

int TabPos; // Tab position (in twips unit) to clear

BOOL repaint; //Repaint the window after this operation

**Description:** Use this function to remove a specified tab stop for the selected text.

When a text block is highlighted before calling this function, the selected lines are affected by this function. Otherwise, only the current paragraph is affected by this operation.

**Return Value:** This function returns TRUE if successful.

### See Also

[ClearAllTabs](#)  
[SetTab](#)



## ClearAllTabs

### Clear all tab stops:

BOOL ClearAllTabs(hWnd,repaint

HWND hWnd; // The handle of the window to be accessed

BOOL repaint; //Repaint the window after this operation

**Description:** Use this function to reset all tab stops for the selected text. The tab stops are reset to their default positions.

When a text block is highlighted before calling this function, the selected lines are affected by this function. Otherwise, only the current paragraph is affected by this operation.

**Return Value:** This function returns TRUE if successful.

### See Also

[SetTab](#)

[ClearTab](#)



## ParaHangingIndent

### Increment or decrement the hanging indents.

BOOL ParaHangingIndent(hWnd, indent, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL indent; // TRUE to increment the indentation, FALSE to decrement the indentation

BOOL repaint; // TRUE to refresh the window after this operation

**Description:** Use this function to increment or decrement the hanging indentation by 1/4 of an inch.

When a text block is highlighted before calling this function, the selected lines are affected by this function. Otherwise, only the current paragraph is affected by this operation.

**Return Value:** This function returns TRUE if successful.

### See Also:

[ParaLeftIndent](#)

[ParaRightIndent](#)

[ParaNormal](#)



## ParaIndentTwips

**Increment or decrement the paragraph indentation.**

BOOL ParaIndentTwips(hWnd, DeltaLeft, DeltaRight, DeltaFirst, repaint)

HWND hWnd;	// Editor window to access
int DeltaLeft;	// amount (twips) of left indentation to apply
int DeltaRight;	// amount (twips) of right indentation to apply
int DeltaFirst;	// amount (twips) of indentation to apply to the first line only.
BOOL repaint;	// TRUE to repaint the screen after this operation.

**Description:** This function can be used to increment or decrement the paragraph indentation amount. This function allows you to affect all three indentation parameters simultaneously.

Please note that the left indentation affects the first line of the paragraph as well. To keep the first line from moving, apply the equal amount of negative indentation to the first line.  
Example:

```
ParaIndentTwips(hWnd,50,0,-50,TRUE);
```

**Return Value:** The function returns TRUE when successful.

### See Also:

[ParaLeftIndent](#)  
[ParaRightIndent](#)  
[ParaHangingIndent](#)  
[TerSetParaIndent](#)



## ParaLeftIndent

**Increment or decrement the left indents.**

BOOL ParaLeftIndent(hWnd, indent, repaint)

HWND hWnd;	// Handle of the window to be accessed
BOOL indent;	// TRUE to increment the indentation, FALSE to

decrement the indentation

BOOL repaint;                      // TRUE to refresh the window after this operation

**Description:** Use this function to increment or decrement the left indentation by 1/4 of an inch.

When a text block is highlighted before calling this function, the selected lines are affected by this function. Otherwise, only the current paragraph is affected by this operation.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[ParaHangingIndent](#)

[ParaRightIndent](#)

[ParaNormal](#)



## ParaRightIndent

**Increment or decrement the right indents.**

BOOL ParaRightIndent(hWnd, indent, repaint)

HWND hWnd;                      // The handle of the window to be accessed

BOOL indent;                    // TRUE to increment the indentation, FALSE to  
decrement the indentation

BOOL repaint;                    // TRUE to refresh the window after this operation

**Description:** Use this function to increment or decrement the right indentation by 1/4 of an inch.

When a text block is highlighted before calling this function, the selected lines are affected by this function. Otherwise, only the current paragraph is affected by this operation.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[ParaHangingIndent](#)

[ParaNormal](#)



## ParaNormal

### Reset paragraph properties

BOOL ParaNormal(hWnd, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL repaint; // TRUE to refresh the window after this operation

**Description:** Use this function to reset the paragraph properties.

When a text block is highlighted before calling this function, the selected lines are affected by this function. Otherwise, only the current paragraph is affected by this operation.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[ParaHangingIndent](#)  
[ParaRightIndent](#)  
[ParaLeftIndent](#)  
[SetTerParaFmt](#)



## SetTab

### Set a tab position:

BOOL SetTab(hWnd, TabType, TabPos, repaint)

HWND hWnd; // The handle of the window to be accessed

int TabType; // Tab type: TAB\_LEFT, TAB\_RIGHT, TAB\_CENTER, TAB\_DECIMAL

int TabPos; // Tab position (in twips unit) to create

BOOL repaint; //Repaint the window after this operation

**Description:** Use this function to create one tab position.

**Note:** This function will be eventually discontinued in favor of the TerSetTab function.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[TerSetTab](#)  
[ClearTab](#)  
[ClearAllTabs](#)



## SetTerParaFmt

### Set paragraph styles

BOOL SetTerParaFmt(hWnd, styles, OnOff, repaint)

HWND hWnd; // Handle of the window to be accessed

WORD styles: Select paragraph styles:

LEFT:	Left justified paragraph
CENTER:	Centered paragraph
RIGHT_JUSTIFY:	Right justified paragraph
JUSTIFY:	Paragraph justified on both margins
DOUBLE_SPACE:	Double spaced paragraph
PARA_KEEP:	Keep the entire paragraph on the same page.
PARA_KEEP_NEXT:	Keep the last line of the current paragraph and the first line of the next paragraph on the same page.
PARA_BOX_TOP:	Apply top paragraph border
PARA_BOX_BOT:	Apply bottom paragraph border
PARA_BOX_BETWEEN	Draw lines between selected paragraphs.
PARA_BOX_LEFT:	Apply left paragraph border
PARA_BOX_RIGHT:	Apply left paragraph border
PARA_BOX:	Apply paragraph borders to all sides (combination of above four flags)
PARA_BOX_DOUBLE:	Double line paragraph border

PARA\_BOX\_THICK:        Thick paragraph border

To specify more than one styles, use the 'logical OR' (|) operator.

BOOL OnOff;                // TRUE to set the styles, FALSE to reset the selected styles.

BOOL repaint;             // TRUE to refresh the window after this operation

**Description:** Use this function to set or reset the specified styles.

When a text block is highlighted before calling this function, the selected lines are affected by this function. Otherwise, only the current paragraph is affected by this operation.

**Return Value:** This function returns TRUE if successful.

**Example:**

```
// center the current paragraph
SetTerParaFmt (hWnd, CENTER, TRUE, TRUE) ;
```

**See Also:**  
[ParaNormal](#)  
[TerSetFlags](#)



## TerCreateBulletId

**Create a paragraph bullet/numbering id.**

int TerCreateBulletId(hWnd, IsBullet, start, level, type)

int TerCreateBulletId2(hWnd, IsBullet, start, level, type, TextBef, TextAft)

int TerCreateBulletId3(hWnd, IsBullet, start, level, type, TextBef, TextAft, flags)

HWND hWnd;                // Handle of the window to be accessed

BOOL IsBullet;            // TRUE to set the paragraph bullet or FALSE to set paragraph numbering.

int start;                // The starting number when setting paragraph numbering. Set to 1 for default.

int level;                // The level number when setting paragraph numbering. Set to 0 for default.

int type;                // The parameter indicates the symbol used for bullets or the letters used for paragraph numbering. Please refer to



the TerSetBulletEx function for the constant symbols used for this parameter.

LPBYTE TextBef;	// Text before the paragraph number (limited to 10 bytes). Set to NULL for default.
LPBYTE TextAft;	// Text After the paragraph number (limited to one byte). Set to NULL for default.
UINT flags;	// Set this argument to BLTFLAG_HIDDEN to create a continuing list item. Set to 0 to create regular list item..

**Return Value:** This function returns the bullet id.

**See Also:**  
[TerSetBulletEx](#)  
[TerSetBulletId](#)  
[TerCreateListBullet](#)



## TerCreateListBullet

**Create a bullet id using the list mechanism.**

**int TerCreateListBullet(hWnd, ListOr, level, repaint)**

HWND hWnd;	// The handle of the window to be accessed
int ListOr;	// The list-override id to create the bullet.
int level;	// The level number to create the bullet. A simple list allows only one list level (level 0). A nested list allows up to 9 levels (0 to 8).

**Return Value:** This function returns a non-zero bullet id when successful. A value of 0 indicates an error condition.

**See Also:**  
[TerCreateParald](#)  
[TerCreateBulletId](#)



## TerCreateParald

**Create a paragraph id.**

int TerCreateParald(hWnd, Reuseld, shared, LeftIndent, RightIndent, FirstIndent, TabId, StyleId, AuxId, Shading, pflags, SpaceBefore, SpaceAfter, SpaceBetween, flags);

```

int TerCreateParalEx(hWnd, Reuseld, shared, LeftIndent, RightIndent, FirstIndent,
Tabld, Styleld, Auxld, Shading, pflags, SpaceBefore, SpaceAfter, SpaceBetween, flags,
Bltd, BkColor);

HWND hWnd;           // The handle of the window to be accessed

int Reuseld;          // Existing paragraph id to modify with new information.
                      // Use -1 to create a new paragraph id.

BOOL shared;          // When TRUE, the editor matches the requested
                      // specification against the existing paragraph ids. If a
                      // matching paragraph id is found, it returns that id.
                      // Otherwise it creates a new id. A TRUE value for this field
                      // is mutually exclusive with a zero or positive value for the
                      // Reuseld field.

int LeftIndent;        // Left indentation (specified in twips). Use 0 for default.

int RightIndent;       // Right indentation (specified in twips). Use 0 for default.

int FirstIndent;       // Indentation for the first line (specified in twips). Use 0
                      // for default.

int Tabld;             // Tab id. Use 0 for default.

int Styleld;           // Paragraph Style id. Use 0 for default. When a non-zero
                      // style id is specified, other parameters values to the
                      // function must be what is indicated by this style id.

int Auxld;             // An application specified id. The editor does not use this
                      // id internally. Use 0 for default.

UINT shading;          // Shading amount Specify a value from 0 (no shading) to
                      // 10000 (darkest shading).

UINT pflags;           // Additional paragraph flags reserved for future use. Use
                      // 0 for default.

int SpaceBefore;       // Space before the paragraph (specified in twips). Use 0
                      // for default.

int SpaceAfter;        // Space after the paragraph (specified in twips). Use 0
                      // for default.

int SpaceBetween;      // Space between the paragraph lines (specified in twips).
                      // Use 0 for default.

UINT flags;            // Paragraph attribute flags. Please refer to the
                      // 'SetTerParaFmt' function for a list of paragraph attribute
                      // ids. Use 0 for default.

```

When creating a paragraph id for use inside a page header, the 'flags' parameter must be ORed with the PAGE\_HDR constant. Similarly, when creating a paragraph id for use inside a page footer, the 'flags' parameter must be ORed with the PAGE\_FTR constant. A paragraph id for use in the regular text must not have either of these constants.

int BltId;	// The bullet id. When a non-zero bullet id is specified, the BULLET flag must also be specified in the 'flags' parameter. Set to 0 for default.
COLORREF BkColor:	Paragraph background color. Set to Hex FFFFFFFF (white) for default.

**Description:** This function is used to create a new paragraph id or to modify an exiting id with new paragraph information. To modify an existing id, specify the old paragraph id using the 'Reuseld' argument, otherwise set the 'Reuseld' parameter to -1. When an existing id is modified, this function automatically updates the text which uses this id with new information.

**Return Value:** When successful, this function returns the id of the new paragraph. Otherwise it returns -1.

**See Also:**  
[TerCreateFont](#)  
[TerAppendText](#)  
[TerGetParaInfo](#)  
[TerSetParalId](#)  
[TerCreateTabId](#)  
[TerCreateBulletId](#)  
[TerCreateListBullet](#)



## TerCreateTabId

**Create a tab id.**

**int TerCreateTabId(hWnd, pTabInfo)**

HWND hWnd;	// The handle of the window to be accessed
struct StrTab far *pTabInfo;	// This structure is used to pass the tab stop information for the new tab id. struct StrTab {
int count;	// number of tab stop (max 20)
int pos[20];	// tab position for each tab stop in twips. The tab positions must be specified in the ascending order
int type[20];	// The tab type for each tab stop:

TAB_LEFT:	Left tab
TAB_RIGHT:	Right tab
TAB_CENTER:	Center tab
TAB_DECIMAL:	Decimal tab

BYTE flags[20];                   // The tab flags for each tab stop:

TAB_NONE	Tab with no leaders (default)
TAB_DOT:	Tab with dot leaders
TAB_HYPH:	Tab with hyphen leaders
TAB_ULINE:	Tab with underline leader

**Description:** The tab id created by this function can be used in the TerCreateParald function.

**Return Value:** This function returns a non-zero tab id if successful, otherwise it returns -1.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

**See Also:**  
[TerCreateParald](#)



## TerGetParalInfo

### Get paragraph attributes.

BOOL TerGetParalInfo(hWnd, LineNo, LeftIndent, RightIndent, FirstIndent, TabId, StyleId, AuxId, Shading, pflags, SpaceBefore, SpaceAfter, SpaceBetween, flags);

BOOL TerGetParalInfo2(hWnd, LineNo, LeftIndent, RightIndent, FirstIndent, TabId, StyleId, AuxId, Shading, pflags, SpaceBefore, SpaceAfter, SpaceBetween, flags, Aux1Id, BkColor);

BOOL TerGetParalInfo3(hWnd, LineNo, IsStyleItem, LeftIndent, RightIndent, FirstIndent, TabId, StyleId, AuxId, Shading, pflags, SpaceBefore, SpaceAfter, SpaceBetween, flags, Aux1Id, BkColor);

BOOL TerGetParalInfo4(hWnd, LineNo, IsStyleItem, LeftIndent, RightIndent, FirstIndent, TabId, StyleId, AuxId, Shading, pflags, SpaceBefore, SpaceAfter, SpaceBetween, flags, Aux1Id, BkColor, LineSpacing);

HWND hWnd;                       // The handle of the window to be accessed

long LineNo;	// Line number to retrieve the paragraph information. To specify a para id instead of a line number, specify a negative value.
BOOL IsStyleItem;	This parameter is applicable to the TerGetParaInfo3 function only. It allows you to retrieve the paragraph information for a style item. When this flag is set to TRUE, the LineNo parameter should be used to pass a style id to retrieve its information. You can also set the LineNo to SID_CUR to get the paragraph information about the style being currently edited.
LPINT LeftIndent;	// Left indentation (in twips).
LPINT RightIndent;	// Right indentation (in twips).
LPINT FirstIndent;	// Indentation for the first line (in twips).
LPINT TabId;	// Tab id.
LPINT StyleId;	// Paragraph Style id.
LPINT AuxId;	// An application specified id.
LPINT shading;	// Shading amount: a value from 0 (no shading) to 10000 (darkest shading).
UINT far *pflags;	// Additional paragraph flags:
	PFLAG_WIDOW: Widow/orphan control
	PFLAG_PAGE_BREAK: Page break before the paragraph
LPINT SpaceBefore;	// Space before the paragraph (in twips).
LPINT SpaceAfter;	// Space after the paragraph (in twips).
LPINT SpaceBetween;	// Space between the paragraph lines (in twips).
UINT far *flags;	// Paragraph attribute flags. Please refer to the 'SetTerParaFmt' function for a list of paragraph attribute ids.
DWORD far *Aux1Id;	// Another application specified id.
COLORREF far *BkColor;	// Paragraph background color.
LPINT LineSpacing;	// Extra line spacing in percentage

**Return Value:** This function returns a TRUE value when successful. Otherwise it returns FALSE.

See Also:

[TerCreateParald](#)  
[TerGetParaParam](#)

TerGetParaParam

Get additional paragraph properties

int TerGetParaParam(hWnd, LineNo, IsStyleItem, type)

HWND hWnd;	// The handle of the window to be accessed	
long LineNo;	// Line number to retrieve the paragraph information. To specify a para id instead of a line number, specify a negative value.	
BOOL IsStyleItem;	This parameter allows you to retrieve the paragraph information for a style item. When this flag is set to TRUE, the LineNo parameter should be used to pass a style id to retrieve its information. You can also set the LineNo to SID_CUR to get the paragraph information about the style being currently edited.	
int type	// Parameter type to retrieve.	
	PARAINFO_TEXT_FLOW	Paragraph text flow. Please refer to the <a href="#">TerSetParaTextFlow</a> function for the list of text flow constants.
	PARAINFO_BORDER_COLOR	Paragraph border color.
	PARAINFO_BK_COLOR	Paragraph background color.

**Return Value:** This function returns the value of the requested parameter. It returns PARAINFO\_ERROR to indicate an error condition.

See Also

[TerGetParaInfo](#)



TerGetTabStop

**Return the parameters for a tab stop and the number of tab stop for a line.**

int TerGetTabStop(hWnd, LineNo, TabNo, pPos, pType, pFlag)

int TerGetTabStop2(hWnd, type, LineNo, TabNo, pPos, pType, pFlag)

HWND hWnd;	// The handle of the window to be accessed
int type;	// This argument specifies the meaning of the 'LineNo' argument:
PID_LINE:	The 'LineNo' parameter specifies the text line number.
PID_PARA:	The 'LineNo' parameter specifies the paragraph id.
PID_TAB:	The 'LineNo' parameter specifies the tab id.
long LineNo;	// Line number (or paragraph or tab id) to get the tab parameters for. Set to -1 to get the tab stop parameters for the current line.
int TabNo;	// Tab number to inquire. To simply get the tab count for the line, set the TabNo parameter to -1.
LPINT pPos;	// The pointer to receive the tab position (in twips).
LPINT pType;	// The pointer to receive the tab type:
TAB_LEFT:	Left aligned tab
TAB_RIGHT:	Right aligned tab
TAB_CENTER:	Center aligned tab
TAB_DECIMAL:	Decimal aligned tab
LPINT pFlag;	// The pointer to receive the tab flag:
TAB_DOT:	Dot leader
TAB_HYPH:	Hyphen leader
TAB_ULINE:	Underline leader

**Return Value:** This function returns the number of tab stops for the line.

**See Also:**

[TerPosTable](#)



## TerSelectParaStyle

**Apply a paragraph stylesheet item.**

BOOL TerSelectParaStyle(hWnd, StyleId, repaint)

HWND hWnd; // The handle of the window to be accessed.

int StyleId; // Paragraph style id to apply

BOOL repaint; // TRUE to refresh the screen after this operation.

**Description:** This function is used to assign the given paragraph style to the current paragraph. If more than one paragraph is highlighted, then all highlighted paragraphs are assigned the specified paragraph style id.

**Return Value:** This function returns TRUE when successful.

### See Also:

[TerSelectCharStyle](#)

[TerEditStyle](#)

[TerGetFontStyleId](#)

## TerSelectParaText

**Select entire text in the current paragraph.**

BOOL TerSelectParaText(hWnd, repaint)

HWND hWnd; // The handle of the window to be accessed.

BOOL repaint; // TRUE to refresh the screen after this operation.

**Return Value:** This function returns TRUE when successful.



## TerSetBullet

**Set the paragraph bullet property.**

BOOL TerSetBullet(hWnd, set, repaint)



```

HWND hWnd;           // The handle of the window to be accessed.

BOOL set;             // TRUE to set the paragraph bullet or FALSE to remove
                     // it.

BOOL repaint;         // Repaint the screen after this operation

```

**Return Value:** This function returns TRUE when successful.



## TerSetBulletEx

**Set the paragraph bullet/numbering property.**

```

BOOL TerSetBulletEx(HWND hWnd, set, IsBullet, start, level, type, repaint)
BOOL TerSetBullet2(HWND hWnd, set, IsBullet, start, level, type, TextBef, TextAft, repaint)
BOOL TerSetBullet3(HWND hWnd, set, IsBullet, start, level, type, TextBef, TextAft, repaint,
flags)

```

```

HWND hWnd;           // The handle of the window to be accessed.

BOOL set;             // TRUE to set the paragraph bullet/numbering or FALSE
                     // to remove it. The 'start', 'level', and 'type' parameters are
                     // ignored when the 'set' parameter is FALSE.

BOOL IsBullet;        // TRUE to set the paragraph bullet or FALSE to set
                     // paragraph numbering.

int start;            // The starting number when setting paragraph
                     // numbering. Set to 1 for default.

int level;            // The level number when setting paragraph numbering.
                     // Set to 0 for default.

int type;             // The parameter indicates the symbol used for bullets or
                     // the letters used for paragraph numbering.

```

When the 'IsBullet' parameter is set to TRUE, use one of the following constants for the 'type' parameter:

```

BLT_ROUND:           Round bullet
BLT_DIAMOND:         Diamond bullet
BLT_SQUARE:          Square bullet
BLT_HOLLOW_SQUARE:   Hollow square bullet
BLT_4_DIAMONDS:      Four diamonds symbol
BLT_ARROW:           Arrow bullet

```

BLT\_CHECK:            Check bullet

When the 'IsBullet' parameter is set to FALSE, use one of the following constants for the 'type' parameter:

NBR\_DEC:            Decimal number

NBR\_UPR\_ALPHA    Uppercase Alphabetic number

NBR\_LWR\_ALPHA    Lowercase Alphabetic number

NBR\_UPR\_ROMAN    Uppercase Roman number

NBR\_LWR\_ROMAN    Lowercase Roman number

LPBYTE TextBef;            // Text before the paragraph number (limited to 10 bytes).  
Set to NULL for default.

LPBYTE TextAft;            // Text After the paragraph number (limited to one byte).  
Set to NULL for default.

UINT flags;                // Set this argument to BLTFLAG\_HIDDEN to create a  
continuing list item. Set to 0 to create regular list item

BOOL repaint;              // Repaint the screen after this operation

**Comment:** This method uses the older method of applying bullet and numbering. You can use the [TerSetListBullet](#) function to use the newer list mechanism to apply bullets and numbers. The new function has a better support for nested lists and multiple lists within a document.

**Return Value:** This function returns TRUE when successful.



## TerSetBulletId

**Assign a bullet id to a paragraph id.**

BOOL TerSetBulletId(hWnd, BulletId, ParaId)

HWND hWnd;                // The handle of the window to be accessed.

int BulletId;              // Bullet id to be assigned to the paragraph id

int ParaId;                // Paragraph id

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerCreateBulletId](#)

[TerCreateParald](#)



## TerSetDefTabWidth

**Set default tab width.**

```
int TerSetDefTabWidth(hWnd, NewWidth, repaint)
```

HWND hWnd; // The handle of the window to be accessed.

int NewWidth; // new tab width in twips

BOOL repaint; // TRUE to repaint the screen after this operation.

**Return Value:** This function returns the previous value of the tab width in twips.

**See Also**

[TerSetDefTabType](#)



## TerSetDefTabType

**Set the default tab type.**

```
BOOL TerSetDefTabType(hWnd, TabType)
```

HWND hWnd; // The handle of the window to be accessed

int TabType; // Tab type: TAB\_LEFT, TAB\_RIGHT, TAB\_CENTER, TAB\_DECIMAL

**Description:** This function allows you to set the tab type for the left-mouse click on the ruler.

**Return Value:** This function returns TRUE if successful

**See Also**

[TerSetDefTabWidth](#)



## TerSetParaAuxId

**Set the Auxiliary id for the paragraph.**

BOOL TerSetParaAuxId(hWnd, FirstLine, LastLine, AuxId)

HWND hWnd;	// The handle of the window to be accessed
long FirstLine;	// The first line of the paragraph. Set this parameter to -1 to select the current paragraph or all paragraphs in the range of selected text (if any text selected)
long LastLine;	// The last line for the paragraph. This argument is not used when 'FirstLine' is set to -1.
BOOL AuxId;	// The new auxiliary id for the paragraph.

**Return Value:** This function returns TRUE when successful.



## TerSetParaBkColor

**Set the background color for the paragraph.**

BOOL TerSetParaBkColor(hWnd, dialog, color, repaint)

HWND hWnd;	// The handle of the window to be accessed
BOOL dialog;	// TRUE to show the dialog box for the user to select a background color. FALSE to use the background color specified by the 'color' parameter.
COLORREF color;	// New background color for the paragraph.
BOOL repaint;	// TRUE to repaint the screen after this operation.

**Return Value:** This function returns TRUE when successful



## TerSetParaBorderColor

**Set the border color for the paragraph.**

BOOL TerSetParaBorderColor(hWnd, color, repaint)

HWND hWnd;	// The handle of the window to be accessed
------------	--

COLORREF color;                      // New border color for the paragraph. This color is effective only if the paragraph borders are enabled. You can enable paragraph borders using the SetTerParaFmt function.

BOOL repaint;                      // TRUE to repaint the screen after this operation.

**Return Value:** This function returns TRUE when successful

**See Also**  
[SetTerParaFmt](#)



## TerSetParald

**Set the paragraph id for the paragraph.**

BOOL TerSetParald(hWnd, FirstLine, LastLine, Parald)

HWND hWnd;                      // The handle of the window to be accessed

long FirstLine;                      // The first line of the paragraph. Set this parameter to -1 to select the current paragraph or all paragraphs in the range of selected text (if any text selected).

long LastLine;                      // The last line for the paragraph. This argument is not used when 'FirstLine' is set to -1.

BOOL Parald;                      // The new paragraph id for the paragraph.

**Return Value:** This function returns TRUE when successful.

**See Also;**  
[TerCreateParald](#)



## TerSetParaList

**Set list numbering for the paragraph.**

BOOL TerSetParaList(hWnd, dialog, Parald, ListOr, level, repaint)

HWND hWnd;                      // The handle of the window to be accessed

BOOL dialog;                      // Set to TRUE to show a dialog box to the user to select list-override and level information.

BOOL Parald;	// The paragraph id to modify. Set to -1 to apply list numbering to the current paragraph, currently selected paragraph, or to the stylesheet item currently being edited.
int ListOr;	// The list-override id to use for the paragraph.
int level;	// The level number to use for the paragraph. A simple list allows only one list level (level 0). A nested list allows up to 9 levels (0 to 8).
BOOL repaint;	// Set to TRUE to repaint the screen after this operation.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerCreateParald](#)  
[TerEditList](#)  
[TerEditListOr](#)  
[TerCreateListBullet](#)  
[TerSetListBullet](#)



## TerSetParaShading

**Set the shading value for the current paragraph.**

BOOL TerSetParaShading(hWnd, shading, refresh)

HWND hWnd;	// The handle of the window to be accessed
int shading;	// shading amount (0 to 10000)
BOOL refresh;	// TRUE to refresh the window after this operation.

**Description:** This function is used to specify the shading amount for the current paragraph or the range of selected paragraphs. The shading value of 10000 indicates the darkest shading, whereas the shading value 0 indicates no shading.

**Return Value:** This function returns TRUE if successful.



## TerSetParaTextFlow

**Set the right-to-left/left-to-right text flow option for the paragraph.**

BOOL TerSetParaTextFlow(hWnd, dialog, TextFlow, refresh)

HWND hWnd; // The handle of the window to be accessed

BOOL dialog; // Set to TRUE to show the user dialog.

int TextFlow; // The text flow constant can be one of the following:

FLOW\_LTR: Left-to-right text flow

FLOW\_RTL Right-to-left text flow

FLOW\_DEF Default text flow. The flow will be determined by the document, section or table level text flow specification.

BOOL refresh; // TRUE to refresh the window after this operation.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[TerSetDocTextFlow](#)

[TerSetSectTextFlow](#)

[TerSetRowTextFlow](#)



## TerSetParaIndent

**Set the paragraph indentation.**

BOOL TerSetParaIndent(hWnd, left, right, first, repaint)

HWND hWnd; // Editor window to access

int left; // The left indentation in twips. Set this value to -1 to leave it unchanged

int right; // The right indentation in twips. Set this value to -1 to leave it unchanged.

int first; // The indentation adjustment (twips) to apply to the first line only. Set this value to -1 to leave it unchanged.

BOOL repaint; // TRUE to repaint the screen after this operation.

**Return Value:** The function returns TRUE when successful.

**See Also:**  
[ParaIndentTwips](#)  
[ParaLeftIndent](#)  
[ParaRightIndent](#)  
[ParaHangingIndent](#)



## TerSetParaSpacing

**Set the spacing parameters for the current paragraph.**

BOOL TerSetParaSpacing(hWnd, SpaceBefore, SpaceAfter, SpaceBetween, refresh)

BOOL TerSetParaSpacing2(hWnd, SpaceBefore, SpaceAfter, SpaceBetween,  
LineSpacing, refresh)

HWND hWnd;	// The handle of the window to be accessed
int SpaceBefore;	// Space before the first line of the paragraph in twips. Set to -1 to leave the previous value unchanged.
int SpaceAfter;	// Space after the last line of the paragraph in twips. Set to -1 to leave the previous value unchanged.
int SpaceBetween;	// Minimum space between the lines of the paragraph. To set the exact spacing, specify a negative value. To set a minimum line spacing, specify a positive value. Set to -9999 to leave the previous value unchanged.
int LineSpacing;	// Applicable to TerSetParaSpacing2 function only. You can use this argument to specify the extra line space in percentage of the current line height. For example, set to 50 to specify 1.5 line spacing, or 100 to specify double line spacing. Set to 0 for default. Set to -1 to leave the previous value unchanged.
BOOL refresh;	// TRUE to refresh the window after this operation.

**Description:** This function is used to specify the paragraph spacing parameters. Use zero to specify the default value for any parameter.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[ParaIndentTwips](#)  
[ParaLeftIndent](#)  
[ParaRightIndent](#)  
[ParaHangingIndent](#)





## TerSetPflags

### Set additional paragraph flags

BOOL TerSetPflags(hWnd, flags, OnOff, repaint)

HWND hWnd;	// Handle of the window to be accessed
WORD flags;	Select paragraph flags: PFLAG_NO_WRAP: Disable word wrapping PFLAG_WIDOW: Set Widow/Orphan control To specify more than one styles, use the 'logical OR' ( ) operator.
BOOL OnOff;	// TRUE to set the flags, FALSE to reset the selected flags.
BOOL repaint;	// TRUE to refresh the window after this operation

**Return Value:** This function returns TRUE if successful.

### See Also:

[ParaNormal](#)  
[SetTerParaFmt](#)



## TerSetTab

### Set a tab position:

BOOL TerSetTab(hWnd, TabType, TabPos, TabLeader, repaint)

HWND hWnd;	// The handle of the window to be accessed
int TabType;	// Tab type: TAB_LEFT, TAB_RIGHT, TAB_CENTER, TAB_DECIMAL
int TabPos;	// Tab position (in twips unit) to create
BYTE TabLeader;	// Tab leader type: TAB_NONE: No tab leader TAB_DOT: Dotted line tab leader TAB_HYPH: Hyphen line tab leader

TAB\_ULINE: Underline tab leader

```
BOOL repaint;           //Repaint the window after this
                           operation
```

**Description:** Use this function to create one tab position.

**Return Value:** This function returns TRUE if successfulSee Also: ClearTab, ClearAllTabs

**See Also:**  
[ClearTab](#)  
[ClearAllTabs](#)



## Section Formatting

### In This Chapter

[TerColBreak](#)  
[TerGetMarginEx](#)  
[TerGetSectBins](#)  
[TerGetSectColWidth](#)  
[TerGetSectInfo](#)  
[TerGetSectParam](#)  
[TerGetPageOrient](#)  
[TerGetSeqSect](#)  
[TerSectBreak](#)  
[TerSetMargin](#)  
[TerSetPaper](#)  
[TerSetSect](#)  
[TerSetSectBorder](#)  
[TerSetSectColWidth](#)  
[TerSetSectLineNbr](#)  
[TerSetSectOrient](#)  
[TerSetSectPageSize](#)  
[TerSetSectParam](#)  
[TerSetSectTextFlow](#)



## TerColBreak

**Create a column break.**

```
BOOL TerColBreak(hWnd,repaint)
```

```
HWND hWnd;           // Handle of the window to be accessed
```

```
BOOL repaint;         //Repaint the window after this operation
```

**Description:** This function is used to place the following text on the new column. This

function is valid only when editing in the 'Print View' and 'Page' modes. Further, this function is valid only for sections containing multiple columns. Please note that a column break can not be created inside an object such as table, frame, text box, etc.

A column break is indicated by a line containing a 'dot and dash' pattern.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerPageBreak](#)  
[TerSectBreak](#)



## TerGetMarginEx

**Retrieve the margin values for a section in the document.**

int TerGetMarginEx(hWnd,sect,left,right,top,bottom,header,footer)

HWND hWnd;	// The handle of the window to be accessed
int sect;	// The section id to modify. This parameter can assume a value between 0 and 'TotalSects-1'. It can also be set to SECT_CUR to specify the current section.
LPINT left;	// The location to retrieve the left margin value in twip units.
LPINT right;	// The location to retrieve the right margin value in twip units.
LPINT top;	// The location to retrieve the top margin value in twip units.
LPINT bottom;	// The location to retrieve the bottom margin value in twip units.
LPINT header;	// The location to retrieve the distance of the header text from the top of the page.
LPINT footer;	// The location to retrieve the distance of the footer text from the bottom of the page.

**Comment:** Any of the location variables can be set to NULL, in which case the editor ignores the argument.

**Return Value:** This function returns the total number of sections in the document if successful. Otherwise it returns 0.

**See Also:**  
[TerSetMargin](#)



## TerGetSectBins

**Retrieve the paper bins used by a section.**

BOOL TerGetSectBins(hWnd, sect, FirstPageBin, NextPageBin)

HWND hWnd;	// Window handle to access.
int sect;	// The section id to retrieve information. This parameter can assume a value between 0 and 'TotalSects-1'. It can also be set to SECT_CUR to specify the current section.
LPINT FirstPageBin;	// The variable to retrieve the first page bin.
LPINT NextPageBin;	// The variable to retrieve the next page bin.

**Return Value:** This function returns true when successful.

### See Also

[TerGetSectInfo](#)



## TerGetSectColWidth

**Retrieve the column width or inter-column spacing for a variable width column section.**

int TerGetSectColWidth(hWnd, sect, col, GetColWidth)

HWND hWnd;	// The handle of the window to be accessed
int sect;	// Section id to access. You can also set this parameter to -1 to specify the current column.
int col;	// The column number (0 to total columns -1) to retrieve the the width or column space values.
BOOL GetColWidth;	// Set to TRUE to return the column width for the specified column. Set to FALSE to return the space after the specified column.

**Return Value:** This function returns the column width or the space after the column for the specified column. The value is returned in twips unit. The function return -1 if an error

occurs.



## TerGetSectInfo

**Retrieve the current section parameters.**

BOOL TerGetSectInfo(hWnd, NumCols, ColSpace, NewPage, FirstPageNo)

HWND hWnd;	// The handle of the window to be accessed
LPINT NumCols;	// The pointer to receive the number of columns for the section
LPINT ColSpace;	// The pointer to receive the space between the columns in Twips.
LPINT NewPage;	// This pointer receives 1 if the section starts on a new page, otherwise it receives a 0.
LPINT FirstPageNo;	// This pointer receives 0 if this section uses continuous page numbering, otherwise it receives the page number of the first page for this section.

**Return Value:** The function returns TRUE when successful.

**See Also:**  
[TerSetSect](#)  
[TerGetSectBins](#)



## TerGetSectParam

**Get the section parameters.**

int TerGetSectParam(hWnd, id, type)

HWND hWnd;	// The handle of the window to be accessed
int id;	// Section id (0 to Total Sections-1) to retrieve parameters.
int type;	// The parameter to retrieve:
SP_FLAGS:	Returns the flags bits associated with the section id. Please use the AND parameter with

the return value to check if one of the following flags is applicable:

SECT_NEW_PAGE:	Start the section on a new page.
SECT_RESTART_PAGE_NO	Restart page numbers.
SECT_VALIGN_CTR	Vertically center align the page text.
SECT_VALIGN_BOT	Vertically bottom align the page text.
SECT_LINE	Section uses line numbering.
SECT_SNAP_LINE_GRID ID	Text within the section is aligned to a grid.

SP\_GUTTER\_MARG      Gutter margin in twips.

SP\_LINE\_STEP:      Returns the steps in which the line numbering is applied when the SECT\_LINE flag (see SP\_FLAGS) is applied. A value of 0 or 1 indicates continuous line numbering.

**Return Value:** The function returns the value for the requested parameter. It returns FP\_ERROR to indicate an error condition.



## TerGetPageOrient

**Get the page orientation and dimensions.**

int TerGetPageOrient(hWnd, PageNum)

int TerGetPageOrientEx(hWnd, PageNum, pWidth, pHeight)

int TerGetPageOrient2(hWnd, PageNum, pWidth, pHeight, pHiddenX, pHiddenY)

HWND hWnd;      // The handle of the window to be accessed

int PageNum;	// Page number between 0 and TotalPages-1
LPINT pWidth;	// Pointer to receive the page width (in twips) after considering the orientation.
LPINT pHeight;	// Pointer to receive the page height (in twips) after considering the orientation.
LPINT pHiddenX;	// Pointer to receive the printer hidden area (in twips) in the x direction.
LPINT pHiddenY;	// Pointer to receive the printer hidden area (in twips) in the y direction.

**Description:** This function is available in the PageMode or PrintView mode only.

**Return Value:** This function returns the following constants to indicate the page orientation:

Portrait:	DMORIENT_PORTRAIT (value 1)
Landscape:	DMORIENT_LANDSCAPE (value 2)



## TerGetSeqSect

**Translate a section id into the sequential section number.**

```
int TerGetSeqSect(hWnd, SectId)
```

HWND hWnd;	// The handle of the window to be accessed
------------	--

int SectId;	// Section id for the text.
-------------	-----------------------------

Description: This function translates the section id into the sequential section numbers. Please note that the section id assigned to the text are not sequential. For example, in a document containing 3 sections, it is not correct to assume id 0 for the first section, or id 1 for the subsequent section of the document. Most APIs involving section need you to specify the section id. A section id can be retrieved using the GetTerFields or TerGetPageSect functions. However, certain functions such as TerPosBodyText and TerPosHdrFtr need the sequential section number for the section argument. For the purpose of differentiation, this manual uses the term 'section id' or 'sequential section number' as appropriate.

**Return Value:** The function returns the sequential section number when successful. Otherwise it returns -1.

**See Also:**

[TerPosBodyText](#)  
[TerPosHdrFtr](#)  
[TerGetPageSect](#)



## TerSectBreak

**Create a new section.**

BOOL TerSectBreak(hWnd,repaint)

HWND hWnd; // The handle of the window to be accessed.

BOOL repaint; //Repaint the window after this operation

**Description:** This function is used to place the following text on the new section. The section break is created before the current line. If you have enabled the editing of header/footer text, please turn it off before calling this function. Please note that a section break can not be created inside an object such as table, frame, text box, etc.

A section break is indicated by a double solid line.

**Return Value:** This function returns TRUE if successful.

**See Also:**

[TerPageBreak](#)  
[TerColBreak](#)



## TerSetMargin

**Set the margin values for the sections in the document.**

BOOL TerSetMargin(hWnd,left,right,top,bottom,repaint)

BOOL TerSetMarginEx(hWnd,sect,left,right,top,bottom,header,footer,repaint)

HWND hWnd; // The handle of the window to be accessed

int sect; // The section id to modify. This parameter can assume a value between 0 and 'TotalSects-1'. It can also be set to SECT\_CUR to modify the current section only, or it can be set to SECT\_ALL to modify all sections in the current document. The TerSetMargin function implicitly uses a value of SECT\_CUR for this parameter

int left; // The left margin value in twip units. Set to -1 to leave this parameter unchanged.



int right;	// The right margin value in twip units. Set to -1 to leave this parameter unchanged.
int top;	// The top margin value in twip units. Set to -1 to leave this parameter unchanged.
int bottom;	// The bottom margin value in twip units. Set to -1 to leave this parameter unchanged.
int header;	// The distance of the header text from the top of the page. Set to -1 to leave this parameter unchanged.
int footer;	// The distance of the footer text from the bottom of the page. Set to -1 to leave this parameter unchanged.
BOOL repaint;	// set to TRUE to repaint the screen after this operation

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerGetMarginEx](#)



## TerSetPaper

### Set custom paper size and orientation.

BOOL TerSetPaper(hWnd, size, orient, refresh)

BOOL TerSetPaperEx(hWnd, size, width, length, orient, refresh)

HWND hWnd;	// The handle of the window to be accessed
int size;	// The paper size is given by one of the Windows SDK DMPAPER_ constant. You can also set this parameter to 0 to use the 'width' and 'length' parameters to specify the paper size (TerSetPaperEx only).
int width;	// Custom paper width. This parameter is valid only for the printers which support custom paper sizes.
int length;	// Custom paper length. This parameter is valid only for the printers which support custom paper sizes.
int orient;	// The paper orientation is given by one of the Windows SDK DMORIENT_ constant. Set this argument to -1 to leave the current document orientation unchanged

BOOL refresh;                               // TRUE to refresh the window after this operation.

**Return Value:** This function returns TRUE when successful.



## TerSetSect

**Set section parameters.**

BOOL TerSetSect(hWnd, NumCols, ColSpace, NewPage)

BOOL TerSetSectEx(hWnd, NumCols, ColSpace, NewPage, FirstPageNo)

BOOL TerSetSect2(hWnd, NumCols, ColSpace, NewPage, FirstPageNo, FirstPageBin, NextPageBin)

BOOL TerSetSect2(hWnd, NumCols, ColSpace, NewPage, FirstPageNo, FirstPageBin, NextPageBin, SectId)

HWND hWnd;                               // Window handle to access

int NumCols;                            // number of columns for the section. Set to 0 to show a user dialog. Set to -1 to leave this value unchanged.

int ColSpace;                          // Space between the columns in Twips.

BOOL NewPage;                          // TRUE to begin the section on a new page.

int FirstPageNo;                      // Page number for the first page of the current section. Set to 0 to use default page numbering.

int FirstPageBin;                      / Bin selection to print the first page of this section. Set to -1 to leave this value unchanged. The bin selection constants are defined by Windows SDK as DMBIN\_\*\*\* constants.

int NextPageBin;                      // Bin selection to print the subsequent pages of this section. Set to -1 to leave this value unchanged.

int SectId;                            // Section id (0 to TotalSect-1). Set to -1 to use the current section.

**Return Value:** The function returns TRUE when successful.

**See Also:**  
[TerSetSectOrient](#)  
[TerGetSectInfo](#)



## TerSetSectBorder

**Set the page border for a section.**

BOOL TerSetSectBorder(hWnd, sect, type, width, space, color, repaint)

HWND hWnd;                      // Window handle to access

int sect;                        // Section id to apply changes. You can also set this parameter to SECT\_CUR to edit the current section, or set it to SECT\_ALL to apply changes to all sections in the document.

int type;                        // Border type. It can be one of the following constants:

BRDRTYPE_SINGLE	Single line border
BRDRTYPE_DBL	Double line border
BRDRTYPE_TRIPLE	Triple line border
BRDRTYPE_SHADOW	Shadow border
BRDRTYPE_THICK_THIN	Thick-thin lines border
BRDRTYPE_THIN_THICK	Thin-thick lines border
BRDRTYPE_THICK_THIN_THICK	Thick-thin-thick border
BRDRTYPE_THIN_THICK_THIN	Thin-thick-thin border
BRDRTYPE_NONE	No Border

int width;                      // Line thickness in twips units

int space;                      // Border distance from the edge of the page in twips units

int color;                      // Border color

BOOL repaint;                  // TRUE to refresh screen after this operation.

**Return Value:** The function returns TRUE when successful.

**See Also:**  
[TerSetSect](#)



## TerSetSectColWidth

**Set the column width and inter-column spacing for a variable width column section.**

BOOL TerSetSectColWidth(hWnd, sect, col, width, ColSpace, repaint)

HWND hWnd;	// The handle of the window to be accessed
int sect;	// Section id to apply changes. You can also set this parameter to -1 to specify the current column.
int col;	// The column number (0 to total columns -1) to apply the width and column space parameters. Set this parameter to -1 to restore the section to use uniform width for the columns.
int width;	// The new width (int twips) for the specified column.
int ColSpace;	// The new inter-column space (in twips) after the specified column.
BOOL repaint;	// TRUE to refresh the window after this operation

**Return Value:** This function returns TRUE if successful.



## TerSetSectLineNbr

**Set line numbering for the section.**

BOOL TerSetSectLineNbr(hWnd, sect, set, repaint)

BOOL TerSetSectLineNbr2(hWnd, sect, set, step, repaint)

HWND hWnd;	// The handle of the window to be accessed
int sect;	// Section id to apply changes. You can also set this parameter to SECT_CUR to edit the current section, or set it to SECT_ALL to apply changes to all sections in the document.
BOOL set;	// Set to TRUE to enable line numbering. Set to false to disable line numbering.
int step;	// Steps in which to draw the numbers. A value of 0 or 1 produces continuous line numbering.
BOOL repaint;	// TRUE to refresh the window after this operation

**Comment:** The line numbers are displayed on the left side of the page. The page-layout (ID\_SHOW\_PAGE\_LAYOUT) display must be turned on to see line numbering.

**Return Value:** This function returns TRUE if successful.



## TerSetSectOrient

**Set the orientation for a section.**

BOOL TerSetSectOrient(hWnd, orient, repaint)

HWND hWnd; // Window handle to access

int orient; // orientation: DMORIENT\_PORTRAIT or  
DMORIENT\_LANDSCAPE

BOOL repaint; // TRUE to refresh screen after this operation.

**Description:** This function is used to set the page orientation for the section..

**Return Value:** The function returns TRUE when successful.

**See Also:**  
[TerSetSect](#)



## TerSetSectPageSize

**Set the page size for a section.**

BOOL TerSetSectPageSize(hWnd, sect, PageSize, PageWidth, PageHeight, repaint)

HWND hWnd; // Window handle to access

int sect; // Section id to apply changes. You can also set this  
parameter to SECT\_CUR to edit the current section, or  
set it to SECT\_ALL to apply changes to all sections in  
the document.

int PageSize; // Use one of the following Windows SDK defined  
constants:

Constant	Value
DMPAPER_LETTER	1
DMPAPER_LEGAL	5
DMPAPER_LEDGER	4
DMPAPER_TABLOID	3

DMPAPER_STATEMENT	6
DMPAPER_EXECUTIVE	7
DMPAPER_A3	8
DMPAPER_A4	9
DMPAPER_A5	11
DMPAPER_B4	12
DMPAPER_B5	13

If you need to use a paper size not listed above, please set the PageSize argument to zero and specify the page width and height using the next two arguments.

int PageWidth;	// The page width in twips units. This argument is ignored if the PageSize is set to one of the defined page sizes listed above.
int PageHeight;	// The page height in twips units. This argument is ignored if the PageSize is set to one of the defined page sizes listed above
BOOL repaint;	// TRUE to refresh screen after this operation.

**Return Value:** The function returns TRUE when successful.

**See Also:**  
[TerSetSect](#)



## TerSetSectParam

### Set section parameters.

BOOL TerSetSectParam(hWnd, select, type, val, repaint)

HWND hWnd;	// Handle of the window to be accessed
int select;	// Section id to apply changes. You can also set this parameter to SECT_CUR to edit the current section, or set it to SECT_ALL to apply changes to all sections in the document.
int type;	Parameter type to set. Select one of the constants:
SP_LINE_BET_COL	Set to 1 to draw a line between columns. Applicable to a multi-column section only.
SP_LEFT_MARG	Get the section left margin value in twips.

SP_RIGHT_MARG	Get the section right margin value in twips.
SP_TOP_MARG	Get the section top margin value in twips.
SP_BOT_MARG	Get the section bottom margin value in twips.
SP_HDR_MARG	Get the distance of the header text from the top of the page in twips.
SP_FTR_MARG	Get the distance of the footer text from the bottom of the page in twips.

int val;                      New value for the parameter specified by the 'type' parameter.

BOOL repaint;                TRUE to repaint the screen after this operation

**Return Value:** This function returns a TRUE value if successful.



## TerSetSectTextFlow

**Set the right-to-left/left-to-right text flow option for the section.**

BOOL TerSetSectTextFlow(hWnd, sect, TextFlow, refresh)

HWND hWnd;                      // The handle of the window to be accessed

int sect;                        // Section id to apply changes. You can also set this parameter to SECT\_CUR to edit the current section, or set it to SECT\_ALL to apply changes to all sections in the document.

int TextFlow;                    // The text flow constant can be one of the following

FLOW\_LTR                      Left-to-right text flow

FLOW\_RTL                      Right-to-left text flow

FLOW\_DEF                      Default text flow. The flow will be determined by the document, table, or paragraph level text flow specification.

```
BOOL refresh;           // TRUE to refresh the window after this operation
```

**Return Value:** This function returns TRUE if successful.

### See Also:

TerSetDocTextFlow

## TerSetParaTextFlow

TerSetRowTextFlow



## Document

## In This Chapter

## GetTerFields

## SetTerFields

## TerGetParam

## TerGetWordCount

TerInsertDateTime

TerInsertTOC

TerInsertToc2

## TerLoadExtFont

TerSetDefDir

TerSetDocTextFlow

## TerSetRtfDocInfo

TerGetRtfDocInfo



## GetTerFields

### Retrieve Window Variables:

BOOL GetTerFields(hWnd,field)

```
HWND hWnd; /* Window handle */
```

```
struct StrTerField *field;      /* information buffer, see below*/
```

Description: This function returns various operational parameters for the current TER window. See the TER.H file for the complete description of the StrTerField structure.**Information Block Structure:**

```
struct StrTerField {
```

The following fields are read/write fields. To update a field you must retrieve the current values by calling the *GetTerFields* function. Modify the fields that you wish to, and then call the *SetTerFields* function to make the new value effective.

int size;	The size of this structure in number of bytes.
-----------	--



int CurColW	Current window column position (0 to one less than the length of the line).
int PaintEnabledW	A FALSE value disables the screen painting and word wrapping until it is re-enabled using another call to the <i>SetTerFields</i> function
int WrapFlagW	Wrap control. This function can be used to temporarily suspend word wrapping. The word wrapping is automatically enabled when the user hits a keystroke or makes a selection from the menu.
long CurRowW	Current window row position ( 0 to the height of the window). This field is not meaningful in Page mode or Fitted View modes.
long BeginLineW	First line number in the window. The editor ensures that CurRow is always equal to CurLine minus BeginLine (this field is not meaningful in Page mode or Fittend View modes).
long CurLineW	Current line number in the file (0 to one less than the total number of lines in the file).
COLORREF TextBkColorW	Background color for the window.
COLORREF StatusBkColorW	Background color of the status line
COLORREF StatusColorW	Foreground color of the status line
int HilightTypeW	Line or character highlighting flag (see HILIGHT_ constants in the TER.H file). Use this flag and the following variables to set or reset text selection.
int HilightBegColW	Beginning column number of the highlighted block
int HilightEndColW	Ending column number of the highlighted block.
long HilightBegRowW	Beginning line number of the highlighted block
long HilightEndRowW	Ending line number of the highlighted block
BOOL StretchHilightW	A TRUE value allows the user to stretch the current highlighted block by using the mouse or arrow keys.

Please do not use the following two fields as they are being phased out. Use the TerGetLine function to retrieve the text and font ids for a line number.

Text data for the current line (NULL terminated). This

char text[MAX_WIDTH]	variable is being phased out. Please use the TerGetLine function to retrieve the line text.
char font[MAX_WIDTH]	Font id for every character in the 'text' array. Use the 'GetFontInfo' function to get further information about an editor font id. This variable is being phased out. Please use the TerGetLine function to retrieve the line fonts.
int pfmtW	Paragraph id of the current line
int LineLenW	Length of the current line
int TextApply	Use this variable to specify how the 'text' and 'font' data should be applied to the current TER window, see APPLY_ constants in the TER.H file. Using this flag you can modify the current line, or insert a new line after or before the current line.
BOOL ReclamesourcesW	TRUE to reuse unused font and paragraph ids.
BOOL ModifyProtectColorW	TRUE to show the protected text in a lighter shade
BOOL LinkDbClickW	TRUE to fire hyperlink on mouse double click. Otherwise single click is used to fire the hyperlink event.
BOOL ShowProtectCaretW	TRUE to display caret even when positioned on protected text.
UINT LinkStyleW	The character style of the hyperlink phrase. When this style is set to HLINK, then the following LinkColorW variable is not used for detecting a link.
COLORREF LinkColorW	The color of the hyperlink phrase.
BOOL SnapToGridW	TRUE to snap tabs and margin on the ruler to an invisible grid
BOOL HtmlModeW	TRUE to enable html mode adjustments
BOOL ShowTableGridLinesW	TRUE to show table grid lines
The following are the read only fields. TER will ignore any modification to these fields.	
HWND hTerWndW	Handle to the editor window
HDC hTerDCW	Handle to TER class DC. Call the TerGetBufferDC function if you wish to retrieve the handle of the associated buffer device context.

RECT TerRectW	Entire client window rectangle
RECT TerWinRectW	Text window rectangle
long TotalLinesW	Total lines in the file
long MouseLineW	Current text line position of the mouse pointer. Current row position is given by MouseLine minus BeginLine.
long MaxColBlockW	Biggest column block allowed
int TotalPfmtsW	Total paragraph ids in use by the current window.
int TotalFontsW	Total font objects in use by the current window
int TotalStylesW	Total number of stylesheet items
int WinWidthW	Current window width in character columns
int WinHeightW	Number of lines displayed in the window. This field is not meaningful in Page mode or Fitted View modes.
int TerWinOrgXW	Window origin x co-ordinates used to set the view port
int MouseColW	Current text column position of the mouse pointer.
BOOL modified	Data modified, user needs to select the 'save' option to save data
BOOL WordWrapW	True when the word wrap is turned on
int ParaLeftIndentW	Paragraph left indent in twips
int ParaRightIndentW	Paragraph right indent in twips
int ParaFirstIndentW	Paragraph first line indent in twips
UINT ParaFlagsW	Paragraph flags. Refer to the SetTerParaFmt function for a list of paragraph flags.
int ParaTabIdW	Paragraph tab id (index into the tab table)
int ParaCellIdW	Paragraph cell id (index into the cell table)
UINT ParaShadingW	Paragraph shading (0 to 10000)
int ParaFrameIdW	Paragraph frame id

int ParaSpaceBeforeW	Space before the paragraph in twips
int ParaSpaceAfterW	Space after the paragraph in twips
int ParaSpaceBetweenW	Minimum space between
int ParaStyleIdW	Paragraph style Id
int ParaAuxIdW	Paragraph aux id
int pflagsW	Paragraph PFLAG_ flag constants
int CurSectW	The section id of the current line. You can use the TerGetSeqSect function to translate the section id into the sequential section number.
int LeftMarginW	Section left margin in twips
int RightMarginW	Section right margin in twips
int TopMarginW	Section top margin in twips
int BotMarginW	Section bottom margin in twips
int columnsW	Number of columns in the current section
int CurPageW	Current page number
int TotalPagesW	Number of pages in the document
int MouseXW	Recent mouse click x position
int MouseYW	Recent mouse click y position
BOOL PrintViewW	TRUE when the print view mode is turned on
BOOL PageModeW	TRUE when the page mode is turned on
BOOL FittedViewW	TRUE when the fitted view mode is turned on
BOOL ShowParaMarkW	TRUE when showing paragraph markers
BOOL ShowHiddenTextW	TRUE when showing the hidden text
int CurCtlIdW	Currently selected control id
UINT ParaFrameFlagsW	Current paragraph frame flags (PARA_FRAME_? constants defined in the ter.h file

**Return Value:** A TRUE value indicates success.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

See Also  
[SetTerFields](#)  
[GetFontInfo](#)



## SetTerFields

### Set Window Variables:

BOOL SetTerFields(hWnd,field)

HWND hWnd;                               // The handle of the window to be accessed

struct StrTerField \*field;               // information buffer (see the GetTerFields function)

**Description:** This function sets various operational parameters for the current TER window. You must first call the GetTerFields function to retrieve the current values of the parameters. You can then change the variables that you need to change. The TER editor validates the information before applying them to the current window.

**Return Value:** A TRUE value indicates success.

See Also:  
[TerGetField](#)  
[TerSetCtlColor](#)



## TerGetParam

### Retrieve miscellaneous operating variables.

BOOL TerGetParam(hWnd, type)

HWND hWnd;                               // The handle of the window to be accessed

int type;                                 // The parameter type to retrieve:

TP\_CUR\_LINE                               Current line number

TP\_CUR\_COL                                Current column number

TP_CUR_SECT	Current section id
TP_MOUSE_X	<p>Returns the pixel x position where the mouse is located.</p> <p>This value is relevant only when retrieved from the PreProcess or Action events for a mouse message.</p>
TP_MOUSE_Y	<p>Returns the pixel y position when the mouse is located.</p> <p>This value is relevant only when retrieved from the PreProcess or Action events for a mouse message.</p>
TP_MOUSE_LINE	<p>Return the text line number where the mouse is located.</p> <p>This value is relevant only when retrieved from the PreProcess or Action events for a mouse message.</p>
TP_MOUSE_COL	<p>Return the text column number where the mouse is located.</p> <p>This value is relevant only when retrieved from the PreProcess or Action events for a mouse message.</p>
TP_MOUSE_FONT_ID	<p>Return the font id for the character where the mouse is located.</p> <p>This value is relevant only when retrieved from the PreProcess or Action events for a mouse message.</p>
TP_MOUSE_PICT_ID	<p>Return the picture id for the picture where the mouse is located. If the mouse is not positioned over a picture object, then the return value would be 0.</p> <p>This value is relevant only when retrieved from the PreProcess or Action events for a mouse message.</p>

TP_MOUSE_FIELD_ID	<p>Return the field-id for the field where the mouse is located. If the mouse is not positioned over a field, then the return value would be 0.</p> <p>This value is relevant only when retrieved from the PreProcess or Action events for a mouse message.</p>
TP_MOUSE_ON_TEXT_LINE	Returns TRUE if the mouse is located on a text line. This value is meaningful only when retrieved after calling the TerPixToTextPos method.
TP_PAGE_BK_COLOR	Page background color
TP_SELECTION_TYPE	<p>Selection type: Set to HIGHLIGHT_OFF if no text is selected. Any other value indicates that a text block is selected.</p> <p><i>The following four TP_SELECTION_ constants are not valid if the selection type is HIGHLIGHT_OFF.</i></p>
TP_SELECTION_START_LINE	Selection start line.
TP_SELECTION_START_COL	Selection start col.
TP_SELECTION_END_LINE	Selection end line.
TP_SELECTION_END_COL	Selection end col.
TP_TOTAL_BLTS	Total number of bullet ids in the document.
TP_TOTAL_CELLS	Total number of cell ids in the document.
TP_TOTAL_CHAR_TAGS	Total number of character tags in the document
TP_TOTAL_FONTS	Total number of font ids in the document.
TP_TOTAL_IMAGE_MAPS	Total number of image maps in the document.

TP_TOTAL_LINES	Total number of lines in the document.
TP_TOTAL_LISTS	Total number of list ids in the document.
TP_TOTAL_LIST_OR	Total number of list-override ids in the document.
TP_TOTAL_PAGES	Total number pages in the document.
TP_TOTAL_PARA_FRAMES	Total number of paragraph frames in the document
TP_TOTAL_PFMTS	Total number of paragraph ids in the document.
TP_TOTAL_SECTS	Total number of section ids in the document.
TP_TOTAL_STYLES	Total number of style ids in the document.
TP_TOTAL_TABS	Total number of tab ids in the document.
TP_TOTAL_TABLE_ROWS	Total number of table row ids in the document.
TP_WATERMARK_WASH	The return value is 1 if watermark picture is washed, otherwise the return value is 0.
TP_WATERMARK_PICT	Returns the picture id for the current watermark picture. Returns 0 if watermark is not set for the document.
TP_TOTAL_REVIEWERS	Total number of reviewer ids. Id# 0 is not used.

**Return Value:** This function returns the value of the requested parameter. It returns -1 to indicate an error condition.

**See Also:**  
[TerSetPictInfo](#)  
[TerPastePicture](#)  
[TerInsertPictureFile](#)



[TerPictureFromFile](#)  
[TerGetPictOffset](#)



## TerGetWordCount

**Count number of words in the document.**

long TerGetWordCount(hWnd, flags)

HWND hWnd;                      // Handle of the window to be accessed

UINT flags;                      // The flag can be one or more of the following bits:

WC\_SELECTION:                  Scan only the selected text.  
If a text block is not  
highlighted or if the  
WC\_SELECTION bit is not  
specified, then the entire  
document will be scanned.

WC\_INCLUDE\_HIDDEN:            Count the hidden words  
also.

WC\_INCLUDE\_HDR\_FTR:          Count the words in the  
header/footer area also.

**Return Value:** This function returns the number of words counted. It returns -1 to indicate an error condition.



## TerInsertDateTime

**Insert a date/time field.**

BOOL TerInsertDateTime(hWnd, format, repaint)

HWND hWnd;                      // The handle of the window to be accessed

LPBYTE format;                  // Date time format (see description). Set this parameter  
to NULL to display a date format selection dialog box.

BOOL repaint;                   //Repaint the window after this operation

**Description:** The format argument accepts a format string. A format string consists of

day, date, month, year, second, hour and delimiter components. Example:

```
TerInsertDateTime(hWnd, "M/d/yy" ,TRUE);
```

```
TerInsertDateTime(hWnd, "M-d-yy " ,TRUE);
```

```
TerInsertDateTime(hWnd, "d/M/yy" ,TRUE);
```

Following is a list of various format components using an example date: June 8, 1999, 2:30:01 PM

Format

String Component Example

-----

D day 8

M month 6

dd day, 0 padded 08

MM month, 0 padded 06

yy year, 2 digits 99

yyyy year, 4 digits 1999

ddd day (abbr) Tue

MMM month (abbr) Jun

dddd day Tuesday

MMMM month June

HH hour, 24 hour format 14

mm minutes 30

ss seconds 01

h 12 hour format 2

hh 12 hour format, 0 padded 02

am/pm AM or PM am/pm

Example date format strings:

"d MMM yy" 8 June 99

"dd/MM/yyyy " 08/06/1999

"dddd, d MMMM yyyy" Tuesday, 8 June 1999

" h:mm" 2:30

"HH:mm" 14:30

"hh:mm:ss am/pm" 02:30:01 PM

"dddd, d MMMM yyyy hh:mm:ss am/pm" Tuesday, 8 June 1999 02:30:01 PM

Note: Date format is case-sensitive.

**Delimiter:** A delimiter may be used to separate the date components. The delimiter could be '/', '-', comma, spaces or any character not used by the date components.

**Return Value:** This function returns TRUE when successful.



## TerInsertTOC

### Insert table of contents.

BOOL TerInsertToc(hWnd, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL repaint; //Repaint the window after this operation

**Description:** This function scans the document to build a table of contents at the current cursor location. It includes a text line in the table of contents if it uses a paragraph style and the name of the paragraph style is in the form of 'heading n', when 'n' is a number from 1 to 9. The heading number is used to specify the indentation level. This function uses paragraph styles 'TOC n' for the assembled heading lines. The top level heading (heading 1) is assigned the style 'TOC 1', and so on. The editor would automatically create any missing 'TOC' style.

To insert a table of contents, first create the heading styles using the TerEditStyle function. For example, if you wish to insert a three level deep table of contents, create heading styles 'heading 1', 'heading 2', and 'heading 3'. Then place the cursor at the heading lines and apply a suitable heading style using the TerSelectParaStyle function. The last step would be to position the cursor where you wish to insert the table of contents and call the TerInsertTOC function.

The table of contents are automatically updated whenever repagination occurs.

**Return Value:** This function returns TRUE when successful.

### See Also:

[TerEditStyle](#)

[TerSelectParaStyle](#)

[TerInsertPageRef](#)



## TerInsertToc2

### Insert table of contents.

BOOL TerInsertToc2(hWnd, TocType, styles,MinLevel, MaxLevel, repaint)

HWND hWnd; // Handle of the window to be accessed

int TocType; // The table-of-contents construction method. Use one of the following types:

Construct the table-of-contents with the text using

TOC_HEADINGS:	the heading styles named following the format 'heading n', where 'n' is a number from 1 to 9.
TOC_OUTLINES:	Construct the table-of-contents with the text using the style with a outline level 0 to 8.
TOC_CUSTOM:	Construct the table-of-contents with the text using the custom styles specified in the 'styles' parameter.
TOC_FIELD:	Construct the table-of-contents with the text using the 'tc' field. The 'tc' field can be applied to the text using the TerSetTcField function.

  

LPBYTE styles;	// The list of custom styles to use to construct the table of contents. Each style name in this list must be delimited using the comma delimiter.  This parameter is used only when the TocType parameter is set to TOC_CUSTOM.
int MinLevel;	// The minimum heading level to use.
int MaxLevel;	// The maximum heading level to use.
BOOL repaint;	// Repaint the window after this operation

**Description:** This function provides additional flexibility for creating the table-of-contents than the simpler method called [TerCreateToc](#).

The table of contents are automatically updated whenever repagination occurs.

**Return Value:** This function returns TRUE when successful.



## TerLoadExtFont

**Load an external font file.**

BOOL TerLoadExtFont(hWnd, typeface, FontFile, type)

HWND hWnd;	// The handle of the window to be accessed.
LPBYTE typeface;	// Actual typeface of the font as specified in the font-file.
LPBYTE FontFile	// Path or name of the FontFile containing the font data.
BOOL UpdateToolBar	// Set to true to update the toolbar immediately.

**Comment:** You can call this method more than once to load multiple fonts. The fonts loaded by this method are effective only during the session.

**Return Value:** This function returns TRUE when successful.

**Example:**

```
TerLoadExtFont(hWnd,"Pirulen","pirulen.ttf",true);
```



## TerSetDefDir

**Set the default directory and file type.**

BOOL TerSetDefDir(hWnd, dir, type)

HWND hWnd; // The handle of the window to be accessed.

LPBYTE dir; // The default directory. Set to "" to use the program directory.

int type; // Input file type:

SAVE\_TER: Native file (.sse or .ter)

SAVE\_RTF: RTF file

SAVE\_DOCX DOCX format

SAVE\_TEXT: Text file

SAVE\_UTEXT: Unicode Text Format (not available in the 16 bit product)

**Description:** This function is used to set the initial directory and the file type display by the File Open dialog.

**Return Value:** This function returns TRUE when successful.

See Also  
[TerSetLinkPictDir](#)



## TerSetDocTextFlow

### Set the right-to-left/left-to-right text flow option for the document.

BOOL TerSetDocTextFlow(hWnd, dialog, TextFlow, refresh)

HWND hWnd;	// The handle of the window to be accessed.
BOOL dialog;	// Set to TRUE to show the user dialog.
int TextFlow;	// The text flow constant can be one of the following:
FLOW_LTR:	Left-to-right text flow
FLOW_RTL	Right-to-left text flow
FLOW_DEF	Default text flow. The flow will be determined by the , section, table, or paragraph level text flow specification.
BOOL refresh;	// TRUE to refresh the window after this operation.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerSetParaTextFlow](#)  
[TerSetSectTextFlow](#)  
[TerSetRowTextFlow](#)



## TerSetRtfDocInfo

### Set the header information about the document.

BOOL TerSetRtfDocInfo(hWnd, type, text)

HWND hWnd;	// Handle of the window to be accessed
int type;	// Information type. Please refer to the TerGetRtfDocInfo function for the constant values for this argument.
LPBYTE text;	// Information text to set.

**Comment:** Information text specified by this function is saved with the document, only if the document is saved in the RTF format. This information is not saved in the native format document.

**Return Value:** This function returns TRUE when successful, otherwise it returns FALSE.

**See Also:**

[TerGetRtfDocInfo](#)



## TerGetRtfDocInfo

**Retrieve the header information about the document.**

```
int TerGetRtfDocInfo(hWnd, type, text)
```

```
HWND hWnd;           // The handle of the window to be accessed
```

```
int type;             // Information type. Use one of the following constants:
```

```
INFO_TITLE           Document title
```

```
INFO_SUBJECT         Subject
```

```
INFO_AUTHOR          Author
```

```
INFO_MANAGER         Manager
```

```
INFO_COMPANY         Company
```

```
INFO_OPERATOR        Operator
```

```
INFO_CATEGORY        Category
```

```
INFO_KEYWORDS        Keywords
```

```
INFO_COMMENT         Comment
```

```
INFO_DOCCOMM         Additional comment
```

```
INFO_HLINKBASE      Hyperlink base path
```

```
LPBYTE text;         // (output) Information text. Set to NULL to simply retrieve  
                     // the length of the information text.
```

**Return Value:** When successful, this function returns the length of the information text, otherwise it returns 0. It also returns the information text using the 'text' argument.

**See Also:**  
[TerSetRtfDocInfo](#)



## Text Selection

### In This Chapter

[DeselectTerText](#)  
[SelectTerText](#)  
[TerGetSelection](#)  
[TerGetTextSel](#)  
[TerLineSelected](#)  
[TerNormalizeBlock](#)



### DeselectTerText

#### Deselect previous selected text

BOOL DeselectTerText(hWnd, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL repaint; // TRUE to refresh the window after this operation.

**Description:** Use this function to deselect previously selected text.

**Return Value:** This function returns TRUE if successful.

### See Also

[SelectTerText](#)



### SelectTerText

#### Select text block

BOOL SelectTerText(hWnd, FirstLine, FirstCol, LastLine, LastCol, repaint)

HWND hWnd; // Handle of the window to be accessed

long FirstLine; // Beginning line number of the block

int FirstCol; // Beginning column number of the block

long LastLine; // Last line number of the block

int LastCol; // Last column number of the block

BOOL repaint; // TRUE to refresh the window after this operation



**Description:** This function is used to select a block of text. When the 'repaint' flag is set, the selected block is shown with a highlight.

The FirstLine and FirstCol determine the beginning of the block. To specify the beginning location in absolute terms (character position from the beginning of the file), set the FirstCol to -1, and specify the absolute location using the FirstLine argument.

The LastLine and LastCol (exclusive) determine the end of the block. To specify the ending location in absolute terms (character position from the beginning of the file), set the LastCol to -1, and specify the absolute location using the LastLine argument.

Note that all characters starting from the beginning location until the last character before the ending location are included in the block.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[DeselectTerText](#)

[TerGetSelection](#)



## TerGetSelection

**Get the beginning and ending positions of a selected text block.**

BOOL TerGetSelection(hWnd, FirstLine, FirstCol, EndLine, EndCol)

HWND hWnd;	// The handle of the window to be accessed
LPLONG FirstLine;	// Beginning line number of the block
LPINT FirstCol;	// Beginning column number of the block
LPLONG EndLine;	// Last line number of the block
LPINT EndCol;	// The first non-selected column position. The ending column position is not included in the selected block.

**Return Value:** This function returns TRUE if a text block is selected, otherwise it returns FALSE.

#### See Also:

[DeselectTerText](#)

[SelectTerText](#)

[TerIsTableSelected](#)



## TerGetTextSel

Retrieve the selected text.

**HANDLE TerGetTextSel(hWnd, size)**

**HANDLE TerGetTextSelNR(hWnd)**

**HANDLE TerGetTextSelU(hWnd, size)**

**HANDLE TerGetTextSelUNR(hWnd)**

HWND hWnd;                               // The handle of the window to be accessed

long far \*size;                           // The size of the output global buffer.

**Return Value:** This function returns the global handle to the buffer containing the selected data as plain text (excluding any hidden text). The size of the global buffer is returned using the second argument. Lock (GlobalLock) the returned handle to access the text. Your application owns this memory handle. When the data is no longer required, use the GlobalFree function to free the global handle.

A NULL value of the handle indicates an error.

The *TerGetTextSelU* function should be use when the text contains unicode characters. This function returns the text in the unicode format. Each character is two bytes long. Therefore the pointer dereference from the global handle (return value) should be treated as a pointer to the WORD array.

A VB application can use the HandleToIntArray function read the contents of the output of the TerGetTextSelU function into an integer array

```
Dim size As Long
Dim hMem As Long
Dim UniText(MAX_WIDTH) as Integer
Dim count As Long

hMem = TOCl.TerGetTextSelU(size)
count = TOCl.HandleToIntArray(UniText(0), size, hMem)
```

The UniText array now contains unicode character values.

*The TerGetTextSelNR method is similar to the TerGetTextSel method, but it does not use the size parameter to return the size of the memory block. This function is useful within a script which does not allow for a 'pass-by-reference' parameter. The size of the returned memory block can be retrieved by immediately calling the [TerGetRefParam](#) method. Similarly, the TerGetTextSelUNR function is a replacement for the TerGetTextSelU unicode function within a script.*

#### See Also:

[TerGetRtfSel](#)  
[GetTerBuffer](#)  
[SetTerBuffer](#)

[ReadTerFile](#)  
[SaveTerFile](#)  
[CloseTer](#)  
[HandleToStr](#)  
[TerSearchReplace](#)  
[HandleToIntArray](#)



## TerLineSelected

**Check if a text line is selected in the highlighted block of text.**

BOOL TerLineSelected(hWnd, LineNo)

HWND hWnd; // Handle of the window to be accessed

long LineNo; // text line number.

**Return Value:** This function returns TRUE if a text block is highlighted in the control and the given line is also highlighted. It **also** returns TRUE if no text block is highlighted in the editor. It returns FALSE if a text block is highlighted but the given line is not.



## TerNormalizeBlock

**Normalize selected text block.**

BOOL TerNormalizeBlock(hWnd)

HWND hWnd; // Handle of the window to be accessed

**Description:** This function is used to adjust the beginning and ending of a selected text block, such that the beginning position is smaller than the ending position.



## Cursor and Text Position

**In This Chapter**  
[GetTerCursorPos](#)  
[SetTerCursorPos](#)  
[TerAbsToRowCol](#)  
[TerEngageCaret](#)  
[TerGetCaretPos](#)

[TerGetVisibleCol](#)  
[TerPixToTextPos](#)  
[TerPosLineAtTop](#)  
[TerPosBodyText](#)  
[TerRowColToAbs](#)  
[TerScrToTwipsX](#)  
[TerScrToTwipsY](#)  
[TerSetCaretPos](#)  
[TerTextPosToPix](#)



## GetTerCursorPos

**Retrieve the current cursor position.**

BOOL GetTerCursorPos(hWnd, CurLine, CurCol)

HWND hWnd;                      window handle to be accessed

long far \*CurLine;              // Pointer to a long variable where the current line  
                                    number or the current absolute cursor position is  
                                    returned

int far \*CurCol;                // Pointer to an integer variable where the current column  
                                    number is returned.

Description: This function returns the current cursor position. The cursor position can be retrieved as the absolute position or in terms of the line number and column number. To get the absolute cursor position, set the CurCol variable to -1 before calling this function. The absolute position (base 0) is returned in the CurLine variable.

Example: int CurCol=-1; long CurLine;

**GetTerCursorPos(hWnd,&CurLine,&CurCol);** // returns absolute position in the CurLine variable.  
**Line,&CurCol);**

To get the line (base 0) and column (base 0) position of the cursor, set the CurCol variable to a value other than -1 before calling this function.

int CurCol=0;

long CurLine;

GetTerCursorPos(hWnd,&CurLine,&CurCol);    // The current line number is returned  
   in the CurLine variable, the current  
   column is returned in the CurCol  
   variable.

**Return Value:** This function returns TRUE when successful.

### See Also

[SetTerCursorPos](#)



## SetTerCursorPos

### Set the cursor position

BOOL SetTerCursorPos(hWnd, line, column, repaint)

HWND hWnd; // The handle of the window to be accessed

long line; // new line position of the cursor. Set to -1 to position at the end of the document.

int column; // new column position of the cursor

BOOL repaint; // TRUE to refresh the window after this operation

**Description:** Use this function to set the new cursor position.

To specify the absolute cursor position, set the 'column' argument to -1, and specify the absolute position using the 'line' argument.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[GetTerCursorPos](#)



## TerAbsToRowCol

### Convert the given character position to the row and column position

void TerAbsToRowCol(hWnd, abs, row, col)

HWND hWnd; // Handle of the window to be accessed

long abs; // character position (0 based) from the beginning of the file.

long far \*row; // location to return the line number (0 based)

int far \*col; // location to return the column number (0 based)

**Description:** This function converts the text position given in a number of characters from the beginning of the file to the row and column position.

**Return Value:** The line and column numbers are returned using the pointer specified by the third and fourth arguments.

**See Also:**  
[TerRowColToAbs](#)



## TerEngageCaret

**Engage the caret manually.**

BOOL TerEngageCaret(hWnd, AtCursorLoc)

HWND hWnd; // Editor window to access

BOOL AtCursorLoc; // Set to TRUE to engage the caret at the current 'cursor' location. Set to FALSE to engage the caret at the current 'caret' location.

**Description:** The caret position indicates the text insertion point, whereas the cursor position indicates a position within currently visible text on the screen. Normally, these positions are the same. However, when the user clicks on the scrollbar, the caret can become disengaged from the cursor position. The editor will reengage the caret automatically when the user conducts any text editing operation. This function allows you to engage the caret manually. This function does not have any effect if the caret is already engaged.

**Return Value:** The function returns TRUE when successful.



## TerGetCaretPos

**Get the current text insertion position.**

long TerGetCaretPos(hWnd)

HWND hWnd; // The handle of the window to be accessed

**Description:** This function returns the current text insertion position or the caret position. Please note that the editor differentiates between the text insertion position and the current cursor position as returned by the GetTerCursorPos function. When the user scrolls the text, the caret position (where the next text input will be inserted) remains the same. However, the cursor position changes in such a way that the cursor position is always maintained within the current visible text on the screen.

**Return Value:** This function returns the caret position. This value is returned as the character position since the beginning of the file. You can use the TerRowColToAbs function to convert this position in to line/column position. This function returns -1 if an error occurs.

**See Also:**

[GetTerCursorPos](#)  
[SetTerCursorPos](#)  
[TerRowColToAbs](#)  
[TerSetCaretPos](#)



## TerGetVisibleCol

**Get the visible column number from a text column number.**

long TerGetVisibleCol(hWnd, LineNo, ColNo)

HWND hWnd;                               // The handle of the window to be accessed

long LineNo;                             // The text line number (zero based). Set this parameter to -1 to specify the current line number.

int ColNo                                // The text column number. Set to -1 to specify the current column number.

**Description:** This function calculates the visible column number (zero based) by ignoring the character not visible on the screen. These characters might include the hidden text when the hidden text is not displayed, field name when field name is not displayed, field data when the field data is not displayed, etc.

**Return Value:** This function returns the visible column number. It returns -1 to indicate an error condition.



## TerPixToTextPos

**Retrieve the text position at a given pixel position.**

BOOL TerPixToTextPos(hWnd, RelativeTo, x, y, pLine, pCol)

HWND hWnd;                               // Handle of the window to be accessed

int RelativeTo;                         // This parameter should be set to one of the following constants:

REL\_SCREEN:                            When specifying the x/y values relative to the top of the screen.

REL\_WINDOW:                            When specifying the x/y values relative to the client area of the edit control.

REL\_TEXT\_BOX: When specifying the x/y values relative to the top of the text box.

```
int x;                // The x position of the pixel.

int y;                // The y position of the pixel.

DWORD far *pLine;     // The pointer to the location to receive the text line
                      // number.

int far *pCol;         // The pointer to the location to receive the text column
                      // number. To retrieve the absolute text position in the pLine
                      // parameter, set the pCol parameter to NULL or set the
                      // column number to -1.
```

**Return Value:** This function returns TRUE when successful.

Example: The example below retrieves the text position at a pixel position  $x=100$ ,  $y=100$ .

long line;

```
int col=0; // set to 0 to retrieve the position in line/column format.
```

```
TerPixToTextPos(hWnd, REL_TEXT_BOX, 100, 100, &line, &col);
```

**See Also:**  
[TerTextPosToPix](#)



## TerPosLineAtTop

**Position the specified line at the top or middle of the window.**

BOOL TerPosLineAtTop (hWnd, LineNo, WinTop)

```
HWND hWnd;           // Handle of the window to be accessed
```

```
long LineNo;           // Line number to position at
```

```

BOOL WinTop;           // TRUE to position the specified line at the top of the
                        // window, FALSE to position the line at the middle of the
                        // window.

```

**Return Value:** This function returns a TRUE value when successful.



## TerPosBodyText



### Position the cursor at the body text outside of header or footer text.

BOOL TerPosBodyText (hWnd, section, pos, repaint)

HWND hWnd;	// Handle of the window to be accessed
int section;	// Sequential section number for the text. Specify a number between 0 (first section) and total section -1.
 This function uses sequential section numbers within the document. Please note that the sequential section numbers can be different from the actual section id for the section. You can use the TerGetSeqSect function to translate a section id into the sequential section number.	
int pos;	// Set to POS_BEG to position at the first character of the body text. Set to POS_END to position at the end of the section.
BOOL repaint;	// TRUE to refresh the screen after this operation.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**  
[TerPosHdrFtr](#)  
[TerPosFrame](#)  
[TerPosTable](#)  
[TerGetSeqSect](#)



## TerRowColToAbs

### Convert the given line/row position to the character position.

long TerRowColToAbs(hWnd, row, col)

HWND hWnd;	// The handle of the window to be accessed.
long row;	// text line number. The text line number must be between 0 and TotalLines - 1.
int col;	// text column position. The text column position must be between 0 and line length minus 1.

**Description:** This function translates the text position given in line number and column number to the character position from the beginning of the file.

**Return Value:** The function returns the text position from the beginning of the file.

**See Also:**  
[TerAbsToRowCol](#)



## TerScrToTwipsX

**Translate screen x position to margin relative x position.**

**BOOL TerScrToTwipsX(hWnd, ScrX, MargX)**

HWND hWnd;	// The handle of the window to be accessed.
int ScrX;	// The screen X position (pixels) to translate.
LPINT MargX;	// The variable to receive the left margin relative position in twips units.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerScrToTwipsY](#)



## TerScrToTwipsY

**Translate screen y position to page relative y position.**

**BOOL TerScrToTwipsY(hWnd, ScrY, MargY)**

HWND hWnd;	// The handle of the window to be accessed.
int ScrY;	// The screen Y position (pixels) to translate.
LPINT PaegY;	// The variable to receive the position (twips) relative to the top of the page.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerScrToTwipsX](#)



## TerSetCaretPos

**Set the current caret position.**

BOOL TerSetCaretPos(hWnd, CaretPos)

HWND hWnd;	// The handle of the window to be accessed.
Long CaretPos;	// New caret position. The value is specified as the character position from the beginning of the document

**Description:** When the caret is engaged, this function simply calls the SetTerCursorPos function to set the cursor position. When the caret is disengaged from the cursor, this value updates an internal variable. When the caret is eventually engaged, the cursor is positioned at the new caret position.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerGetCaretPos](#)

**TerTextPosToPix****Retrieve the pixel position of the text.**

BOOL TerTextPosToPix(hWnd, RelativeTo, line, col, x, y)

HWND hWnd;	// Handle of the window to be accessed
int RelativeTo;	// This parameter should be set to one of the following constants:  REL_SCREEN: To return the x/y values relative to the top of the screen. REL_WINDOW: To return the x/y values relative to the client area of the edit control. REL_TEXT_BOX: To return the x/y values relative to the top of the text box.
long line;	// The line number of the text to find position. Set this parameter to -1 to retrieve the pixel position of the text at the current caret location.
int col;	// The column number of the text to find position. To specify an absolute location, set the col to -1, and specify the absolute position in the 'line' argument.
LPINT x;	// The parameter to receive the x pixel position.

LPINT y; // The parameter to receive the y pixel position.

**Return Value:** This function returns TRUE when successful.

**Example:** The example below retrieves the pixel position at the current cursor location relative to the top of the text box.

int x,y,

TerTextPosToPix(hWnd, REL\_TEXT\_BOX, -1, -1, &x, &y);

#### See Also

[TerPixToTextPos](#)



## Table

### In This Chapter

[TerAdustHtmlTable](#)  
[TerCellBorder](#)  
[TerCellBorderColor](#)  
[TerCellColor](#)  
[TerCellRotateText](#)  
[TerCellShading](#)  
[TerCellVertAlign](#)  
[TerCellWidth](#)  
[TerCreateCellId](#)  
[TerDeleteCells](#)  
[TerDeleteCellText](#)  
[TerCreateTable](#)  
[TerGetCellBorderWidth](#)  
[TerGetCellBorderColor](#)  
[TerGetCellInfo](#)  
[TerGetCellInfo2](#)  
[TerGetCellParam](#)  
[TerGetRowCellCount](#)  
[TerGetRowInfo](#)  
[TerGetTableId](#)  
[TerGetTableLevel](#)  
[TerGetTablePos](#)  
[TerHtmlCellWidthFlag](#)  
[TerInsertTableCol](#)  
[TerInsertTableRow](#)  
[TerIsTableSelected](#)  
[TerMarkCells](#)  
[TerPosAfterTable](#)  
[TerPosTable](#)  
[TerReformatTable](#)  
[TerRowHeight](#)  
[TerRowPosition](#)  
[TerSelectCellText](#)  
[TerSelectCol](#)  
[TerSelectRow](#)  
[TerSelectTable](#)

[TerSetCellInfo2](#)  
[TerSetCellParam](#)  
[TerSetHdrRow](#)  
[TerSetRowKeep](#)  
[TerSetRowTextFlow](#)  
[TerSetTableColWidth](#)  
[TerSetTableId](#)



## TerAdjustHtmlTable

**Adjust the HTML table width.**

BOOL TerAdjustHtmlTable(hWnd)

HWND hWnd; // Handle of the window to be accessed

**Description:** This function is used in conjunction with HTML add-on product. When you add/delete a column or change a column width, you can call this function to recalculate the cell width. The cell width is calculated using the html table width specification, cell contents, and the current editor window width.

**Return Value:** The function returns TRUE when successful.



## TerCellBorder

**Set the borders for the table cells.**

BOOL TerCellBorder(hWnd, select, TopWidth, BotWidth, LeftWidth, RightWidth, repaint)

BOOL TerCellBorder2(hWnd, select, TopWidth, BotWidth, LeftWidth, RightWidth, outline, repaint)

HWND hWnd; // Handle of the window to be accessed

int select; // Cell selection for this operation:

SEL\_ALL: Select the entire table

SEL\_CELLS: Select the current cell or all highlighted cells

SEL\_COLS: Select the current column or all highlighted columns

SEL\_ROWS:     Select the current row or all highlighted rows

Set the 'select' parameter to 0 to invoke the user selection dialog box.

int TopWidth;             Width of the top border in twips.

int BotWidth;             Width of the bottom border in twips

int LeftBorder;           Width of the left border in twips

int RightBorder;          Width of the right border in twips

BOOL outline;             Set to True to draw the outline around the selected cells.

BOOL repaint;             TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function. The maximum border width should be less than the cell text margin. Any width parameter can be set to -1 to leave the current value unchanged.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerPosTable](#)  
[TerSetTableColWidth](#)  
[TerInsertTableRow](#)  
[TerCellShading](#)  
[TerCellBorderColor](#)



## TerCellBorderColor

**Set the borders color for the table cells.**

BOOL TerCellBorderColor(hWnd, select, top, bot, left, right, repaint)

HWND hWnd;                // Handle of the window to be accessed

int select;                // Cell selection for this operation:

SEL\_ALL:           Select the entire table

SEL\_CELLS:        Select the current cell or all highlighted cells

SEL\_COLS:         Select the current column or all

highlighted columns

SEL\_ROWS:     Select the current row or all highlighted rows

Set the 'select' parameter to 0 to invoke the user selection dialog box.

COLORREF top;                 // Color for the top border. Set the this parameter to CLR\_INVALID to leave it unchanged.

COLORREF bot;                // Color for the bottom border. Set the this parameter to CLR\_INVALID to leave it unchanged.

COLORREF left;                // Color for the left border. Set the this parameter to CLR\_INVALID to leave it unchanged.

COLORREF right;               // Color for the right border. Set the this parameter to CLR\_INVALID to leave it unchanged.

BOOL repaint;                 // TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerPosTable](#)  
[TerSetTableColWidth](#)  
[TerInsertTableRow](#)  
[TerCellShading](#)  
[TerCellBorder](#)



## TerCellColor

**Set the cell background color.**

BOOL TerCellColor(hWnd, select, color, repaint)

HWND hWnd;                    // Handle of the window to be accessed

int select;                    // Cell selection for this operation:

SEL\_ALL:                       Select the entire table

SEL\_CELLS:                    Select the current cell or all highlighted cells

SEL\_COLS:                      Select the current column or all highlighted columns

SEL\_ROWS:                      Select the current row or all highlighted rows

Set the 'select' parameter to 0 to invoke the user selection dialog box.

COLORREF color;                // Cell background color

BOOL repaint;                 // TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerPosTable](#)  
[TerCellBorder](#)  
[TerCellShading](#)  
[TerCellWidth](#)



## TerCellRotateText

**Set the text rotation angle within a table cell.**

BOOL TerCellRotateText(hWnd, select, direction, repaint)

HWND hWnd;                    // Handle of the window to be accessed

int select;                    // Cell selection for this operation:

SEL\_ALL:                      Select the entire table

SEL\_CELLS:                    Select the current cell or all highlighted cells

SEL\_COLS:                     Select the current column or all highlighted columns

SEL\_ROWS:                     Select the current row or all highlighted rows

Set the 'select' parameter to 0 to invoke the user selection dialog box.

int direction;                // Set it to one of the following values:

TEXT\_HORZ:                    Horizontal text flow.

TEXT\_TOP\_TO\_BOT:            Top-to-Bottom vertical text flow.



TEXT\_BOT\_TO\_TOP: Bottom-to-Top vertical text flow.

BOOL repaint; // TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.



## TerCellShading

**Set the shading percentage for the table cells.**

BOOL TerCellShading(hWnd, select, percent, repaint)

HWND hWnd; // Handle of the window to be accessed

int select; // Cell selection for this operation:

SEL\_ALL: Select the entire table

SEL\_CELLS: Select the current cell or all highlighted cells

SEL\_COLS: Select the current column or all highlighted columns

SEL\_ROWS: Select the current row or all highlighted rows

Set the 'select' parameter to 0 to invoke the user selection dialog box

int percent; // Shading percentage (0 to 100)

BOOL repaint; // TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerPosTable](#)  
[TerCellBorder](#)  
[TerCellColor](#)  
[TerCellWidth](#)



## TerCellVertAlign

**Set the vertical alignment for the text inside a table cell.**

BOOL TerCellVertAlign(hWnd, select, align, repaint)

HWND hWnd;                      // Handle of the window to be accessed

int select;                      // Cell selection for this operation:

SEL\_ALL:                      Select the entire table

SEL\_CELLS:                      Select the current cell or all highlighted cells

SEL\_COLS:                      Select the current column or all highlighted columns

SEL\_ROWS:                      Select the current row or all highlighted rows

Set the 'select' parameter to 0 to invoke the user selection dialog box.

int align;                      // Set this parameter to 0 to use the default 'top' alignment, or set it to one of the following values:

CFLAG\_VALIGN\_CTR:      Center alignment

CFLAG\_VALIGN\_BOT:      Bottom alignment

CFLAG\_VALIGN\_BASE:      Align base line of the text

BOOL repaint;                      // TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerPosTable](#)  
[TerCellBorder](#)  
[TerCellColor](#)



## TerCellWidth

**Set the cell width and cell margin.**

BOOL TerCellWidth(hWnd, select, width, margin, repaint)

HWND hWnd;	// Handle of the window to be accessed
int select;	// Cell selection for this operation:
SEL_ALL:	Select the entire table
SEL_CELLS:	Select the current cell or all highlighted cells
SEL_COLS:	Select the current column or all highlighted columns
SEL_ROWS:	Select the current row or all highlighted rows

Set the 'select' parameter to 0 to invoke the user selection dialog box.

int width;	// Cell width in twips unit. Set to -1 to leave this value unchanged.
int margin;	// Cell Margin in twips unit. Set to -1 to leave this value unchanged.
BOOL repaint;	// TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerPosTable](#)  
[TerCellBorder](#)  
[TerCellColor](#)



## TerCreateCellId

### Create a cell id.

int TerCreateCellId(HWND hWnd, NewRow, PrevCell, RowAlign, RowPos, RowMinHeight, CellWidth, shading, LeftWidth, RightWidth, TopWidth, BotWidth, RowSpan, ColSpan, flags)

HWND hWnd;	// The handle of the window to be accessed
BOOL NewRow;	// Set to TRUE when creating the first cell of a new row.
int PrevCell;	// Set this value to 0 when creating the first cell id for first row of the table. Otherwise set it to the cell id of the

previous cell in the table.

int RowAlign;	// LEFT,RIGHT, CENTER. (set to 0 for default).
int RowPos;	// Row position in twips (set to 0 for default).
int RowMinHeight;	// Minimum row height in twips (set to 0 for default).
int CellWidth;	// Cell width in twips.
int shading;	// Shading percentage (set to 0 for default)
int LeftWidth;	// The left border width (set to 0 for default).
int RightWidth;	// The right border width (set to 0 for default).
int TopWidth;	// The top border width (set to 0 for default).
int BotWidth;	// The bottom border width (set to 0 for default).
int RowSpan;	// Number of rows spanned by the cell (set to 1 for default).
int ColSpan;	// Number of columns spanned by the cell (set to 1 for default).
int flags;	// CFLAG_ constants defined in the ter.h file (set to 0 for default).

**Description:** This function is used to create a new cell id. This function is useful for creating tables very efficiently.

The following example creates a 3 row by 2 column table:

PrevCell=0;

```
for (row=0;row<3;row++) {
    for (col=0;col<2;col++) {
        if (col==0) NewRow=TRUE;
        else NewRow=FALSE;
        CellId=TerCreateCellId(hWnd,NewRow,PrevCell,0,0,0,
                               2000,0,0,0,0,0,1,1,0);
        string = "cell text" + chr$(CELL_CHAR);    // append cell
                                                    delimiter to the cell text
        TerAppendTextEx(hWnd,string,-1,-1,CellId,-1,FALSE);
        PrevCell=CellId;
    }
    string = chr$(ROW_CHAR);    // insert a row delimiter
```

```

    TerAppendTextEx(hWnd,string,-1,-1,CellId,-1,FALSE);
}

```

**Return Value:** When successful, this function returns the id of the new cell. Otherwise it returns 0

#### See Also:

[TerCreateParald](#)  
[TerAppendTextEx](#)  
[TerSetCellInfo2](#)



## TerDeleteCells

**Delete table cells.**

BOOL TerDeleteCells(hWnd,select, repaint)

HWND hWnd; // The handle of the window to be accessed

int select; // This flag can be set to one of the following values:

SEL\_CELLS: Delete the current cell or all selected cells.

SEL\_ROWS: Delete the current row or all selected rows.

SEL\_COLS: Delete the current column or all selected columns.

You can set the 'select' parameter to 0 to invoke a dialog box to accept this parameter from the user.

BOOL repaint; // repaint the screen after this operation.

**Return Value:** This function returns TRUE when successful.

#### See Also

[TerDeleteCellText](#)



## TerDeleteCellText

**Delete cell contents.**

BOOL TerDeleteCellText(hWnd,select, repaint)

HWND hWnd;	// The handle of the window to be accessed
int select;	// This flag can be set to one of the following values:
SEL_CELLS:	Delete the contents of the current cell or all selected cells.
SEL_ROWS:	Delete the contents of the current row or all selected rows.
SEL_COLS:	Delete the contents of the current column or all selected columns.
You can set the 'select' parameter to 0 to invoke a dialog box to accept this parameter from the user.	
BOOL repaint;	// repaint the screen after this operation.

**Comment:** This function deletes the contents of the table cell, but the table structure is not affected.

**Return Value:** This function returns TRUE when successful.

#### See Also

[TerDeleteBlock](#)

[TerDeleteCells](#)



## TerCreateTable

**Create a text table.**

**BOOL TerCreateTable(HWND, row, col, repaint)**

**BOOL TerCreateTable2(HWND, row, col, AutoWidth, repaint)**

HWND hWnd;	// The handle of the window to be accessed
int row;	// number of text rows in the table.
int col;	// number of text columns in the table
BOOL AutoWidth	// Set to TRUE to let the table cell automatically expand as the user types text into a table cell. This parameter is applicable to only the TerCreateTable2 function.
BOOL repaint;	//Repaint the window after this operation

**Description:** This function is used to create a text table. The number of rows and columns are specified by the 'row' and 'col' arguments. Specify a -1 value for the 'row' if

you wish to activate a user dialog for the row and column selection.

The table is inserted after the current line. After this operation the cursor is placed in the first cell of the table.

Please note that the PageMode must be turned on at the design-time for this function to work properly. If the PageMode is not turned on, the tables are displayed as a series of dashed lines.

**Return Value:** This function returns TRUE if successful.



## TerGetCellBorderWidth

**Retrieve the border width about a specific cell id.**

```
int TerGetCellBorderWidth(hWnd, CellId, pLeft, pRight, pTop, pBottom);
```

HWND hWnd;	// The handle of the window to be accessed
int CellId;	// Cell id to retrieve the information for. Set to -1 to select the current table cell.
LPINT pLeft;	// The location to receive the left border width in twips.
LPINT pRight;	// The location to receive the right border width in twips.
LPINT pTop;	// The location to receive the top border width in twips.
LPINT pBottom;	// The location to receive the bottom border width in twips.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**

[TerGetCellInfo](#)



## TerGetCellBorderColor

**Retrieve the border color for a specific cell id.**

```
int TerGetCellBorderColor(hWnd, CellId, pLeft, pRight, pTop, pBottom);
```

HWND hWnd;	// The handle of the window to be accessed
int CellId;	// Cell id to retrieve the information for. Set to -1 to select

the current table cell.

```
COLORREF far * pLeft;      // The location to receive the left border color.
COLORREF far * pRight;     // The location to receive the right border color.
COLORREF far * pTop;       // The location to receive the top border color.
COLORREF far * pBottom;    // The location to receive the bottom border color.
```

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerGetCellInfo](#)



## TerGetCellInfo

**Retrieve the information about a specific cell id.**

```
int TerGetCellInfo(hWnd, CellId, RowId, PrevCell, NextCell, width, border, shading,
RowSpan, ColSpan, flags);
```

```
HWND hWnd;                // The handle of the window to be accessed
int CellId;                // Cell id to retrieve the information for.
LPINT RowId;               // Pointer to receive the row id for the cell
LPINT PrevCell;            // Pointer to receive the previous cell in the row
LPINT NextCell;            // Pointer to receive the next cell in the row
LPINT width;               // Pointer to receive cell width in twips
LPINT border;              // Pointer to receive border (TRUE/FALSE)
LPINT shading;             // Pointer to receive the shading percentage
LPINT RowSpan;             // Pointer to receive the row span for the cell
LPINT ColSpan;             // Pointer to receive the column span for the cell
UINT far *flags;           // Pointer to receive the cell flags (CFLAG_ constants).
```

**Return Value:** This function returns a TRUE value if successful.



**See Also:**  
[TerGetCellBorderWidth](#)  
[TerGetRowCellCount](#)



## TerGetCellInfo2

**Retrieve the information about a specific cell id.**

```
int TerGetCellInfo2(HWND, CellId, BackColor, margin)

HWND hWnd;           // The handle of the window to be accessed

int CellId;          // Cell id to retrieve the information for.

COLOREF* BackColor   // Pointer to receive the cell background color

LPINT margin;        // Pointer to receive the cell margin value
```

**Return Value:** This function returns a TRUE value if successful.



## TerGetCellParam

**Get additional cell parameters.**

```
BOOL TerGetCellParam(HWND, type, id, value)
long TerGetCellParam2(HWND, type, id)

HWND hWnd;           // The handle of the window to be accessed

int type;            // The parameter to retrieve:

    CP_TEXT_ROTATION:  Return the text rotation type. Please refer
                        to the TerCellRotateText function for a list
                        of text rotation type constants.

    CP_PARENT_CELL     The parent cell id for the requested cell.

    CP_LEVEL           Nesting level for the cell.

    CP_TEXT_ROTATION   Returns the following values for text
                        direction:
```

```

TEXT_TOP_TO_BOT: Top To
bottom
TEXT_BOT_TO_TOP: Bottom to
top
TEXT_HORZ:          Horizontal

```

CP_WIDTH	Cell width in twips.
CP_ROW	Row id which contains the specified cell.
CP_ID	Id of the specified cell. Same as the 'id' parameter, unless the 'id' parameter is set to -1.
CP_NEXT	Id of the next cell in the current row.
CP_PREV	Id of the previous cell in the current row.
CP_ROW_WIDTH	Width (in twips) of the row which contains the specified cell.
CP_PAD_LEFT	Cell left padding in twips.
CP_PAD_RIGHT	Cell right padding in twips.
CP_PAD_TOP	Cell top padding in twips.
CP_PAD_BOT	Cell bottom padding in twips.

```

int id;          // Cell id to retrieve parameters. Set to -1 to use the cell id of the current
                 line.

```

```

LPINT value      // The variable to receive the requested value.

```

**Return Value:** If successful, the TerGetCellParam function returns a TRUE value, and the requested value is returned using the 'value' parameter. A FALSE value indicates an error.

If successful, the TerGetCellParam2 method returns the value of the requested parameter. A return value of CP\_ERROR indicates an error.



## TerGetRowCount

### Retrieve table row or cell count.

int TerGetRowCellCount(hWnd, GetRowCount)

HWND hWnd;	// The handle of the window to be accessed
BOOL GetRowCount;	// Set to TRUE to return the number of rows in the current table. Set to FALSE to get the number of cells in the current row.

**Return Value:** The return value is as described above. A value of 0 indicates an error.

See Also  
[TerGetCellInfo](#)



## TerGetRowInfo

### Retrieve information about a table row id.

BOOL TerGetRowInfo(hWnd, RowId, height, MinHeight, FixWidth, PrevRow, NextRow, indent, flags, border, CurWidth)

HWND hWnd;	// The handle of the window to be accessed
int RowId;	// The row id to extract information. Set to a negative value to specify a cell id in the row.
LPINT height;	// The location to receive the current row height in printer units.
LPINT MinHeight;	// The location to receive the minimum height specification for the row. This value is a negative number to indicate exact row height in twips, or a positive number to indicate the minimum row height in twips, or zero to indicate auto row height.
LPINT FixWidth;	// The location to receive the width specification for the table. This value is negative number to indicate the row with as percentage of the current screen width. A positive value indicates the row width in twips.
LPINT PrevRow;	// The location to receive the previous row id.
LPINT NextRow;	// The location to receive the next row id.
LPINT indent;	// The location to receive the row indent in twips.

LPINT flags;	// The location to receives the row flags. The row flags (ROWFLAG_?) constants are defined in the ter.h file.
	ROWFLAG_HDR      Header row
	ROWFLAG_RTL      Right-to-left cell placement row
LPINT border;	// The location to receive the table border specification. This parameter is reserved for future.
LPINT CurWidth;	// This location receives the current table width in twips units.

**Return Value:** The function returns TRUE if successful.

**See Also:**  
[TerRowHeight](#)



## TerGetTableId

**Retrieve the table id.**

int TerGetTableId(hWnd, row)

HWND hWnd;	// The handle of the window to be accessed
int row;	// A table row id within a table for which to retrieve the table id. You can set this parameter to -1 to indicate the current table

**Return Value:** This function returns 0 or a positive value for the table id. A value of -1 indicates an error.

**See Also:**  
[TerSetTableId](#)



## TerGetTableLevel

**Get the nested table level.**

int TerGetTableLevel(hWnd, LineNo)

HWND hWnd;	// The handle of the window to be accessed
------------	--

int LineNo; // The line number to get the table level. Set to a negative value to specify a cell id instead of a line number.

**Return Value:** This function returns 0 or a positive value to indicate the table level number. It returns 0 if the line (or cell id) is at outer most table or if the line does not belong to a table.



## TerGetTablePos

**Get the current table position.**

int TerGetTablePos(hWnd, TableNo, RowNo, ColNo)

int TerGetTablePos2(hWnd, TableNo, RowNo, ColNo, ParentCell)

HWND hWnd; // The handle of the window to be accessed

LPINT TableNo; // The location to receive the current table number. The tables are assigned a sequential number starting with 0 from the beginning of the document.

LPINT RowNo; // The location to receive the current table row number. The table rows are assigned a sequential number starting with 0 from the beginning of the current table.

LPINT ColNo; // The location to receive the current table column number. The table columns are assigned a sequential number starting with 0 from the beginning of the current table row.

int ParentCell; // The parent cell id to locate a nested table. Set to 0 for default.

**Return Value:** This function returns TRUE when successful. It returns a FALSE value if the cursor is not positioned inside a table.

**See Also:**  
[TerPosTable](#)



## TerHtmlCellWidthFlag

**Set the cell width flag when using the HTML add-on.**

BOOL TerHtmlCellWidthFlag(hWnd, select, flag, repaint)

HWND hWnd; // The handle of the window to be accessed

int select; // Cell selection for this operation:

SEL\_ALL: Select the entire table

SEL\_CELLS: Select the current cell or all highlighted cells

SEL\_COLS: Select the current column or all highlighted columns

SEL\_ROWS: Select the current row or all highlighted rows

Set the 'select' parameter to 0 to invoke the user selection dialog box.

UINT flag; // Use one of the following values:

0: Best Fit

CFLAG\_FIX\_WIDTH Current Width

CFLAG\_FIX\_WIDTH\_PCT Current width as the percentage of the table width.

BOOL repaint; // TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.



## TerInsertTableCol

**Insert or append a table column.**

BOOL TerInsertTableCol(hWnd, insert, AllRows, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL insert; // TRUE to insert a table column before the current

column, or FALSE to append a column to the table.

BOOL AllRows;                      // TRUE to insert/append a column to all table rows,  
FALSE to insert/append a column to the current row only

BOOL repaint;                      // TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerPosTable](#)  
[TerSetTableColWidth](#)  
[TerInsertTableRow](#)  
[TerCellBorder](#)  
[TerCellShading](#)



## TerInsertTableRow

**Insert or append a table row.**

BOOL TerInsertTableRow(hWnd, insert, repaint)

HWND hWnd;                      // Handle of the window to be accessed

BOOL insert;                      // TRUE to insert a table row before the current row, or  
FALSE to append a row to the table.

BOOL repaint;                      // TRUE to repaint the screen after this operation

**Description:** The cursor must be positioned in a table cell before calling this function.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerPosTable](#)  
[TerSetTableColWidth](#)  
[TerInsertTableCol](#)  
[TerCellBorder](#)  
[TerCellShading](#)



## TerIsTableSelected

**Check if any table row or cell is selected.**

BOOL TerIsTableSelected(hWnd)

HWND hWnd;                      // Handle of the window to be accessed

**Return Value:** This function returns a TRUE value if a table row or cell is selected.

See Also  
[TerGetSelection](#)



## TerMarkCells

**Set the cell selection flags for the selected text.**

BOOL TerMarkCell(hWnd, select)

HWND hWnd;                      // Handle of the window to be accessed

int select;                      // Cell selection for this operation:

SEL\_ALL:                      Select the entire table

SEL\_CELLS:                      Select the current cell or all highlighted cells

SEL\_COLS:                      Select the current column or all highlighted columns

SEL\_ROWS:                      Select the current row or all highlighted rows

**Description:** This function can be called after selecting the table cells to set the cell selection flags (CFLAG\_SEL1 and CFLAG\_SEL2). You can then use the TerGetCellInfo function to retrieve the cell flags and then test against the CFLAG\_SEL1 and CFLAG\_SEL2 constants to check if the cell is selected:

```
TerGetCellInfo(CellId, &RowId, &PrevCell, &NextCell, &width,  
               &border, &shading, &RowSpan, &ColSpan, &flags);  
if (flags and (CFLAG_SEL1 or CFLAG_SEL2)) then .. cell  
                                   selected.
```

**Return Value:** This function returns a TRUE value if successful.



## TerPosAfterTable



#### Position after the current table.

BOOL TerPosAfterTable (hWnd, OuterMost, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL OuterMost; // Set to TRUE to position after the outer-most table, or set to FALSE to position after the current nested table. Set to TRUE for default.

BOOL repaint; // TRUE to refresh the screen after this operation.

**Description:** This function is available in the Page Mode only. This function places the cursor after the current table. The cursor must already be placed inside a table before calling this function.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**  
[TerPosHdrFtr](#)  
[TerPosTable](#)  
[TerPosBodyText](#)



## TerPosTable

#### Position the cursor at a table cell.

BOOL TerPosTable (hWnd, TableNo, RowNo, ColNo, pos, repaint)

BOOL TerPosTable2 (hWnd, TableNo, RowNo, ColNo, pos, ParentCell, repaint)

BOOL TerPosTable3 (hWnd, TableId, RowNo, ColNo, pos, repaint)

HWND hWnd; // Handle of the window to be accessed

int TableNo; // The table number of the table to position on. The first table in the document is considered the table number zero. You can also set the TableNo to -1 to specify the current table.

int TableId; // The TerPosTable3 function uses TableId instead of TableNo. The table id is an application assigned id for a table. A table id can be assigned using the TerSetTableId function.

int RowNo; // The table row number (zero based).

int ColNo; // The column number (zero based).

int pos;	// Set to POS_BEG to position before the first character of the existing cell text. Set to POS_END to position at the end of the existing cell text.
Int ParentCell;	// Parent cell id for the nested tables. Set to 0 for default.
BOOL repaint;	// TRUE to refresh the screen after this operation.

**Description:** This function is available in the Page Mode only.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**  
[TerPosHdrFtr](#)  
[TerGetTablePos](#)  
[TerPosAfterTable](#)  
[TerGetTableId](#)



## TerReformatTable

**Recalculate the widths for the auto-width tables cells and repaginate the document.**

int TerReformatTable(hWnd, repaint)

HWND hWnd;	// Handle of the window to be accessed
BOOL repaint;	// TRUE to repaint the document after this operation

**Return Value:** The function returns TRUE when successful



## TerRowHeight

**Set the minimum table row height.**

BOOL TerRowHeight(hWnd, MinHeight, AllRows, refresh)

HWND hWnd;	// The handle of the window to be accessed.
int MinHeight;	// Minimum table row height in twips.
BOOL AllRows;	// TRUE to apply the given height to all the rows in the table. FALSE to apply the height to the current row or all

the highlighted rows.

BOOL refresh;                      // TRUE to repaint the screen after this operation.

**Return Value:** The function returns TRUE when successful.

**See Also:**  
[TerRowPosition](#)  
[TerGetRowInfo](#)



## TerRowPosition

**Position a table row.**

BOOL TerRowPosition(hWnd, JustFlag, AllRows, refresh)

BOOL TerRowPositionEx(hWnd, JustFlag, indent, AllRows, refresh)

HWND hWnd;                      // The handle of the window to be accessed.

UINT JustFlag;                      // The position of the table row:  
    LEFT:                      Left justified  
    RIGHT\_JUSTIFY: Right justified  
    CENTER:                      Centered

int indent;                      // The row indentation in twips unit. The JustFlag  
parameter is ignored when the 'indent' parameter is non-  
zero.

BOOL AllRows;                      // TRUE to apply the given position to all the rows in the  
table. FALSE to apply the position to the current row or  
the all highlighted rows.

BOOL refresh;                      // TRUE to repaint the screen after this operation.

**Return Value:** The function returns TRUE when successful.

**See Also:**  
[TerRowHeight](#)



## TerSelectCellText

**Select entire text in the current table cell.**

BOOL TerSelectCellText(hWnd, repaint)

HWND hWnd; // The handle of the window to be accessed.

BOOL repaint; // TRUE to refresh the screen after this operation.

**Return Value:** This function returns TRUE when successful.

**See Also**

[TerSelectCol](#)

[TerSelectTable](#)



## TerSelectCol

**Select the current table column.**

BOOL TerSelectCol(hWnd, repaint)

HWND hWnd; // The handle of the window to be accessed.

BOOL repaint; // TRUE to refresh the screen after this operation.

**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerSelectRow](#)

[TerSelectCellText](#)

[TerSelectTable](#)



## TerSelectRow

**Select the current table row.**

BOOL TerSelectRow(hWnd, repaint)

HWND hWnd; // The handle of the window to be accessed.

BOOL repaint; // TRUE to refresh the screen after this operation.

**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerSelectCol](#)

[TerSelectTable](#)



## TerSelectTable

**Select the current table.**

BOOL TerSelectTable(hWnd, level, repaint)

HWND hWnd;	// The handle of the window to be accessed.
int level;	// Table level number. Set this value to -1 to specify the current level. Set this value to 0 to specify the outmost table.
BOOL repaint;	// TRUE to refresh the screen after this operation.

**Return Value:** This function returns TRUE when successful.

### See Also

[TerSelectCellText](#)  
[TerSelectRow](#)  
[TerSelectCol](#)



## TerSetCellInfo2

**Set additional information for a cell id.**

BOOL TerSetCellInfo2(hWnd, CellId, BackColor, margin, ParentCell)

HWND hWnd;	// The handle of the window to be accessed.
int CellId;	// The cell id to set information.
COLORREF BackColor;	// The background color (RGB) for the cell.
int margin;	// The cell margin in twips. Set to -1 to leave this value unchanged.
int ParentCell;	// The parent cell id for this cell. Set to -1 to leave this value unchanged.

**Return Value:** This function returns TRUE when successful.

### See Also:

[TerCreateCellId](#)



## TerSetCellParam

### Set cell parameters.

BOOL TerSetCellParam(hWnd, select, type, val, repaint)

HWND hWnd; // Handle of the window to be accessed

int select; // Cell selection for this operation:

SEL\_ALL: Select the entire table

SEL\_CELL Select the current cell or all highlighted cells  
S:

SEL\_COLS Select the current column or all highlighted  
: columns

SEL\_ROW Select the current row or all highlighted rows  
S:

Set the 'select' parameter to negative value to specify a specific cell id. For example, to specify CellId value 2, set this parameter to -2.

int type; Parameter type to set. Select one of the constants:

CP\_PAD\_LEFT Specify left padding in twips.

CP\_PAD\_RIGHT Specify right padding in twips.

CP\_PAD\_TOP Specify top padding in twips.

CP\_PAD\_BOT Specify bottom padding in twips.

int val; New value for the parameter specified by the 'type' parameter.

BOOL repaint; TRUE to repaint the screen after this operation

**Return Value:** This function returns a TRUE value if successful.



## TerSetHdrRow

**Set the header row for a table.**

BOOL TerSetHdrRow(hWnd, CellId, set, repaint)

HWND hWnd;	// The handle of the window to be accessed
int CellId;	// A cell id for a cell in the row. Set to 0 to assume the current table row.
BOOL set;	// Set to TRUE to turn the current table row (or selected rows) into a header row. Set to FALSE to remove this attribute.
BOOL repaint;	// Set to TRUE to repaint the screen after this operation

**Return Value:** This function returns TRUE when successful.

### See Also:

[TerSetRowKeep](#)



## TerSetRowKeep

**Set/reset the flag to keep a table row in one page.**

BOOL TerSetHdrRow(hWnd, CellId, set, repaint)

HWND hWnd;	// The handle of the window to be accessed
int CellId;	// A cell id for a cell in the row. Set to 0 to assume the current table row.
BOOL set;	// TRUE to set this flag, or FALSE to reset this flag.
BOOL repaint;	// Set to TRUE to repaint the screen after this operation.

**Description:** The editor moves the entire table row to the next page when this flag is set for the row and the page break occurs within the row.

**Return Value:** This function returns TRUE when successful.

---

### See Also:

[TerSetHdrRow](#)



## TerSetRowTextFlow

**Set the right-to-left/left-to-right text flow option for the table row.**

BOOL TerSetRowTextFlow(hWnd, dialog, AllRows, TextFlow, refresh)

HWND hWnd;	// The handle of the window to be accessed
BOOL dialog;	// Set to TRUE to show the user dialog.
BOOL AllRows;	// Set to TRUE to apply the changes to all the rows in the table. Set to FALSE to apply the changes to the current row or the selected rows
int TextFlow;	// The text flow constant can be one of the following:
FLOW_LTR	Left-to-right text flow
FLOW_RTL	Right-to-left text flow
FLOW_DEF	Default text flow. The flow will be determined by the document, or section level text flow specification.
BOOL refresh;	// TRUE to refresh the window after this operation.

**Return Value:** This function returns TRUE if successful.

**See Also:**

[TerSetDocTextFlow](#)  
[TerSetSectTextFlow](#)



## TerSetTableColWidth

**Set the width of the table columns.**

BOOL TerSetTableColWidth(hWnd, width, repaint)

HWND hWnd;	// Window handle to access
int width;	// column width specified in twips.



```
BOOL repaint;           // TRUE to repaint the screen after this operation
```

**Description:** This function sets the width of the table column where the cursor is positioned.

**Return Value:** This function returns a TRUE value if successful.

### See Also:

## TerPosTable

TerInsertTableCol

TerInsertTableRow

## TerCellBorder

## TerCellShading



## TerSetTableId

## Set an id for a table.

BOOL TerSetTableId(hWnd, row, id)

```
HWND hWnd;           // The handle of the window to be accessed
```

```
int row;           // A table row id within a table for which to set the table
                   // id. You can set this parameter to -1 to indicate the
                   // current table.
```

```
int id;           // The table id. Use a negative number to place the id on
                  // the current table row instead of the entire table. A value
                  // of 0 assigns the default table id.
```

**Return Value:** This function returns TRUE when successful.

### See Also:

TerGetTableId



## Hyperlink

## In This Chapter

TerApplyHyperlink

TerDeleteHypertext

TerFindHlinkField

TerGetHypertextEx

[TerInsertHyperlink](#)

TerSetLinkDbClick

TerUpdateHyperlinkCode

TerUpdateHyperlinkText



## TerApplyHyperlink

**Apply hyperlink field to the selected text.**

int TerApplyHyperlink(hWnd, code, repaint)

HWND hWnd; // Handle of the window to be accessed

LPBYTE code; // The url or other information for the hyperlink

BOOL repaint; //Repaint the window after this operation

**Comment:** Please note that the hyperlink cursor must be enabled using the ID\_SHOW\_HYPERLINK\_CURSOR to show the hyperlink cursor.

**Return Value:** This function returns TRUE if successful.

### See Also:

[TerInsertHyperlink](#)

[TerUpdateHyperlinkText](#)

[TerUpdateHyperlinkCode](#)

[TerFindHlinkField](#)



## TerDeleteHypertext

**Delete the hypertext phrase and the associated hidden text.**

BOOL TerDeleteHypertext(hWnd, LineNo, ColNo, repaint)

HWND hWnd; // The handle of the window to be accessed

long LineNo; // Line number of the hypertext phrase. Set to -1 to use the current line number and column number.

int ColNo; // Column number of the hypertext phrase. This parameter is not used when the 'LineNo' argument is set to -1.

BOOL repaint; // repaint the screen after this operation.

**Return Value:** This function returns TRUE if a hypertext phrase is found at the given location and is deleted successfully.



## TerFindHlinkField

**Locate the next hyperlink field.**

BOOL TerFindHlinkField(hWnd, CodePart1, CodePart2, pLine, pCol)

HWND hWnd;	// The handle of the window to be accessed
LPBYTE CodePart1;	// The part of the hyperlink code or url to search for
LPBYTE CodePart2;	// Another part of the hyperlink code or url to search for
long far *pLine;	// The line number to start the search. This variable also receives the line number of the hyperlink on successful search.
Long far *pCol;	// The column number to start the search. This variable also receives the column number of the hyperlink on successful search.

**Description:** This function examines the hyperlinks in the document. A hyperlink is matched if either CodeString1 or CodeString2 is found in the hyperlink code or url.

**Return Value:** This function returns TRUE when a hyperlink is located. On successful search, the line and column number of the hyperlink is returned using the pLine and pCol variables.

### see Also:

[TerInsertHyperlink](#)  
[TerUpdateHyperlinkCode](#)  
[TerUpdateHyperlinkText](#)



## TerGetHypertextEx

**Retrieve the hypertext information at the cursor position.**

BOOL TerGetHypertextEx(hWnd, text, code, select)

BOOL TerGetHypertext2(hWnd, LineNo, ColNo, text, code, select)

BOOL TerGetHypertextU(hWnd, LineNo, ColNo, UnicodeText, code, select)

HWND hWnd;	// The handle of the window to be accessed
long LineNo;	// The line number to examine. Set to -1 to use the current line and current column number. The TerGetHypertextEx function automatically examines the

	current text position.
int ColNo;	// The text column number to examine.
LPBYTE text;	// Pointer to receive the text part of the current hypertext
LPWORD UnicodeText;	// Pointer to receive the text part of the current hypertext. This parameter is used only by the TerGetHypertextU function.
LPBYTE code;	// Pointer to receive the hidden code part of the current hypertext.
BOOL select;	// TRUE to select (highlight) the current hypertext code and phrase.

**Return Value:** This function returns TRUE if hypertext is found at the current cursor location.



## TerInsertHyperlink

### Insert hyperlink.

int TerInsertHyperlink(hWnd, text, code, PictId, repaint)

int TerInsertHyperlinkU(hWnd, UnicodeText, code, repaint)

HWND hWnd;	// The handle of the window to be accessed
------------	--

LPBYTE text;	// The text phrase for the hyperlink.
--------------	---------------------------------------

The style and color for the hyperlink can be modified by changing the LinkStyle and LinkColor variables using the GetTerFields/SetTerFields function before inserting the hyperlink.

This parameter is ignored when the PictId parameter is non-zero.

LPWORD UnicodeText;	// The unicode text phrase for the hyperlink. This parameter is used only by the TerInsertHyperlinkU method.
---------------------	--

LPBYTE code;	// The url or other information for the hyperlink. This information is not displayed on the screen.
--------------	--

int PictId;	// The picture id for the hyperlink if this a picture link. Set this parameter to 0 to insert text type hyperlink.
-------------	---

BOOL repaint;                               //Repaint the window after this operation

**Comment:** Please note that the hyperlink cursor must be enabled using the ID\_SHOW\_HYPERLINK\_CURSOR to show the hyperlink cursor.

**Return Value:** This function returns the font id or the picture id for the newly inserted hyperlink. It returns -1 to indicate an error condition.

#### See Also:

[TerApplyHyperlink](#)  
[TerUpdateHyperlinkText](#)  
[TerUpdateHyperlinkCode](#)  
[TerFindHlinkField](#)



## TerSetLinkDbClick

**Set mouse click type (single or double click) to invoke a link.**

bool TerSetLinkDbClick(hWnd, DbClick)

HWND hWnd;                               // Handle of the window to be accessed

BOOL DbClick;                           //Set to TRUE to invoke hyperlink on a double-click. Set to FALSE to invoke hyperlink on a single click.

**Return Value:** This function returns the previous value of the DbClick variable.



## TerUpdateHyperlinkCode

**Update the hyperlink url**

BOOL TerUpdateHyperlinkCode(hWnd, code)

HWND hWnd;                               // The handle of the window to be accessed

LPBYTE code;                           // New code or url information for the hyperlink. Set the 'code' parameter to "" to convert the hyperlink text to normal text.

**Description:** This function is used to modify the hyperlink code or url for the hyperlink under the cursor.

**Return Value:** This function returns TRUE when successful.

#### See Also

[TerInsertHyperlink](#)

[TerUpdateHyperlinkText](#)

[TerFindHlinkField](#)



## TerUpdateHyperlinkText

### Update the hyperlink text

BOOL TerUpdateHyperlinkText(hWnd, text, repaint)

BOOL TerUpdateHyperlinkTextU(hWnd, UnicodeText, repaint)

HWND hWnd // The handle of the window to be accessed

LPBYTE text; // New text phrase for the hyperlink.

LPWORD UnicodeText; // New unicode text phrase for the hyperlink. This parameter is used only the the TerUpdateHyperlinkTextU function.

BOOL repaint; // refresh the screen after this operation.

**Description:** This function is used to modify the hyperlink text for the hyperlink under the cursor.

**Return Value:** This function returns TRUE when successful.

#### See Also

[TerInsertHyperlink](#)

[TerUpdateHyperlinkCode](#)

[TerFindHlinkField](#)



## Mail-merge

This chapter includes the mail-merge APIs. Please refer to the [Mail Merge Support](#) chapter for additional information.

#### In This Chapter

[TerChangeField](#)

[TerChangeFieldPicture](#)

[TerChangeFieldRtf](#)

[TerDeleteField](#)

[TerGetField](#)

[TerInsertField](#)

[TerLocateField](#)  
[TerMergeFields](#)  
[TerSelectField](#)



## TerChangeField

**Change the value for a data field.**

BOOL TerChangeField(hWnd, name, data, repaint)

BOOL TerChangeFieldU(hWnd, name, UnicodeData, repaint)

HWND hWnd; // Handle of the window to be accessed

LPBYTE name; // Name of the field to modify

LPBYTE data; // New data text for the field. This parameter is used by the TerChangeField function only.

LPWORD UnicodeData; // New unicode data text for the field. This parameter is used by the TerChangeFieldU function only.

BOOL repaint; // TRUE to repaint the screen after this operation

**Description:** This function changes the data for all occurrence of the specified field name.

**Return Value:** This function returns a TRUE value if successful.

### See Also:

[TerInsertField](#)  
[TerDeleteField](#)



## TerChangeFieldPicture

**Apply a picture to a field.**

BOOL TerChangeFieldPicture(hWnd, name, PictPath, repaint)

HWND hWnd; // Handle of the window to be accessed

LPBYTE name; // Name of the field to modify

LPBYTE PictPath // The picture file name or path to insert

BOOL repaint;                               // TRUE to repaint the screen after this operation

**Description:** This function applies the picture to all occurrence of the specified field name.

**Return Value:** This function returns a TRUE value if successful.



## TerChangeFieldRtf

**Apply the rtf data to a field.**

BOOL TerChangeFieldRtf(hWnd, name, rtf, RtfLen, repaint)

BOOL TerChangeFieldRtfU(hWnd, name, UnicodeRtf, RtfLen, repaint)

HWND hWnd;                               // Handle of the window to be accessed

LPBYTE name;                            // Name of the field to modify

LPBYTE rtf;                             // New RTF data for the field. This parameter is used by the TerChangeFieldRtf function only.

LPWORD UnicodeRtf;                    // New RTF data in the unicode format for the field. This parameter is used by the TerChangeFieldRtfU function only.

int RtfLen;                             // The character length of the RTF data.

BOOL repaint;                           // TRUE to repaint the screen after this operation

**Description:** This function changes the data for all occurrence of the specified field name.

**Return Value:** This function returns a TRUE value if successful.



## TerDeleteField

**Delete the data field at the current cursor position.**



BOOL TerDeleteField(hWnd, repaint)

HWND hWnd; // The handle of the window to be accessed

BOOL repaint; // TRUE to repaint the screen after this operation

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerInsertField](#)  
[TerLocateField](#)  
[TerChangeField](#)



## TerGetField

**Retrieve the text for a field name or field data.**

long TerGetField(hWnd, LineNo, ColNo, type, text)

long TerGetFieldU(hWnd, LineNo, ColNo, type, UnicodeText)

HWND hWnd; // The handle of the window to be accessed

long LineNo; // The line number to examine. Set to -1 to use the current line and current column number.

int ColNo; // The text column number to examine.

int type; // Set to one of the following constants:

FIELD\_NAME: Retrieve the field name.

FIELD\_DATA: Retrieve the field data.

LPBYTE text; // Pointer to receive the text part for the field name or field data. Set this field to NULL if you wish only to get the text length. This parameter is used by the TerGetField function only.

LPWORD UnicodeText; // Pointer to receive the unicode text part for the field name or field data. Set this field to NULL if you wish only to get the text length. This parameter is used by the TerGetFieldU function only.

**Return Value:** This function returns the length of the retrieved text.

**See Also:**  
[TerInsertField](#)

[TerLocateField](#)



## TerInsertField

### Insert a data field.

BOOL TerInsertField(hWnd, name, data, repaint)

BOOL TerInsertFieldU(hWnd, name, UnicodeData, repaint)

HWND hWnd; // The handle of the window to be accessed

LPBYTE name; // Field name

LPBYTE data; // Field data. Set to NULL to insert a field without field data. This parameter is used by the TerInsertField function only.

LPWORD UnicodeData; // Field data in unicode format. Set to NULL to insert a field without field data. This parameter is used by the TerInsertFieldU function only.

BOOL repaint; // TRUE to repaint the screen after the operation.

**Description:** This function inserts a field name and field data in the document. Please note that a data field is different from a mail merge field. A field inserted using this function can not be used for mail merge. Refer to 'Mail Merge Support' chapter for information about mail merge fields.

**Return Value:** This function returns TRUE when successful.

### See Also:

[TerGetField](#)

[TerLocateField](#)

[TerChangeField](#)

[TerDeleteField](#)



## TerLocateField

### Locate a data field.

BOOL TerLocateField(hWnd, location, name, exact, repaint)

HWND hWnd; // Handle of the window to be accessed

int location; // Use one of the following constants:

TER_FIRST:	Search from the top of the file and locate the first occurrence of the field.
TER_LAST:	Search from the bottom of the file and locate the last occurrence of the field.
TER_NEXT:	Find the next occurrence of the field.
TER_PREV:	Find the previous occurrence of the field.

LPBYTE name: // Field name to search for.

BOOL exact: // Set to TRUE to match the field names in the document exactly to the field name given by the 'name' parameter. Set to FALSE to match only the length of the name parameter. Consider a document containing two fields 'company1' and 'company2'. If the name parameter is set to 'company' and the 'exact' parameter is set to FALSE, then both the fields will be matched.

To match any field name, set the 'name' argument to "" and the 'exact' argument to FALSE.

BOOL repaint: // Set to TRUE to repaint the screen after this operation.

**Please note that the repaint operation can change the cursor position to adjust for any invisible text in the line. Therefore, the 'repaint' parameter should be set to FALSE if your application relies on the cursor position set by this function for subsequent APIs.**

**Return Value:** This function returns a TRUE value when successful. When successful, it positions the cursor on the field name.

**See Also:**  
[TerGetField](#)  
[TerInsertField](#)  
[TerChangeField](#)  
[ActiveX Control](#)  
[TerLocateFieldChar](#)



## TerMergeFields

**Replace field names with field data strings.**

BOOL TerMergeFields(hWnd,names,data,repaint)

HWND hWnd;	// Handle of the window to be accessed
LPBYTE names;	// This argument points to a list of field names. The field names must be separated by a ' ' character. The list must be NULL terminated.
LPBYTE data;	// This argument points to a list containing data strings for the corresponding field names in the 'names' argument. The data strings must be separated by a ' ' character. The list must be NULL terminated.
BOOL repaint;	//Repaint the window after this operation

**Description:** This function is used to replace the field names in the current editing window with the corresponding field data strings. Refer to the 'Mail/Merge Support' chapter on how to denote field names during the editing session.

If the document uses a field name which is not contained in the field name table, the editor sends a TER\_MERGE message to the parent window. The 'lParam' parameter for this message contains the pointer to the field name string. If your application processes this message, it should return the pointer to the field data string.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerMergePrint](#)



## TerSelectField

**Select the data field at the current cursor position.**

BOOL TerSelectField(HWND, SelectData, repaint)

HWND hWnd;	// The handle of the window to be accessed
BOOL SelectData	// Set to TRUE to select field-data, set to false to select field-name.
BOOL repaint;	// TRUE to repaint the screen after this operation

**Return Value:** This function returns a TRUE value if successful.



## Picture and OLE objects

**In This Chapter**

[TerCreateOcxObject2](#)  
[TerDeleteObject](#)  
[TerGetControllId](#)  
[TerGeTOCxObject](#)  
[TerGetOlePtr](#)  
[TerGetPictCropping](#)  
[TerGetPictInfo](#)  
[TerGetPictOffset](#)  
[TerInsertControl](#)  
[TerInsertObjectId](#)  
[TerInsertOleFile](#)  
[TerInsertOleObject](#)  
[TerInsertPictureFile](#)  
[TerPastePicture](#)  
[TerPictAltInfo](#)  
[TerPictLinkName](#)  
[TerPictureFromFile](#)  
[TerPictureFromWmf](#)  
[TerSetBkPictId](#)  
[TerSetLinkPictDir](#)  
[TerSeTOCxIntProp](#)  
[TerSeTOCxTextProp](#)  
[TerSetPictCropping](#)  
[TerSetPictFrame2](#)  
[TerSetPictInfo](#)  
[TerSetPictOffset](#)  
[TerSetPictSize](#)  
[TerSetPlaceholderPict](#)  
[TerSetWatermarkPict](#)  
[TerShrinkPictureToPage](#)  
[TerXlateControllId](#)



## TerCreateOcxObject2

**Embed an ActiveX control in the document.**

```
int TerCreateOcxObject2(hWnd, ProgId, width, height)
```

```
HWND hWnd;           // Editor window to access
```

```
int ProgId;           // The program id of the ActiveX control to embed.
```

```
int width;            // The width of the ActiveX control.
```

```
int height;           // The height of the ActiveX control
```

### Example:

```
TerCreateOcxObject2(hWnd, "SAMPLECONTROL.SampleControlCtrl.1",  
                    2000, 2000)
```

**Return Value:** When successful this function returns object id of the control. Otherwise it returns 0.

**See Also:**

[TerGetTOCxObject](#)  
[TerSeTOCxIntProp](#)  
[TerSeTOCxTextProp](#)



## TerDeleteObject

**Delete a font, picture or an ole object id.**

BOOL TerDeleteObject(hWnd, id)

HWND hWnd;	// The handle of the window to be accessed
int id;	// A font id to delete. A font id might represent a font, picture or an ole object. Your application must ensure that the font id is <b>not</b> in use in the document before using this function to delete it.

**Return Value:** This function returns TRUE if successful.

**See Also:**

[TerInsertPictureFile](#)



## TerGetControlId

**Retrieve the control id for a picture id.**

int TerGetControlId(hWnd, PictId)

HWND hWnd;	// The handle of the window to be accessed
int PictId;	// Picture id to translate into control id

**Return Value:** This function returns the control id for the picture id. It returns -1 when unsuccessful.

**See Also:**

[TerInsertControl](#)  
[TerXlateControlId](#)



## TerGeTOCXObject

**Retrieve an ActiveX object reference.**

TerGeTOCXObject(ObjectId)

HWND hWnd; // The handle of the window to be accessed

Long ObjectId; // The object id as returned by the TerCreateOcxObject2 function.

**Description:** This method is available only when TE is used as an ActiveX control.

**Return Value:** This function returns the object reference.

**Example:**

Dim ObjId as long

ObjId=TOC1.TerCreateOcxObject2("MyMathML.Component",3000,3000)

MathMLEq=TerGeTOCXObject(ObjId)

**MathMLEq.Data="<math><mi>a</mi><mo>+</mo><mi>b</mi></math>"**



## TerGetOlePtr

**Get OLE object pointers.**

LPVOID TerGetOlePtr(hWnd, obj, type)

HWND hWnd; // The handle of the window to be accessed

int obj; // The object id.

int type; // The pointer type:

OLEPTR\_OBJECT: Object pointer

OLEPTR\_STORAGE: Istorage pointer

**Return Value:** This function returns the ole pointer for the type specified. It returns NULL to indicate an error condition.

**See Also:**  
[TerInsertOleObject](#)



## TerGetPictCropping

**Retrieve picture cropping values.**

int TerGetPictCropping(hWnd, pict, type)

HWND hWnd;	// The handle of the window to be accessed
int pict;	// Id of the picture. Must be a valid number between 0 and TotalFonts - 1.
int type;	// Cropping type. Use one of the following variables:
	CROP_LEFT: Left cropping
	CROP_RIGHT: Right cropping
	CROP_TOP: Top cropping
	CROP_BOT: Bottom cropping

**Return Value:** This function returns the picture cropping value (in twips unit) for the selected side of the picture. The function returns -1 to indicate an error condition.

See Also

[TerSetPictCropping](#)



## TerGetPictInfo

**Retrieve assorted information for a picture type object.**

BOOL TerGetPictInfo(hWnd, pict, style, rect, align, aux)

HWND hWnd;	// The handle of the window to be accessed
int pict;	// Id of the picture. Must be a valid number between 0 and TotalFonts - 1.
LPINT style;	// pointer to receive the style bits
LPRECT rect;	// pointer to receive the current screen location and size (in device units) of the picture.
LPINT align;	// pointer to receive the picture alignment flags.
LPINT aux;	// pointer to receive the auxiliary id associated with the picture.

**Comment:** While using this function as an OCX, the 'rect' parameter must be passed as



NULL (or nothing) because 'rectangle' (LPRECT) is not a common data type. Use this API as a DLL function if you need to retrieve the picture rectangle.

**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerSetPictInfo](#)  
[TerPastePicture](#)  
[TerInsertPictureFile](#)  
[TerPictureFromFile](#)  
[TerGetPictOffset](#)  
[TerGetCurFont](#)  
[TerGetFontParam](#)



## TerGetPictOffset

**Retrieve the offset value for picture placement.**

```
int TerGetPictOffset(hWnd, pict)
```

HWND hWnd;                               // The handle of the window to be accessed

int pict;                                 // Id of the picture. Must be a valid number between 0  
and TotalFonts - 1.

**Return Value:** This function returns picture offset when successful. A value of -1 indicates an error condition.

**See Also:**

[TerGetPictInfo](#)  
[TerSetPictInfo](#)  
[TerSetPictOffset](#)



## TerInsertControl

**Insert another control inside the TER control.**

```
int TerInsertControl(hWnd, class, style, width, height, align, id, insert)
```

HWND hWnd;                               // The handle of the window to be accessed

LPBYTE class;                            // control class name

DWORD style;                             // control style bits. The control must be of WS\_CHILD  
style. It must not have the caption and menu.

int width;	// width (twips) of the control.
int height;	// height (twips) of the control.
int align;	// control alignment relative to the baseline of the text: ALIGN_BOT (default), ALIGN_TOP, ALIGN_MIDDLE.
int id;	// control id. Specify a unique id with a value above 2000.
BOOL insert;	// TRUE to insert the control into text. FALSE to simply return the object id without inserting the control into text.

**Description:** This function inserts a control of the specified window class into the current text position. The control notification messages are conveyed to your window (the parent of the TER window) via the WM\_COMMAND message.

When the 'insert' argument is TRUE, the object is inserted at the current cursor location.

**Return Value:** This function returns a non-zero object id, if successful. Otherwise it returns zero.

**See Also:**  
[TerInsertObjectId](#)  
[TerGetPictInfo](#)  
[TerSetPictInfo](#)  
[TerGetControlId](#)



## TerInsertObjectId

**Insert an object into text.**

BOOL TerInsertObjectId(hWnd, ObjectId, repaint)

HWND hWnd;	// Editor window to access
------------	----------------------------

int ObjectId;	// id of an existing object to insert.
---------------	--

BOOL repaint;	// TRUE to repaint the window after this operation
---------------	--

**Description:** This function inserts the specified object at the current cursor location.

**Return Value:** This function returns TRUE when successful.



## TerInsertOleFile

### Insert a file as an OLE object.

int TerInsertOleFile(hWnd, FileName, embed, icon, line, col)

HWND hWnd;	// Editor window to access
LPBYTE FileName;	// The name of the file to insert. The file extension is used to select appropriate ole object server.
BOOL embed;	// Set to TRUE to embed the file, or set to FALSE to create a link to the file.
BOOL icon;	// Set to TRUE to display the object icon, or set to FALSE to display the contents.
Long line;	// The text line number to insert the object. Set to -1 to insert the file at the current cursor location.
int col;	// The text column position to insert the object. This parameter is ignored when the 'line' parameter is set to -1.

**Return Value:** This function returns the object id of the inserted object. A value of 0 indicates an error condition.

**See Also:**  
[TerGetOlePtr](#)



## TerInsertOleObject

### Insert an OLE object into text.

int TerInsertOleObject(hWnd, ProgName)

HWND hWnd;	// Editor window to access
LPBYTE ProgName;	// Program name of the server as found in the registration database. Examples: excel.exe, c:\excel\excel.exe or 123w.exe

**Description:** This function invokes the specified OLE server to allow the user to create an object. The new object is placed at the current cursor location.

**Return Value:** This function returns the object id of the inserted object. A value of 0 indicates an error condition.



## TerInsertPictureFile

**Embed or link a bitmap or a metafile from a disk file.**

```
int TerInsertPictureFile(hWnd, FileName, embed, align, insert)
```

```
int TerInsertPictureFileXY(hWnd, FileName, embed, align, insert, x, y)
```

```
int TerInsertPictureFileXY2(hWnd, FileName, embed, align, insert, PageRelative x2, y2)
```

```
int TerInsertPictureFileU(hWnd, FileNameW, embed, align, insert)
```

HWND hWnd;                      // The handle of the window to be accessed

LPBYTE FileName;                // Name of the disk image file.

LPWORD FileNameW;               // Name of the unicode disk image file. This parameter is used by the TerInsertPictureFileU method only.

BOOL embed;                     // TRUE to embed the picture in the document file, FALSE to create a link to the picture file.

int align;                       // picture alignment relative to the baseline of the text: ALIGN\_BOT (default), ALIGN\_TOP, ALIGN\_MIDDLE.

BOOL insert;                    // TRUE to insert the picture into text. FALSE to simply return the picture id without inserting the picture into text.

int x;                           // Pixel X position relative to the client-window to insert the picture. This parameter is used by the TerInsertPictureFileXY method only.

int y;                           // Pixel Y position relative to the client-window to insert the picture. This parameter is used by the TerInsertPictureFileXY method only.

BOOL PageRelative;              // Set to TRUE to interpret the values of the x2 and y2 relative to the top of the page. Set to FALSE to insert a paragraph-relative picture.

int x2                           // When PageRelative parameter is set to TRUE: specify the x position in twips unit relative to the left edge of the page.

When PageRelative parameter is set to FALSE; specify the x position in twips unit relative to the left margin of the page.

This parameter is used by the TerInsertPictureFileXY2 method only.

int y2

// When PageRelative parameter is set to TRUE: specify the y position in twips unit relative to the top edge of the page.

When PageRelative parameter is set to FALSE; specify the y position in twips unit relative to the current paragraph.

This parameter is used by the TerInsertPictureFileXY2 method only.

**Description:** A file selection dialog box is displayed if the FileName argument is set to NULL. The TerInsertPictureFile function inserts the picture at the current text position. The TerInsertPictureFileXY function inserts the picture at the pixel location given by the x,y position. The x,y pixel location is relative to the top left corner of the client area of the window. The TerInsertPictureFileXY2 method is used to insert a page or paragraph relative picture.

**Return Value:** This function returns a non-zero picture id, if successful. Otherwise it returns zero.

#### See Also:

[TerPastePicture](#)  
[TerPictureFromFile](#)  
[TerPictureFromWmf](#)  
[TerInsertObjectId](#)  
[TerSetBkPictId](#)  
[TerSetPictFrame2](#)  
[TerShrinkPictureToPage](#)



## TerPastePicture

### Paste a picture from the buffer or clipboard.

int TerPastePicture(hWnd, format, hData, ParaFrameld, align, insert)

HWND hWnd; // Handle of the window to be accessed

UINT format; // Picture format:

CF\_DIB: Device independent bitmap format

CF\_BITMAP: Bitmap format

CF\_METAFILEPICT: Metafile picture format

CF\_ENHMETAFILE: Enhanced metafile picture (win32)

HGLOBAL hData; // picture data. The hData value for the supported picture formats should be specified as following:

CF_DIB:	Global memory handle containing the DIB picture header and picture data.
CF_BITMAP:	Bitmap handle
CF_METAFILEPICT:	Global memory handle containing the clipboard metafile header and metafile data.
CF_ENHMETAFILE:	Enhanced metafile handle.
int ParaFrameld;	// Id of the frame to insert the picture into. Set 0 for default.
int align;	// picture alignment relative to the baseline of the text: ALIGN_BOT (default), ALIGN_TOP, ALIGN_MIDDLE.
BOOL insert;	// TRUE to insert the picture into text. FALSE to simply return the picture id without inserting the picture into text.

**Description:** This function pastes a picture contained in the hData global memory handle. If the hData argument is NULL, the picture is retrieved from the clipboard. When the 'insert' argument is TRUE, the picture is inserted at the current cursor location.

**Return Value:** This function returns a non-zero picture id, if successful. Otherwise it returns zero.

**See Also:**  
[TerInsertPictureFile](#)  
[TerPictureFromFile](#)  
[TerInsertObjectId](#)  
[TerGetPictInfo](#)  
[TerSetPictInfo](#)  
[TerSetBkPictId](#)



## TerPictAltInfo

**Set or retrieve the alternate string for a picture.**

BOOL TerPictAltInfo(hWnd, id, get, AltInfo)

HWND hWnd; // Handle of the window to be accessed

int id; // picture id

BOOL get; // TRUE to retrieve the alternate string, FALSE to set a new alternate string.

LPBYTE AltInfo; // The new alternate name string for the picture, or the pointer to receive the current string.

**Comment:** The alternate string is saved using the 'alt' tag when the file is saved in the HTML format.

**Return Value:** This function returns TRUE if successful.



## TerPictLinkName

**Set or retrieve the link file name for a picture.**

BOOL TerPictLinkName(hWnd, id, get, FileName)

BOOL TerPictLinkNameU(hWnd, id, get, FileNameW)

HWND hWnd; // Handle of the window to be accessed

int id; // picture id

BOOL get; // TRUE to retrieve the file name, FALSE to set a new name.

LPBYTE FileName; // The new link file name for the picture, or the pointer to receive the current picture name. This parameter is used by the TerPictLinkName method only.

LPWORD FileNameW; // The new unicode link file name for the picture, or the pointer to receive the current picture name. This parameter is used by the TerPictLinkNameU method only.

**Return Value:** This function returns TRUE if successful.



## TerPictureFromFile

**Paste a picture from a disk bitmap file.**

BOOL TerPictureFromFile(hWnd, FileName, align, insert)

HWND hWnd;	// Handle of the window to be accessed
LPBYTE FileName;	// Name of the disk image file.
int align;	// picture alignment relative to the baseline of the text: ALIGN_BOT (default), ALIGN_TOP, ALIGN_MIDDLE.
BOOL insert;	// TRUE to insert the picture into text. FALSE to simply return the picture id without inserting the picture into text.

**Description:** This function pastes a picture bitmap from the specified disk file. A file selection dialog box is displayed if the FileName argument is set to NULL.

**Return Value:** This function returns a non-zero picture id, if successful. Otherwise it returns zero.

#### See Also:

[TerPastePicture](#)  
[TerInsertPictureFile](#)  
[TerPictureFromWmf](#)  
[TerInsertObjectId](#)  
[TerSetBkPictId](#)



## TerPictureFromWmf

### Paste a picture from a Windows metafile file.

BOOL TerPictureFromWmf(HWND hWnd, LPBYTE FileName, int align, BOOL insert)

HWND hWnd;	// Handle of the window to be accessed
LPBYTE FileName;	// Name of the disk image file.
int align;	// picture alignment relative to the baseline of the text: ALIGN_BOT (default), ALIGN_TOP, ALIGN_MIDDLE.
BOOL insert;	// TRUE to insert the picture into text. FALSE to simply return the picture id without inserting the picture into text.

**Description:** This function pastes a picture from the specified disk file. The file must be in the Windows placeable metafile format. A file selection dialog box is displayed if the FileName argument is set to NULL.

**Return Value:** This function returns a non-zero picture id, if successful. Otherwise it returns zero.

#### See Also:

[TerPastePicture](#)  
[TerInsertPictureFile](#)



[TerPictureFromFile](#)  
[TerInsertObjectId](#)  
[TerSetBkPictId](#)



## TerSetBkPictId

**Set the background picture.**

BOOL TerSetBkPictId(hWnd, PictId, PictFlag, repaint)

HWND hWnd; // The handle of the window to be accessed.

int PictId; // Picture id

UINT PictFlag; // This can be set to one of these values:

BKPICT\_STRETCH: Stretch the picture to occupy the text area

BKPICT\_TILE: Tile the picture to occupy the text area

Or you can set this argument to 0 to draw the picture in its original size without tiling.

BOOL repaint; // Repaint the screen after this operation

**Description:** The picture can be an ID returned by any of these functions: TerPastePicture, TerInsertPictureFile, TerPictureFromFile, TerPictureFromWmf. *The 'insert' argument for these function calls must be set to FALSE.* To remove an existing picture background, call this function with the PictId set to 0. Call this function with the PictId set to -1 to show a dialog box to the user. This dialog box allows the user to select a bitmap or a metafile file name.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerPastePicture](#)  
[TerInsertPictureFile](#)  
[TerPictureFromFile](#)  
[TerPictureFromWmf](#)  
[TerSetWatermarkPict](#)



## TerSetLinkPictDir

**Set the default directory to read linked pictures from an RTF file.**

BOOL TerSetLinkPictDir(hWnd, dir)

HWND hWnd; // The handle of the window to be accessed.

LPBYTE dir; // The default directory. Set to "" to use the program directory.

**Description:** This directory is used to located the linked pictures which do not contain full path specification.

**Return Value:** This function returns TRUE when successful.



## TerSeTOCxIntProp

**Set an integer property for an embedded ActiveX control.**

BOOL TerSeTOCxIntProp(hWnd, ObjectId, PropName, PropValue)

HWND hWnd; // The handle of the window to be accessed

int ObjectId; // An object id as returned by the TerCreateOcxObject2 function.

LPBYTE PropName; // The property name to set.

Long PropValue; // The property value to assign.

**Return Value:** This function returns TRUE if successful.

### See Also:

[TerCreateOcxObject2](#)  
[TerSeTOCxTextProp](#)



## TerSeTOCxTextProp

**Set a text property for an embedded ActiveX control.**

BOOL TerSeTOCxTextProp(hWnd, ObjectId, PropName, PropString)

HWND hWnd; // The handle of the window to be accessed

int ObjectId;	// An object id as returned by the TerCreateOcxObject2 function.
LPBYTE PropName;	// The property name to set.
LPBYTE PropString;	// The string value to assign.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerCreateOcxObject2](#)



## TerSetPictCropping

**Set the picture cropping values.**

BOOL TerSetPictCropping(hWnd, pict, type, CropLeft, CropTop, CropRight, CropBot, repaint)

HWND hWnd;	// The handle of the window to be accessed
int pict;	// Id of the picture. Must be a valid number between 0 and TotalFonts - 1.
int CropLeft;	// Left cropping value in twips.
int CropTop;	// Top cropping value in twips.
int CropRight;	// Right cropping value in twips.
int CropBot;	// Bottom cropping value in twips.
BOOL repaint;	// Repaint the screen after this operation.

**Return Value:** This function returns TRUE when successful.

**See Also**  
[TerGetPictCropping](#)



## TerSetPictFrame2

**Set a floating frame for a picture.**

BOOL TerSetPictFrame2(hWnd, pict, type, x, y, repaint)

HWND hWnd;	// The handle of the window to be accessed
int pict;	// Id of the picture. Must be a valid number between 0 and TotalFonts - 1.
int type;	// Picture frame type: PFRAME_FLOAT: Free floating picture frame PFRAME_LEFT: Left aligned picture frame PFRAME_RIGHT: Right aligned picture frame PFRAME_NONE: No picture frame
int x;	// The x location of the frame relative to the left edge of the page. This value is used only when frame type is set to PFRAME_FLOAT.
int y;	// The y location of the frame relative to the top edge of the page. This value is used only when frame type is set to PFRAME_FLOAT.
BOOL repaint;	// Repaint the screen after this operation.

**Description:** This function enclosed the give picture in a frame. The document text wraps around the frame.

**Return Value:** This function returns the frame id of the picture frame when successful. It returns a value of -1 to indicate an error condition.

**See Also:**  
[TerGetPictInfo](#)  
[TerSetPictSize](#)  
[TerPastePicture](#)  
[TerInsertPictureFile](#)



## TerSetPictInfo

**Set assorted information for a picture type object.**

BOOL TerSetPictInfo(hWnd, pict, style, align, aux)

HWND hWnd;	// The handle of the window to be accessed
int pict;	// Id of the picture. Must be a valid number between 0 and TotalFonts - 1

UINT style;	// Picture style constant
int align;	// Alignment flags: ALIGN_TOP, ALIGN_BOT, ALIGN_MIDDLE
int aux;	// Auxiliary id associated with the picture. This id is not used internally by the editor. It is solely for the use of an application.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerGetPictInfo](#)  
[TerSetPictSize](#)  
[TerPastePicture](#)  
[TerInsertPictureFile](#)  
[TerPictureFromFile](#)



## TerSetPictOffset

**Set the offset from the baseline to place the picture in the text.**

BOOL TerSetPictOffset(hWnd, pict, offset, repaint)

HWND hWnd;	// The handle of the window to be accessed
int pict;	// Id of the picture. Must be a valid number between 0 and TotalFonts - 1.
int offset;	// The offset (twips) for the picture. This value is used to depress the bottom of the picture against the text baseline.
BOOL repaint;	// TRUE to repaint the screen after this operation.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerGetPictInfo](#)  
[TerSetPictInfo](#)  
[TerGetPictOffset](#)



## TerSetPictSize

### Set width and height for a picture type object.

BOOL TerSetPictSize(hWnd, pict, width, height)

HWND hWnd;	// The handle of the window to be accessed
int pict;	// Id of the picture. Must be a valid number between 0 and TotalFonts - 1.
int width;	// New picture width in the screen units. Use the negative values to specify in twips units. Set to -1 to leave the picture width unchanged.
int height;	// New picture height in the screen units. Use the negative values to specify in twips units. Set to -1 to leave the picture height unchanged.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerGetPictInfo](#)  
[TerSetPictInfo](#)



## TerSetPlaceholderPict

### Insert a place-holder picture name or picture path.

BOOL TerSetPlaceholderPict( hWnd, path)

HWND hWnd;	// Handle of the window to be accessed
LPBYTE path;	// Name of the picture file, or path-name of the picture file to insert when a linked picture is missing during the RTF read process.

**Return Value:** This function returns a TRUE value when successful.



## TerSetWatermarkPict

### Set the watermark picture.

BOOL TerSetWatermarkPict(hWnd, PictId, wash, repaint)

```

HWND hWnd;           // The handle of the window to be accessed.

int PictId;           // Picture id

BOOL wash;            // Show the watermark with a 'washed' look.

BOOL repaint;         // Repaint the screen after this operation

```

**Description:** The picture can be an ID returned by any of these functions: TerPastePicture, TerInsertPictureFile, TerPictureFromFile, TerPictureFromWmf. *The 'insert' argument for these function calls must be set to FALSE.* To remove an existing picture background, call this function with the PictId set to 0. Call this function with the PictId set to -1 to show a dialog box to the user. This dialog box allows the user to select a picture file name.

**Return Value:** This function returns TRUE when successful.

**See Also**  
[TerSetBkPictId](#)



## TerShrinkPictureToPage

**Shrink the picture to fit within a page, column, or table cell.**

```

BOOL TerShrinkPictureToPage(hWnd, line, PictId)

```

```

HWND hWnd;           // The handle of the window to be accessed.

long line;            // The document line number where the picture is
                        // located. Set to -1 to specify the current line number.

int PictId;           // Picture id

```

**Description:** The picture can be an ID returned by any of these functions: TerPastePicture, TerInsertPictureFile, TerPictureFromFile, TerPictureFromWmf. *The 'insert' argument for these function calls must be set to FALSE.*

**Return Value:** This function returns TRUE when successful.

**See Also**  
[TerInsertPictureFile](#)



## TerXlateControlId

**Retrieve the picture id for a control id.**

## **int TerXlateControlld(hWnd, CtlId)**

HWND hWnd; // The handle of the window to be accessed

int XlatId; // The control id to translate into the picture id

**Return Value:** This function returns the picture id for the control id. It returns -1 when unsuccessful.

### **See Also**

[TerInsertControl](#)

[TerGetControlld](#)



## **Page Header/Footer**

### **In This Chapter**

[TerCreateFirstHdrFtr](#)

[TerDeleteFirstHdrFtr](#)

[TerCreateLeftRightHdrFtr](#)

[TerDeleteHdrFtr](#)

[TerGetHdrFtrPos](#)

[TerHdrFtrExists](#)

[TerPosHdrFtr](#)



## **TerCreateFirstHdrFtr**

**Create a first page header or footer area.**

BOOL TerCreateFirstHdrFtr(hWnd, HdrFtr)

HWND hWnd; // The handle of the window to be accessed

BOOL HdrFtr; // Set to TRUE to create a first page header area. Set to FALSE to create a first page footer area.

**Return Value:** This function returns TRUE when successful.

### **See Also:**

[TerDeleteFirstHdrFtr](#)



## **TerDeleteFirstHdrFtr**



**Delete a first page header or footer area.**

**BOOL TerDeleteFirstHdrFtr(hWnd, HdrFtr, msg)**

HWND hWnd;	// The handle of the window to be accessed
BOOL HdrFtr;	// Set to TRUE to delete a first page header area. Set to FALSE to delete a first page footer area.
BOOL msg;	// Set to FALSE to suppress user confirmation before the deletion.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerCreateFirstHdrFtr](#)



## TerCreateLeftRightHdrFtr

**Create a left or right page header or footer area.**

**BOOL TerCreateLeftRightHdrFtr(hWnd, HdrFtr)**

HWND hWnd;	// The handle of the window to be accessed	
int HdrFtr;	// Specify one of the constants to create a header or footer:	
	RHDR_CHAR:	Right (odd) page header
	RFTR_CHAR:	Right (odd) page footer
	LHDR_CHAR:	Left (even) page header
	LFTR_CHAR:	Left (even) page footer

**Return Value:** This function returns TRUE when successful.



## TerDeleteHdrFtr

**Delete a page header or footer area.**

**BOOL TerDeleteHdrFtr(hWnd, HdrFtr, msg)**

HWND hWnd;	// The handle of the window to be accessed
BYTE HdrFtr;	// Use one of the following constants to specify the type of the header/footer area to delete:
HDR_CHAR:	Regular header
FTR_CHAR:	Regular footer
FHDR_CHAR:	First page header
FFTR_CHAR:	First Page footer
RHDR_CHAR:	Right (odd) page header
LHDR_CHAR:	Left (even) page header
RFTR_CHAR:	Right (odd) Page footer
LFTR_CHAR	Left (even) Page footer
BOOL msg;	// Set to FALSE to suppress user confirmation before the deletion.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerCreateFirstHdrFtr](#)



## TerGetHdrFtrPos

**See Also**  
[TerPosHdrFtr](#)

**Check if a line is location in a header or footer area**

**int TerGetHdrFtrPos(hWnd, line)**

HWND hWnd;	// The handle of the window to be accessed
long line;	// The line number to find the position. Set to -1 to find the location for the current line.

**Return Value:** This function returns the following values to indicate the line position:

LFLAG_HDR	The line is location in a regular page header
-----------	---

LFLAG_FTR	The line is location in a regular page footer
LFLAG_FHDR	The line is location in first page header
LFLAG_FFTR	The line is location in first page footer
LFLAG_RHDR	The line is location in right (odd) page header
LFLAG_RFTR	The line is location in right (odd) page footer
LFLAG_LHDR	The line is location in left (even) page header
LFLAG_LFTR	The line is location in left (even) page footer
0	The line is location in page body text



## TerHdrFtrExists

### See Also

[TerGetHdrFtrPos](#)

[TerGetSeqSect](#)

### Check if headers or footers exists in the document

**int TerHdrFtrExists(hWnd, SectId)**

HWND hWnd; // The handle of the window to be accessed

int SectId; // The sequential section id of the section to search the headers or footers. You can also set this parameter to SECT\_ALL to search the entire document, or set to SECT\_CUR to search the current section only.

**Return Value:** This function returns a flag value. The following constant bits in the flag value indicate the existence of the headers and footers. You can use the 'And' operator to check if a particular header or footer exists.

A zero return value indicates that no header or footer exists in the document.

LFLAG_HDR	Regular page header
LFLAG_FTR	Regular page footer
LFLAG_FHDR	First page header

LFLAG_FFTR	First page footer
LFLAG_RHDR	Right (odd) page header
LFLAG_RFTR	Right (odd) page footer
LFLAG_LHDR	Left (even) page header
LFLAG_LFTR	Left (even) page footer

#### Example:

```
int flag=TerHdrFtrExists(hWnd, tc.SECT_ALL);
if ((flag and LFLAG_HDR) <> 0) then RegularPageHeader
                                found.
```



## TerPosHdrFtr

### Position the cursor at the header or footer text.

BOOL TerPosHdrFtr (hWnd, section, header, pos, repaint)

BOOL TerPosHdrFtrEx(hWnd,section,HdrFtrId,pos,repaint)

HWND hWnd;                               // Handle of the window to be accessed

int section;                               // section number for the header. Specify a number  
between 0 (first section) and total section -1.

This function uses sequential section numbers within the document. Please note that the sequential section numbers can be different from the actual section id for the section. You can use the TerGetSeqSect function to translate a section id into the sequential section number.

int header;                               // TRUE to position at the header text or FALSE to  
position at the footer text (used by the TerPosHdrFtr  
function only).

int HdrFtrId;                             // Use one of the following constants to specify the type of  
the header/footer area to position at (used by the  
TerPosHdrFtrEx function only):

HDR\_CHAR:                               Regular header

FTR\_CHAR:                               Regular footer

FHDR_CHAR:	First page header
FFTR_CHAR:	First Page footer
RHDR_CHAR:	Right (odd) page header
RFTR_CHAR:	Right (odd) Page footer
LHDR_CHAR:	Left (even) page header
LFTR_CHAR:	Level (even) Page footer

int pos;                      // Set to POS\_BEG to position before the first character of the existing header or footer text. Set to POS\_END to position at the end of the existing header or footer text.

BOOL repaint;                // TRUE to refresh the screen after this operation.

**Description:** This function is available in the Page Mode only. The function toggles the header/footer edit mode before positioning the cursor.

Please note that this function automatically turns on the editing of header/footers, if not already enabled.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**  
[TerPosTable](#)  
[TerGetSeqSect](#)  
[TerGetHdrFtrPos](#)



## Frame and Drawing Objects

### In This Chapter

[TerConnectTextBoxes](#)  
[TerCreateParaFrameId](#)  
[TerGetDrawObjectInfo](#)  
[TerGetFrameParam](#)  
[TerGetFrameSize](#)  
[TerInsertDrawObject](#)  
[TerInsertLineObject](#)  
[TerInsertParaFrame](#)  
[TerMoveParaFrame](#)  
[TerMovePictFrame](#)  
[TerPosFrame](#)  
[TerRotateFrameText](#)  
[TerSelectFrameText](#)  
[TerSetFrameMarginDist](#)  
[TerSetFrameTextDist](#)  
[TerSetNewFrameDim](#)  
[TerSetObjectAttrib](#)

[TerSetObjectWrapStyle](#)  
[TerSetFrameYBase](#)



## TerConnectTextBoxes

**Link two text boxes.**

BOOL TerConnectTextBoxes(hWnd, FromFID, ToFID)

HWND hWnd;	// Handle of the window to be accessed
int FromFID;	// Source frame id. Specify a number between 1 and total frames -1.
int ToFID;	// Target frame id. Specify a number between 1 and total frames -1.

**Comment:** This method is used to link two text boxes. The text overflow from the first text box is displayed in the target text-box. You can use this method to construct a linked list of two or more than two text boxes.

**Return Value:** This function returns a TRUE value when successful.

### See Also:

[TerMoveParaFrame](#)  
[TerPosFrame](#)  
[TerSetNewFrameDim](#)  
[TerInsertParaFrame](#)



## TerCreateParaFrameId

**Create a paragraph frame id without inserting it in the document.**

int TerCreateParaFrameId(hWnd, x, y, width, height)

HWND hWnd;	// Editor window to access
int x;	// X location of the frame rectangle specified in the Twips unit. The x location is relative to the left margin of the page. Specify -1 to assume the current cursor position for the x value.
int y;	// Y position of the frame rectangle specified in the Twips unit. The Y position is relative to the beginning of the current paragraph. Specify -1 to assume the current cursor position for the y value.

int width;	// Initial width of the frame rectangle in the Twips unit. Specify -1 to use the default value.
int height;	// Initial height of the frame rectangle in the Twips unit. Specify -1 to use the default value.

**Return Value:** When successful this function returns the paragraph frame id of the new paragraph frame. Otherwise it return 0.



## TerGetDrawObjectInfo

**Retrieve information about a drawing object.**

BOOL TerGetDrawObjectInfo(hWnd, FrameId, width, height, LineWidth, LineColor, FillColor, flags)

HWND hWnd;	// The handle of the window to be accessed
int FrameId;	// The frame id of the drawing object to inquire
LPINT width;	// The location to received the width (twips) of the text box or the rectangle object
LPINT height;	// The location to received the height (twips) of the text box or the rectangle object
LPINT LineWidth;	// The location to receive the line width (twips) of the line object
COLORREF far *LineColor;	// The location to receive the color of the line object
COLORREF far *FillColor;	// The location to received the fill color for the text box or the rectangle object
UINT far *flags;	// The location to receive the object flags:
PARA_FRAME_TEXT_BOX:	Indicates a text box
PARA_FRAME_LINE:	Indicates a line object
PARA_FRAME_RECT:	Indicates a rectangle object
PARA_FRAME_BOXED:	The text box or the rectangle object is boxed.

PARA\_FRAME\_DOOTED:       Indicates dotted border

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerInsertDrawObject](#)



## TerGetFrameParam

**Get the frame object parameters.**

int TerGetFrameParam(hWnd, id, type)

HWND hWnd;                                // The handle of the window to be accessed

int id;                                     // Frame item id to retrieve parameters.

int type;                                  // The parameter to retrieve:

FP_TEXT_ROTATION:	Return the text rotation type. Please refer to the <a href="#">TerRotateFrameText</a> function for a list of text rotation type constants.
FP_WRAP_STYLE	Return the text wrap style. Please refer to the <a href="#">TerSetObjectWrapStyle</a> function for a list of text wrap style.
FP_YBASE	Return the vertical base position for the frame. Please refer to the <a href="#">TerSetFrameYBase</a> function for a list of base constants.
FP_FILL_PATTERN	Fill pattern: 0=Transparent, 1=Solid
FP_TEXT_DIST	Text distance from the frame in twips.

**Return Value:** The function returns the value for the requested parameter. It returns FP\_ERROR to indicate an error condition.



## TerGetFrameSize



### Retrieve the location and size of the frame.

BOOL TerGetFrameSize(hWnd, Frameld, x, y, width, height)

HWND hWnd;	// The handle of the window to be accessed
int FontId;	// Frame id to inquire. If the frame id is 0, the function returns the values for the frame at the current cursor location.
LPINT x;	// pointer to receive the x location relative to the left edge of the page in twips units.
LPINT y;	// pointer to receive the y location relative to the top edge of the page in twips units.
LPINT width;	// pointer to receive the width of the frame in twips.
LPINT height;	// pointer to receive the height of the frame in twips.

**Return Value:** This function returns TRUE when successful.

#### See Also:

[TerMoveParaFrame](#)



## TerInsertDrawObject

### Insert a drawing object.

int TerInsertDrawObject(hWnd, type, x, y, width, height)

HWND hWnd;	// Editor window to access
int type;	// Drawing object type:
DOB_TEXT_BOX:	A box containing text
DOB_RECT:	A rectangle
DOB_LINE:	A horizontal line. To draw other than a horizontal line, please use the TerInsertLineObject function instead.
int x;	// X location of the frame rectangle specified in the Twips unit. The x location is relative to the left margin of the page. Specify -1 to assume the current cursor position

for the x value.

int y;	// Y position of the frame rectangle specified in the Twips unit. The Y position is relative to the beginning of the current paragraph. Specify -1 to assume the current cursor position for the y value.
int width;	// Initial width of the frame rectangle in the Twips unit. Specify -1 to use the default value.
int height;	// Initial height of the frame rectangle in the Twips unit. Specify -1 to use the default value.

**Description:** This function creates a specified drawing object.

**Return Value:** When successful this function returns the paragraph frame id of the new object. Otherwise it returns 0.

#### See Also:

[TerSetObjectAttrib](#)  
[TerSetFrameYBase](#)  
[TerPosFrame](#)  
[TerSetNewFrameDim](#)  
[TerInsertLineObject](#)  
[TerGetDrawObjectInfo](#)



## TerInsertLineObject

**Insert a line object.**

int TerInsertLineObject(HWND, x1, y1, x2, y2)

HWND hWnd;	// Editor window to access
int x1;	// X position of the first point of the line.
int y1;	// Y position of the first point of the line.
int x2;	// X position of the second point of the line.
int y2;	// Y position of the second point of the line.

**Comments:** The point positions are specified in the twips units. The x locations are relative to the left margin of the page. The Y positions are relative to the beginning of the current paragraph.

**Return Value:** When successful this function returns the paragraph frame id of the new line object. Otherwise it returns 0.



## TerInsertParaFrame

**Insert a paragraph frame.**

```
int TerInsertParaFrame(hWnd, x, y, width, height,boxed)
```

HWND hWnd;	// Editor window to access
int x;	// X location of the frame rectangle specified in the Twips unit. The x location is relative to the left margin of the page. Specify -1 to assume the current cursor position for the x value.
int y;	// Y position of the frame rectangle specified in the Twips unit. The Y position is relative to the beginning of the current paragraph. Specify -1 to assume the current cursor position for the y value.
int width;	// Initial width of the frame rectangle in the Twips unit. Specify -1 to use the default value.
int height;	// Initial height of the frame rectangle in the Twips unit. Specify -1 to use the default value.
BOOL boxed;	// TRUE to create a border around the frame

**Description:** This function creates an empty paragraph frame and positions the cursor inside the frame.

**Return Value:** When successful this function returns the paragraph frame id of the new paragraph frame. Otherwise it returns 0.

### See Also:

[TerMoveParaFrame](#)  
[TerPosFrame](#)  
[TerSetNewFrameDim](#)  
[TerCreateParaFrameId](#)  
[TerSetFrameMarginDist](#)



## TerMoveParaFrame

**Move or resize the current paragraph frame or the drawing object.**

```
BOOL TerMoveParaFrame(hWnd, ParaFID, x, y, width, height)
```

HWND hWnd;	// Handle of the window to be accessed
int ParaFID;	// Id of the frame which is being moved.
int x;	// X location of the frame rectangle specified in the Twips unit. The x location is relative to the left margin of the page.
int y;	// Y position of the frame rectangle specified in the Twips unit. The y position is relative to the top of the page when the page header/footer is visible. Otherwise it is relative to the top margin of the page.
int width;	// New width of the frame rectangle in the Twips unit. Specify -1 to use the current value.
int height;	// New height of the frame rectangle in the Twips unit. Specify -1 to use the current value.

**Description:** This function is used to move or resize an exiting frame.

**Return Value:** This function return a TRUE value when successful.

**See Also:**  
[TerInsertParaFrame](#)  
[TerInsertDrawObject](#)  
[TerGetFrameSize](#)  
[TerMovePictFrame](#)



## TerMovePictFrame

**Move a picture frame.**

BOOL TerMovePictFrame(HWND, PictId, x, y)

HWND hWnd;	// Handle of the window to be accessed
int PictId;	// Id of the picture object. This id value must be between 0 and TotalFonts-1, and must correspond to a picture object.
int x;	// New X location of the picture frame specified in the Twips unit. The X location is relative to the left margin. Set to PARAM_IGNORE to let this value remain unchanged.
int y;	// New Y location of the picture frame specified in the Twips unit. The Y location is relative to the page top, top

margin, or current paragraph as defined by the current frame Y alignment attribute. Set to PARAM\_IGNORE to let this value remain unchanged.

**Description:** This function is used to move or resize an exiting picture frame.

**Return Value:** This function return a TRUE value when successful.

#### See Also

[TerSetPictSize](#)



## TerPosFrame

**Position the cursor in a frame or drawing object.**

BOOL TerPosFrame (hWnd, FrameNo, pos, repaint)

HWND hWnd;	// Handle of the window to be accessed
int FrameNo;	// Frame id to position at. Specify a number between 1 and total frames -1.
int pos;	// Set to POS_BEG to position at the beginning of the frame. Set to POS_END to position at the end of the frame.
BOOL repaint;	// TRUE to refresh the screen after this operation.

**Return Value:** This function returns a TRUE value when successful.

#### See Also:

[TerPosBodyText](#)

[TerInsertParaFrame](#)

[TerInsertDrawObject](#)



## TerRotateFrameText

**Rotate the text within a frame or a text box.**

BOOL TerRotateFrameText(hWnd, dialog, LineNo, direction, repaint)

HWND hWnd;	// The handle of the window to be accessed.
BOOL dialog;	// Set to TRUE to show a dialog box to the user.

long LineNo;                      // The line number within a frame. You can also specify the frame id by specifying a negative value.

int direction:                    // Text flow direction. Choose one of the following values:

TEXT\_HORZ:                      Horizontal text flow

TEXT\_TOP\_TO\_BOT:              Top-to-bottom text flow

TEXT\_BOT\_TO\_TOP:              Bottom-to-top text flow

BOOL repaint;                    // Set to TRUE to repaint the screen after this operation.

Return Value: The function returns TRUE when successful. Otherwise, it returns a FALSE value.

#### See Also:

[TerSetObjectAttrib](#)  
[TerGetFrameParam](#)

## TerSelectFrameText

**Select entire text in the current frame or text drawing object.**

BOOL TerSelectFrameText(hWnd, repaint)

HWND hWnd;                      // The handle of the window to be accessed.

BOOL repaint;                    // TRUE to refresh the screen after this operation.

**Return Value:** This function returns TRUE when successful.



## TerSetFrameMarginDist

**Set the frame margin distance.**

BOOL TerSetFrameMarginDist(hWnd,dist)

HWND hWnd;                      // The handle of the window to be accessed

int dist;                        // The frame margin distance in twips unit (default 1440 twips)

**Description:** The dist argument controls the minimum distance between a frame and the left or right margin at which the text starts flowing around the frame.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerInsertParaFrame](#)



## TerSetFrameTextDist

**Set the frame to text distance.**

BOOL TerSetFrameTextDist(hWnd,ParaFID, dist)

HWND hWnd;	// The handle of the window to be accessed
int ParaFID;	// The paragraph frame id to the set the distance.
int dist;	// The frame text distance in twips unit (default 180 twips)

**Description:** The dist argument controls the minimum distance between a frame and the text flowing around the frame.

**Return Value:** This function returns TRUE when successful.



## TerSetNewFrameDim

**Set the default dimensions for a new frame.**

BOOL TerSetNewFrameDim(hWnd,x,y,width,height,PageTop)

HWND hWnd;	// The handle of the window to be accessed
int x;	// The default x location for the new frame. Set this parameter to -1 to insert the new frame at the current cursor location.
int y;	// The default y location for the new frame. Set this parameter to -1 to insert the new frame at the current cursor location.
int width;	// The default width for the new frame. Set this parameter to 0 to leave it unchanged.
int height;	// The default height for the new frame. Set this parameter to 0 to leave it unchanged.

```

BOOL PageTop;           // Set to TRUE to create the frames relative to the top of
                        // the page.

```

**Description:** The values passed by this function are used by any subsequent calls to the TerInsertParaFrame and TerInsertDrawObject functions as default values.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerInsertParaFrame](#)  
[TerInsertDrawObject](#)



## TerSetObjectAttrib

**Set the drawing object attributes.**

```

BOOL TerSetObjectAttrib(hWnd,ObjectId, LineType, LineThickness, LineColor, FillSolid,
FillColor)

```

```

BOOL TerSetObjectAttribEx(hWnd,ObjectId, LineType, LineThickness, LineColor,
FillSolid, FillColor, ZOrder)

```

```

HWND hWnd;           // The handle of the window to be accessed

int ObjectId;        // id of the drawing object. Set to -1, to display the user
                    // selection dialog box. This dialog box is displayed only if
                    // the cursor is positioned on a drawing object.

int LineType;        // Border line type:

                    DOB_LINE_NONE:      No border

                    DOB_LINE_SOLID:     Solid border line

                    DOB_LINE_DOTTED:    Dotted border line

int LineThickness;   // Border line thickness in twips.

COLORREF LineColor;  // Border line color

BOOL FillSolid;      // TRUE to fill the background, FALSE to leave the text
                    // box transparent. This option is available for a text box
                    // type drawing object only.

COLORREF FillColor;  // Background color for the drawing object.

int ZOrder;          // Z Order for the object. Set to -9999 to leave this value
                    // unchanged.

```



**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerInsertDrawObject](#)  
[TerSetFrameYBase](#)  
[TerRotateFrameText](#)  
[TerSetObjectWrapStyle](#)



## TerSetObjectWrapStyle

**Set the text wrap style for a drawing object.**

BOOL TerSetObjectWrapStyle(hWnd, ObjectId, WrapStyle)

HWND hWnd;	// The handle of the window to be accessed
int ObjectId;	// id of the drawing object. Set to -1, to specify the object at the current cursor position.
int WrapStyle;	// Text wrap style:
	SWRAP_NO_WRAP: Do not place text on the left and right of the object.
	SWRAP_AROUND: Flow the text around the object.
	SWRAP_THUR: Flow the text through the object.

**Return Value:** This function returns TRUE when successful.

**See Also**

[TerSetObjectAttrib](#)



## TerSetFrameYBase

**This function sets the vertical base position for a frame or a drawing object.**

BOOL TerSetFrameYBase(hWnd, FrameId, base)

HWND hWnd;	// Window handle to access
int FrameId;	// id of the frame or the drawing object
int base;	// The base can be set to one of the following:

BASE_PAGE:	Vertical position relative to the top of the page.
BASE_MARG:	Vertical position relative to the top margin.
BASE_PARA:	Vertical position relative to the top of the anchor paragraph.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerInsertDrawObject](#)  
[TerSetObjectAttrib](#)



## Footnote, Endnote, Bookmark, Tag

### In This Chapter

[TerDeleteBookmark](#)  
[TerGetBookmark](#)  
[TerDeleteTag](#)  
[TerGetTag](#)  
[TerGetTagPos](#)  
[TerInsertBookmark](#)  
[TerInsertFootnote](#)  
[TerPosTag](#)  
[TerPosBookmark](#)  
[TerSetTag](#)



## TerDeleteBookmark

**Delete a bookmark.**

BOOL TerDeleteBookmark(hWnd, name)

HWND hWnd;                               // The handle of the window to be accessed

LPBYTE name;                            // The name of the bookmark to delete.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerInsertBookmark](#)  
[TerPosBookmark](#)



## TerGetBookmark

**Retrieve a bookmark name.**

int TerGetBookmark(hWnd, index, name)

HWND hWnd; // The handle of the window to be accessed

int index; // The index (0 to Total Bookmarks -1) of the bookmark to retrieve. Set to -1 to retrieve the total number of bookmarks in the document.

LPBYTE name; // The location to retrieve the bookmark name.

**Return Value:** This function returns the total number of bookmarks in the document.

### See Also:

[TerInsertBookmark](#)

[TerPosBookmark](#)

[TerDeleteBookmark](#)



## TerDeleteTag

**Delete a tag at the specified text position:**

int TerDeleteTag (hWnd, line, col, type, name)

HWND hWnd; // The handle of the window to be accessed

long line; // The line position of the text. Set to -1 to use the current cursor position.

int col; // The column position of the text. This parameter is ignored if the 'line' parameter is set to -1.

int type; // Tag type:

TERTAG_BKM:	Bookmark tags
TERTAG_USER:	Generic tags

LPBYTE name; // The tag name to delete

**Return Value:** This function deletes the specified tag and return the tag id of the deleted tag. It returns 0 if no tag is found at the location, or if an error is encountered.



## TerGetTag

**Retrieve the tag at the specified text position:**

int TerGetTag (hWnd, line, col, name, AuxText, AuxInt, flags)

int TerGetTagEx (hWnd, line, col, type, name, AuxText, AuxInt, flags)

HWND hWnd;	// The handle of the window to be accessed
long line;	// The line position of the text. Set to -1 to use the current cursor position.
int col;	// The column position of the text. This parameter is ignored if the 'line' parameter is set to -1.
int type;	// Tag type (used by TerGetTagEx function only):
	TERTAG_BKM:                Bookmark tags
	TERTAG_USER:             Generic tags
LPBYTE name;	// The location to retrieve the tag name.
LPBYTE AuxText;	// The location to retrieve any auxiliary text string associated with the tag.
LPLONG AuxInt;	// The location to retrieve any auxiliary numeric data associated with the tag.
UINT far *flags;	// The location to retrieve tag flags (reserved for future use).

**Return Value:** This function returns the unique tag id at the specified location. It returns 0 if no tag is found at the location, or if an error is encountered.

**See Also:**  
[TerPosTag](#)  
[TerSetTag](#)



## TerGetTagPos

### Get the text position of a tag.

int TerGetTagPos(hWnd, TagId, name, type)

BOOL TerGetTagPos2(hWnd, TagId, name, type, line, col)

HWND hWnd;	// Handle of the window to be accessed
int type;	// Tag type (used by the TerPosTagEx function only):
	TERTAG_BKM:           Bookmark tag
	TERTAG_USER:         Generic tag
int TagId;	// The tag id to search for. This parameter is ignored if the 'name' parameter is specified.
LPBYTE name;	// The tag name to search for. Please note that tag names are not unique within a document.
LPINT line;	// The line number where the tag is located. This parameter is used by the TerGetTagPos2 function only.
LPINT line;	// The columber number where the tag is located. This parameter is used by the TerGetTagPos2 function only.

**Return Value:** The TerGetTagPos function searches the given tag to return the absolute character position of the tag. The function returns -1 if the tag is not found in the document.

The TerGetTagPos2 function searches the given tag to return the line/column position of the tag. The function returns a false value if the tag is not found in the document.



## TerInsertBookmark

### Insert a bookmark.

int TerInsertBookmark(hWnd, line, col, name)

HWND hWnd;	// The handle of the window to be accessed
long line;	// The text line number where to insert the bookmark. Set to -1 to insert the bookmark at the current cursor

location.

int col; // The text column position to insert the bookmark. This argument is ignored if the 'line' argument is set to -1.

LPBYTE name; // Bookmark name. The name may consist of regular alphabetic characters, but it may not contain spaces.

**Return Value:** This function returns a non-zero bookmark id, if successful. Otherwise it returns zero.

**See Also:** TerDeleteBookmark, TerGetBookmark, TerPosBookmark



## TerInsertFootnote

**Insert a footnote.**

int TerInsertFootnote(hWnd, FnMarker, FnText, style, repaint)

int TerInsertFootnote2(hWnd, FnMarker, FnText, style, IsFootnote, repaint)

HWND hWnd; // The handle of the window to be accessed

LPBYTE FnMarker; // footnote marker

LPBYTE FnText; // footnote text. Set this parameter to NULL to invoke a dialog box for the user to enter the footnote parameters.

UINT style; // footnote marker style. The style can be any combination of BOLD, ULINE, ITALIC, and SUPSCR. Typically, the style is set to SUPSCR.

BOOL IsFootnote; // Set to TRUE to insert a footnote. Set to FALSE to insert an endnote. This argument is applicable to the TerInsertFootnote2 function only. The TerInsertFootnote function only inserts a footnote.

BOOL repaint; // TRUE to repaint the screen after the operation.

**Description:** This function inserts a footnote at the current cursor location. The cursor is positioned after the footnote text upon the completion of this operation.

**Return Value:** This function returns TRUE when successful.



## TerPosTag

**Position the cursor at the specified tag position.**

BOOL TerPosTag (hWnd, TagId, name, scope, repaint)

BOOL TerPosTag Ex(hWnd, type, TagId, name, scope, repaint)

HWND hWnd;	// Handle of the window to be accessed
int type;	// Tag type (used by the TerPosTagEx function only):
	TERTAG_BKM:           Bookmark tag
	TERTAG_USER:          Generic tag
int TagId;	// The tag id to search for. This parameter is ignored if the 'name' parameter is specified.
LPBYTE name;	// The tag name to search for. Please note that tag names are not unique within a document.
int scope;	// Search scope:
	SCOPE_BEGIN:          Search from the beginning of the document.
	SCOPE_FORWARD:        Search below the cursor position.
	SCOPE_BACKWARD:       Search above the cursor position.
	SCOPE_ANY:            A fast method of locating an instance of the requested bookmark.
BOOL repaint;	// TRUE to refresh the screen after this operation.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**  
[TerSetTab](#)  
[TerGetTag](#)



## TerPosBookmark

#### Position at a bookmark.

OOL TerPosBookmark(hWnd, name, repaint)

HWND hWnd;	// Handle of the window to be accessed
LPBYTE name;	// The name of the bookmark to position at.
BOOL repaint;	// Set to TRUE to refresh the screen after this operation

**Return Value:** This function returns TRUE when successful.

#### See Also:

[TerInsertBookmark](#)

[TerGetBookmark](#)

[TerDeleteBookmark](#)



## TerSetTag

#### Set a tag at the specified text position:

int TerSetTag(hWnd, line, col, name, AuxText, flags)

HWND hWnd;	// The handle of the window to be accessed
long line;	// The line position of the text. Set to -1 to use the current cursor position.
int col;	// The column position of the text. This parameter is ignored if the 'line' parameter is set to -1.
LPBYTE name;	// The tag name string. A tag name does not need to be unique within the document.
LPBYTE AuxText;	// Any auxiliary text string associated with the tag.
long AuxInt;	// Any auxiliary numeric data associated with the tag
UINT flags;	// Reserved for future use. Must be set to 0.

**Description:** Use this function to set a tag at the specified character position. If a tag already exists at the current text position, then the existing tag is updated with the new information.

**Return Value:** This function returns the unique tag id if successful. Otherwise it returns 0.

#### See Also:



[TerPosTag](#)  
[TerGetTag](#)



## Stylesheet

### In This Chapter

[TerCancelEditStyle](#)  
[TerDeleteStyle](#)  
[TerEditStyle](#)  
[TerGetStyleId](#)  
[TerGetStyleInfo](#)  
[TerGetStyleParam](#)  
[TerSetStyleParam](#)



## TerCancelEditStyle

**Cancel the editing of the current style and restore its original value.**

BOOL TerCancelEditStyle(hWnd)

HWND hWnd; // The handle of the window to be accessed

**Return Value:** This function returns TRUE if successful.

### See Also

[TerEditStyle](#)



## TerDeleteStyle

**Delete a stylesheet item.**

BOOL TerDeleteStyle(hWnd, StyleId, name)

BOOL TerDeleteStyleU(hWnd, StyleId, name)

HWND hWnd; // The handle of the window to be accessed

int StyleId; // Style id to delete. Please note that the default style ids 0 and 1 can not be deleted. Set this parameter to -1 to specify the name of the style item to delete.

LPBYTE name; // Name of style item to delete. This parameter is ignored when StyleId is non-zero.

Any paragraph or font id referring the style id being deleted are modified to refer to the default style ids instead.

The TerDeleteStyleU method uses a unicode style item name string.

**Return Value:** This function returns TRUE if successful.

**See Also:**  
[TerEditStyle](#)



## TerEditStyle

### Create or edit a stylesheet style item.

int TerEditStyle(hWnd, BeginRecording, name, new, type, repaint)

int TerEditStyleU(hWnd, BeginRecording, name, new, type, repaint)

HWND hWnd;	// The handle of the window to be accessed
BOOL BeginRecording;	// TRUE to begin recording a stylesheet item, FALSE to end the recording of a stylesheet item.
LPBYTE name;	<p>// Name of the stylesheet item. When this parameter is NULL, the editor shows a dialog box to the user to prompt for the parameters.</p> <p>The TerEditStyleU method uses a Unicode style item name string.</p>
BOOL new;	// TRUE if creating a new stylesheet item and FALSE if modifying an existing item.
int type;	// Stylesheet item type:
SSTYPE_CHAR:	Character style
SSTYPE_PARA:	Paragraph style
BOOL repaint;	// TRUE to repaint the screen after this operation.

**Description:** This function is used to create a new stylesheet item or to modify an existing stylesheet item. The name and the type of the stylesheet item are specified by the 'name' and the 'type' arguments. Call this function with the 'BeginRecording' argument set to TRUE to begin recording the properties for a stylesheet item. To apply the paragraph and character formatting properties to a stylesheet being recorded, use the menu, toolbar, ruler or any of the APIs (example: SetTerParaFmt) which modifies these

properties. While a character stylesheet item is being recorded, only the character properties should be edited. When a paragraph stylesheet item is being recorded, both the character and the paragraph attributes can be edited.

To end the property recording of the current stylesheet item, call this function again with the 'BeginRecording' argument set to FALSE.

**Return Value:** This function returns the new style id if successful. Otherwise, it returns -1.

#### See Also:

[TerSelectCharStyle](#)

[TerSelectParaStyle](#)

[TerGetFontStyleId](#)

[TerDeleteStyle](#)

[TerGetParaInfo](#)

[TerCancelEditStyle](#)



## TerGetStyleId

**Translate a stylesheet style name to style id.**

int TerGetStyleId(hWnd, name)

int TerGetStyleIdU(hWnd, name)

HWND hWnd; // The handle of the window to be accessed

LPBYTE name; // Style name.

The TerGetStyleIdU method uses a Unicode style item name string.

**Return Value:** The function returns the style id for the given style name when successful. Otherwise it returns -1.

#### See Also:

[TerDeleteStyle](#)

[TerGetStyleInfo](#)



## TerGetStyleInfo

**Retrieve information for a style item id.**

int TerGetStyleInfo(hWnd, id, name, type)

int TerGetStyleInfoU(hWnd, id, name, type)

HWND hWnd;	// The handle of the window to be accessed
int id;	// Style item id to retrieve information.
LPBYTE name;	// The location to receive the style name. The TerGetStyleInfoU method uses a Unicode style item name string.
LPINT type;	// The location to receive the style type. The style type can be one of the following constants:
	SSTYPE_PARA:      Paragraph style
	SSTYPE_CHAR:      Character style

**Return Value:** The function returns total number of style items available in the document  
A return value of -1 indicates an error.

**See Also:**  
[TerGetStyleId](#)



## TerGetStyleParam

**Get the style parameters.**

int TerGetStyleParam(HWND, id, type)

HWND hWnd;	// The handle of the window to be accessed
int id;	// Style item id to retrieve parameters.
int type;	// The parameter to retrieve:
SSINFO_CHAR_SPACE:	Return the additional character spacing value in twips. The negative value indicates the compression of character spacing in twips unit.
SSINFO_CHAR_OFFSET	Return the character offset (twips) from the baseline.
SSINFO_NEXT	Returns next style id to be applied when the user hits Enter at the end of a line.

**Return Value:** The function returns the value for the requested parameter. It returns -1 to indicate an error condition.



## TerSetStyleParam

**Set the style parameters.**

BOOL TerSetStyleParam(hWnd, id, type, IntParam, TextParam, repaint)

BOOL TerSetStyleParamU(hWnd, id, type, IntParam, TextParam, repaint)

HWND hWnd; // The handle of the window to be accessed

int id; // Style item id to set parameters.

int type; // The parameter to set

SSINFO\_NAME      New style name. Specify using the TextParam argument.

SSINFO\_NEXT      The style id to be applied to the new line when the user hits Enter at the end of a line. Specify using the IntParam argument.

int IntParam;      The value for the numeric type parameter. This value is ignored for the string type parameters.

LPBYTE TextParam;      The value for the string type parameter. This value is ignored for the numeric type parameters.

The TerSetStyleParamU method uses a Unicode TextParam string.

BOOL repaint;      TRUE to repaint the screen after this operation.

**Return Value:** The function returns TRUE when successful.



## Control Creation

If you wish to use the editor as an ActiveX control, please refer to the [ActiveX Control](#) chapter.

This section includes the functions to create the control window directly using the DLL.

#### In This Chapter

[CloseTer](#)

[CreateTerWindow](#)

[LoadTerControl](#)

[TerCreateWindowAlt](#)

[TerOverrideStyles](#)



## CloseTer

### Close a TER Window:

```
BOOL CloseTer(hWnd,ForcedClose)
```

```
HWND hWnd;           // The handle to the window to be closed
```

```
BOOL ForcedClose;    // Set to TRUE to save the modifications and close the
                     // window unconditionally. Set to FALSE if the user
                     // verification is desired to close the window. The user
                     // verification is sought only if the text was modified.
```

**Description:** This function can be used by your application to close a TER window. You can also allow the user to close the window by using a menu option.

*Before a TER window is actually closed, TER sends the TER\_CLOSE message (see Message Communication) to your application window. The wParam variable for the message contains the handle of the window being closed (see DEMO.C). When your application receives this message, it can use the GetTerBuffer function to retrieve the updated text buffer.*

**Return Value:** This function returns a TRUE value if the window is closed, otherwise it returns a FALSE value.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

#### See Also

[CreateTerWindow](#)



## CreateTerWindow

## Open a TER Window:

HWND CreateTerWindow(ptr)

struct arg\_list far \*ptr;

The *ptr* argument points to a structure that provides the initial parameters to open a window. This structure includes the following parameters:

struct arg\_list {

<b>int</b> x	Initial X position of the editing window. You may specify CW_USEDEFAULT here to use the default value. The CW_USEDEFAULT constant is defined by Windows as hex 8000 for 16 bit and hex 80000000 for 32 bit.
<b>int</b> y	Initial Y position of the editing window. You may specify CW_USEDEFAULT here to use the default value.
<b>int</b> width	Window width. You may specify CW_USEDEFAULT here to use the default value
<b>int</b> height	Window Height. You may specify CW_USEDEFAULT here to use the default value.
<b>int</b> LineLimit	Maximum number of lines allowed for text editing. Set this field to 0 to allow unlimited number of lines.
<b>BOOL</b> WordWrap	If set to TRUE, this flag enables the word wrapping feature.
<b>BOOL</b> PrintView	Normally (PrintView = FALSE) the text lines are wrapped at the right edge of the window. This is a convenient editing mode with better performance. In the print view edit mode, however, the text lines are wrapped as they would be wrapped when printed at the current printer.
<b>BOOL</b> PageMode	Set to TRUE to facilitate editing of documents one page at a time. This feature is useful when editing multiple column documents. The text is displayed in side-by-side columns. This mode also implies the PrintView mode.
<b>BOOL</b> FittedView	Special case of the page mode in which the text wraps to the window width and the soft page breaks are not displayed
<b>BOOL</b> ShowStatus	If set to TRUE, the editor will display a status line describing the current cursor location, and the insert/overtyping indicator.
<b>BOOL</b> ShowMenu	If set to TRUE, the editor displays the editor menu under the caption bar. When set to FALSE, the editor commands must be selected using only the accelerator keys.

<b>BOOL</b> ShowHorBar	If set to TRUE, the editor displays the horizontal scroll bar.
<b>BOOL</b> ShowVerBar	If set to TRUE, the editor displays the vertical scroll bar.
<b>BOOL</b> ruler	Set this variable to TRUE to show a ruler with tab stops.
<b>BOOL</b> ToolBar	Set this variable to TRUE to show the tool bar.
<b>BOOL</b> UserCanClose	Set this variable to TRUE if you want your user to be able to close the TER window by selecting the exit option from the TER menu. Otherwise, your application will need to close the window by using the <i>CloseTer</i> function.
<b>BOOL</b> BorderMargins	Set this field to TRUE to create a thin margin around the text area within the TER window. This margin area is for cosmetic purpose only.
<b>BOOL</b> ReadOnly	This is a browser mode which does not allow text modifications.
<b>int</b> InitLine	The first line number to position on when the TER routine is called.
<b>char</b> InputType	This field specifies the input type to the editor. If you wish to pass a text file name to the editor, set this flag to 'F'. However, if the text input will be passed in a global memory buffer, set this flag to 'B'.
<b>char</b> file[299]	If the InputType field is set to 'F', this field specifies the full path of the input file.
<b>HANDLE</b> hBuffer	If the InputType field is set to 'B', this field specifies the handle to the global memory containing the input text data. To specify an empty buffer, allocate a buffer for 1 byte, place the 'delim' character in the first byte and set the 'BufferLen' field to 1. The input memory handle must be unlocked before calling the editor routine. You can get the updated text by using the <i>GetTerBuffer</i> prior to closing the TER window.
<b>long</b> BufferLen	When the InputType field is set to 'B', the BufferLen field specifies the length of the input buffer.
<b>char</b> delim	This character should be set to 13 (carriage return character).
<b>int</b> SaveFormat	This field determines the format of the output file or buffer: <div> <div>SAVE_DEFAULT:</div> <div>Save in the format of the input file.</div> <div>SAVE_TER:</div> <div>Save in the native TER format.</div> </div>



	SAVE_RTF:	Save in the Rich Text Format.
	SAVE_DOCX	DOCX format
	SAVE_TEXT:	Save in ASCII format.
	SAVE_TEXT_LINES:	Save in ASCII format with line breaks.
	SAVE_UTEXT	Save in the Unicode text format (not available in 16 bit)
<b>HANDLE</b>	hInst	Handle for the current instance of the application.
<b>HWND</b>	hParentWnd	Handle of the parent window. The DLL sends the TER_CLOSE message to this function before closing a TER window (see the CloseTer API function).
<b>HWND</b>	hTextWnd	The editor fills in this field with the window handle of the editor window. The editor does this as soon as the window is created. You will need this handle to use other TER API functions.
<b>DWORD</b>	style	The window style that you provide here is simply passed to the CreateWindow function.
char	FontTypeFace[30]	Default font typeface for the document. Example: "Helv", "Courier", "System" etc. A menu option in the TER window provides multiple selections of typeface and point sizes to choose from.
int	PointSize	Point size of the default font. Please remember that 72 points are equal to one inch. Most written correspondences use a letter size of 10 or 12 points.
<b>BOOL</b>	open	The editor set this variable to a TRUE value after initializing a TER window.
<b>BOOL</b>	modified	This variable is reserved for internal use.
}		

**Description:** This function opens a TER window.

**Return Value:** If successful, the function returns the window handle of the new window. Otherwise it returns a NULL value. The handle to the newly opened window is also returned to you using the hTextWnd variable within the arg\_list structure. If your application provides the data using a global buffer, the handle to the specified global buffer becomes the property of the DLL. Your application MUST not try to lock or free this handle. To get the updated text data, you should use the GetTerBuffer function.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

See Also  
[CloseTer](#)  
[SetTerBuffer](#)



## LoadTerControl

### Load the TER Edit Control DLL

BOOL LoadTerControl(void)

**Description:** This function must be called in the beginning of your program when using the editor as a custom control. This function call ensures that the DLL file is loaded and the "TerClass" ("Ter32Class" for WIN32) class created. The library is automatically unloaded when your program ends.

This function needs to be called only when creating the TE window using the CreateWindow Windows SDK API or when TE control is placed in a dialog box. You do *not* need to use this function when using the product as an ActiveX control, or when creating the TE window using the CreateTerWindow or TerCreateWindowAlt functions.

**Return Value:** This function always returns TRUE.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function



## TerCreateWindowAlt

**Alternate method of creating a TER window.**

HWND TerCreateWindowAlt(x, y, width, height, IsContained, hParentWnd, styles)

int x;	// x position of the window in pixel units
int y;	// y position of the window in the pixel units
int width;	// width of the window in the pixel units
int height;	// height of the window in the pixel units
BOOL IsContained;	// TRUE if this window is to be contained inside another window
HWND hParentWnd;	// handle of the parent window.

DWORD styles; // TER window styles. Please refer to the TER\_xxx constants defined in the 'Getting Started' chapter. Set to 0 to use the default styles.

**Description:** This is a quick method of creating a TER window. For a more elaborate method of creating a TER window, please refer to the CreateTerWindow function.

**Return Value:** This function returns the window handle of the new window. It returns NULL if the window creation fails.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

See Also:  
[CreateTerWindow](#)



## TerOverrideStyles

**Override the control creation flags.**

DWORD TerOverrideStyles(TerStyles)

DWORD TerStyles; // The styles to use to create the next control.

**Description:** This function is useful when your code does not allow you to specify the style bits or control properties while creating a new control. Call this function to set the style bits before you create the new control. For a description of the style bit constants (TER\_XXXXX) please refer to the 'Getting Started' chapter.

**Return Value:** This function returns the old values for the override flags.



## Control Flags

### In This Chapter

[TerSetFlags](#)  
[TerSetFlags2](#)  
[TerSetFlags3](#)  
[TerSetFlags4](#)  
[TerSetFlags5](#)  
[TerSetFlags6](#)  
[TerSetFlags7](#)  
[TerSetFlags8](#)  
[TerSetFlags9](#)  
[TerSetFlags10](#)



## TerSetFlags

**Set certain flags or retrieve the values of the flags.**

DWORD TerSetFlags(hWnd, set, flags)

HWND hWnd;                      // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set;                        // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags;                    // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG\_RESIZE\_BITMAP:            Resize the bitmap when inserted into a frame.

TFLAG\_METRIC:                  Show the ruler and dialog measurements in metric units.

TFLAG\_APPLY\_PRT\_ORIENT:        This flag overrides the paper orientation of the document with the paper orientation of the currently selected printer.

TFLAG\_RETURN\_MSG\_ID :          Store the message ids so that they can be retrieved by using the TerGetLastMessage function.

TFLAG\_IGNORE\_PICT\_LINK:        This flag instructs the save functions not to write the file name of the linked pictures to the disk file.

TFLAG\_DISABLE\_ACCEL:          Disable the internal editor accelerator table.

TFLAG\_SHOW\_CARET:              Show caret even in the read-only mode.

TFLAG\_UNPROTECTED\_DEL:        Allows for unprotected deletion of text blocks

TFLAG\_NO\_HOUR\_GLASS:          Do not display the hour glass cursor during lengthy operations.

TFLAG\_NO\_CHILD\_TOP:            Don't force child to the top of Z order.

TFLAG\_NO\_WRAP:                Turn-off word wrapping temporarily. This

	flag should not be used in the Page Mode.
TFLAG_EXCLUDE_HIDDEN_SEL:	Exclude hidden text from both ends of a selected block of text.
TFLAG_AUTO_VSCROLL_BAR:	Enable/disable vertical scroll bar depending upon the height of the text in the window. This option is not available in Page Mode.
TFLAG_NO_PALETTE:	This flag disables the internal use of the color palettes for the picture display.
TFLAG_KEEP_PICT_ASPECT:	This flag maintains the picture aspect ratio when being resized using the mouse.
TFLAG_KEEP_FRAME_ASPECT:	This flag maintains the frame aspect ratio when being resized using the mouse.
TFLAG_PICT_IN_FRAME:	This flag creates a frame around the picture being dropped into the editor.
TFLAG_NO_PRINTER:	Disable the use of the printer.
TFLAG_NO_DRAG_TEXT:	Disable the drag/drop feature for text.
TFLAG_BETTER_256:	Improves the 256 color picture production at a small cost of picture load performance. This flag is effective only when displaying a 256 colors bitmap in the 256 color display mode.
TFLAG_NO_EDIT_OLE:	Disable the editing of OLE objects
TFLAG_NO_EDIT_PICT:	Disable the editing of picture objects.
TFLAG_SHOW_BREAKS:	Show the break lines even in the read-only mode.
TFLAG_SELECT_FULL_HLINK:	Select complete hypertext phrase during text selection.
TFLAG_NO_OLE:	Disable the OLE features.
TFLAG_ROW_PASTE:	Paste the table data in the control as rows instead of as columns.
TFLAG_NO_AUTO_FULL_CELL_SEL:	Do not select the entire cell when the last character of the cell is highlighted.
TFLAG_SWAP_DECIMAL:	Swap decimal and comma characters in

the dialog boxes.

**Return value:** This function returns the new value of all the flags. Call this function with the 'flags' parameter set to zero to retrieve flag values without modifying it.

**See Also:**

[TerSetFlags2](#)

[TerSetFlags3](#)

[TerSetFlags4](#)



## TerSetFlags2

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags2(hWnd, set, flags)

HWND hWnd; // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set; // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG2\_RETAIN\_BKND: Display the text over the existing background. To be effective, this flag must be set before the TER window is created (set hWnd to NULL).

TFLAG2\_USE\_PAL\_FOR\_TEXT: Use the currently selected palette to draw the text

TFLAG2\_CAN\_MERGE\_PROT\_TEXT: Allow the deletion of the last character between two protected blocks of text.

TFLAG2\_BKPICT\_OVER\_PAGE\_BORDER: When the background picture as well as the page border is displayed, this option ensures that the background picture is displayed in place of the border shading around the page.

TFLAG2\_NO\_CARET: Do not display the caret even in the edit mode.

TFLAG2\_SHOW\_SECT\_PAGE\_NO: Show the section page number on the

	status bar and while scrolling.
TFLAG2_NO_CURSOR_CHANGE:	Do not change the cursor shape as the mouse moves over the editor window.
TFLAG2_VERT_THUMB_TRACK:	Scroll the screen while dragging the vertical scroll bar.
TFLAG2_NO_AUTO_REPAGE:	Do not do automatic repagination when the document is edited.
TFLAG2_NO_BKP_FILE:	Do not save the original file as backup before saving the document.
TFLAG2_SELECT_FRAME_PICT:	When the frame containing a picture is clicked, select the picture instead of the frame.
TFLAG2_HIDE_PAGE_BREAK:	Don't show the soft page break lines in the Page Mode.
TFLAG2_PROTECT_FORMAT:	Protect the formatting of the protected text.
TFLAG2_NO_HIDDEN_RTF_TEXT:	Do not write hidden text to the RTF file.
TFLAG2_NO_SHADE_FIELD_TEXT:	Do not apply shading to the field text.
TFLAG2_IGNORE_TIMER:	Suspend timer activity.
TFLAG2_NO_AUTO_HDR_FTR:	Do not display header/footer automatically at file input or paste operation.
TFLAG2_WRITE_FIRST_RTF_COLOR:	Write the initial default color to the rtf color table.
TFLAG2_FULL_REPAINT:	Always fully repaint the text box.
TFLAG2_KEEP_PRINTER_OPEN:	Keep the printer driver open after the last edit window is closed. When this flag is set, your application should call the TerClosePriner function to close the printer driver manually after the last edit window is closed.
TFLAG2_ALT_PARA_SYM:	Display alternate paragraph symbol ( ). This flag should be set before creating the edit window.
TFLAG2_ALT_LINE_SYM:	Display alternate line break symbol ( ). This flag should be set before creating the edit

window.

TFLAG2_NO_LINE_FITTING:	Do not attempt to shrink longer screen lines to fit within the printer defined width.
TFLAG2_NO_PRT_CANCEL_DLG:	Do not show the print cancel dialog box during printing.
TFLAG2_INDENT_FRAMES:	When indenting text, indent the selected frames as well.
TFLAG2_INDENT_TABLES:	When indenting text, indent the selected table as well.
TFLAG2_NO_ADJUST_CURSOR:	Do adjust cursor when placed over hidden or protected or hidden text.
TFLAG2_CURSOR_BEH_HIDDEN:	When the cursor is in the middle of hidden text, move it before the hidden text. By default, the cursor is moved after the hidden text.
TFLAG2_NO_CURSOR_ON_PROTECT:	Do not allow cursor in the middle of the protected text.
TFLAG2_WRAP_SPACES:	Wrap spaces at the end of a line to the next line.

**Return value:** This function returns the new value of all the flags. Call this function with the 'flags' parameter set to zero to retrieve flag values without modifying it.

**See Also:**  
[TerSetFlags](#)  
[TerSetFlags3](#)  
[TerSetFlags4](#)



## TerSetFlags3

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags3(hWnd, set, flags)

HWND hWnd;                      // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set;                        // TRUE to set the given flags, FALSE to reset the given



## flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG3_WRAP_SPACES:	Wrap additional spaces at the end of the line to the next line.
TFLAG3_NO_EDIT_TABLE_COL:	Disable the editing of table column width and indentation using the mouse.
TFLAG3_TABLE_STATUS_LINE:	Consider the table rows as lines for the status line display.
TFLAG3_PLAIN_TABLE_BORDER:	Display one line table border in HTML mode.
TFLAG3_GRAY_READ_ONLY:	Display the text in gray color in the read-only mode. To affect the color change during run-time, you also need to call the TerRepaint function after setting the read-only mode using the TerSetReadOnly function.
TFLAG3_CURSOR_IN_CELL:	Restrict the cursor to the current cell.
TFLAG3_NO_SCROLL:	Do not scroll down from the top of the document. This flag is effective only in the PageMode and only if the document does not contain framed text.
TFLAG3_NO_FULL_CELL_COPY:	Do not select the cell marker for cut/copy operations when the selection includes only one table cell.
TFLAG3_HTML_CONT_TABLE:	For HTML output, always treat two contiguous table rows as part of one table.
TFLAG3_SELECT_FIRST_FIELD:	Position at the first form-field in the document. This flag should be set before a file is read into the control.
TFLAG3_MULTIPLE_RTF_GROUPS:	Support consecutive 'rtf' groups in the RTF file.
TFLAG3_OLD_WORD_FORMAT:	Write RTF syntax for previous versions of MS Word.
TFLAG3_DATA_FIELD_INPUT:	Data field input mode. Protects the field name from replacement during text input to the data field.

TFLAG3_GET_BUF_HDR_FTR:	Retrieve the rtf header/footer when inserting within an existing document and when the existing document does not already have header/footer. Normally, the header/footer from the document being copied into another document is always ignored.
TFLAG3_NO_RTF_BKND_COLOR:	Do not read or write the document background color in the RTF file.
TFLAG3_NO_SAVE_UNDO:	Do not collect undo information.
TFLAG3_LARGE_PARA_BORDER:	Include the paragraph before and after spaces within the paragraph borders.
TFLAG3_LINE_SCROLL	Scroll up or down one line at a time when the up or down arrow is pressed.
TFLAG3_STYLES_ON_TOOLBAR	This flags creates a style item selection combobox in the toolbar (default). To remove the this combobox, turn-off this flag BEFORE the control is created.
TFLAG3_PRINT_BKND_PICT	Print any background picture when printing the document.
TFLAG3_CLIP_CELL_OVERFLOW	Do not print text lines which do not fit inside a fixed height table cell.
TFLAG3_EXACT_SCREEN_FONT	In page mode, create the exact screen font without matching it to the printer font.
TFLAG3_ZERO_CELL_HEIGHT	Do not display empty table rows.
TFLAG3_READ_PNG	Read PNG images from the RTF file.
TFLAG3_NO_TEXT_COLOR_ADJ	Do not adjust text color to contrast against the background color.
TFLAG3_NO_MOUSE_SEL	Disable text selection using mouse.

**See Also:**  
[TerSetFlags](#)  
[TerSetFlags2](#)  
[TerSetFlags4](#)



**TerSetFlags4**

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags4(hWnd, set, flags)

HWND hWnd; // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set; // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG4\_COUNT\_PCHAR\_AS\_CRLF: Count a paragraph character as a cr/lf pair (2 characters) in TerAbsToRowCol and TerRowColToAbs functions.

TFLAG4\_SKIP\_PROT\_TEXT: Do not allow cursor anywhere on the protected text.

TFLAG4\_READONLY\_CONTROLS: Enable or disable the embedded controls and form fields when calling the TerSetReadOnly function.

TFLAG4\_NO\_REPAGINATE: Hold repagination.

TFLAG4\_SMOOTH\_SCROLL: Scroll screen smoothly when selecting text using mouse.

TFLAG4\_AUTO\_SPELL: Invoke auto spelling (SpellTime required for this feature)

TFLAG4\_BINARY\_RTF\_PICT: Save rtf pictures in the binary format (default is the hex format)

TFLAG4\_UNDO\_WINDOW\_OVERFLOW: Undo any editing task which makes the text go past the current window height. This flag is not effective in PageMode or FittedView modes.

TFLAG4\_IME\_UNICODE: Convert DBCS characters to Unicode during IME input.

TFLAG4\_MOD\_END\_MARK\_FONT: Set the font of the ending paragraph marker same as the font for the previous character.

TFLAG4_ONE_ROW_TOOLBAR:	Display only the second row of the toolbar.
TFLAG4_NO_OLE_DROP:	Disable dropping of OLE objects.
TFLAG4_ALWAYS_INOKE_OLE:	Invoke ole double-click even in read-only mode.
TFLAG4_DISABLE_DATE_UPDATE:	Do not update the date/time field value.
TFLAG4_SAVE_BMP_AS_PNG:	Write the the BMP images into the PNG format to the RTF output file.
TFLAG4_SAVE_SHAPE_WITH_DRAW_OBJECT:	Save shape and drawing object in one rtf file. Warning: Using this flag might generate an RTF file incompatible with MSWord.
TFLAG4_HTML_INPUT:	The input to the editor is assumed to be in the HTML format. HTML Add-on is called (if installed) to read the input data.
TFLAG4_NO_DRAG_PROT_TEXT:	Do not allows drag/drop of protected text.
TFLAG4_FULL_DRAG_PROT_TEXT:	Allow drag/drop of protected text fully. The protected text at the source location will be deleted.
TFLAG4_NO_MERGE_TABLE:	Do allow deletion of the paragraph marker before a table.
TFLAG4_NO_TOC_UPDATE:	Do not update table of contents
TFLAG4_NO_RESET_DC	Do not reset the printer device context
TFLAG4_NO_SHARE_BORDER:	Do not draw shared cell borders
TFLAG4_ADJ_LEFT_TABLE_COL	During table column adjust using the mouse, adjust only the left column.
TFLAG4_DONT_FIX_NEG_INDENT	Do not adjust for negative indents in an RTF file.
TFLAG4_PASTE_LAST_PARA_PROP	Paste the properties of the last paragraph when text is pasted into an empty paragraph.
TFLAG4_PRINT_WMF_AS_BMP	Print the metafiles as bitmap. This option is useful if the target device context is not able to handle the metafiles properly.

**See Also:**  
[TerSetFlags](#)  
[TerSetFlags2](#)  
[TerSetFlags3](#)



## TerSetFlags5

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags5(hWnd, set, flags)

HWND hWnd; // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set; // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG5_NO_EXT_DROP:	Disable text drop from other applications. This flag must be set before creating a control.
TFLAG5_GROUP_UNDO	Group the operations into one undo.
TFLAG5_NO_EXACT_ROW_HEIGHT	Translate 'Exact' row height to 'Minmum' row height during RTF input.
TFLAG5_RTL_CURSOR	Reverse the direction of the left/right cursor keys for the RTL text.
TFLAG5_NO_KB_SEL	Disable text selection using the keyboard.
TFLAG5_WRITE_DOB	Generates the older "do" drawing object constuct during RTF output.
TFLAG5_NO_OBJ_IN_STATUS_LINE	Ignore the embedded objects while display the status line number.
TFLAG5_NO_CLEAR_SPL_HIST	Do not clear the spell-checker word history.
TFLAG5_NO_NORMALIZE_FNOTE	Do not modified text selection to fully include footnote text and footnote marker.

TFLAG5_INPUT_TO_UNICODE	Convert keyboard input or rtf input for a foreign language to unicode.
TFLAG5_NO_NORMALIZE_FIELD	Do not modified text selection to fully include data field text in field enclosure tags in the rtf output.
TFLAG5_BEF_AND_AFT_HIDDEN	Allow the caret to stop both before and after the hidden text.
TFLAG5_SHOW_PAGE_SETUP	Show the page-setup options instead of print-setup options on the printer setup dialog box.
TFLAG5_FULL_REPAGINATE	Fully repaginate the document during file read irrespective of the size of the document.
TFLAG5_PNG_RGB	Save png data in RGB format (true color format)
TFLAG5_NO_SHOW_SPACE_SYM	Do not show the space symbol.
TFLAG5_OLD_HLINK	Allow the old hyperlink mode supporting the HLINK style and hidden/double-underline text.
TFLAG5_NO_DRAG_ROW_LINE	Do not show the drag cursor to drag the row break line.
TFLAG5_NO_SHARE	Do not share font and printer resources between different instances of TE control.
TFLAG5_VARIABLE_PAGE_SIZE	<p>Automatically adjust the page size to contain the entire content of the window. When this flag is set, TE fires the PageSizeChanging event (or TER_PAGE_SIZE_CHANGING message) to allow the user to override or modify the suggested page size.</p> <p>This flag is effective only in the Page Mode.</p>
TFLAG5_SAVE_PICT_AS_WMF	Save jpeg, png, gif images as Windows Metafile image.
TFLAG5_NO_DRAG_CELL_LINE	Do not show the drag cursor to drag the cell divider line and row indentation line.
TFLAG5_RULER_INDENT_FIXED	When apply indentation using the ruler to set of selected lines, set all lines to same

indentation.

TFLAG5_TOP_ROW_TOOLBAR	Display only the top row of the toolbar. The TerRecreateToolbar function must be called after setting this flag to re-display the toolbar.
TFLAG5_NO_ADJ_FOR_TABLE	Do not adjust the text selection within table.
TFLAG5_UNLIMITED_OLE_SPACE	Provide for unlimited ole storage space.
TFLAG5_OLD_RULER	Display the old style ruler.
TFLAG5_FRAME_TEXT_ONLY	Do not write the frame structure while saving selected frame text to a buffer.
TFLAG5_PROTECT_DATA_FIELD	Make al data field read-only.



## TerSetFlags6

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags6(hWnd, set, flags)

HWND hWnd; // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set; // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG6_WRITE_DBCS:	Write DBCS character codes for the Unicode characters.
TFLAG6_WRITE_DEFAULT_COLOR	Write the default color to the rtf output during clipboard copy operation.
TFLAG6_DONT_WRITE_PICT_PATH	Write linked pictures without full path.
TFLAG6_HI_RES	Use a printer-independent hi-resolution metrics to format the text. Use this flag to obtain printer independent word-wrapping

and pagination.

TFLAG6_DONT_PROCESS_TAB	Set to True to enable the use of the Tab key to move out of the TOC ActiveX control.
TFLAG6_SHOW_PICT_IN_PIXEL_SIZE	Embed the picture in real pixel size (ignore the recommended size).
TFLAG6_DONT_PROCESS_BULLET_KEYS	Do not use the Enter/Tab keys for manipulating bullets.
TFLAG6_XLATE_UNICODE	Enable unicode keyboard input processing for the languages that do not use an IME editor for input.
TFLAG6_ALLOW_CELL_OVERFLOW	Allow text to overflow in a fixed height table cell.
TFLAG6_INSERT_DROP_PICT_AS_LINK	Insert dropped picture files as linked pictures (and not as embedded pictures).
TFLAG6_LIST_TO_TEXT_IN_HTML	Convert lists to text before html save.
TFLAG6_SUPPORT_DRAGON_SPEECH	Support messages for Dragon Speech program.
TFLAG6_NO_LINK_MSG	Do not display a pop-up message when the mouse hovers over a text link.
TFLAG6_NO_TRACK_MSG	Do not display a pop-up message when the mouse hovers over a track-change text.
TFLAG6_SWAP_CR_LINE_BREAK	Generate line-break when Return is pressed, and generate Return when Shift-Return is pressed.
TFLAG6_DEL_CELL_TEXT	On deletion, delete cell content, but not cell structure. This flag is turned on by default.

**Return value:** This function returns the new value of all the flags. Call this function with the 'flags' parameter set to zero to retrieve flag values without modifying it.





## TerSetFlags7

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags7(hWnd, set, flags)

HWND hWnd; // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set; // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG7_NO_RTL_FONT:	Do not use right-to-left fonts.
TFLAG7_AUTO_NEW_ROW	Create a new row if the tab key is hit at the last column of the last row of a table. This flag is turned on by default.
TFLAG7_NO_TRACK_CHANGE_LINE	Do not draw track-change indicator line.
TFLAG7_SHOW_SPACES_AT_PARA_END	Show ending spaces before the end of the paragraph.
TFLAG7_SHRINK_PICT_TO_PAGE	Shrink to size the picture to fit within a page, column, or a table cell.
TFLAG7_DISABLE_RULER	Disable ruler.
TFLAG7_ALWAYS_FIRE_MODIFY	Always fire the modified event. By default, the editor fires the modified event only for the first modification.
TFLAG7_WRITE_LISTTEXT	Generate listtext tag on RTF output so Crystal Report can show bullets.
TFLAG7_SET_BOX_CLIPPING	Draw the table cell and frame text within its boundaries.
TFLAG7_NO_SPELLCHECK_FIELDS	Do not spell-check the field text.
TFLAG7_NO_INTERNET	Disable Internet access.
TFLAG7_NO_TABLE_AUTO_FIT	Disable automatic table column width adjustment.

**Return value:** This function returns the new value of all the flags. Call this function with the 'flags' parameter set to zero to retrieve flag values without modifying it.



## TerSetFlags8

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags8(hWnd, set, flags)

HWND hWnd; // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set; // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG8\_AUTO\_REFORMAT\_TABLES: Automatically recalculate the width of the auto-width tables in the fitted-view mode

TFLAG8\_DONT\_SELECT\_LAST\_CHAR Do not select the last character of the document on the select\_all operation.

TFLAG8\_ENABLE\_GDIPLUS Enable the gdi-plus features. The included ssgp.dll dll must be copied to the project folder to enable this feature.  
Also, Microsoft's gdi-plus package must be installed to use this feature.

TFLAG8\_HIDE\_PAGE\_BLANK\_AREA For short documents, disable the display of the non-text area by hiding the vertical scroll bar

TFLAG8\_MERGE\_TABLES\_ON\_PASTE Merge the paste table to the preceding table.

TFLAG8\_INSERT\_ROWS\_ON\_PASTE Insert the table rows on paste instead of overlaying it over the existing rows.

TFLAG8\_DONT\_SPELL\_CHECK\_HDR\_ Do not spell-check header/footer text.

FTR

**Return value:** This function returns the new value of all the flags. Call this function with the 'flags' parameter set to zero to retrieve flag values without modifying it.



## TerSetFlags9

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags9(hWnd, set, flags)

HWND hWnd; // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set; // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG9_TREAT_DATA_FIELD_AS_ATOM	Select entire data-field for deletion or copying to clipboard.
TFLAG9_NO_COMMENTS	Do not show comment.
TFLAG9_ENABLE_POPUP_MENU	Enable the default right-click menu.
TFLAG9_INVOKE_HYPERLINK	Enable automatic invoking of the hyperlinks without requiring to handle the Hypertext event within your application.
TFLAG9_FIRE_FONT_CHECK	Fire OverrideFont event to allow an override of current font during rtf input or keyboard entry.

**Return value:** This function returns the new value of all the flags. Call this function with the 'flags' parameter set to zero to retrieve flag values without modifying it.



## TerSetFlags10

**Set additional flags or retrieve the values of the flags.**

DWORD TerSetFlags10(hWnd, set, flags)

HWND hWnd; // The handle of the window to be accessed. If hWnd is set to NULL, this function sets the common initialization flags. When a new window is created, it inherits all common flags.

BOOL set; // TRUE to set the given flags, FALSE to reset the given flags

DWORD flags; // Flags (bits) to set or reset. Currently, the following flag values are available:

TFLAG10_DONT_MARK_WITH_SEL_COL	Don't select table cells using the table column selection method
TFLAG10_DONT_PAD_TABLE_HEIGHT	Don't add the bottom padding to the exact height table rows
TFLAG10_MERGE_NESTED_RTF_PROPS	Merge nested rtf color tables
TFLAG10_EXTEND_AUTO_VSCROLL	Extend the auto-vertical-scroll bar feature to hide the vertical bar when not needed
TFLAG10_NO_HLINK_BOOKMARK	Do not generate bookmark for internal hyperlinks
TFLAG10_NO_LAST_PARA_MARKER	Don't read or write the last para-marker to the rtf file
TFLAG10_DONT_COMBINE_UNDOS	Don't combine undos
TFLAG10_AUTO_HSCROLL_BAR	Activate the automatic horizontal scroll bar feature
TFLAG10_DOTTED_GRID_LINES	Show the dotted table grid lines (instead of light blue lines)
TFLAG10_NO_CRLF	Don't insert cr/lf in the RTF text
TFLAG10_USE_LAST_PARA_PROPS	When last paragraph marker is deleted upon RTF read, use its property for the resulting last paragraph
TFLAG10_WRITE_RANDOM_LIST_TMPL_ID	Randomize the list template id when writing the RTF output
TFLAG10_LOCK_DATA_FIELD	Write the fldlock tag for the data fields

TFLAG10_SHRINK_RTF_IMAGES	Shrink images to fit during rtf input
TFLAG10_PREFER_PLACEHOLDER_PICT	Use placeholder pictures if available for the linked images
TFLAG10_NO_PICT_READ_MSG	do not show the message encountered during picture data import error
TFLAG10_DONT_USE_BKSP_FONT	Use the font of the character being deleted for subsequent insertions.

**Return value:** This function returns the new value of all the flags. Call this function with the 'flags' parameter set to zero to retrieve flag values without modifying it.



## List Numbering

The list mechanism allows you create very complex lists. Here we will described how to create simple lists.

The list mechanism consists of a list id and a list-or (list override id). These ids can be created using the TerEditList and TerEditListOr functions. One or more list-or ids can be created for a list id. However, it is sufficient in most cases to create just one or two list-or ids for each list id. The list and list-or id can have multiple levels. Each level can designate its own list numbering format (decimal, alpha, etc).

To apply list numbering to text, you would first create a list and its corresponding list-or id. Then you would apply the list-or id to the selected text (one or more paragraphs) using the TerSetParaList function. You also specify the list level when calling the TerSetParaList function.

When a list needs to be restarted from 1, it is simpler to create another list-id (and associated list-or ids).

The product also includes TerSetBulletList function which wraps the basic list functions in one easy method call.

### In This Chapter

- [TerEditList](#)
- [TerEditListLevel](#)
- [TerEditListOr](#)
- [TerGetBulletInfo](#)
- [TerGetListInfo](#)
- [TerGetListLevelInfo](#)
- [TerGetListOrInfo](#)
- [TerSetDefListFormat](#)
- [TerSetListBullet](#)
- [TerSetListLevel](#)



## TerEditList

### Create or edit a list table item.

int TerEditList(hWnd, NewList, ListId, PropDialog, ListName, nested, flags)

HWND hWnd; // The handle of the window to be accessed

BOOL NewList; // Set to TRUE to create a new list item. Set to FALSE to modify an existing list item.

int ListId; // The id of the list item to modify. This parameter is ignored when the 'NewList' parameter is set to TRUE.

BOOL PropDialog; // Show a dialog box for the user to enter the list properties. When this parameter is set to TRUE, the values for the remaining parameters are ignored.

LPBYTE ListNme; // Name of the list

BOOL nested; // Set to TRUE to create a multi-level list. Set to FALSE to create a single level list. A multi-level list can have up to 9 levels.

DWORD flags; The following flag bits are supported currently:

LISTFLAG\_RESTART\_SEC: Restart the numbering at a section break.

**Description:** This function allows you to create a list table item. Typically, the following steps are needed to uselistnumbering for a paragraph. The first step is to use the TerEditList function to create a list table item. The second step is to use the TerEditListOr function to create one or more overrides for the list. Finally, you can use the TerSetParaList function with the list-override id and list-level parameters to assign list numbering for a paragraph. The list or list-override level properties can be modified using the TerEditListLevel function.

**Return Value:** When successful, this function returns a valid list id. A return value of -1 indicates an error condition.

### Example:

```
int ListId, ListOrId;
```

```
ListId= TerEditList(hWnd, TRUE, 0, FALSE, "MyFirstList",TRUE, 0); // create a new multi-level list
```

```
TerEditListLevel(hWnd, TRUE,ListId, 0,1, LIST_DEC, LISTAFT_TAB, "(~1~)", 0, 0); // print the first level in (1), (2), (3)...format.
```

```

TerEditListLevel(hWnd, TRUE, ListId, 1, 1, LIST_LWR_ALPHA, LISTAFT_TAB, "~1~.~2~",
0, 0); // print the second level in 1.a, 1.b. 1.c ... format.

ListOrId=TerEditListOr(hWnd, TRUE, 0, FALSE, ListId, 0, 0); // create a list-override id for
our list

SetTerCursorPos(hWnd, 10,0,TRUE); // position at line number 10

TerSetParaList(hWnd,FALSE,-1, ListOrId, 0, TRUE); // apply top level numbering to this
line

SetTerCursorPos(hWnd, 11,0,TRUE); // position at line number 11

TerSetParaList(hWnd,FALSE,-1, ListOrId, 1, TRUE); // apply second level numbering to
this line

```

#### See Also:

[TerEditListOr](#)  
[TerEditListLevel](#)  
[TerSetParaList](#)  
[TerCreateListBullet](#)



## TerEditListLevel

**Edit the level properties for a list or a list-override item.**

```

BOOL TerEditListLevel(hWnd, IsList, id, level, StartAt, NumType, CharAft, ListText,
FontId, flags)

```

HWND hWnd;	// The handle of the window to be accessed
BOOL IsList;	// Set to TRUE to modify the level properties for a list item. Set to FALSE to modify the level properties for a list-override item. A list-override item is allowed only if it has the 'OverrideLevels' flag set (please refer to the TerEditListOr function).
int id;	// The id of the list or list-override item to modify.
int level;	// A valid level number. A simple list has only one level (level number 0). A nested list has 9 levels (0 to 8).
int StartAt;	// The starting number for a level. A typical value would be 1.
int NumType;	// Use one of the following number type constants:
LIST_DEC	Decimal number
LIST_UPR_ROMAN	Uppercase roman letters

	LIST_LWR_ROMAN	Lowercase roman letters
	LIST_UPR_ALPHA	Uppercase alphabets
	LIST_LWR_ALPHA	Lowercase alphabets
	LIST_DEC_PAD	Padded decimal numbers
	LIST_BLT	Bullet (no numbering)
	LIST_NO_NUM	Hidden
int CharAft;	// The character between the bullet text and the body text:	
	LISTAFT_TAB	Tabbed space
	LISTAFT_SPACE	Single space
	LISTAFT_NONE	No space
LPBYTE ListText.	<p>// The text printed for the paragraph number. The level number information can be embedded in this text surrounded by a pair of '~' characters.</p> <p>For example, specify the following list text to print the numbers in the n.n formats:</p> <p>~1~.~2~</p> <p>The editor will replace the ~1~ string by the current number value for level 1. Similarly, the editor will replace the ~2~ string by the current number value for level 2. The result might be as following:</p> <p>1.1 Item 1 1.2 Item 2 1.3 Item 3</p> <p>Similarly, a ListText of (~1~) would print the paragraph numbers as following:</p> <p>(1) Item 1 (2) Item 2 (3) Item 3</p> <p>A bullet character can also be specified by passing a one character string. The first character of this string would contain the BLT_ constant. Please refer to the <a href="#">TerSetListBullet</a> function for a description of the available BLT_constants.</p>	



int FontId;	// A valid font id (0 to TotalFonts-1) to print the paragraph number text.
DWORD flags;	// One or more of the following flags bits can be specified:
LISTLEVEL_RESTART	Restart the paragraph number for a list-override level. This flag is valid only for a list-override level. When this flag is not specified, the numbering may be continued among multiple list-overrides for sharing the same list.
LISTLEVEL_REFORMAT	Use the font information as specified by the list-override level. This flag is valid only for a list-override level.
LISTLEVEL_LEGAL	This allows the previous upper level number to be printed in the Arabic numbering format.
LISTLEVEL_NO_RESET	Do not restart the level number when an upper level text is encountered.

**Description:** This function allows you edit the default properties for a list or list-override level.

**Return Value:** When successful, this function returns a TRUE value.

**See Also:**  
[TerEditList](#)  
[TerEditListOr](#)  
[TerSetParaList](#)  
[TerCreateListBullet](#)



## TerEditListOr

**Create or edit a list-override item.**

int TerEditListOr(hWnd, NewListOr, ListOrId, PropDialog, ListId, OverrideLevels, flags)

HWND hWnd; // The handle of the window to be accessed

BOOL NewListOr; // Set to TRUE to create a new list-override item. Set to

FALSE to modify an existing list-override item.

int ListOrId;	// The id of the list-override item to modify. This parameter is ignored when the 'NewListOr' parameter is set to TRUE.
BOOL PropDialog;	// Show a dialog box for the user to enter the list-override properties. When this parameter is set to TRUE, the values for the remaining parameters are ignored.
int ListId;	// The id of the list item to override.
BOOL OverrideLevels;	// Set to TRUE to create a new set of level information for the list-override. The level information includes properties such as number-format, starting-number, fonts, etc. When this parameter is FALSE, the list-override item simply points to the corresponding list item without any modification.
DWORD flags;	// Set to Zero. This parameters is reserved for future use.

**Description:** The list-override items are used to override the list items. Multiple list overrides can be created for a list item. This allows for various fragments of a continuous list to be displayed in different formats. Please refer to the TerEditList function for further description of list numbering mechanism..

**Return Value:** When successful, this function returns a valid list-override id. A return value of -1 indicates an error condition.

#### See Also:

[TerEditList](#)  
[TerEditListLevel](#)  
[TerSetParaList](#)  
[TerCreateListBullet](#)



## TerGetBulletInfo

**Get the paragraph bullet/numbering information.**

```
int TerGetBulletInfo(hWnd, QueryType, id, IsBullet, start, level, type, flags)
int TerGetBulletInfo2(hWnd, QueryType, id, IsBullet, start, level, type, ListOrId, flags)
int TerGetBulletInfo3(hWnd, QueryType, id, IsBullet, start, level, type, ListOrId, flags, ListText)
```

HWND hWnd;	// The handle of the window to be accessed
int QueryType;	// This parameter can be set to one of the following

values:

PID_LINE:	Get the bullet information for the given line.
PID_PARA:	Get the bullet information for the given paragraph id.
PID_BULLET:	Get the bullet information for the given bullet id.
PID_STYLE	Get the bullet information for the style id specified by the 'id' argument.

long id;	// The value for this parameter depends upon the value specified for the 'QueryType' parameter. When the 'QueryType' is set to PID_LINE, the 'id' parameter must contain a text line number. You can set the line number value to -1 to specify the current line. when the 'QueryType' is set to PID_PARA, then the 'id' parameter must contain a paragraph id. When the 'QueryType' is set to PID_BULLET, then the 'id' parameter must contain a valid bullet id. When the 'QueryType' is set to PID_STYLE, the 'id' parameter must contain a stylesheet id.
LPINT IsBullet;	// This pointer returns TRUE if the paragraph has bullets. It returns FALSE if the paragraph numbering is turned on.
LPINT start;	// This pointer returns the starting number when setting paragraph numbering is turned on.
LPINT level;	// This pointer returns the level number when paragraph numbering is turned on.
LPINT type;	// The pointer returns the symbol used for paragraph bullet/numbering. Please refer to the TerSetBulletEx function for the description of this variable.
LPINT ListOrId;	<p>// The pointer to return the list override id if this bullet is created using the list table. A zero return value for this parameter indicates a regular bullet or a regular paragraph number.</p> <p>A non-zero value for this field indicates a bullet or number created using the list-mechanism. In this case, the values returned by the 'type' and 'ListText' are not applicable. Please use the <a href="#">TerGetListLevelInfo</a> to retrieve the list information for this list.</p>
LPINT flags;	// reserved for future use.

LPBYTE ListText // List text. Please refer to the [TerEditListLevel](#) function for a description for this parameter. The ListText is valid only when ListOrId is returned as non-zero.

**Return Value:** This function returns the current bullet id. A value of 0 indicates that the bullet/numbering is not turned on for the paragraph. A value of -1 indicates an error condition.

**See Also:**  
[TerCreateBulletId](#)  
[TerSetBulletEx](#)



## TerGetListInfo

**Retrieve information for a list id.**

BOOL TerGetListInfo(hWnd, ListId, ListName, LevelCount, flags)

BOOL TerGetListInfo2(hWnd, ListId, ListName, LevelCount, flags, RtfId, TmplId)

HWND hWnd; // The handle of the window to be accessed

int ListId; // The id of the list item. Set this parameter to -1 to retrieve the information about the list id for the current line.

The TerGetListOrInfo function can be used to retrieve the list id associated with a list override id.

LPBYTE ListName; // The location to retrieve the list name

LPINT LevelCount; // The location to retrieve the number of list levels supported by the list. A nested list supports 9 levels. A simple list supports one level.

LPLONG flags; // The location to retrieve the list flag (LISTFLAG\_?). Please refer to the TerEditList for an available list of list flags.

LPLONG RtfId; // The location to retrieve the unique RTF id for this list.

LPLONG TmplId; // The location to retrieve the template id for this list.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerEditList](#)  
[TerGetListOrInfo](#)  
[TerGetListLevelInfo](#)



## TerGetListLevelInfo

**Retrieve the level properties for a list or a list-override item.**

BOOL TerGetListLevelInfo(hWnd, IsList, id, level, StartAt, NumType, CharAft, ListText, FontId, flags)

HWND hWnd;	// The handle of the window to be accessed
BOOL IsList;	// Set to TRUE to retrieve the level properties for a list item. Set to FALSE to retrieve the level properties for a list-override item. A list-override item is allowed only if it has non-zero list levels (please refer to the TerGetListOrInfo function).
int id;	// The id of the list or list-override item to retrieve. You can set this parameter to -1 to retrieve the information about the list level for the current line.
int level;	// A valid level number. A simple list has only one level (level number 0). A nested list has 9 levels (0 to 8). This parameter is ignored when the 'id' parameter is set to -1.
LPINT StartAt;	// The location to retrieve the starting number for a level.
LPINT NumType;	// The location to retrieve the number type used for the bullet. Please refer to the TerEditListLevel function for a list of number type constants.
LPINT CharAft;	// The location to retrieve the character between the bullet text and the body text. Please refer to the TerEditListLevel function for a list of values returned by this parameter.
LPBYTE ListText.	// The location to retrieve the text printed for the paragraph number. Please refer to the TerEditListLevel function for the description of this parameter.
LPINT FontId;	// The location to retrieve the font id to print the paragraph number text.
LPDWORD flags;	// The location to retrieve the list level flags. Please refer to the TerEditListLevel function for a description of the list level flags.

**Return Value:** When successful, this function returns a TRUE value.

**See Also:**  
[TerGetListInfo](#)  
[TerGetListOrInfo](#)  
[TerEditListLevel](#)



## TerGetListOrInfo

**Retrieve information for a list override id.**

BOOL TerGetListOrInfo(hWnd, ListOrId, ListId, LevelCount, flags)

HWND hWnd;	// The handle of the window to be accessed
int ListOrId;	// The id of the list override item. Set this parameter to -1 to retrieve the information about the list override id for the current line.  The TerGetBulletInfo2 function can be used to retrieve the list override id associated with the current line or a paragraph id.
LPINT ListId;	// The location to retrieve the list id for this list override id.
LPINT LevelCount;	// The location to retrieve the number of list levels supported by the list override id. A non-zero value indicates that this list override id actually overrides the level information for the underlying list id.
LPLONG flags;	// The location to retrieve the list override flags. This parameter is not used currently and always returns a zero value.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerGetBulletInfo](#)  
[TerEditListOr](#)  
[TerGetListInfo](#)  
[TerGetListLevelInfo](#)



## TerSetDefListFormat

**Specify default list level properties to apply when creating a new list.**

BOOL TerSetDefListFormat(hWnd, level, NumType, ListText)

HWND hWnd;	// The handle of the window to be accessed
int id;	// The id of the list or list-override item to modify.
int level;	// Level number to apply the properties (0 to 8).
int NumType;	// Use one of the following number type constants:
	LIST_DEC                      Decimal number
	LIST_UPR_ROMAN              Uppercase roman letters
	LIST_LWR_ROMAN              Lowercase roman letters
	LIST_UPR_ALPHA              Uppercase alphabets
	LIST_LWR_ALPHA              Lowercase alphabets
	LIST_DEC_PAD                Padded decimal numbers
	LIST_BLT                     Bullet (no numbering)
	LIST_NO_NUM                 Hidden
LPBYTE ListText.	<p>// The text printed for the paragraph number. The level number information can be embedded in this text surrounded by a pair of '~' characters.</p> <p>For example, specify the following list text to print the numbers in the n.n formats:</p> <p>~1~.~2~</p> <p>The editor will replace the ~1~ string by the current number value for level 1. Similarly, the editor will replace the ~2~ string by the current number value for level 2. The result might be as following:</p> <p>1.1 Item 1 1.2 Item 2 1.3 Item 3</p> <p>Similarly, a ListText of (~1~) would print the paragraph numbers as following:</p> <p>(1) Item 1 (2) Item 2 (3) Item 3</p>

**Description:** The properties specified using this method is applicable only to a new list subsequently created using any of the list creation methods.

**Return Value:** When successful, this function returns a TRUE value.



## TerSetListBullet

**Set the paragraph bullet/numbering property.**

BOOL TerSetListBullet(hWnd, set, NumType, level, start, TextBef, TextAft, repaint)

BOOL TerSetListBullet2(hWnd, set, NumType, level, start, TextBef, TextAft, ListText, repaint)

HWND hWnd;	// The handle of the window to be accessed.
BOOL set;	// TRUE to set the paragraph bullet/numbering or FALSE to remove it. The 'start', 'level', and 'type' parameters are ignored when the 'set' parameter is FALSE.
int NumType	// This parameter allows you to apply a bullet or a numbered list. Please refer to the <a href="#">TerEditListLevel</a> function for the list of available constants.
int level;	// list level (0 to 8). The default value for this parameter is 0.
int start;	// The starting number when setting paragraph numbering. Set to 1 for default.
LPBYTE TextBef;	// Text before the paragraph number. Set to "" for default. This parameter is ignored when ListText is not blank.
LPBYTE TextAft;	// Text After the paragraph number. Set to "" for default. This parameter is ignored when ListText is not blank.
LPBYTE ListText;	// List text. For NumType parameter value other than NUM_BLT, please refer to the <a href="#">TerEditListLevel</a> function for a description for this parameter.

When NumType parameter is LIST\_BLT, the bullet character can be specified by passing one of the following constants as a one-character string to this parameter:

BLT\_ROUND              Round bullet

BLT\_SQUARE            Square bullet

BLT\_HOLLOW\_SQUA      Hollow square bullet



RE

BLT\_CHECK              Check mark bullet

BLT\_DIAMOND            Diamond bullet

BLT\_4\_DIAMONDS        4 Diamonds bullet

BLT\_ARROW              Arrow bullet

BOOL repaint;                      // Repaint the screen after this operation

### Description:

The TerSetListBullet method is a wrapper to the basic list mechanism functions: TerEditList, TerEditListOr, TerEditListLevel and TerSetParaList. This method provides most used features of the basic list functions. However, you might like to use the basic list function if you need further control of the lists. Please refer to the [List Numbering](#) chapter for a discussion of basic list functions.

### Examples:

**Create a decimal numbered list of level 0 where the numbering prints as:**

**(1), (2), (3), Etc**

```
TOC.TerSetListBullet2(true,LIST_DEC,0,1,"",""("~1~"),true);
```

**Create a decimal numbered list of level 0 where the numbering prints as:**

**1., 2., 3., Etc.**

```
TOC.TerSetListBullet2(true,LIST_DEC,0,1,"",""~1~.",true);
```

**Create a decimal numbered list of level 1 where the numbering prints as:**

**1.1, 1.2, 1.3, etc.**

```
TOC.TerSetListBullet2(true,LIST_DEC,1,1,"",""~1~.~2~",true);
```

**Create a standard bullet list of level 1 with round bullet:**

```
TOC.TerSetListBullet2(true,LIST_BLT,1,1,"",""~1~.",true);
```

**Create a diamond shaped bullet list of level 1:**

```
bullet$=chr$(BLT_DIAMOND)
```

```
TOC.TerSetListBullet2(true,LIST_BLT,1,1,"",""bullet$,true);
```

**Return Value:** This function returns TRUE when successful.

See Also  
[TerSetListLevel](#)



## TerSetListLevel

**Set the level for a list.**

BOOL TerSetListLevel(hWnd, level, increment, repaint)

HWND hWnd;	// The handle of the window to be accessed.
int level;	// list level (0 to 8). The outmost level is 0. Set this parameter to -1 to increase or decrease the level using the 'increment' parameter.
int increment	// Specify the number to increase or decrease the current level. For example, a value of 1 will increase the level by one. A value of -1 will decrease the level by one.  This parameter is used only when the 'level' parameter is set to -1.

**Return Value:** This function returns TRUE when successful.

See Also  
[TerSetListBullet](#)



## Toolbar

### In This Chapter

[TerAddToolBarIcon](#)  
[TerEditTooltip](#)  
[TerHideToolBarIcon](#)  
[TerRecreateToolBar](#)  
[TerSetToolBarComboWidth](#)  
[TerUpdateToobar](#)



## TerAddToolBarIcon

**Add a new icon to the toolbar**

BOOL TerAddToolBarIcon(hWnd, LineNo, TlbId, CmdId, BmpFile, balloon)

HWND hWnd;	// The handle of the window to be accessed
int LineNo;	// Set to 0 to add the icon to the top row of the toolbar. Set to 1 to add this icon to the bottom row of the toolbar.
int TlbId;	// Set this parameter to one of the toolbar ids. Please refer to TerEditTooltip function for a list of toolbar icon ids. This would add an existing toolbar icon. The CmdId and BmpFile parameters are ignored when this parameter is non-zero.  Set this parameter to 0 to add a new tool bar icon.
int CmdId;	// When the TlbId parameter is set to zero, use the CmdId parameter to specify the command id for the new toolbar icon. Please refer to the <a href="#">'Command'</a> property for a list of existing command ids.  You can use command ids ID_USER1 to ID_USER9 to implement functionality not provided by the existing command ids. For example, you can use ID_USER1 to implement PDF output icon. You would intercept ID_USER1 using the PreProcess event and execute your code there. Then call the TerIgnoreCommand method to tell the editor to ignore this command.
LPBYTE BmpFile;	// The bitmap file to draw the toolbar icon. The toolbar icon consists of 20x20 pixels. Therefore, the picture must be at least 20 pixels wide and 20 pixels tall. Only first 20x20 pixels are used. The remaining pixels are ignored.
LPBYTE balloon;	// The tool tip text string.

### Example:

```
// Add a custom icon
TerAddToolBarIcon(hWnd,0,0,ID_PARA_KEEP,"pict.bmp","Keep");

// Add spacers and vertical separating lines
TerAddToolBarIcon(hWnd,0,TLB_SPACER,0,NULL,NULL);
TerAddToolBarIcon(hWnd,0,TLB_LINE,0,NULL,NULL);
TerAddToolBarIcon(hWnd,0,TLB_SPACER,0,NULL,NULL);

// Add an existing icons
TerAddToolBarIcon(hWnd,0,TLB_PASTE,0,NULL,NULL);

// more spacers and vertical lines
```

```

TerAddToolBarIcon(hWnd, 0, TLB_SPACER, 0, NULL, NULL);
TerAddToolBarIcon(hWnd, 0, TLB_LINE, 0, NULL, NULL);

// display the modified toolbar
TerRecreateToolBar(hWnd, TRUE);

```

**Return Value:** This function returns a TRUE value if successful.

#### See Also

[TerEditTooltip](#)

[TerRecreateToolBar](#)



## TerEditTooltip

**Specify a new tooltip text.**

```
int TerEditTooltip(hWnd, id, tooltip)
```

HWND hWnd;                      // The handle of the window to be accessed

int id;                            // The toolbar icon id. The following is a list of toolbar icon ids:

TLB_LINE	Vertical line in the toolbar. Tooltip is not applicable to this id.
TLB_TYPEFACE	Font typeface combobox
TLB_POINTSIZE	Font pointsize combobox
TLB_BOLD	Bold style
TLB_ITALIC	Italic style
TLB_ULINE	Underline style
TLB_ALIGN_LEFT	Align left
TLB_ALIGN_RIGHT	Align right
TLB_ALIGN_CENTER	Paragraph centering
TLB_ALIGN_JUSTIFY	Align both

TLB_INC_INDENT	Increase paragraph indentation
TLB_DEC_INDENT	Decrease paragraph indentation
TLB_STYLE	Stylesheet item combobox
TLB_ZOOM	Zoom combobox
TLB_CUT	Clipboard cut
TLB_COPY	Clipboard copy
TLB_PASTE	Clipboard paste
TLB_SPACER	Space between toolbar icons
TLB_NEW	File New
TLB_OPEN	File Open
TLB_SAVE	File Save
TLB_PRINT	Print
TLB_HELP	Help
TLB_PAR	Show paragraph markers
TLB_PREVIEW	Print preview
TLB_NUMBER	Set paragraph numbering
TLB_BULLET	Set paragraph bullet
TLB_UNDO	Undo
TLB_REDO	Redo
TLB_FIND	Text search
TLB_DATE	Insert a date field
TLB_PAGE_NUM	Insert a page number field
TLB_PAGE_COUNT	Insert a page count field
TLB_PAINT_FORMAT	Format copy/paste

TLB\_BULLET\_DROPD   Bullet library drop down  
OWN

LPBYTE tooltip;                   // The new tooltip text.

**Return Value:** This function returns TRUE when successful.



## TerHideToolbarIcon

**Hide or redisplay a toolbar icon.**

BOOL TerHideToolbarIcon(id, hide)

int id;                            // The id of the icon to hide. Please refer to  
TerEditTooltip function for a list of toolbar icon ids.

BOOL hide;                        // Set to TRUE to hide the icon, or set to FALSE to  
redisplay a previously hidden icon.

**Description:** Please note that the changes made by this function are not displayed on an 'existing' TE control window until the toolbar is recreated using the TerRecreateToolbar function. You would typically call this function multiple times to hide more than one icons and then call the TerReceateToolbar function once to redisplay the modified toolbar. When a new control is created, it would automatically hide the icons flagged by this function.

**Return Value:** This function returns a TRUE value if successful.

**See Also:**  
[TerRecreateToolbar](#)  
[TerEditTooltip](#)



## TerRecreateToolbar

**Recreate the toolbar.**

BOOL TerRecreateToolbar(hWnd, show)

HWND hWnd;                       // The handle of the text window to be accessed

BOOL show;                        // Show the toolbar after it is recreated.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerHideToolbarIcon](#)



## TerSetToolbarComboWidth

**Set the width of a combo-box on the toolbar.**

BOOL TerUpdateToolbar(hWnd, id, width, recreate)

HWND hWnd;	// The handle of the window to be accessed
int id;	// Set this parameter to one of the toolbar ids corresponding to a combo-box toolbar item. Please refer to TerEditTooltip function for a list of toolbar icon ids.
int width;	// New width of the combo-box in screen pixels.
BOOL recreate;	// Set to TRUE to recreate the toolbar with new combo-width.

**Return Value:** This function returns TRUE when successful.



## TerUpdateToolbar

**Update toolbar.**

BOOL TerUpdateToolbar(hWnd)

HWND hWnd;	// The handle of the window to be accessed
------------	--

**Description:** This function should be called to update the toolbar after calling an API function which might need refreshing of the toolbar.

**Return Value:** This function returns TRUE when successful.



## Undo

**In This Chapter**

[TerFlushUndo](#)  
[TerSetMaxUndo](#)  
[TerSetUndoRef](#)



## TerFlushUndo

**Flush undo/redo buffer.**

BOOL TerFlushUndo(hWnd)

HWND hWnd; // The handle of the window to be accessed

**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerSetUndoRef](#)



## TerSetMaxUndo

**Set maximum undo or redo levels.**

int TerSetMaxUndo(hWnd, MaxUndo)

HWND hWnd; // Window handle to access

int MaxUndo; // The maximum number of undo or redo allowed. The default value is 40.

Return Value: This functions returns TRUE when successful.

See Also: TerFlushUndo, TerSetUndoRef



## TerSetUndoRef

Set or retrieve undo reference id.

int TerSetUndoRef(hWnd, UndoRef)

HWND hWnd; // Window handle to access



```
int UndoRef; // The undo reference id. Set this parameter to -1 to
              simply retrieve the current undo reference id.
```

**Comment:** This function can be used to connect multiple API calls to one undo buffer. You would first call this function before any API is called to retrieve the current undo reference count. You would then call this function before each API call to reset the undo reference to the initial value.

**Return Value:** This function returns the previous value of the undo reference id when successful. Otherwise it returns -1.

#### See Also:

[TerFlushUndo](#)  
[TerSetMaxUndo](#)



## Input Field

#### In This Chapter

[TerGetCheckboxInfo](#)  
[TerGetComboboxInfo](#)  
[TerGetInputFieldInfo](#)  
[TerGetTextFieldInfo](#)  
[TerInsertCheckBoxField](#)  
[TerInsertComboBoxField](#)  
[TerInsertTextInputField](#)  
[TerLocateInputField](#)  
[TerSetCheckboxInfo](#)  
[TerSetComboboxInfo](#)  
[TerSetInputFieldInfo](#)  
[TerSetTextFieldInfo](#)



## TerGetCheckboxInfo

**Retrieve the information for a checkbox input field.**

```
BOOL TerGetCheckboxInfo(hWnd, id, checked)
```

```
HWND hWnd; // The handle of the window to be accessed
```

```
int id; // Input field id to retrieve information.
```

```
LPINT checked; // This location receives a TRUE value if the checkbox is
                checked. Otherwise it returns a FALSE value.
```

**Return Value:** This function returns TRUE when successful.

#### See Also:

[TerLocateInputField](#)  
[TerGetInputFieldInfo](#)  
[TerGetTextFieldInfo](#)  
[TerSetCheckboxInfo](#)



## TerGetComboboxInfo

**Retrieve the information for a combo-box input field.**

int TerGetComboboxInfo(hWnd, id, out items)

HWND hWnd; // The handle of the window to be accessed

int id; // Id of the combo-box field to retrieve information.

LPBYTE items; // A pointer of a string to receive the list of combo-box items. Each item in the list is delimited by a '|' character.

**Return Value:** This function returns a zero based index of the selected item when successful. A value of -1 indicates an error condition.



## TerGetInputFieldInfo

**Retrieve the common input field information.**

BOOL TerGetInputFieldInfo(hWnd, id, name, type, border)

HWND hWnd; // The handle of the window to be accessed

int id; // Input field id to retrieve information.

LPBYTE name; // The location to receive the field name.

LPINT type; // The location to receive the field type:

FIELD\_TEXTBOX      Text box field

FIELD\_CHECKBOX      Checkbox field

LPINT border; // The location to receive the border information. The argument returns TRUE if the field has border, otherwise it returns a FALSE value.

**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerLocateInputField](#)  
[TerGetCheckboxInfo](#)  
[TerGetTextFieldInfo](#)  
[TerSetInputFieldInfo](#)



## TerGetTextFieldInfo

**Retrieve the information for a textbox input field.**

BOOL TerGetTextFieldInfo(hWnd, id, data, MaxChars, width, typeface, TwipsSize, style)

HWND hWnd;	// The handle of the window to be accessed
int id;	// Input field id to retrieve information.
LPBYTE data;	// The location to receive the current text data in the text box.
LPINT MaxChars;	// The location to receive the current maximum text length for the text box.
LPINT width;	// The location to receive the text box width in twips.
LPBYTE typeface;	// The location to receive the font typeface for the text in the text box.
LPINT TwipsSize;	// The location to receive the font typeface size in twips (20 twips = 1 point).
LPINT style;	// The location to receive the style information for the font:

BOLD	Bold style
ITALIC	Italic style
ULINE	Underline style

**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerLocateInputField](#)  
[TerGetInputFieldInfo](#)  
[TerGetCheckboxInfo](#)  
[TerSetTextFieldInfo](#)



## TerInsertCheckBoxField

**Insert a checkbox field at the cursor position.**

int TerInsertCheckBoxField(hWnd, name, twipsize, checked, repaint)

HWND hWnd;	// The handle of the window to be accessed
LPBYTE name;	// field name
int twipsize;	// The width of the checkbox specified in Twips unit.
BOOL checked;	// Set to TRUE to create the checkbox initially checked.
BOOL Insert;	// Set to TRUE to insert the checkbox. Set to FALSE to create a check-box object, but do not insert it in the text.
BOOL repaint;	//Repaint the window after this operation

**Description:** This function inserts a checkbox.

**Return Value:** This function returns the picture id of the field. It returns 0 to indicate an error condition.

### See Also:

[TerInsertTextField](#)  
[TerLocateInputField](#)  
[TerGetCheckboxInfo](#)



## TerInsertComboBoxField

**Insert a combo-box field at the cursor position.**

int TerInsertComboBoxField(hWnd, name, items, SelectedItem, insert, repaint)

HWND hWnd;	// The handle of the window to be accessed
LPBYTE name;	// field name
LPBYTE items;	// List of items in the combo-box. Each item must be delimited by a ' ' character. Example: "Red Blue Green Yellow".
LPBYTE SelectedItem;	// The value of the selected item. Example: "Blue".

BOOL Insert;	// Set to TRUE to insert the combo-box. Set to FALSE to create a combo-box object, but do not insert it in the text.
BOOL repaint;	//Repaint the window after this operation

**Description:** This function inserts a combo-box.

**Return Value:** This function returns the picture id of the field. It returns 0 to indicate an error condition.



## TerInsertTextInputField

**Insert a text input field at the cursor position.**

int TerInsertTextInputField(hWnd, name, InitText, MaxLen, border, typeface, twipsize, style, color, insert, repaint)

HWND hWnd;	// Handle of the window to be accessed
LPBYTE name;	// field name
LPBYTE InitText;	// Initial text
int MaxLen;	// Maximum number of characters allowed in the field. Set to 0 to specify no limit.
BOOL border;	// Set to TRUE to draw the box around the text field.
LPBYTE typeface;	// font typeface for the field text.
int twipsize;	// Font pointsize specified in Twips unit. (20 Twips = 1 Point).
UINT style;	// Style (BOLD, ULINE, ITALIC) for the text. Use the logical OR operator to specify more than one style constants.
COLORREF color;	// Foreground color for the text. This argument is ineffective currently. Set to 0.
BOOL insert;	// Set to TRUE to insert the new input field into the document. Set to FALSE to simply return the picture id of the new field without inserting into the text.
BOOL repaint;	//Repaint the window after this operation

**Description:** This function inserts a text box. The user can input data into the text box.

**Return Value:** This function returns the picture id of the text input field. It returns 0 to indicate an error condition.

**See Also:**

[TerInsertCheckBoxField](#)

[TerLocateInputField](#)

[TerGetInputFieldInfo](#)

[TerGetTextFieldInfo](#)



## TerLocateInputField

**Locate an input field in the document.**

int TerLocateInputField(hWnd, location, repaint)

HWND hWnd; // Handle of the window to be accessed

int location; // Use one of the following constants:

TER\_FIRST: Search from the top of the file and locate the first occurrence of the input field.

TER\_LAST: Search from the bottom of the file and locate the last occurrence of the input field.

TER\_NEXT: Find the next occurrence of the input field.

TER\_PREV: Find the previous occurrence of the input field.

TER\_CUR: Current field with focus

BOOL repaint; // Set to TRUE to repaint the screen after this operation.

**Return Value:** This function returns the object id of the located field when successful. Otherwise, it return 0.

**See Also:**

[TerInsertCheckBoxField](#)

[TerGetInputFieldInfo](#)

[TerSetInputFieldInfo](#)



## TerSetCheckboxInfo

**Set the information for a checkbox input field.**

BOOL TerGetCheckboxInfo(hWnd, id, checked)

HWND hWnd; // The handle of the window to be accessed.

int id; // Input field id to retrieve information.

BOOL checked; // Set to TRUE to check the checkbox.

**Return Value:** This function returns TRUE when successful.

### See Also:

[TerLocateInputField](#)

[TerSetInputFieldInfo](#)

[TerSetTextFieldInfo](#)

[TerGetCheckboxInfo](#)



## TerSetComboboxInfo

**Set the information for a combo-box input field.**

BOOL TerSetComboboxInfo(hWnd, id, Items, SellItemIdx, repaint)

HWND hWnd; // The handle of the window to be accessed.

int id; // Input field id to retrieve information.

LPBYTE items; // A string containing a list of new combo-box items. Each item in the list should be delimited by a '|' character.  
Example: "Red|Blue|Green|Yellow"

int SellItemIdx; // A zero based index of the item in the 'items' parameter to be selected automatically. For example, to select the item 'Blue', you would pass a value of 1 for this parameter.

BOOL repaint // Set to TRUE to repaint the control after this operation.

**Return Value:** This function returns TRUE when successful.



## TerSetInputFieldInfo

**Set the common input field information.**

BOOL TerSetInputFieldInfo(hWnd, id, name, border)

HWND hWnd;	// The handle of the window to be accessed
int id;	// Input field id to retrieve information.
LPBYTE name;	// The new field name.
BOOL border;	// Set to TRUE to set the border around the field. This argument is applicable to the textbox input field only.

**Return Value:** This function returns TRUE when successful.

### See Also:

[TerLocateInputField](#)

[TerSetCheckboxInfo](#)

[TerSetTextFieldInfo](#)

[TerGetInputFieldInfo](#)



## TerSetTextFieldInfo

**Set new information for a textbox input field.**

BOOL TerSetTextFieldInfo(hWnd, id, data, MaxChars, width, typeface, TwipsSize, style)

HWND hWnd;	// The handle of the window to be accessed
int id;	// Input field id to retrieve information.
LPBYTE data;	// The new text data for the text box.
int MaxChars;	// The maximum text length for the text box. Set to 0 for no maximum limit.
int width;	// This is a reserved field. It should be set to 0.
LPBYTE typeface;	// The font typeface for the text in the text box.
int TwipsSize;	// The font typeface size in twips (20 twips = 1 point).
uint style;	// The style information for the font. Set to 0 for default. The following styles can be selected for the text box.



BOLD      Bold style  
ITALIC    Italic style  
ULINE     Underline style

Please use the logical OR operator to specify more than one flag.

**Return Value:** This function returns TRUE when successful.

#### See Also:

[TerLocateInputField](#)  
[TerSetInputFieldInfo](#)  
[TerSetCheckboxInfo](#)



## Page

This chapter includes page manipulation functions. Please also refer to the [Section Formatting](#) page for other related functions.

#### In This Chapter

[TerGetDispPageNo](#)  
[TerGetPageBorderDim](#)  
[TerGetPageCount](#)  
[TerGetPageFirstLine](#)  
[TerGetPageNumFmt](#)  
[TerGetPageParam](#)  
[TerGetPagePos](#)  
[TerGetPageSect](#)  
[TerInsertPageRef](#)  
[TerPageBreak](#)  
[TerPageFromLine](#)  
[TerPosPage](#)  
[TerRepaginate](#)  
[TerSetPageBkColor](#)  
[TerSetPageNumFmt](#)  
[TerSetPagePos](#)



## TerGetDispPageNo

**Get the display page number corresponding to an actual page number.**

```
int TerGetDispPageNo(hWnd, PageNo)
```

HWND hWnd;                                      // The handle of the window to be accessed

int PageNo; // Actual page number between 0 to TotalPages-1.

**Return Value:** The display page number can be different from the actual page number since the display page number can be reset at the section breaks.

**See Also:**  
[TerGetPagePos](#)  
[TerGetField](#)



## TerGetPageBorderDim

**Return the page border dimension.**

long TerGetPageBorderDim(hWnd, pWidth, pHeight)

HWND hWnd; // The handle of the window to be accessed

LPINT pWidth; // The location to retrieve the page border width (left and right) in twips.

LPINT pHeight; // The location to retrieve the top page border height in twips.

**Description:** This function is available in the PageMode or PrintView mode only.

**Return Value:** This function returns TRUE when successful.



## TerGetPageCount

**Get the page count and current page number.**

BOOL TerGetPageCount(hWnd, TotalPages, CurPage)

HWND hWnd; // The handle of the window to be accessed

LPINT TotalPages; // This variable receives the total number of pages in the document

LPINT CurPage; // This variable receives the current page number (Zero based)

**Comment:** This function is valid in the PageMode only.

**Return Value:** This function returns TRUE when successful.

**See Also:**  
[TerSetPagePos](#)  
[TerGetDispPageNo](#)



## TerGetPageFirstLine

**Return the first line number of the specified page.**

```
long TerGetPageFirstLine(hWnd, PageNum)
```

HWND hWnd; // The handle of the window to be accessed

int PageNum; // Page number between 0 and TotalPages-1

**Description:** This function is available in the PageMode or PrintView mode only.

**Return Value:** This function returns the first line of the specified page number.

**See Also:**  
[TerPageFromLine](#)



## TerGetPageNumFmt

**Retrieve the print format for the page number string.**

```
int TerGetPageNumFmt(hWnd, sect, format)
```

HWND hWnd; // The handle of the window to be accessed

int sect; // Section id to retrieve the page number format. You can also set this parameter to SECT\_CUR to specify the current section.

**Return Value:** This function returns one of the following constants.

LIST_DEC	Decimal number
LIST_UPR_ROMAN	Uppercase roman letters
LIST_LWR_ROMAN	Lowercase roman letters
LIST_UPR_ALPHA	Uppercase alphabets
LIST_LWR_ALPHA	Lowercase alphabets

A return value of -1 indicates an error condition.

**See Also**  
[TerSetPageNumFmt](#)



## TerGetPageParam

**Get the page parameters.**

int TerGetPageParam(hWnd, pageNo, type)

HWND hWnd;	// The handle of the window to be accessed
int pageNo;	// Page number (zero based)
int type;	// The parameter to retrieve:
PP_PAGE_HDR_HT	Height of the page header area in twips unit.
PP_PAGE_BODY_HT	Height of the page body area (area between the header and footer) in twips unit.
PP_PAGE_FTR_HT	Height of the page footer area in twips unit.
PP_PAGE_BODY_AVAIL_HT	Height of the available area on the body of the page in twips unit.
PP_TOP_SECT	Section id at the top of the page.
PP_FIRST_LINE	First line of the page
PP_LAST_LINE	Last line of the page
PP_DISP_NO	Display page number.

**Return Value:** The function returns the value for the requested parameter. It returns FP\_ERROR to indicate an error condition.



## TerGetPagePos

**Get the current page number and the display offset.**

BOOL TerGetPagePos(hWnd, page, y)

HWND hWnd; // The handle of the window to be accessed

LPINT page; // This variable receives the current page number

LPINT y; // This variable receives the offset of the top of the window relative to the top of the page. This value is returned in twips unit.

**Return Value:** This function returns TRUE when successful.

**See Also:**

[TerGetPageCount](#)

[TerSetPagePos](#)

[TerGetDispPageNo](#)



## TerGetPageSect

**Return the section id for a page.**

int TerGetPageSect(hWnd, PageNum)

HWND hWnd; // The handle of the window to be accessed

int PageNum; // Page number between 0 and TotalPages-1

**Description:** This function is available in the PageMode or PrintView mode only.

**Return Value:** This function returns the section number containing the first line of a page.



## TerInsertPageRef

**Insert page reference field.**

BOOL TerInsertPageRef(hWnd, bookmark, IsHyperlink, IsAlphabetic, repaint)

HWND hWnd; // The handle of the window to be accessed

LPBYTE bookmark; // The name of the bookmark to build a reference. The bookmark must already exists in the document. You can use the TerInsertBookmark function to insert a

bookmark.

BOOL IsHyperlink;                      // Set to True to treat this page reference as a hyperlink. When the user click on this page-reference hyperlink, the cursor will jump to the referenced bookmark.

The hyperlink cursor must be turned on using the ID\_SHOW\_HYPERLINK\_CURSOR command to display the hyperlink cursor.

BOOL IsAlphabetic;                      // Set to True to display the page number in the alphabetic format.

BOOL repaint;                            //Repaint the window after this operation

**Comment:** The page-reference field displays the page number where the referenced bookmark is located.

**Return Value:** This function returns a True value when successful, otherwise it returns a false value.

#### See Also

[TerInsertBookmark](#)  
[TerInsertTOC](#)



## TerPageBreak

**Create a hard page break.**

BOOL TerPageBreak(hWnd,repaint)

HWND hWnd;                              // Handle of the window to be accessed

BOOL repaint;                            //Repaint the window after this operation

**Description:** This function is used to place the following text on the new page. The page break is created before the current line. Please note that a page break can not be created inside an object such as table, frame, text box, etc.

A page break is indicated by a solid line.

**Return Value:** This function returns TRUE if successful.

#### See Also:

[TerColBreak](#)  
[TerSectBreak](#)



## TerPageFromLine

**Retrieve the page number containing the given line.**

int TerPageFromLine(hWnd,LineNo)

HWND hWnd; // Handle of the window to be accessed

long LineNo; // Line number (0 to TotalLines - 1)

**Return Value:** This function returns the page number that contains the given text line number. A value of -1 indicates an error condition.

**See Also:**

[TerGetPageFirstLine](#)



## TerPosPage

**Position at the specified page number.**

BOOL TerPosPage (hWnd, NewPage)

HWND hWnd; // Handle of the window to be accessed

intnew Page; // Page number (0 to TotalPages - 1) to position at.

**Description:** This function is available in the Page Mode only.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**

[TerPosTable](#)

[TerGetSeqSect](#)



## TerRepaginate

**Repaginate the document.**

int TerRepaginate(hWnd, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL repaint; // TRUE to repaint the document after this operation

**Description:** This function rewraps and repaginates a document. This function should be used instead of the TerRewrap function in the PrintView, Page mode, and Fitted View

modes.

**Return Value:** The function returns TRUE when successful

**See Also:**

[TerRewrap](#)

[TerReformatTable](#)



## TerSetPageBkColor

**Set page background color.**

BOOL TerSetPageBkColor(hWnd, BkColor)

HWND hWnd;                               // Handle of the window to be accessed

COLORREF BkColor;                       //The background color for the page

**Description:** This function is used set a background color for the page. If the page borders are active, then the color is applied to the area inside the page borders.

Use the TerGetParam function to retrieve the current page background color.

**Example:**

```
TerSetPageBkColor(hWnd,0xFF); // set page color to red
TerRepaint(hWnd,FALSE); // repaint to show the color
```

**Return Value:** This function returns TRUE if successful.



## TerSetPageNumFmt

**Set the print format for the page number string.**

BOOL TerSetPageNumFmt(hWnd, sect, format)

HWND hWnd;                               // The handle of the window to be accessed

int sect;                                 // Section id to apply changes. You can also set this parameter to SECT\_CUR to edit the current section, or set it to SECT\_ALL to apply changes to all sections in the document.

int format;                               // The 'format' parameter can be set to one of the following constants:



NBR_DEC	Decimal number
NBR_UPR_ROMAN	Uppercase roman letters
NBR_LWR_ROMAN	Lowercase roman letters
NBR_UPR_ALPHA	Uppercase alphabets
NBR_LWR_ALPHA	Lowercase alphabets

BOOL refresh;                      // TRUE to refresh the window after this operation

**Return Value:** This function returns TRUE if successful.

#### See Also

[TerGetPageNumFmt](#)



## TerSetPagePos

**Position at the specified page number and at the specified display offset with the page.**

BOOL TerSetPagePos(hWnd, page, y)

HWND hWnd;                      // The handle of the window to be accessed

int page;                      // This variable specifies the page number (0 to TotalPages -1) to position at.

int y;                      // This variable specifies the offset of the top of the window relative to the top of the page. This value is specified in the twips unit.

**Return Value:** This function returns TRUE when successful.

#### See Also:

[TerGetPagePos](#)



## Search, Replace, Locate

#### In This Chapter

[TerLocateFieldChar](#)

[TerSearchReplace](#)

[TerSearchReplaceU](#)

[TerSetSearchString](#)



## TerLocateFieldChar

**Locate the character with the given field id.**

BOOL TerLocateFieldChar(hWnd,FieldId,FieldCode, present, StartLine, StartCol, forward)

HWND hWnd;	// Handle of the window to be accessed
UINT FieldId;	// Select one of the following field ids to locate:
	FIELD_NAME: Mail-merge field name
	FIELD_DATA: Mail-merge field data
	FIELD_HLINK: Hyperlink field
	FIELD_TEXTBOX Input text field
LPBYTE FieldCode;	// Field code associated with Field Id. Pass a NULL value for the field ids (such as FIELD_NAME and FIELD_DATA) that do not use field code.
BOOL present;	// TRUE to test for the presence of the given field id, or FALSE to test for the absence of the given field id.
long far *StartLine;	(INPUT/OUTPUT) Specifies the pointer to the line number to start the search. On a successful search, this parameter contains the line number of the located text.
int far *StartCol;	(INPUT/OUTPUT) Specifies the pointer to the column number to start the search. On a successful search, this parameter contains the column number of the located text.
BOOLforward;	// TRUE to scan the text in the forward direction, or FALSE to scan the text in the backward direction.

**Return Value:** This function returns a TRUE value when successful.

**See Also:**

[TerLocateField](#)



## TerSearchReplace

### Search, replace or retrieve text.

BOOL TerSearchReplace(hWnd, search, replace, flags, StartPos, pEndPos, pBufSize)

Int TerSearchReplace2(hWnd, search, replace, flags, StartPos, EndPos)

HWND hWnd;	// The handle of the window to be accessed.
LPBYTE search;	// search text string
LPBYTE replace;	// replace text string
UINT flags;	// mode flags
long StartPos;	// start text position
long far *pEndPos;	// pointer to the end text position. This parameter is not used by the TerSearchReplace2 function.
Long EndPos;	// The end position of the string to replace. This parameter is not used by the TerSearchReplace function.
long far *pBufSize;	// pointer to the size of the retrieved text. This parameter is not used by the TerSearchReplace2 function.

Description: This function has three operational modes: search, replace and retrieve.

**Search Mode:** To initiate this mode, set the SRCH\_SEARCH flag in the 'flags' parameter. You can also set the following bits in the 'flags' parameter:

SRCH_CASE	Case sensitive search
SRCH_WORD	Match whole words
SRCH_SCROLL	Scroll the located text into view.
SRCH_SKIP_HIDDEN_TEXT	Skip over the hidden text
SRCH_BACK	Search in the backward direction
SRCH_NO_REPLACE_PROT_TEXT	Do not replace protected text.

The search string is passed via the 'search' parameter. The search string consists of regular text and certain special characters. The special characters are inserted using the '^' prefix:

- ^p Paragraph character
- ^t Tab character
- ^m Manual page break
- ^b Section break
- ^+ Em dash
- ^- En dash
- ^^ ^ character.

The initial search location is specified by the 'StartPos' parameter. This parameter specifies the absolute character position since the beginning of the file.

If the search string is located, the *TerSearchReplace* function returns its absolute character position via the 'pEndPos' long pointer.

If the search string is located, the **TerSearchReplace2** function returns its absolute character position via function return value. If the string is not found, then this function returns -1.

**Replace Mode:** To initiate this mode, set the SRCH\_REPLACE flag in the 'flags' parameter. This function replaces the text between the 'StartPos' and 'pEndPos' (or EndPos for the TerSearchReplace2 function) absolute character positions by the specified replacement text. The replacement text is specified by the 'replace' argument.

**Retrieve Mode:** To initiate this mode, set the SRCH\_RETRIEVE flag in the 'flags' parameter. This function retrieves the text between the 'StartPos' and 'EndPos' absolute character positions. The size of the retrieved text is returned via the 'BufSize' long pointer. The actual text is returned via the 'search' argument. This function returns even the hidden text.

Comments: This **Retrieve** mode is not available when this function is used as an ActiveX control method. The function must be used as a DLL function to use the retrieve mode.

#### **Return Value for the TerSearchReplace function:**

This function returns TRUE if successful. Otherwise, it returns a FALSE value.

#### **Return Value for the TerSearchReplace2 function:**

When the SRCH\_SEARCH flag is specified, the return value indicates the absolute character position of the string found. It returns -1 if the string is not found.

When the SRCH\_REPLACE flag is specified, this function returns TRUE if successful. Otherwise, it returns a FALSE value

#### **Example:**

```
pos=0;

while (pos>=0) {
    pos=TerSearchReplace2(hWnd,SearchString, "",
        SRCH_SEARCH+SRCH_WORD + SRCH_CASE, pos, 0);

    if (pos>=0) TerSearchReplace2(hWnd,"", ReplaceString,
        SRCH_REPLACE, pos, pos+strlen(SearchString)-1);
}
```

See Also:  
[TerSearchReplaceU](#)



## TerSearchReplaceU

**Search, replace or retrieve Unicode text.**

BOOL TerSearchReplaceU(HWND, pSearch, pReplace, flags, StartPos, EndPos, BufSize)

HWND hWnd;	// The handle of the window to be accessed.
LPWORD pSearch;	// The text to be searched. A Unicode text string passed as an array of short integers (16 bit). Each array position holds one Unicode character to be searched. The array must end with a NULL terminator character.
LPBYTE pReplace;	// The replacement text. A Unicode text string passed as an array of short integers (16 bit). Each array position holds one Unicode character to be searched. The array must end with a NULL terminator character.
UINT flags;	// mode flags
long StartPos;	// start text position
long far *EndPos;	// pointer to the end text position
long far *BufSize;	// pointer to the size of the retrieved text

**Description:** Please refer to the TerSearchReplace function for the description of flags, StartPos, EndPos and BufSize arguments.

**Return Value:** This function returns TRUE if successful. Otherwise, it returns a FALSE value

**Comment:** A VB application might find easier to use the following variable of these methods available through ActiveX interface:

*BOOL TerSearchReplaceU2(SearchText, ReplaceText, flags, StartPos, EndPos, BufSize)*

This method allows you to pass unicode SearchText and ReplaceText parameters as simple VB strings. The remaining parameters and the result values are the same as described above.

**See Also:**  
[TerSearchReplace](#)



## TerSetSearchString

**Set the string for backward and forward search menu commands.**

BOOL TerSetSearchString(hWnd, SearchFor, CaseSensitive)

HWND hWnd; // Window handle to access

LPBYTE SearchFor; // search string

BOOL CaseSensitive; // Set to TRUE for the case sensitive search.

**Return Value:** The function returns TRUE when successful.



## Track Changes

This chapter includes document modification tracking functions. TE can track modification made by multiple reviewers. You can search for each modified text string. You can accept individual modification, or you can accept all modification at once. The process of acceptance merges the modification to the main document.

In This Chapter
<a href="#">TerAcceptChanges</a>
<a href="#">TerDeleteReviewer</a>
<a href="#">TerEnableTracking</a>
<a href="#">TerFindNextChange</a>
<a href="#">TerGetReviewerInfo</a>
<a href="#">TerRejectChanges</a>



## TerAcceptChanges

**Accept modified text.**

```
int TerAcceptChanges(hWnd, all, msg, repaint)
```

```
HWND hWnd; // Handle of the window to be accessed
```

```
BOOL all; // Set to TRUE to accept all changes. Set to false to  
accept the current change.
```

```
BOOL msg; // Set to true to display confirmation and completion  
messages.
```

```
BOOL repaint; // Repaint the window after this operation
```

**Description:** This function merges the modified text to the document. The 'deleted' text is removed from the document. The 'inserted' text attribute is set to normal.

The tracking mode should be turned off to enable the use of this function.

The ID\_ACCEPT\_CHANGE and ID\_ACCEPT\_ALL\_CHANGES ids can also be used with the TerCommand function or the Command property to accept modified text.

**Return Value:** This function returns the number of text strings accepted. It returns -1 to indicate an error condition.



## TerDeleteReviewer

### Delete a reviewer

```
BOOL TerDeleteReviewer(hWnd, RevId)
```

```
HWND hWnd; // Handle of the window to be accessed
```

```
int RevId; // Reviewer id to delete
```

**Comment:** This function deletes the specified reviewer id from the current document. The reviewer id should be deleted only if the document does not contain any changes associated with the reviewer.

**Return Value:** This function returns TRUE when successful.



## TerEnableTracking

### Enable tracking of document modification.

```
BOOL TerEnableTracking(hWnd, enable, name, UseDefaultClrStyle, InsStyle, InsColor,
```

DelStyle, DelColor)

HWND hWnd;	// The handle of the window to be accessed
BOOL enable;	// Set to TRUE to enable tracking. Set to false to disable tracking.
LPBYTE name;	// The reviewer name. Set to "" to assume currently logged user name.
BOOL UseDefaultClrStyle;	// Use the default value for color and style for the deleted and inserted text. When this parameter is set to TRUE, the following color and style parameters are ignored.
DWORD InsStyle;	<p>// The style to apply to the newly inserted text. Please refer to the <a href="#">SetTerCharStyle</a> function for the list of available character style.</p> <p>This parameter is ignored if the UseDefaultClrStyle parameter is set to TRUE.</p>
COLORREF InsColor;	<p>// The color to apply to the newly inserted text.</p> <p>This parameter is ignored if the UseDefaultClrStyle parameter is set to TRUE.</p>
DWORD DelStyle;	<p>// The style to apply to the deleted text. Please refer to the <a href="#">SetTerCharStyle</a> function for the list of available character style.</p> <p>This parameter is ignored if the UseDefaultClrStyle parameter is set to TRUE.</p>
COLORREF DelColor;	<p>// The color to apply to the deleted text.</p> <p>This parameter is ignored if the UseDefaultClrStyle parameter is set to TRUE.</p>

**Comment:** The ID\_TRACK\_CHANGES id can also be used with the TerCommand function or the Command property to enable/disable tracking of modification.

**Return Value:** This function returns TRUE when successful.



## TerFindNextChange

**Find next or previous modified text.**



BOOL TerFindNextChange(hWnd, forward, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL forward; // Set to TRUE to find the next modified string. Set to false to find the previous modified string.

BOOL repaint; //Repaint the window after this operation

**Comment:** The ID\_NEXT\_CHANGE and ID\_PREV\_CHANGE ids can also be used with the TerCommand function or the Command property to locate modified text.

**Return Value:** This function returns TRUE if the string is located, otherwise it returns FALSE.



## TerGetReviewerInfo

### Retrieve reviewer info

BOOL TerGetReviewerInfo(hWnd, RevId, RevName)

HWND hWnd; // Handle of the window to be accessed

int RevId; // Reviewer id to retrieve information

LPBYTE RevName; // (output) Variable to receive reviewer name

**Return Value:** This function returns TRUE when successful.



## TerRejectChanges

### Reject modified text.

int TerRejectChanges(hWnd, all, msg, repaint)

HWND hWnd; // Handle of the window to be accessed

BOOL all; // Set to TRUE to reject all changes. Set to false to reject the current change.

BOOL msg; // Set to true to display confirmation and completion messages.

BOOL repaint; // Repaint the window after this operation

**Description:** This function annuls the modification to the document. The 'inserted' text is removed from the document. The 'deleted' text is reinstated in the document.

The tracking mode should be turned off to enable the use of this function.

The ID\_REJECT\_CHANGE and ID\_REJECT\_ALL\_CHANGES ids can also be used with the TerCommand function or the Command property to reject modified text.

**Return Value:** This function returns the number of text strings rejected. It returns -1 to indicate an error condition.



## Comments

### In This Chapter

[TerApplyComment](#)

[TerEditComment](#)



## TerApplyComment

### Apply comment to the selected text.

int TerApplyComment(hWnd, ShowDialog, AuthorName, AuthorInitials, CommentText, repaint)

HWND hWnd; // The handle of the window to be accessed

BOOL ShowDialog; // Set to TRUE to show a dialog to accept the author information and the comment text. Set to false to use the information from the remaining parameters passed to this method

LPBYTE AuthorName; // The author name for the comment

LPBYTE AuthorInitials; // Author initials

LPBYTE CommentText; // Comment text. Use the cr/lf sequence to include a paragraph break.

BOOL repaint; // refresh the screen after this operation

**Comment:** Some text must be selected before calling this method. The comment is applied to the selected text. The selected text is highlighted and pointed to the comment text in the right-margin area. The comments are displayed horizontally when the right-margin is sufficiently wide. Otherwise, it is displayed vertically.

**Return Value:** This function returns the comment-id when successful, otherwise it returns a -1 value.



## TerEditComment

**Edit or delete an existing comment.**

BOOL TerEditComment(hWnd, ShowDialog, id, pAuthorName, pAuthorInitials, pCommentText, repaint)

HWND hWnd;	// The handle of the window to be accessed
BOOL ShowDialog;	// Set to TRUE to show a dialog to accept the new author information and the comment text. Set to false to use the information from the remaining parameters passed to this method
int id;	// Comment id to modify
LPBYTE AuthorName;	// The new author name for the comment
LPBYTE AuthorInitials;	// The new author initials
LPBYTE CommentText;	// New Comment text. Use the cr/lf sequence to include a paragraph break. Set to "" to delete the comment.
BOOL repaint;	// refresh the screen after this operation

**Return Value:** This function returns TRUE when successful.



## Menu Command

### In This Chapter

[TerCommand](#)

[TerMenuEnable](#)

[TerMenuSelect](#)

[TerIgnoreCommand](#)



## TerCommand

### Execute a TER menu command.

BOOL TerCommand(hWnd, CommandId)

BOOL TerCommand2(hWnd, CommandId, send)

HWND hWnd; // Handle of the window to be accessed

int CommandId; // Please refer to the 'command' property under the Visual Basic Interface chapter for a list of available command ids.

BOOL send; // Set to TRUE to execute the command immediately, or set to FALSE to 'post' the command for delayed execution.

**Return Value:** This function always returns TRUE.



## TerMenuEnable

### Get menu item 'enable' status

UINT TerMenuEnable(hWnd, MenuId)

BOOL TerMenuEnable2(hWnd, MenuId)

HWND hWnd; // Editor window to access

int MenuId; // menu id can be any of the constants defined in the ter\_cmd.h file.

**Description:** If your program creates a menu outside the editor window, you can use this function to test if a menu item should be enabled or grayed.

Return Value: The TerMenuEnable function returns one of the following constants:

MF\_ENABLED = Enable menu item

MF\_GRAYED = Gray out the menu item

The TerMenuEnable2 function returns TRUE when a menu item is enabled. Otherwise it returns a FALSE value.

Example:

```
#include "ter_dlg.h"

BOOL status;

status = TerMenuEnable2(hWnd,ID_CUT);

// The 'status' variable will be TRUE if a text block is highlighted to be copied to the
// clipboard.
```

**See Also:**  
[TerMenuSelect](#)



## TerMenuSelect

### Get menu item selection status

```
UINT TerMenuSelect(hWnd, MenuId)
BOOL TerMenuSelect2(hWnd, MenuId)
```

```
HWND hWnd;                // Editor window to access

int MenuId;                // menu id can be any of the constants defined in the
                           // ter_cmd.h file.
```

**Description:** If your program creates a menu outside the editor window, you can use this function to test if a menu item should be *checked*.

**Return Value:** The TerMenuSelect function returns one of the following constants:

```
MF_CHECKED = Check the menu item
MF_UNCHECKED = Uncheck the menu item
```

The TerMenuSelect2 function returns a TRUE value if a menu item is to be checked. Otherwise it returns a FALSE value.

Example:

```
#include "ter_dlg.h"

BOOL status;

status = TerMenuSelect2(hWnd,ID_BOLD_ON);

// The 'status' variable will be TRUE if the current character has the bold style.
```

**See Also:**  
[TerMenuEnable](#)



## TerIgnoreCommand

**Ignore the current preprocess command.**

BOOL TerIgnoreCommand(hWnd)

HWND hWnd;                               // The handle of the window to be accessed

**Description:** This function can be used while processing the TER\_PREPROCESS message or the 'Preprocess' event. This function sets a flag which instructs the editor to skip processing the current command.

**Return Value:** This function returns TRUE when successful.



## Screen Drawing

### In This Chapter

[TerEnableRefresh](#)  
[TerGetTextHeight](#)  
[TerRepaint](#)  
[TerRewrap](#)  
[TerScrLineHeight](#)  
[TerSetBorderColor](#)  
[TerSetBorderLineColor](#)  
[TerSetFocus](#)  
[TerSetWinBorder](#)  
[TerSetWrapWidth](#)  
[TerSetZoom](#)  
[TerSetStatusColor](#)



## TerEnableRefresh

**Enable or disable screen refresh.**

BOOL TerEnableRefresh(hWnd,enable)

HWND hWnd;                               // The handle of the window to be accessed

BOOL enable;                             // True to enable the screen refresh.

**Description:** This function is used to enable or disable the screen painting.

**Return Value:** This function returns TRUE if successful.



## TerGetTextHeight

**Return the total text height in twips.**

```
int TerGetTextHeight(hWnd)
```

HWND hWnd;                               // The handle of the window to be accessed.

**Description:** This function returns the total body text height of all pages in the document. The body text height does not include the header/footer text, or the table header text.

**Return Value:** This function returns the text height in twips.

**See Also**

[TerScrLineHeight](#)



## TerRepaint

**Repaint the TER control.**

```
BOOL TerRepaint(hWnd, ClearBackground)
```

HWND hWnd;                               // Handle of the window to be accessed

BOOL ClearBackground;               // TRUE to clear the background before repainting

**Description:** This function repaints every aspect of the TER control.

**Return Value:** The function returns TRUE when successful



## TerRewrap

**Word wrap the entire document on demand.**

```
int TerRewrap(hWnd)
```

HWND hWnd;                               // Window handle to access

**Description:** This function can be used to rewrap the entire document on demand.

**Return Value:** The function returns TRUE when successful.

**See Also:**

[TerRepaginate](#)



## TerScrLineHeight

**Return the line height in screen pixels.**

int TerScrLineHeight(hWnd, line)

HWND hWnd; // The handle of the window to be accessed.

long line; // line number (0 to TotalLines -1) to return the height for.

**Return Value:** This function returns the line height in the screen pixels.

**See Also**

[TerGetTextHeight](#)



## TerSetBorderColor

**Set the color of the border area around the text window.**

BOOL TerSetBorderColor(hWnd, color)

HWND hWnd; // The handle of the window to be accessed.

COLORREF color; // Color of the border area. The border color gets reset when you set the text background color using the [SetTerFields](#) function.

**Return Value:** This function returns TRUE when successful.



## TerSetBorderLineColor

**Set the color of the border line around the page in the page-layout mode.**

BOOL TerSetBorderLineColor(hWnd, color)

HWND hWnd; // The handle of the window to be accessed.

COLORREF color; // Color of the border line.



**Return Value:** This function returns TRUE when successful.



## TerSetFocus

**Set the focus cursor.**

BOOL TerSetFocus(hWnd)

HWND hWnd; // The handle of the window to be accessed

**Description:** Normally, when the editor window is activated, the caret (cursor) shows up in the editor window. However, within certain programming environment, the caret does not appear in the editor window automatically. This function can be used to display the caret in this situation.

**Return Value:** This function returns TRUE if successful.



## TerSetWinBorder

**Set the editor window border.**

BOOL TerSetWinBorder(hWnd, BorderType)

HWND hWnd; // Window handle to access

int BorderType; // Border type: 0= No border, 1= Single line border, 2= Double line sizable border

**Return Value:** This function returns TRUE when successful.



## TerSetWrapWidth

**Set the wrap width.**

BOOL TerSetWrapWidth(hWnd, WidthChars, WidthTwips, repaint)

HWND hWnd; // Window handle to access

int WidthChars;	// This parameter specifies the maximum number of characters allowed in a line to trigger word wrapping.
int WidthTwips;	// This parameter specifies the length of the line in twips for word wrapping. The actual width of the text line is calculated as following:  WidthTwips - Left Margin - Left Paragraph Indentation - Right Margin - Right Paragraph Indentation.
BOOL repaint;	// TRUE to repaint after this operation.

**Description:** This function is available in simple word-wrap mode only (not available when the PrintView or Page Mode is turned on). To specify the wrap width in terms of the number of characters, set the WidthTwips parameter to 0. To specify the wrap width in terms of 'twips', set the WidthChars parameter to 0. To reset to regular word wrapping, set both the WidthChars and WidthTwips parameters to 0.

**Return Value:** This function returns TRUE when successful.



## TerSetZoom

**Set the zoom percentage.**

int TerSetZoom(hWnd, ZoomPercent)

HWND hWnd;	// Window handle to access
------------	----------------------------

int ZoomPercent;	// Specify a value between 10 and 1000, the 100 being the normal display. You can set this argument to - 2 (minus 2) to simply retrieve the current zoom percent with changing it.
------------------	--

**Return Value:** This function returns the previous zoom percent. It returns -1 if an error occurs.



## TerSetStatusColor

**Set the color of the status, ruler and toolbar area around the text window.**

BOOL TerSetStatusColor(hWnd, color, BkColor)

HWND hWnd;	// The handle of the window to be accessed
COLORREF color;	// The foreground color for the status bar text
COLORREF BkColor;	// The background color for the status, ruler and toolbar area.

**Return Value:** This function returns TRUE when successful.



## PDF Output

TE Edit control can export the current document into PDF format using another one of products called WinPDF Converter. WinPDF Converter must be purchased separately.

The pdc32.dll file included with WinPDF Converter must be copied to the same directory which includes the ter32.dll file. This dll provides the access to the functions listed in this chapter.

The TerPdcStart function is used to start a PDF export process. The TerPdcPrintPage function is used to print one of more pages or the currently loaded document. The TerPdcEnd function is used to generate the PDF file. You can load multiple documents into TE within one invocation of the TerPdcStart and TerPdcEnd functions. This allows you to combine more than one document into one PDF file.

### In This Chapter

[TerPdcEnd](#)

[TerPdcPrintPage](#)

[TerPdcStart](#)



## TerPdcEnd

### Terminate the PDF generation process

BOOL TerPdcEnd(PdfId)

int PdfId;	// The PdfId value as returned by the TerPdcStart function.
------------	---

**Description:** This function terminates the PDF generation process and generates the PDF output file.

**Return Value:** This function returns TRUE if successful.

See Also

[TerPdcPrintPage](#)  
[TerPdcStart](#)



## TerPdcPrintPage

**Print document pages to the PDF output file.**

BOOL TerPdcPrintPage(hWnd, PdfId, PageNo)

HWND hWnd;	// The handle of the window to be accessed
int PdfId;	// The PdfId value as returned by the TerPdcStart function.
int PageNo;	// The page number to export to the PDF file. Set the parameter to -1 to export all pages.

**Return Value:** This function returns TRUE if successful.

**See Also**  
[TerPdcStart](#)  
[TerPdcEnd](#)



## TerPdcStart

**Begin the PDF generation process.**

int TerPdcStart(LicKey, OutFile, author, producer, title, subject, keywords, date, ModDate, flags)

int TerPdcStart2(LicKey, OutFile, author, producer, title, subject, keywords, date, ModDate, flags, PermFlags, OwnerPassword, UserPassword)

LPBYTE LicKey;	// Your license key for WinPDF Converter. <i>Your license key for WinPDF Converter is e-mailed to you after your order for WinPDF Converter is processed.</i>
----------------	---

Please pass "" for this parameter when using WinPDF in the evaluation mode.

LPBYTE OutFile;	// The name of the PDF output file.
-----------------	-------------------------------------

LPBYTE author;	// The optional author name for the document. Pass a blank string as default.
----------------	---

LPBYTE producer;	// The optional producer name for the document. Pass a
------------------	--

	blank string as default.						
LPBYTE title;	// The optional title for the document. Pass a blank string as default.						
LPBYTE subject;	// The optional subject string for the document . Pass a blank string as default.						
LPBYTE keywords;	// The optional keywords for the document. Pass a blank string as default.						
LPBYTE date;	// The optional creation date for the document. Pass a blank string as default.						
LPBYTE ModDate;	// The optional modification date for the document. Pass a blank string as default.						
DWORD flags;	// The following flag constants are available:						
	<table border="0"> <tr> <td>PDFFLAG_COMPRESS_TEXT</td> <td>Compress text in the PDF file.</td> </tr> <tr> <td>PDFFLAG_NO_BOOKMARK</td> <td>Do not convert table-of-contents into PDF bookmarks.</td> </tr> <tr> <td>PDFFLAG_EMBED_FONTS</td> <td>Always embed fonts.</td> </tr> </table>	PDFFLAG_COMPRESS_TEXT	Compress text in the PDF file.	PDFFLAG_NO_BOOKMARK	Do not convert table-of-contents into PDF bookmarks.	PDFFLAG_EMBED_FONTS	Always embed fonts.
PDFFLAG_COMPRESS_TEXT	Compress text in the PDF file.						
PDFFLAG_NO_BOOKMARK	Do not convert table-of-contents into PDF bookmarks.						
PDFFLAG_EMBED_FONTS	Always embed fonts.						
DWORD PermFlags;	// Use this flag to specify the permissions granted when the PDF document is being viewed or manipulated without using the owner password. You can use one or more of the following flags using the OR operator:						
	<table border="0"> <tr> <td>PERM_PRINT</td> <td>Allow printing operation</td> </tr> <tr> <td>PERM_COPY</td> <td>Allow copying operation</td> </tr> <tr> <td>PERM_MOD</td> <td>Allow document modification</td> </tr> </table>	PERM_PRINT	Allow printing operation	PERM_COPY	Allow copying operation	PERM_MOD	Allow document modification
PERM_PRINT	Allow printing operation						
PERM_COPY	Allow copying operation						
PERM_MOD	Allow document modification						
LPWORD OwnerPassword;	<p>// Optional document owner password.</p> <p>When either an owner or a user password is specified, the PDF document is written out using Adobe standard encryption mechanism.</p> <p>An owner password in the PDF document requires a PDF editor to prompt the user for the owner password and allow PDF modification only when the supplied owner password matches the encrypted owner password found in the file.</p>						
LPWORD OwnerPassword;	// Optional user password						

When either an owner or a user password is specified, the PDF document is written out using Adobe standard encryption mechanism.

A user password in the PDF document requires a PDF viewer to prompt the user for the user password and allow PDF display only when the supplied user password matches the encrypted user password (or owner password) found in the file.

**Description:** This function begins PDF document creation.

**Return Value:** This function returns the PdfId. The PdfId value is needed for passing to other Pdf APIs. This function returns 0 to indicate an error condition.

#### See Also

[TerPdcEnd](#)

[TerPdcPrintPage](#)



## SpellChecking

SpellTime must be installed to use these functions. The spell32.dll (stw5.dll for Win64) and the SpellTime dictionaries (dict25.\* files) should be placed where the ter32.dll file is located. The folder containing the dictionary files must be available at run-time. One method to ensure this would be to include the folder containing the dictionary files in the environment 'Path' property.

After a TE control instance is created, please call the [TerSetStKey](#) function to set the product key for SpellTime. The spell-time license key can also be set using the SpellTimeKey property of the Toc30.ocx ActiveX control.

You can use the ID\_SPELL and ID\_AUTO\_SPELL commands to use on-demand or as-you-type spell-checking.

You can also use the TerSpellCheck function to invoke on-demand spell-checking.

If you need the spell-time dictionaries to be accessed from a different directory, you can use the [TerSetStDictPath](#) dll function, or the SpellTimeDictPath ActiveX property.

#### In This Chapter

[TerSetStDictPath](#)

[TerSetStCharSet](#)

[TerSetStKey](#)

[TerSpellCheck](#)

[TerSpellCheckerPopped](#)



## TerSetStDictPath

**BOOL** TerSetStDictPat(DictName)

**LPBYTE** DictName;                      (input) New name of the main dictionary excluding the file suffix. Example: dict26

**Description:** The standard dictionary name is dict26. If you are using a dictionary with a different name, use this function to specify the new name. This method can also be used to specify the full path of the dictionary files, example: "c:\temp\dict26".

This function must be called before creating an instance of the TE Edit Control.

**Return:** This function returns TRUE when successful.

**Comment:** The dictionary path can also be specified using the SpellTimeDictPath ActiveX control property:

toc.SpellTimeDictPath = "c:\temp\dict26"



## TerSetStCharSet

**Set the character set for spell-checking.**

**BOOL** TerSetStCharSet(hWnd, NewCharSet)

HWND hWnd;                              // The handle of the window to be accessed.

BYTE NewCharSet;                      // New character set.

BOOL repaint;                          // Repaint the screen after this operation

**Description:** Use this function to set the character set corresponding to the chosen dictionary for SpellTime. This call would be needed only if the chosen SpellTime dictionary uses non-ANSI characters (character set value greater than 2).

**Return Value:** This function returns TRUE when successful.



## TerSetStKey

**Set the license key for SpellTime.**

**bool** TerSetStKey(LicenseKey)

string LicenseKey                      // The license key for SpellTime is e-mailed to you after your order for SpellTime is processed.



# TerSpellCheck

```
toc.SpellTimeKey="xxxxx-yyyyy-zzzzz"
```

10

```
HWND hWnd;           // Window handle to access.
```



**Return Value:** This function returns TRUE if the previous right mouse click invoked the spell-checker pop menu.



## Html Add-on Interface

Html Add-on must be installed to use these functions. The hts27.dll should be placed where the ter32.dll file is located.

After a TE control instance is created, please call the [TerSetHtmlAddOnKey](#) function to set the product key for SpellTime. The Html Add-on license key can also be set using the HtmlAddOnKey property of the Toc30.ocx ActiveX control.

You can use TE Edit control to create and edit the document, and then use HTML Add-on simply to save (or import) the document. You would set the SAVE\_HTML output format to instruct TE to use HTML add-on to save the current contents in the HTML format.

```
TerSetOutputFormat(SAVE_HTML)
```

```
SaveTerFile("myfile.htm"); // or use the Data property of the toc ActiveX control
```

Similarly, to load an html file

```
TerSetFlags4(TRUE,TFLAG4_HTML_INPUT)
```

```
ReadTerFile("myfile.htm"); // or use the Data property of the toc ActiveX control
```

### In This Chapter

[TerSetHtmlAddOnKey](#)



## TerSetHtmlAddOnKey

### Set the license key for Html Add-on.

```
bool TerSetAddOnKey(LicenseKey)
```

```
string LicenseKey           // The license key for Html Add-on is e-mailed to you
                             after your order for Html Add-on is processed.
```

The license key for Html Add-on is e-mailed to you after your order for Html Add-on is processed. You would set the license key for Html Add-on using the TerSetHtmlAddOnKey static function. This should be preferably be done before creating a TE control instance to avoid eval pop-up screens.

```
TerSetHtmlAddOnKey("xxxxx-yyyyy-zzzzz")
```

Replace the 'xxxxx-yyyyy-zzzzz' by your license key for Html Add-on. Please note that your license key for TE Edit Control is not valid for Html Add-on.

**Return Value:** This function returns True if successful.

**Comment:** The license key for Html Add-on can also be set using the HtmlAddOnKey property of the ActiveX control:

```
toc.HtmlAddOnKey="xxxxx-yyyyy-zzzzz"
```



## Hyphenation Support

TE Edit Control can interface with hyphenation library offered by Circle Noetics ([www.circlenoetics.com](http://www.circlenoetics.com)). This third party product needs to be purchased directly from Circle Noetics if you wish to incorporate hyphenation feature into TE Edit Control. Please contact Circle Noetics to check if a particular language is supported by this product.

The DLL called dashes.dll found in this hyphenation library can be accessed by TE using the TerEnableDashes method.

You would call the TerEnableDashes method after creating a TE window to select a language for hyphenation, and to specify the hyphenation granulation level.

```
BOOL TerEnableDashes(hWnd, lang, level, enable)
```

HWND hWnd;	// The handle of the window to be accessed
int lang;	// The language id to enable the hyphenation feature. The available language id values are found in a Dashes library documentation.
int level;	// Hyphenation granulation level. The applicable values for this parameter are available in a Dashes library documentation. A value of 3 generally provides an acceptable granulation level.
BOOL enable;	// Set to TRUE to enable hyphenation, or set to FALSE to disable it.

**Comment:** The dashes.dll file must be copied to the project folder for hyphenation feature to be available.

**Return Value:** This function returns TRUE if successful.



## Miscellaneous

In This Chapter

[TerAnd](#)  
[TerDataVersion](#)  
[TerEnableScrollBar](#)  
[TerEnableSpeedKey](#)  
[TerGetBufferDC](#)  
[TerGetDeviceRes](#)  
[TerOr](#)  
[TerInsertUserField](#)  
[TerGetLastMessage](#)  
[TerResetLastMessage](#)  
[TerGetRefParam](#)  
[TerSetAccelHandle](#)  
[TerSetCustomMessage](#)  
[TerSetCtlColor](#)  
[TerSetParamRect](#)  
[TerSetUserField](#)  
[TerUpdateDynField](#)  
[Callback](#)  
[String Manipulation](#)



## TerAnd

**Return bitwise 'AND' value of two variables.**

```
int TerAnd(var1, var2)
```

```
int var1;                // First variable.
```

```
Int var2;                // Second variable
```

**Description:** This function is useful in the programming environments which does not provide a built-in bitwise AND operator.

**Return Value:** The function returns the bitwise AND value of the input variables.

### See Also:

[TerOr](#)



## TerDataVersion

**Get the version of the data structures used for the TER DLL interface.**

```
int TerDataVersion(void);
```

**Description:** This function returns a version id for the data structures contained in the TER.H file.



## TerEnableScrollBar

**Enable or disable the horizontal or vertical scroll bar.**

BOOL TerEnableScrollBar(hWnd, HScrollBar, enable)

HWND hWnd;                               // The handle of the window to be accessed

BOOL HScrollBar;                        // Set to TRUE to enable/disable the horizontal scroll bar.  
Set to FALSE to enable/disable the vertical scroll bar.

BOOL enable;                            // Set to TRUE to enable the specified scroll bar. Set to  
FALSE to disable it.

**Return Value:** The function returns the TRUE when successful.



## TerEnableSpeedKey

**Enable or disable a speed key.**

BOOL TerEnableSpeedKey(hWnd, CommandId, enable)

HWND hWnd;                               // The handle of the window to be accessed

int CommandId;                           // Id of the command to set the key for. The command  
ids are defined in the TER\_CMD.H file.

int enable;                               // True to enable or False to disable the speed key. To  
disable the entire accelerator table, please refer to the  
TerSetFlags function.

**Return Value:** The function returns the previous status of the speed key



## TerGetBufferDC

**Get the handle of the buffer device context.**

HDC TerGetBufferDC(hWnd)

```
HWND hWnd;           // The handle of the window to be accessed
```

**Return Value:** This function returns the handle of the internal buffer device context. The buffer device context is used internally to create a temporary image of the screen before actually transferring to the window. It returns NULL to indicate an error condition. It can also return NULL when the internal buffer is disabled by using the TerSetFlags function.

**Please call the GetTerFields function to retrieve the actual device context of the editor window.**



## TerGetDeviceRes

**Get the screen or current printer resolution.**

```
BOOL TerGetDeviceRes(ResX, ResY, ForScreen);
```

```
LPINT ResX;           // The variable to receive pixels per inch in the x  
                      // direction.
```

```
LPINT ResY;           // The variable to receive pixels per inch in the y  
                      // direction.
```

```
BOOL ForScreen;       // Set to TRUE to retrieve the resolution for the screen.  
                      // Set to FALSE to retrieve the resolution for the current  
                      // printer.
```

**Return Value:** This function returns TRUE when successful.



## TerOr

**Return bitwise 'OR' value of two variables.**

```
int TerAnd(var1, var2)
```

```
int var1;             // First variable.
```

```
Int var2;             // Second variable
```

**Description:** This function is useful in the programming environments which does not provide a built-in bitwise OR operator.

**Return Value:** The function returns the bitwise OR value of the input variables.

**See Also:**  
[TerAnd](#)

**See Also**    
[TerSetUserField](#)

## TerInsertUserField

**Insert a dynamic user field at the current cursor position.**

```
int TerInsertUserField(hWnd, repaint)
```

HWND hWnd;                               // The handle of the window to be accessed

BOOL repaint;                            //Repaint the window after this operation

**Description:** This function inserts a dynamic user field. The editor fires an event called SetUserField when it encounters a user-field in the document as the page is being rendered. Your application can specify the content of the user field by using the TerSetUserField function within this event.

The text length of the user field should not exceed one text line. When the user-field is placed in a header/footer area, the text length of the field should not change from one call to another.

**Return Value:** This function returns TRUE when successful.



## TerGetLastMessage

**Get the last message.**

```
int TerGetLastMessage(TerMessage, DebugMessage);
```

LPBYTE TerMessage;                    // Returns the default user message text in English

LPBYTE DebugMsg;                      // Returns any debug message associated with the last message. The debug message need not be displayed to the user.

**Return Value:** This function returns the last message generated by the editor. This value is valid only if saving of the messages is enabled by setting the TFLAG\_RETURN\_MSG\_ID flag. This flag is set using the TerSetFlags function.

The message string constants (MSG\_) are defined in the TER.H file. The description for the message ids can be found in the TER\_MSG.H file.

**See Also:**  
[TerResetLastMessage](#)



## TerResetLastMessage

**Reset the last editor message.**

BOOL TerResetLastMessage()

**Description:** This function can be called before calling any other TER function to reset the last error message.

**Return Value:** The function returns TRUE when successful.

### See Also:

[TerGetLastMessage](#)



## TerGetRefParam

**Get the value of a 'pass-by-reference' parameter to function call within a script.**

int TerGetRefParam(hWnd, ParamNum)

int TerGetRefParamStr(hWnd, ParamNum)

ActiveX method definition:

string TerGetRefParamStr(ParamNum)

HWND hWnd;                                      // The handle of the window to be accessed

int ParamNum;                                      // A zero-based parameter number to the previous call for a 'pass-by-reference' function. The numeric and string parameters are numbered separately. For example, the ParamNum 0 would indicate the first numeric and string parameter.

**Return Value:** This function returns the numeric value of the specified parameter for the previous function call.



## TerSetAccelHandle

**Set a custom accelerator table handle for the editor to use.**

BOOL TerSetAccelHandle(hWnd, hAcc)

```
HWND hWnd;           // The handle of the window to be accessed.

HANDLE hAcc;          // Handle of the new accelerator table. Set to NULL to
                      // use the original accelerator table.
```

**Description:** If you are building your own accelerator table, please review the ter.rc file for the items included in the editor's original accelerator table.

**Return Value:** This function returns a TRUE value when successful.



## TerSetCustomMessage

**Set custom message text for a message id.**

```
BOOL TerSetCustomMessage(id, message);
```

```
int id;                // The message id to set the custom text. The message
                      // id constants (MSG_) are defined in the ter.h, ter.bas or
                      // ter.pas files. The original English version of the message
                      // text is available in the ter_msg.h file.
```

```
LPBYTE message;        // New message text for the message id.
```

**Return Value:** This function returns True when successful.



## TerSetCtlColor

**Set the background color for the control.**

```
BOOL TerSetCtlColor(hWnd,color,repaint)
```

```
HWND hWnd;             // The handle of the window to be accessed
```

```
COLORREF color;         // new background color
```

```
BOOL repaint;           // TRUE to repaint the screen after this operation.
```

**Return Value:** This function returns TRUE when successful.





## TerSetParamRect

**Set the rectangle parameter for a subsequent function call.**

BOOL TerSetParamRect(hWnd, left, right, top, bottom)

HWND hWnd;	// Handle of the window to be accessed
int left;	// The left coordinate value for the rectangle.
int right;	// The right coordinate value for the rectangle.
int top;	// The top coordinate value for the rectangle.
int bottom;	// The top coordinate value for the rectangle.

**Comment:** Certain functions such TerPrintPreview require you to pass a pointer to the rectangle. If your programming language does not allow passing a rectangle pointer, then you can use this function to set the rectangle before hand. Then call the intended function using the null value for the rectangle parameter. The function would then use the value that you passed using the TerSetParamRect function.

**Return Value:** This function returns TRUE when successful, otherwise it returns FALSE.

See Also

[TerInsertUserField](#)



## TerSetUserField

**Set the text for the user field at the current cursor position.**

BOOL TerSetUserField(hWnd, text)

HWND hWnd;	// The handle of the window to be accessed
LPBYTE text;	// The text for the user field.

**Description:** The editor fires an event called SetUserField when it encounters a user-field in the document as the page is being rendered. Your application can specify the content of the user field by using the TerSetUserField function within this event.

The text length of the user field should not exceed one text line. When the user-field is placed in a header/footer area, the text length of the field should not change from one call to another.

**Return Value:** This function returns TRUE when successful.



## TerUpdateDynField

**Update the dynamic fields such as page number, page-count.**

BOOL TerUpdateDynField(hWnd, repaint)

HWND hWnd; // The handle of the window to be accessed

BOOL repaint; // Set to TRUE to refresh the screen after the operation.

**Return Value:** The function returns the previous status of the speed key



## Callback

### In This Chapter

[TerRegisterMsgCallback](#)

[TerRegisterPrintCallback](#)



## TerRegisterMsgCallback

**Register a message callback process.**

BOOL TerRegisterMsgCallback(hWnd, MsgCallback)

HWND hWnd; // The handle of the text window to be accessed

MSG\_CALLBACK  
MsgCallback; // The callback process in your application to receive the editor messages.

**Description:** Your application can use this call to setup a message callback function. When such a function is registered, the editor messages are routed to this function.

**Return Value:** This function returns TRUE when successful.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.



## TerRegisterPrintCallback

### Register a callback for the TerMergePrint function.

BOOL TerRegisterPrintCallback(PrintCallback)

PRINT\_CALLBACK                      // The callback process in your application to receive the  
PrintCallback;                      TerMergePrint message. Set to NULL to reset the  
callback procedure.

**Description:** This function can be used to register a callback procedure before calling the TerMergePrint function. When such a callback procedure is set, the TerMergePrint function calls this procedure immediately after creating a hidden TER window and after reading the specified file or buffer. Your callback procedure should have the following syntax:

int PrintCallback(HWND hParentWnd, HWND hTerWnd)

The TerMergePrint function sets the 'hParentWnd' argument to the parent window handle as specified in the StrPrint structure. The 'hTerWnd' argument is set to the window handle of the hidden TER window. The return value from the callback function is not used by the TerMergePrint function.

**Return Value:** This function returns TRUE when successful.

**Comments:** This function is not available as an ActiveX control method. It must be used as a DLL function.

### Example:

Normally, the TerMergePrint function can not print the HTML file. But this example uses the print callback function and HTML Add-on to print an HTML file.

```
TerRegisterPrintCallback(PrintCallback); // set the callback
```

```
struct StrPrint print;  
print.rect=... // set the StrPrint structure  
print.OnePage=...  
....  
....  
print.InputType='F';  
lstrcpy(print.file,""); // do not specify the HTML file  
here  
TerMergePrint(&print);
```

```
PrintCallback(HWND hParentWnd, HWND hTerWnd)  
{  
    HtsInitialize(hTerWnd,hParentWnd);  
    HtsRead(hTerWnd,HTML_FILE,"test.htm",NULL,0,NULL);  
}
```

**See Also:**  
[TerMergePrint](#)



## String Manipulation

### In This Chapter

[TerLparam2String](#)  
[TerMakeMbcs](#)  
[TerMbcsChar](#)  
[TerMbcsLen](#)  
[TerSplitMbcs](#)  
[TerString2Lparam](#)



### TerLparam2String

**Copy text from IParam pointer to the given string.**

BOOL TerLparam2String(IParam,text)

HWND hWnd; // Handle of the window to be accessed

long IParam; // The 'IParam; value to copy text from

LPBYTE text; // The string to copy data to

**Description:** This function can be used within a message handler to retrieve text data from a IParam value.

**Return value:** This function returns TRUE when successful.

### See Also:

[TerString2Lparam](#)



### TerMakeMbcs

**Build an MBCS string from lead and tail bytes.**

int TerMakeMbcs(string, tail, lead,length)

HWND hWnd; // Handle of the window to be accessed

LPBYTE string; // The location to retrieve the mbcs string.

LPBYTE tail; // The pointer to the tail bytes of the mbcs string.

// The pointer to the lead bytes of the mbcs string. This

LPBYTE lead;                      string should have the same number of bytes as the 'tail' string. For the single-byte characters, the 'lead' byte should be set to NULL.

int length;                      // Length of the 'tail' and 'lead' strings.

**Description:** This function can be used to combine the lead and tail bytes of an mbcs (Multi-byte Character Set) string to return the combined mbcs string.

Return Value: This function returns the byte length of the mbcs string.

Example:

```
dim Mbcstext as string*2000 ' location to retrieve the mbcs string
dim tail as string*MAX_WIDTH
dim lead as string*MAX_WIDTH
dim StringLen as integer
dim LineLen as Integer
tail=space$(MAX_WIDTH)
lead=space$(MAX_WIDTH)
LineLen=TerGetLineEx(hWnd,LineNo,tail,lead,NULL) ' get the lead and tail bytes for a
text line
Mbcstext=space$(MAX_WIDTH) ' allocate space for the parameter
StringLen = TerMakeMbcstext(Mbcstext,tail,lead,length(tail)) ' build an mbcs string
```

---

**See Also:**

[TerSplitMbcstext](#)  
[TerMbcstextLen](#)



## TerMbcstextChar

**Retrieve a character from an MBCS string.**

WORD TerMbcstextChar(string, pos)

LPBYTE string;                      // The pointer to the mbcs string.

long pos;                      // The position (zero based) of the character to extract.

**Return Value:** If the given string is a single-byte string, then this function returns the byte value at the specified position. If the given string is a multi-byte string, then this function returns the MBCS character at the given character position. For example, if the 'pos' parameter was set to 2, then this function will return the 3rd MBCS character from the string..

**See Also:**  
[TerMakeMbc](#)  
[TerSplitMbc](#)



## TerMbcLen

**Get the length of an MBCS string.**

```
long TerMbcLen(string)
```

LPBYTE string;                      // The pointer to the mbcs string.

**Return Value:** If the given string is a single-byte string, then this function returns the byte length of the string. If the given string is a multi-byte string, then this function returns number of characters in the strings. For example, consider a strings containing 2 double-byte characters and 2 single-byte characters. Then this function will return 4 (whereas the actual byte length of the string would be 6).

**See Also:**  
[TerMakeMbc](#)  
[TerSplitMbc](#)



## TerSplitMbc

**Split an MBCS string into lead and tail bytes.**

```
long TerSplitMbc(string, tail, lead)
```

LPBYTE string;                      // The pointer to the mbcs string.

LPBYTE tail;                        // The location to receive the tail bytes of the mbcs string.

LPBYTE lead;                        // The location to the receive the lead bytes of the mbcs string.

**Description:** This function can be used to split an mbcs (Multi-byte Character Set) into lead and tail bytes.

**Return Value:** This function returns the byte length of the lead or tail string. Both lead and tail byte strings have the same length. The single-byte characters within the mbcs strings are assigned NULL for the 'lead' byte.

**Example:**

```
dim tail as string*MAX_WIDTH  
dim lead as string*MAX_WIDTH
```

```

        dim len as integer

        tail=space$(MAX_WIDTH)      ' allocate space for the
output                                strings.

        lead=space$(MAX_WIDTH)

len = TerMakeMbcs(MbcsText,tail,lead)

```

---

#### See Also

[TerMakeMbcs](#)

[TerMbcsLen](#)

#### See Also

[TerLparam2String](#)



### TerString2Lparam

**Copy text into IParam pointer from the given string.**

BOOL TerString2Lparam(text, IParam)

LPBYTE text;                               // The string to copy data from

long IParam;                                // The IParam; value to copy text to

**Description:** This function can be used within a message handler to retrieve text data from a IParam value.

**Return value:** This function returns TRUE when successful.



## ActiveX Control

Your application can use the TER editor either as an OCX or a DLL. You can also drop the toc30.ocx ActiveX control in your application and then use a mix of ActiveX properties, events, methods and direct DLL function calls.

#### In This Chapter

[Using the Editor as an OCX](#)



## Using the Editor as an OCX

To use the editor as an OCX, copy the toc30.ocx, ter32.dll and txml2.dll files to your application's directory. Now register the toc30.ocx control and drop it into your application form. The ter32.dll file is indirectly used by the toc30.ocx file. However, the ter32.dll or txml2.dll files do not need registering.

The data in the edit control can be manipulated by using the properties, methods and the DLL API functions. The 'properties' can be used to manipulate the basic text attributes, whereas, the API functions can be used to manipulate complex attributes.



### Properties:

This control includes a collection of design-time and run-time properties.

#### Design-time properties:

<b>Word Wrap:</b>	Turn on word wrap.
<b>Print View:</b>	Edit document in the Print View mode. In this mode the lines are wrapped as they would be wrapped when printed to the selected printer (see 'Editing Mode' chapter).
<b>Page Mode:</b>	Edit document one page at a time. This mode is useful when editing the documents containing multiple columns.
<b>FittedView:</b>	Special case of the page mode in which the text wraps to the window width and the soft page breaks are not displayed.
<b>Vertical Scroll:</b>	Enable the vertical scroll bar.
<b>Horizontal Scroll:</b>	Enable the horizontal scroll bar.
<b>Show Status Bar:</b>	Show status bar indicating the cursor position.
<b>Show Ruler:</b>	Show the ruler with tab stops and indentation indicators.
<b>Show Toolbar:</b>	Enables tool bar.
<b>Border Margin:</b>	Reserves a think blank area around the text box.
<b>Read Only:</b>	The editor displays the text, but modifications are not allowed.



<b>InAccess</b>	This property should be set to TRUE when using the control within an Access application.
<b>TerKey</b>	This property provides an alternate method of setting your license key.
<b>Output Rtf:</b>	The output is saved in the RTF format. When this option is turned off, the output is saved in the same format as the input buffer (also see SaveFormat property).

## Run-time Properties:

### Command:

Description: This property is used to invoke the menu commands. The menu is not accessible when the TER editor is used as a control. This property allows you to access the menu commands indirectly.

Usage:

control.Command=Command\_id

Example:

TOC1.command = ID\_PASTE

The command id can be one of the following:

ID_ACCEPT_ALL_CHANGES	Accept all modifications
ID_ACCEPT_CHANGE	Accept current modification
ID_AUTO_SPELL	Invoke automatic spell checking. (SpellTime required for this feature)
ID_BACK_TAB	Enter a reverse tab
ID_BK_COLOR	Set background color
ID_BKND_PICT	Set background picture
ID_BLOCK_COPY	Copy a highlighted block
ID_BLOCK_MOVE	Move a highlighted block
ID_BORDER_MARGIN	Toggle border margin around the text box.
ID_BOLD_ON	Set bold on
ID_BOX_ON	Enable boxed option
ID_BULLET	Enable bullet option

ID_CAPS_ON	All capital letters
ID_CENTER	Center the paragraph
ID_CHAR_NORMAL	Reset the character styles
ID_CHAR_SCALEX	Expand or shrink characters horizontally
ID_CHAR_SPACE	Character spacing
ID_CHAR_STYLE	Character style
ID_COL_BREAK	Insert a column break
ID_COLOR	Choose colors
ID_CREATE_FIRST_FTR	Create the first page footer
ID_CREATE_FIRST_HDR	Create the first page header
ID_CREATE_LIST	Create a list table item
ID_CREATE_LIST_OR	Create a list override table item
ID_COPY	Copy text to clipboard
ID_CTRL_DOWN	Position at the first column of the next line.
ID_CTRL_TAB	Enter a tab character within a table
ID_CTRL_UP	Position at the first column of the previous line.
ID_CUT	Cut text to clipboard
ID_DEL	Delete the current character
ID_DEL_LINE	Delete the current line
ID_DEL_NEXT_WORD	Delete next word
ID_DEL_PREV_WORD	Delete the previous word.
ID_DELETE_FIRST_FTR	Delete the first page footer
ID_DELETE_FIRST_HDR	Delete the first page header
ID_DOC_RTL	Set the default right-to-left property for the document

ID_DOUBLE_SPACE	Double space the paragraph lines
ID_DOWN	Arrow down
ID_EDIT_DOB	Edit drawing object
ID_EDIT_ENOTE	Toggle editing of endnote
ID_EDIT_FNOTE	Toggle editing of footnote
ID_EDIT_HDR_FTR	Edit header/footers
ID_EDIT_INPUT_FIELD	Edit an input field
ID_EDIT_LIST	Edit the list table items
ID_EDIT_LIST_OR	Edit the list override table items
ID_EDIT_LIST_LEVEL	Edit list level properties
ID_EDIT_OLE	Edit OLE object
ID_EDIT_PICT	Edit the picture size
ID_EDIT_STYLE	Edit the style
ID_EMBED_PICT	Insert embedded picture
ID_FILE_BEGIN	Position at the beginning of the file
ID_FILE_END	Position at the end of the file
ID_FONTS	Choose fonts and point sizes
ID_FRAME_ROTATE_TEXT	Rotate the frame text
ID_FRAME_YBASE	Vertical base position
ID_HANGING_INDENT	Create or increment hanging indentation
ID_TER_HELP	Show the help window
ID_HIDDEN_ON	Set the hidden attribute on
ID_HIDE_CHANGES	Suppress the track-change display effects for text insertion and deletion.
ID_HIGHLIGHT_LINE	Highlight the line block

ID_HIGHLIGHT_TEXT	Highlight the selected text.
ID_HLINK_ON	Set the hyperlink style
ID_INLINE_IME	Inline Ime
ID_INS_AFT	Insert a line after the current line
ID_INS_BEFO	Insert a line before the current line
ID_INSERT	Toggle the insert mode
ID_INSERT_BOOKMARK	Insert, delete or position at a bookmark
ID_INSERT_CHECKBOX	Insert a checkbox field
ID_INSERT_COMBOBOX	Insert a combo-box field.
ID_INSERT_DATA_FIELD	Insert a data field name and text
ID_INSERT_DATE_TIME	Insert date/time field.
ID_INSERT_DRAW_OBJECT	Insert a drawing object
ID_INSERT_ENOTE	Insert an endnote
ID_INSERT_FNOTE	Insert a footnote
ID_INSERT_HLINK	Insert hyperlink
ID_INSERT_HYPH	Insert an optional hyphen
ID_INSERT_INPUT_FIELD	Insert an input field
ID_INSERT_NBDASH	Insert a non-breaking dash character
ID_INSERT_NBSPACE	Insert a non-breaking spacing character
ID_INSERT_OBJECT	Insert an OLE object
ID_INSERT_PAGE_COUNT	Insert the total page count string
ID_INSERT_PAGE_NUMBER	Insert page number string
ID_INSERT_PARA_FRAME	Insert a text frame
ID_INSERT_TOC	Insert table of contents

ID_ITALIC_ON	Set italic on
ID_JOIN_LINE	Append the next line to the current line
ID_JUMP	Jump to a line number
ID_JUSTIFY	Justify paragraph on both margins
ID_LEFT	Arrow left
ID_LEFT_JUSTIFY	Left align the paragraph
ID_LEFT_INDENT	Create or increment left indentation
ID_LEFT_INDENT_DEC	Decrement left indentation.
ID_LINE_BEGIN	Position at the beginning of the line
ID_LINE_END	Position at the end of the line
ID_LINK_PICT	Insert link picture
ID_NEW	Begin a new file
ID_NEXT_CHANGE	Position at the next modified text string
ID_NEXT_WORD	Position at the next word
ID_OPEN	Open an existing file
ID_PAGE_BREAK	Insert a page break
ID_PAGE_BREAK_BEFORE	Insert page break before the paragraph
ID_PAGE_OPTIONS	Set the page options
ID_PAINT_FORMAT	Paint the current formatting to the text selection
ID_PARA_BK_COLOR	Assign background color to paragraph
ID_PARA_BORDER	Create paragraph borders and shading
ID_PARA_KEEP	Paragraph keep together
ID_PARA_KEEP_NEXT	Paragraph keep with next
ID_PARA_LIST	Assign list numbering to the paragraph

ID_PARA_NORMAL	Reset the paragraph attributes
ID_PARA_NBR	Assign numbering to the paragraph
ID_PARA_RTL	Set the right-to-left property for the paragraph
ID_PARA_SPACING	Assign spacing to the paragraph
ID_PARA_STYLE	Assign style to the paragraph
ID_PASTE	Paste text from the clipboard
ID_PASTE_SPEC	Paste special clipboard formats
ID_PASTE_TEXT	Paste in the plain text format
ID_PGDN	Page down
ID_PGUP	Page up
ID_PICT_FROM_FILE	Import a bitmap from a disk file
ID_PREV_CHANGE	Position at the previous modified text string
ID_PREV_WORD	Position at the previous word
ID_PRINT	Print text
ID_PRINT_OPTIONS	Set the print parameters
ID_PRINT_PREVIEW	Print preview
ID_PROTECT_FORM	Toggle 'form protection' mode. This mode allows the user to input data into the input fields.
ID_PROTECT_ON	Set character protection on
ID_PROTECTION_LOCK	Toggle the protection lock for the document
ID_QUIT	Quit the editing session
ID_REDO	Reverse the previous undo operation
ID_REJECT_ALL_CHANGES	Reject all changes
ID_REJECT_CHANGE	Reject current change
ID_REPAGINATE	Repaginate now

ID_REPLACE	Replace text
ID_RETURN	Process the <Enter> key
ID_RIGHT	Arrow right
ID_RIGHT_INDENT	Create or increment right indentation
ID_RIGHT_JUSTIFY	Right justify the paragraph
ID_RULER	Toggle ruler display
ID_SAVE	Save the current file
ID_SAVEAS	Save data to another file name
ID_SCAPS_ON	Small capital letters
ID_SEARCH	Search a text string
ID_SEARCH_BACK	Search for the previous text string
ID_SEARCH_FOR	Search for the next text string
ID_SECT_BREAK	Insert a section break
ID_SECT_OPTIONS	Set the section parameters
ID_SECT_RTL	Set the default right-to-left property for the section
ID_SELECT_ALL	Select the entire document
ID_SHOW_FIELD_NAMES	Show the field names
ID_SHOW_HIDDEN	Show the hidden characters
ID_SHOW_HYPERLINK_CURSOR	Toggle the display of hyperlink cursor
ID_SHOW_PAGE_BORDER	Show page borders in PageMode
ID_SHOW_PAGE_LAYOUT	Show page layout in PageMode
ID_SHOW_PARA_MARK	Toggle the paragraph marker character display
ID_SNAP_TO_GRID	Snap to grid
ID_SPELL	Invoke a spell checking session (SpellTime required for

this feature)

ID_SPLIT_LINE	Split the current line at the current position
ID_STATUS_RIBBON	Toggle the status ribbon
ID_STRIKE_ON	Set strike style on
ID_SUBSCR_ON	Set subscript on
ID_SUPSCR_ON	Set superscript on
ID_TAB	Insert a tab
ID_TAB_CLEAR	Clear a tab stop position for a paragraph
ID_TAB_CLEAR_ALL	Clear all tab stop positions for a paragraph
ID_TAB_SET	Set tab positions for a paragraph
ID_TABLE_CELL_BORDER	Edit table cell border width
ID_TABLE_CELL_BORDER_COLOR	Edit table cell border color
ID_TABLE_CELL_COLOR	Set table cell background color
ID_TABLE_CELL_SHADE	Set table cell shading
ID_TABLE_CELL_VALIGN	Set the vertical alignment for the text inside a table cell.
ID_TABLE_HDR_ROW	Toggle the 'header' attribute of the current table row
ID_TABLE_INSERT	Insert new table
ID_TABLE_CELL_WIDTH	Modify table cell width
ID_TABLE_DEL_CELLS	Delete table cells
ID_TABLE_INSERT_COL	Insert new table column
ID_TABLE_INSERT_ROW	Insert new table row
ID_TABLE_MERGE_CELLS	Merge table cells
ID_TABLE_ROW_HEIGHT	Position row height
ID_TABLE_ROW_KEEP	Keep the table row together in one page.



ID_TABLE_ROW_POS	Position table rows
ID_TABLE_ROW_RTL	Set the right-to-left property for the table
ID_TABLE_SEL_COL	select the current table column
ID_TABLE_SHOW_GRID	Show table grid lines
ID_TABLE_SPLIT_CELL	Split current table cell horizontally.
ID_TABLE_SPLIT_CELL_VERT	Split the current table cell vertically.
ID_TABLE_CELL_VTEXT	Set text rotation for a the table cell text.
ID_TOOL_BAR	Toggle the toolbar display
ID_TRACK_CHANGES	Toggle tracking of text modification
ID_ULINE_ON	Set underline on
ID_ULINED_ON	Set double underline attribute on
ID_UNDO	Undo previous edit
ID_UP	Arrow up
ID_USER1 to ID_USER9	Unused command ids to be used with new toolbar icons. It is generally used to implement functionality not provided by the existing command ids. For example, you can use ID_USER1 to implement PDF output icon. You would intercept ID_USER1 using the PreProcess event and execute your code there. Then call the TerIgnoreCommand method to tell the editor to ignore this command.
ID_VIEW_HDR_FTR	Show page header/footers
ID_VRULER	Toggle the display of the vertical ruler.
ID_WATERMARK	Let user select a watermark picture.
ID_WIDOW_ORPHAN	Enable widow/orphan control for a paragraph.
ID_ZOOM	Enable Zoom

#### **Data:**

Description: Use this property to assign or retrieve text from the control. Usage:  
control.Data = string or

string = control.Data

Example:

control.Data = "This is a test data"

See Also: SaveFormat

### **hWnd:**

Description: This property is used to access the window handle of the control. The window handle is generally the first required argument when calling a DLL function.

Example: TerSetReadOnly(control.hWnd,TRUE)

### **ParaIndentHanging:**

Description: This property is used to increase (True) or decrease (False) the left hanging indentation for a paragraph. Each increment or decrement is equal to a 1/4 of an inch. The hanging indent is not applied to the first line of a paragraph.

Usage: control. ParaIndentHanging=True|False

### **ParaIndentLeft:**

Description: This property is used to increase (True) or decrease (False) the left indentation for a paragraph. Each increment or decrement is equal to a 1/4 of an inch.

Usage: control.ParaIndentLeft=True|False

### **ParaIndentRight:**

Description: This property is used to increase (True) or decrease (False) the right indentation for a paragraph. Each increment or decrement is equal to a 1/4 of an inch.

Usage: control.ParaIndentRight=True|False

### **ParaReset:**

Description: This function is used to reset all attributes for the current paragraph.

Usage: control.ParaReset=True

### **ParaStyle:**

Description: This function is used to set the centering, right justification, and double space attributes for a paragraph:

LEFT:	Paragraph left justified
CENTER:	Paragraph centered
RIGHT_JUSTIFY:	Paragraph right justified
JUSTIFY:	Paragraph justified on both margins
DOUBLE_SPACE:	Paragraph lines double spaced.
PARA_KEEP:	Paragraph keep together
PARA_KEEP_NEXT:	Paragraph keep with next

A justification flag can be specified along with the double space attribute by using the 'or' operator.

Usage: control.ParaStyle = style

Example:

`control.ParaStyle = CENTER` or `DOUBLE_SPACE`.

#### **ReadFile:**

Description: Use this property to load a specified file in the control.

Usage: `control.ReadFile = filename`

Example: `control.ReadFile = "test.doc"`

See Also: `SaveFile`

#### **Repaint:**

Description: When this property is set to `False`, the screen refresh is suspended for the subsequent calls to other properties.

Usage: `control.Repaint=True | False`

#### **ResetCharStyle:**

Description: This property is used to reset character styles such as bold, underline, italic, etc.. More than one style can be reset by using the 'or' operator.

If a text block is highlighted prior to setting this property, the entire text block is subject to this command. Otherwise, the specified style is reset for new text input.

Usage: `control.ResetCharStyle=constant`

The following constants are available:

`BOLD:`     `Bold`  
`ULINE:`   `Underline`  
`ULINED:` `Double underline`  
`ITALIC:` `Italic`  
`STRIKE:` `Strikethrough`  
`SUPSCR:` `Superscript`  
`SUBSCR:` `Subscript`  
`HIDDEN:` `Hidden text`  
`CAPS:`    `All caps`  
`SCAPS:`   `Small caps`

Example:

`control.ResetCharStyle=BOLD` or `ULINE`.

See Also: `SetCharStyle`

#### **SaveFile:**

Description: Use this property to save the contents of the control to a disk file. Any previous content of the file is destroyed.

Usage: `control.SaveFile = filename`

Example:

`control.SaveFile = "test.doc"`

See Also: `ReadFile`

### SaveFormat:

Description: This property controls the format of the data when it is retrieved from the control. This property is typically set before invoking the 'Data' property.

Usage: `control.SaveFormat=format`

The 'format' can be one of the following:

<code>SAVE_TER:</code>	Save in the control's native format.
<code>SAVE_TEXT:</code>	Save in the text format. The formatting information is lost in the text format.
<code>SAVE_TEXT_LINES:</code>	Save in the text format with line breaks. The formatting information is lost in this format.
<code>SAVE_RTF:</code>	Save in the Rich Text Format.
<code>SAVE_DOCX:</code>	Save in the DOCX format.
<code>SAVE_UTEXT:</code>	Save in the Unicode text format.
<code>SAVE_DEFAULT:</code>	Save in the format of the input data.

Example:

```
control.SaveFormat=SAVE_TER
```

```
string=control.Data
```

See Also: Data

### SetCharStyle:

Description: This property is used to set character styles such as bold, underline, italic, etc.. More than one style can be specified by using the 'or' operator.

If a text block is highlighted prior to setting this property, the entire text block is assigned the character style. Otherwise, the specified style is assigned to new text input.

Usage: `control.SetCharStyle=constant`

Please refer to the `ResetCharStyle` property for the available style constants.

Example: `control.SetCharStyle=BOLD or ULINE`.

See Also: `ResetCharStyle`



## Events

### Standard events:

The control supports these standard events: Click, Double click, Got Focus, and Lost Focus, KeyDown, KeyPress, KeyUp. The KeyDown, KeyPress and KeyUp event take one integer parameter which passes the value of the key code.

**Action:**

This event is fired after completing a user action. Please refer to the ['Message Communication'](#) chapter for a complete description of this event.

**Hypertext event:**

This event is fired when the user double clicks over a hypertext phrase. This event has two string type arguments. The first argument contains the content of the hypertext phrase (the double underlined text). The second argument contains the content of the hypertext code (the hidden text before the underlined text).

**TextDrag**

This event is fired before text drag/drop is begun from an external application, or from within the editor.

**TextBeforeDrop**

This event is fired immediately before the text is dropped from an external application, or from within the editor.

**TextDrop**

This event is fired when text is dropped from an external application, or from within the editor.

**PreProcess:**

This event is fired before completing a user action. Please refer to the ['Message Communication'](#) chapter for a complete description of this event.

**UpdateToolbar:**

This event notifies your application to update any external toolbar the application might be using.

**UpdateStatusbar:**

This event notifies your application to update any external status bar the application might be using.

**PageSizeChanging:**

This event is fired before TE adjusts the page size. Please refer to the ['Message Communication'](#) chapter for a complete description of this event.

**SetUserField:**

This event is fired when the editor encounters a user-field as the text is being rendered. This event uses two arguments. The first argument specifies the page number (zero based) where the user-field was encountered. The second argument provides the absolute text position of the field. The absolute text position can be converted into the line/column position using the TerAbsToRowCol method.

Typically your application would set the text value of the current user field when this event is encountered. The text for the field is set using the TerSetUserField method.



## Methods

All most all the functions that are exported by ter32.dll are also exposed by the ActiveX control. Please refer to the '[Application Interface Functions](#)' chapter for the list of DLL functions and ActiveX methods. There are a few DLL functions which are not exposed as ActiveX methods. We have noted them in the comment section in function description.

In the programming environments (such as VisualBasic and Delphi) where you can use DLL functions as easily as the ActiveX methods, we would recommend that you use the DLL functions. The DLL functions are slightly faster than ActiveX methods and they can accept complex data types. However, you can use ActiveX methods if for some reason if you prefer ActiveX methods over DLL functions. You can also use a mix of methods and functions.

In the VisualBasic environment, the DLL functions are declared in ter.bas file. Once you include ter.bas file in your project you can call any API as simple function calls. Most API need the control window handle as the first argument. This would be TOC1.hWnd for a control named TOC1.

### An example of calling an API function:

```
Call SetTerCharStyle(TOC1.hWnd, BOLD, TRUE, TRUE)
```

String output from an API function: Certain API such as GetFontInfo return a string output. A string variable in VB must be assigned spaces before passing an output string variable to an API function:

```
Dim typeface as string
```

```
Dim pointsize as long
```

```
Dim style as long
```

```
Typeface = space$(50) ' allocate space before calling the API
```

```
Call GetFontInfo(TOC1.hWnd, typeface, pointsize, style)
```

### An example of calling an ActiveX method:

The method name in the toc30.ocx will be the same as the API function names. The parameter and return values are also the same, with one exception. The methods do not need to be passed the first parameter (window handle). This is because the control already knows the window handle.

```
Dim typeface as string
```

```
Dim pointsize as long
```

```
Dim style as long
```

```
Call TOC1.GetFontInfo(typeface, pointsize, style)
```

Please note that in the TOC1.GetFontInfo method, we are omitting the window handle (the first parameter). All other parameters are the same.



## Visual Basic Interface

Your Visual Basic application can use the TER editor either as an OCX or a DLL.

### In This Chapter

[Using the Editor as an OCX](#)

[Using the Editor as a DLL](#)



## Using the Editor as an OCX

To use the editor as an OCX, copy the toc30.ocx, ter32.dll and txml2.dll files to your application's directory. Now include the TER.BAS file in your project and select the 'Ter Edit OCX' control from the Custom Control menu option. The editor icon will be displayed in the Visual Basic toolbar. You can create an editor control using this icon from the toolbar.

For an example of the OCX interface, refer to the **DMO\_OCX** (decompress from vbdemo.zip) demo project included in the package.

The data in the edit control can be manipulated by using the properties, methods and the API functions. The 'properties' can be used to manipulate the basic text attributes, whereas, the methods and API functions can be used to manipulate complex attributes. The API functions are declared in the TER.BAS module. Some API functions are described later in this chapter. Please refer to the '[Application Interface Functions](#)' chapter for the complete description of methods and API functions.

Please refer to the *ActiveX control* chapter for a list of ActiveX control properties and events.



## Using the Editor as a DLL

This method can be used to create an editor window which is not contained within a Visual Basic form. The editor window can be independently moved and sized. You can enable the menu bar. Thus, using this method, you can create a word processor window which handles all user interaction. The initial window is created using the 'CreateTerWindow' API function. If you simply need to edit a file, you do not need to use any other API functions. However, the package provides a list of API functions that can be used for various text manipulation tasks.

The API functions for the use with Visual Basic are defined in the TER.BAS file. The product provides three additional functions: 'DiscardNull', 'StrToHandle' and 'HandleToStr' specifically for the use in the Visual Basic environment. All other functions are the same as those available for the 'C' environment. A brief description of often used

API functions for the Visual Basic environment is presented here in alphabetic order. **However, please refer to the 'Application Interface Function' chapter for a complete list of API functions.** Also, please study the demo program (dmo\_vb) for a practical example of using these functions.

#### CloseTer

CloseTer(hWnd as integer, force as integer) as integer

Description: Use to close a TER window.

The function returns a TRUE value if the window is closed.

#### DiscardNull

DiscardNull(InString as string) as string

Description: Unlike other functions, the *body* of this function is defined in the TER.BAS file. This function is needed to discard the terminating NULL character that a 'C' DLL appends to the string output data. This function is not needed when a string is used for input only

#### GetTerFields

GetTerFields(hWnd as integer, field as StrTerField) as integer

Description: Retrieves the operational variables for the specified TER window. This information is returned in the 'StrTerField' structure type specified by the second argument. The 'StrTerField' structure type is defined in the TER.BAS file. If you wish to modify a TER operation variable, you must call this function before calling the 'SetTerFields' function.

This function returns a TRUE value if successful.

#### HandleToStr

HandleToStr(OutString as string, StringLen as long, hMem as integer) as integer

Description: This function can be used to copy the contents a global memory block to a Visual Basic string. The third argument specifies the handle of the global memory block. You must specify the length of data to retrieve using the second argument. The length must not exceed the size of the global memory block. The first argument specifies the Visual Basic string that will receive the data. This string must be large enough to hold the specified length of data. Use the 'space' (VB) function to set the string size. Example:

```
Dim OutString as string
```

```
Dim result as integer
```

```
OutString = space(BufferLen)
```

```
result = HandleToStr(OutString,BufferLen,hMem)
```

This function also frees the global memory handle after copying. To simply free the block without retrieving the data, call this function with the second argument as zero:

```
result = HandleToStr(" ",0,hMem)
```

This function returns a TRUE value if successful.



## SetTerFields

SetTerFields(hWnd as integer, field as StrTerField) as integer

Description: Use this function to modify the operational variables for the specified TER window. Your program should call the 'GetTerFields' to get the existing variables in a StrTerField structure type variable. You can then modify within the structure the variables that you wish to change and call the 'SetTerFields' function to make the changes effective.

This function returns a TRUE value if successful.

## StrToHandle

StrToHandle(InString as string, StringLen as long) as integer

Description: This function creates a global memory handle and copies the specified string to the global memory block. The length of the string to copy is specified by the second argument. This function returns the global memory handle if successful. Otherwise it returns a zero value. Example:

```
Dim InString as string
```

```
Dim hMem as integer
```

```
InString = "This is a test line"
```

```
hMem=StrToHandle(InString,len(InString))
```

```
if hMem = 0 then MsgBox "Error creating global memory block"
```

## CreateTerWindow

CreateTerWindow(arg as arg\_list) as integer

Description: This most frequently used function is called to open a new TER window. The parameter to open the TER window is specified by the 'arg\_list' argument. The 'arg\_list' structure type is defined in the TER.BAS file. After the call, the handle of the TER window is returned in the 'hTerWnd' member variable.

This function returns a TRUE value if successful.



## Visual C++ Interface

A Microsoft Visual C++ application can interface with the editor using one of the following three methods:

- Use the editor control as an OCX

- Use CTer class interface

- Use CTerView class interface

## In This Chapter

[CTer Class Interface:](#)

[CTerView Class Interface:](#)

[Recompile the DLL](#)



## CTer Class Interface:

The CTer class interface is provided by the TER\_MFC.H and TER\_MFC.CPP files. Your application module that uses the CTer class object should include the TER\_MFC.H file. Your application project should include the TER\_MFC.CPP and TER.LIB files. Also the TER.DLL file must be available in the current directory or a directory included in the path statement.

CTer is a class derived from the CWnd class provided by Microsoft's MFC library. Your application can create an object directly from CTer class or from a class derived from CTer class.

The editor window creation is a two step process. The first step is to call the constructor, and the second step is to call the 'Create' function.

The editor control sends the ON\_EN\_CHANGE message to the parent window. The parent window should include an entry in the message map to handle this message. The message map takes the form of 'ON\_Notification(ControllId, MemberFunction)'.

Your application can communicate with the editor control using the class member functions as well as the DLL functions described in the '*Application Interface Functions*' chapter. The class member functions provide rudimentary functionality similar to the Microsoft's CEdit class functions. The DLL functions should be used for additional manipulation of the control data.

### Class Member Functions:

CanUndo:	Returns TRUE if the undo buffer is not empty.
Clear:	Delete the selected text.
Copy:	Copy the selected data to the clipboard.
Create:	Create the edit control and attach it to the CTer object.
CTer:	Constructor.
Cut:	Cut the selected data to the clipboard.
DefaultEditStyles:	Return the default editor window styles which can be used to create the control window.

DeleteContents:	Delete the current contents of the edit control.
GetFirstVisibleLine:	Get the line index of the topmost line in the window.
GetHandle:	Retrieve the control data.
GetLine:	Retrieve the text for a line.
GetLineCount:	Get the total number of lines in the edit control.
GetModify:	Returns TRUE if the text is modified
GetSel:	Retrieve the selected text.
LineFromChar:	Get the line number from a given character index.
LineIndex:	Retrieve the character index for a line.
LineLength:	Get the length of the given line.
LineScroll:	Scroll the control window.
MenuEnable:	Returns TRUE if a menu item should be enabled.
MenuSelect:	Returns TRUE if a menu item should be checked.
Paste:	Paste the text data from the clipboard.
ReplaceSel:	Replace the selected text or insert new text.
Serialize:	Save the contents of the control to the archive.
SerializeRaw:	Save the contents of the control to the standalone archive file.
SetHandle:	Assign new data to the control.
SetModify:	Set/reset the modification flag.
SetReadOnly	Set/reset the ReadOnly attribute.
SetSel:	Select text.
TerMenu:	Execute an editor menu command.
Undo:	Reverse the previous edit operation.

### **Class Member Function Description**

### CanUndo:

Returns TRUE if the undo buffer is not empty.

BOOL CanUndo();

See Also: Undo

### Clear:

Delete the selected text.

void Clear();

See Also: SetSel(), DeleteContents()

### Copy:

Copy the selected data to the clipboard.

void Copy();

See Also: Cut(), Paste()

### Create:

Create the edit control and attach it to the CTer object.

BOOL Create(dwStyle, rect, pParentWnd, nID);

DWORD dwStyle; // Window style bits. Please refer to the 'Create Window' section in the 'Getting Started' chapter for a list of control style constants. These constants have a prefix of 'TER\_', such as TER\_WORD\_WRAP, TER\_PRINT\_VIEW, etc.. More than one styles can be used by using the logical OR (|) operator.

Set this argument to 0 to use the default styles as given by the CTer::DefaultEditStyles function.

const RECT &rect; // Window rectangle. The rectangle coordinates are specified in the pixel units relative to the top-left of the parent window.

Cwnd \*pParentWnd; // Pointer to the parent window object

UINT nId; // Control id. Each control within a parent window should have a unique control id.

Remarks: This function calls the CWnd::Create function passing the above arguments. The window class is given by the TER\_CLASS constant defined in the TER.H file.

The dwStyle argument should always include the WS\_CHILD style. Usually the dwStyle argument will also include the WS\_VISIBLE and WS\_BORDER styles.

Windows sends the initial window creation messages which can be handled by overriding the default handlers for the OnCreate, OnNcCreate, OnNcCalcSize and OnGetMinMaxInfo functions.

Return Value: The function returns a TRUE value when successful. Otherwise it returns 0.

See Also: DefaultEditStyles()

## **CTer**

Constructor.

CTer();

## **Cut:**

Cut the selected data to the clipboard.

void Cut();

## **DefaultEditStyles:**

Return the default editor window styles which can be used to create the control window.

DWORD DefaultEditStyles();

Remarks: This function returns the default style bits that can be used to create a control window. The default style bits include the following styles:

Return Value: This function returns the window style bits as listed above.

See Also: Create()

## **DeleteContents:**

Delete the entire contents of the edit control.

void DeleteContents();

Remarks: This function clears the contents of the edit control by assigning it an empty buffer.

See Also: Clear()

## **GetFirstVisibleLine:**

Returns the line index of the topmost line in the window.

long GetFirstVisibleLine();

## **GetHandle:**

Retrieve the control data.

HGLOBAL GetHandle(BufferLen);

long far \*BufferLen; // this argument receives the length of the buffer.

Remarks: This function retrieves the current contents of the control in a global memory handle. The buffer contains the text as well as the formatting information. Your application is responsible for freeing this handle when you no longer need it.

The format of the data in the buffer will be the same as the input buffer. You can, however, get the data in an alternate format by calling the ::TerSetOutputFormat DLL function before calling this function.

Return Value: The function returns the handle to a global memory block containing the

contents of the control.

See Also: SetHandle(), ::TerSetOutputFormat()

### **GetLine:**

Retrieve the text for a line.

```
int GetLine(index, buffer);
```

```
int GetLine(index, buffer, MaxLength);
```

long index; // index of the line to retrieve the text. The line number must be between 0 and TotalLines -1.

LPSTR buffer; The buffer pointer to receive the text. The first WORD of the buffer stores the maximum length of the buffer.

When calling the first implementation of the function, your application is responsible for assigning the maximum buffer length to the first word before calling this function. In the second implementation, the GetLine function assigns the MaxLength argument to the first word.

int MaxLength; // maximum size of the text to return.

Remarks: The text data returned by this function does not include the format information. The text string is *not* NULL terminated.

Return Value: The function returns the length of the text returned in the buffer.

See Also: GetLineCount()

### **GetLineCount:**

Get the total number of lines in the edit control.

```
long GetLineCount();
```

### **GetModify:**

Returns TRUE if the text is modified

```
BOOL GetModify();
```

See Also: SetModify()

### **GetSel:**

Retrieve the position of the selected text.

```
void GetSel(StartPos, EndPos);
```

long &StartPos; // Starting character index of the highlighted block

long &EndPos; // Index of the first non-highlighted character past the selected block

Remarks: If a block is not highlighted, both the StartPos and EndPos variables are set to zero.

See Also: SetSel()

### LineFromChar:

Get the line number from a given character index.

```
long LineFromChar(nIndex);
```

long nIndex; // character index of the location

See Also: LineIndex()

### LineIndex:

Retrieve the character index for a line.

```
long LineIndex(nLine);
```

long nLine; // Line index of a line. The line index must be between 0 and TotalLines - 1.

See Also: LineFromChar(), GetLineCount()

### LineLength:

Get the length of the given line.

```
int LineLength(nLine);
```

long nLine; // Line index of a line..

Return Value: If the nLine argument is between 0 and TotalLines - 1, this function returns the length of the specified line.

If nLine is -1, and a block is not highlighted, the function then returns the length of the current line.

If nLine is -1 and a block is highlighted, then the function returns the number of unhighlighted characters in the highlighted lines. For example, if the selected block contains characters starting from the third character of the second line through the 25th character of the sixth line, and if the sixth line is 30 characters long, then this function will return 7 (2 for the second line and 5 for the sixth line).

### LineScroll:

Scroll the control window.

```
void LineScroll(nLines, nChars);
```

long nLines; // number of lines to scroll the window vertically. If nLines is negative, the window is scrolled up.

int nChars; // number of characters to scroll the window horizontally. If nChars is negative, the window is scrolled toward the left.

### MenuEnable:

Returns TRUE if a menu item should be enabled.

```
BOOL MenuEnable(Menuitem);
```

int MenuItem: // menu item number to test. The MenuItem can be one of the constants defined in the TER\_CMD.H file.

Remarks: This function is helpful when your application uses the menu options to manipulate the editor control. Typically, your application will have menu options similar to the demo program. The MenuEnable function can be used to enable or disable a menu option.

For Example, consider the clipboard 'Cut' option in the menu. The following statement in the update handler for this menu option will enable or disable the 'Cut' menu selection:

```
void CMyView::OnUpdateEditCut(CCmdUI *pCmdUI)
{
    pCmdUI->Enable(ter.MenuEnable(ID_CUT));
}
```

The 'ID\_CUT' constant is defined in the TER\_CMD.H file.

Return Value: This function returns TRUE if the menu option is to be enabled.

See Also: MenuSelect(), TerMenu()

### MenuSelect:

Returns TRUE if a menu item should be checked.

BOOL MenuSelect(MenuItem);

int MenuItem: // menu item number to test. The MenuItem can be one of the constants defined in the TER\_CMD.H file.

Remarks: This function is helpful when your application uses the menu options to manipulate the editor control. Typically, your application will have menu options similar to the demo program. The MenuSelect function can be used to 'check' a menu option.

For Example, consider the 'Bold' option in the font menu. The following statement in the update handler for this menu option will check or uncheck the 'Bold' menu selection:

```
void CMyView::OnUpdateFontBold(CCmdUI *pCmdUI)
{
    pCmdUI->SetCheck(ter.MenuSelect(ID_BOLD_ON));
}
```

The 'ID\_BOLD\_ON' constant is defined in the TER\_CMD.H file.

Return Value: This function returns TRUE if the menu option is to be checked.

See Also: MenuEnable(), TerMenu()

### Paste:

Paste the text data from the clipboard.

void Paste();

### ReplaceSel:

Replace the selected text or insert new text.



```
void ReplaceSel(NewText);
```

char huge \*NewText; // pointer to the new text which will replace the old text.

Remarks: This function replaces a highlighted block of text with the new text specified by the argument. If a block is not highlighted, the new text is simply inserted at the current caret location.

### Serialize:

Save the contents of the control to the archive.

```
void Serialize(ar);
```

CArchive &ar; // archive object reference

Remarks: This function retrieves and stores the control data from the archive file. The text is preceded by a 4 byte header block that stores the length of the control data buffer.

See Also: SerializeRaw()

### SerializeRaw:

Save the contents of the control to the standalone archive file.

```
void SerializeRaw(ar);
```

CArchive &ar; // archive object reference

Remarks: This function retrieves and stores the control data from the archive file. Unlike the 'Serialize' function, this function does *not* use a header block. The archive file is expected to be a standalone file.

See Also: Serialize()

### SetHandle:

Assign new data to the control.

```
BOOL SetHandle(hBuffer, BufferLen, title, release);
```

HANDLE hBuffer: // The global handle to the buffer containing the new text and format data.

long BufferLen; // The size of the hBuffer buffer

LPBYTE title; // new title for the window. Specify a NULL value if you do not wish to change the window title

BOOL release; // Release the buffer after applying

Description: You can use this function to set new data in an existing TER window. *The existing text in the window is discarded.* The data in the buffer can be provided in one of these formats:

Text Format

Rich Text Format

TER Native Format

If the 'release' flag is set, the hBuffer handle becomes the property of the TER window. Your application must not try to lock or free this buffer.

Return Value: This function returns a TRUE value if successful. Otherwise it returns a FALSE value.

See Also: GetHandle()

### **SetModify:**

Set/reset the text modification flag.

```
void SetModify(bModified);
```

BOOL bModified; // new status of the modification flag

Remarks: The editor automatically sets an internal flag when the user modifies the text. This flag is used to prompt the user to save the text before closing the window. You can use this function to override the status of the modification flag.

See Also: GetModify()

### **SetReadOnly**

Set/reset the ReadOnly attribute.

```
BOOL SetReadOnly(bReadOnly);
```

BOOL bReadOnly; // new status of the ReadOnly flag

Return Value: This function returns the previous status of the ReadOnly flag.

### **SetSel:**

Select text.

```
void SetSel(nStartChar, nEndChar, bNoScroll);
```

long nStartChar; // Starting character index of the block

long nEndChar; // Ending character index of the block

BOOL bNoScroll; // Set to TRUE to scroll the selection ending character into view.

Remarks: The highlighting is automatically turned off when the user inputs a new character or hits any direction key.

See Also: GetSel()

### **TerMenu:**

Execute an editor menu command.

```
void TerMenu(MenuItem)
```

int MenuItem: // menu item number to test. The MenuItem can be one of the constants defined in the TER\_CMD.H file.

Remarks: This function is helpful when your application uses the menu options to manipulate the editor control. Typically, your application will have menu options similar to the demo program. The TerMenu function is used to call the editor DLL to perform a menu option.

For Example, consider the 'Bold' option in the font menu. The following statement in the

handler for this menu option will invoke the corresponding editor DLL handler:

```
void CMyView::OnFontBold()  
{  
    ter.TerMenu( ID_BOLD_ON );  
}
```

The 'ID\_BOLD\_ON' constant is defined in the TER\_CMD.H file.

See Also: MenuEnable(), MenuSelect()

### Undo:

Reverse the previous edit operation.

```
void Undo();
```

See Also: CanUndo()



## CTerView Class Interface:

The CTerView class interface is provide by the TER\_VIEW.H and TER\_VIEW.CPP files. Your application module that uses the CTerView class object should include the TER\_VIEW.H file. Your application project should include the TER\_VIEW.CPP and TER.LIB files. Also the TER.DLL file must be available in the current directory or a directory included in the path statement.

CTerView is a class derived from the CView class provided by Microsoft's MFC library. Your application can add the CTerView class or a class derived from CTerView to the document template.

The CTerView class provides the menu handlers for each menu option. The command ids for the menu handlers are defined in the TER\_VIEW.RCH file. You can use the Visual C++ Application Studio to connect these menu ids to the menu option in your application frame window. For Application Studio to recognize these symbols, the TER\_VIEW.RCH file must be specified as a read only symbol file. Select the 'Set Include' option from the 'File' menu in Application Studio and add the TER\_VIEW.RCH file to the list of 'Read Only Symbol Directive' list box.

Your application can communicate with the editor control using the class member functions as well as the DLL functions described in the '*Application Interface Functions*'. The class member functions provide rudimentary functionality similar to the Microsoft's CView class functions. The DLL functions should be used for additional manipulation of the control data.

### Class Member Functions:

CTerView:                      Constructor.

EditStyles:                    Returns the default editor window styles which are used to

	create the control window.
DeleteContents:	Delete the current contents of the edit control.
MenuEnable:	Returns TRUE if a menu item should be enabled.
MenuSelect:	Returns TRUE if a menu item should be checked.
OnXXXX	Handlers for the menu commands
OnUpdateXXXX	Update handlers for the menu commands
Serialize:	Save the contents of the control to the archive.
SerializeRaw:	Save the contents of the control to the standalone archive file.
TerMenu:	Execute an editor menu command.

## Class Member Function Description

### CTerView:

Initialize internal variables.  
CTerView();

### EditStyles:

Returns the editor attribute bits.  
WORD EditStyles();  
Remarks: This function is called by the 'CTerView::PreCreateWindow' function to modify the editor window style bits. By default, the function returns TER\_WORD\_WRAP, TER\_VSCROLL, and TER\_BORDER\_MARGIN styles. You can override this function to provide other styles. Please refer to the 'Create Window' section in the 'Getting Started' chapter for a list of control style constants

### DeleteContents:

Delete the entire contents of the edit control.  
void DeleteContents();  
Remarks: This function clears the contents of the edit control by assigning it an empty buffer.

### MenuEnable:

Returns TRUE if a menu item should be enabled.  
BOOL MenuEnable(Menuitem);  
int Menuitem: // menu item number to test. The Menuitem can be one of the constants

defined in the TER\_CMD.H file.

Remarks: This function is called internally to enable or disable a menu option in an OnUpdateXXXX type handler.

Return Value: This function returns TRUE if the menu option is to be enabled.

See Also: MenuSelect(), TerMenu()

### MenuSelect:

Returns TRUE if a menu item should be checked.

BOOL MenuSelect(Menuitem);

int Menuitem: // menu item number to test. The Menuitem can be one of the constants defined in the TER\_CMD.H file.

Remarks: This function is called internally to 'check' a menu option in an OnUpdateXXXX type handler.

Return Value: This function returns TRUE if the menu option is to be checked.

See Also: MenuEnable(), TerMenu()

### OnXXXX

### OnUpdateXXXX

These functions provide the handlers for each menu option that your application's frame window can incorporate to allow the user to interact with the editor. The menu item ids are defined in the TER\_VIEW.RCH file. This file must be included in the Application Studio's ReadOnly Symbol Directives. The following is a brief description of each id:

TID_LINE_BEGIN	Position at the beginning of the line
TID_LINE_END	Position at the end of the line
TID_FILE_BEGIN	Position at the beginning of the file
TID_FILE_END	Position at the end of the file
TID_INS_AFT	Insert a line after the current line (non-wordwrap mode only)
TID_INS_BEF	Insert a line before the current line (non-wordwrap mode only)
TID_DEL_LINE	Delete the current line
TID_SPLIT_LINE	Split the current line (non-wordwrap mode only)
TID_JOIN_LINE	Append the line at the end of the current line (non-wordwrap mode only)
TID_NEXT_WORD	Position at the next word
TID_PREV_WORD	Position at the previous word
TID_HILIGHT_LINE	Highlight a line block (non-wordwrap mode only)

TID_BLOCK_COPY	Copy a block (non-wordwrap mode only)
TID_BLOCK_MOVE	Move a block (non-wordwrap mode only)
TID_SELECT_ALL	Select the entire document
TID_CUT	Cut the selected block to the clipboard
TID_COPY clipboard	Copy the selected block to the
TID_PASTE	Paste text from the clipboard
TID_PASTE_SPEC	Paste OLE objects from the clipboard
TID_PICT_FROM_FILE	Import a picture from a bitmap file
TID_SEARCH	Search a text string
TID_SEARCH_FOR	Find the next occurrence of the text string
TID_SEARCH_BACK	Find the previous occurrence of the text string
TID_REPLACE	Replace a text string with another text string
TID_HELP	Show the editor help window
TID_UNDO	Undo the previous edit operation
TID_INSERT	Toggle the insert/overtyping mode
TID_INSERT_DRAW_OBJECT	Insert a drawing object
TID_INSERT_OBJECT	Insert an object into text
TID_INSERT_PAGE_NUMBER	Insert the page number string
TID_INSERT_PARA_FRAME	Insert a text frame
TID_PRINT	Print the document to the current printer
TID_PRINT_PREVIEW	Print preview
TID_PAGE_OPTIONS	Edit page setup options
TID_JUMP	Jump to a specified line number
TID_CHAR_NORMAL	Reset all character styles
TID_BOLD_ON	Toggle bold character style
TID_ULINE_ON	Toggle underline character style
TID_ITALIC_ON	Toggle italic character style
TID_STRIKE_ON	Toggle strikethrough character style
TID_SUPSCR_ON	Toggle superscript character style
TID_SUBSCR_ON	Toggle subscript character style
TID_COLOR	Choose text foreground color
TID_BK_COLOR	Choose text background color
TID_FONTS	Choose text font
TID_PARA_NORMAL	Reset all paragraph styles
TID_CENTER	Center the current paragraph

TID_RIGHT_JUSTIFY	Right justify the current paragraph
TID_JUSTIFY	Paragraph justification on both edges.
TID_LEFT_INDENT	Left indent the current paragraph
TID_RIGHT_INDENT	Right indent the current paragraph
TID_DOUBLE_SPACE	Double space the current paragraph
TID_HANGING_INDENT	Create hanging indents for the current paragraph
TID_TAB_CLEAR	Clear a tab position for a paragraph
TID_TAB_CLEAR_ALL	Clear all tab positions for a paragraph
TID_PAGE_BREAK	Insert a page break
TID_PARA_KEEP	Paragraph keep together
TID_PARA_KEEP_NEXT	Paragraph keep with next
TID_REPAGINATE	Repaginate now
TID_SECT_BREAK	Insert a section break
TID_SECT_OPTIONS	Edit section options
TID_FITTED_VIEW	Toggle fitted view
TID_COL_BREAK	Insert a column break
TID_RULER	Toggle the ruler display
TID_TOOL_BAR	Toggle the toolbar display (use the framework toolbar instead)
TID_STATUS_RIBBON	Toggle the status ribbon display (use the framework status ribbon instead)
TID_SHOW_HYPERLINK_CURSOR	Toggle the display of hyperlink cursor.
TID_EDIT_PICT	Edit picture size
TID_SHOW_HIDDEN	Show hidden text
TID_HIDDEN_ON	Toggle hidden character attribute
TID_ULINED_ON	Toggle double underline character attribute
TID_PROTECT_ON	Toggle protect character attribute
TID_PROTECTION_LOCK	Toggle protection lock for the document
TID_PARA_BORDER	Edit paragraph borders
TID_SHOW_PARA_MARK markers.	Toggle the display of paragraph markers.
TID_INSERT_OBJECT	Insert an OLE object from a file
TID_TABLE_INSERT	Insert a new table
TID_TABLE_INSERT_ROW	Insert a table row
TID_TABLE_SPLIT_CELL	Split current table cell
TID_TABLE_MERGE_CELLS	Merge selected table cells
TID_TABLE_DEL_CELLS	Delete table cells

TID_TABLE_ROW_POS	Table row positioning
TID_TABLE_CELL_BORDER	Edit cell border
TID_TABLE_CELL_SHADE	Edit cell shading
TID_TABLE_SHOW_GRID	Show table grid lines

### Serialize:

Save the contents of the control to the archive.

```
void Serialize(ar);
```

CArchive &ar; // archive object reference

Remarks: This function retrieves and stores the control data from the archive file. The text is preceded by a 4 byte header block that stores the length of the control data buffer.

See Also: SerializeRaw()

### SerializeRaw:

Save the contents of the control to the standalone archive file.

```
void SerializeRaw(ar);
```

CArchive &ar; // archive object reference

Remarks: This function retrieves and stores the control data from the archive file. Unlike the 'Serialize' function, this function does *not* use a header block. The archive file is expected to be a standalone file.

See Also: Serialize()

### TerMenu:

Execute an editor menu command.

```
void TerMenu(Menuitem)
```

int Menuitem: // menu item number to test. The Menuitem can be one of the constants defined in the TER\_CMD.H file.

Remarks: This function is called internally to call the DLL menu handler for an OnXXXX type handler implementation.

See Also: MenuEnable(), MenuSelect()



## Recompile the DLL

**If you need to modify the DLL source code and recompile within the Visual C++ environments, follow these steps to create a project:**

Files: TER\*.C, TER.DEF and TER.RC

Executable Type: Windows DLL



Alignment (Compiler Option): 1 Byte

Optimization: Minimum size

Remaining parameters should be left at their default values.

The 32 bit product should also include the imm32.lib as an additional linker library.

### Using Borland C++ Builder

1. Create a new project (File->New) of type 'DLL' and name it ter32. This process would create a new file called ter32.cpp.

2. Select 'Project->Add To Project' option to add ter\*.c (no cpp files), ter.def, and ter.rc files.

3. Select 'Project->Options' menu to set the 'Compiler' warning to none and Data Alignment (Advance Compiler options) to 'Byte'.

4. Select the Project->view to edit the main project file, ter32.cpp. This file contains the DllEntryPoint function. Modify this function to include a call to the TerInit function to initialize the DLL as following:

```
#include "ter.h"

int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason,
                        void *)
{
    return TerInit(hinst,reason);
}
```

5. Save the project as ter32. This step ensures that the resulting dll would be called ter32.dll. Now build the project. The build process will create ter32.dll and ter32.lib files.



## PowerBuilder Interface

*Please note that Sub Systems, Inc. does not test this product with PowerBuilder, however we have a number of users who are successfully using this product under PowerBuilder. The remainder of this chapter is a contribution from one of our kind PowerBuilder developers as a guide to using TE under PowerBuilder.*

A Sybase PowerBuilder application can incorporate the editor as an OCX control for 32 bit applications. The instructions presented here are specific to version 6.5 of PowerBuilder. PowerBuilder versions 6.0 and 5.0 are sufficiently similar that these instructions should be applicable. Versions of PowerBuilder earlier than 5.0 supported the creation of objects with VBX controls for building 16 bit versions of your application. If you have an earlier version of PowerBuilder that supports creation of VBX user objects, you may be able to create a VBX user object and migrate it in PowerBuilder to the present version. The principles for using OCX controls presented here must be adapted to work with a migrated VBX object.

## In This Chapter

[Editor Control as an OCX:](#)

[TER Events and PowerBuilder Events:](#)

[Initialize the OCX control:](#)

[OCX Events, Properties and Methods:](#)

[Using DLL Functions with the OCX:](#)



## Editor Control as an OCX:

In PowerBuilder, an OCX is incorporated into your application as a Standard User Object. You need to create a new Standard User Object of type olecontrol. Select the user object button in the toolbar. Select New. In the New User Object window, select the button for Standard Visual and click OK. In the Select Standard Visual Type window, select olecontrol and click OK. The Insert Object window will open.

In the Insert Object window, select the InsertControl tab. The TER control will appear in the list of OCX controls that have been registered. If the TER control does not appear in the list, click the Register New button. Locate the toc30.ocx file in the browse window, select it and click the open button. This will register the TER control and TER will now appear in the list of registered controls. Double click on TER in the list to create your user object. The TER OCX will open in the user object painter. Save your user object and give it a name, i.e. u\_ter. This PowerBuilder user object consists of the TER OCX in a PowerBuilder OCX container.

With the user object u\_ter open in the user object painter, you can change the initial OCX properties. Right click on the control and select OCX Properties. Check the appropriate properties for your application.

The user object, u\_ter, can be placed on windows in your application just as you would place datawindows and other PowerBuilder objects.



## TER Events and PowerBuilder Events:

PowerBuilder provides a set of standard events including clicked, doubleclicked, etc. Many of these events are not supported by u\_ter under the PowerBuilder event names. You should use the u\_ter events instead. The u\_ter events include click, dblclick, keydown, keyup, keypress, html, etc. You can use the PowerBuilder events of constructor and destructor to initialize the u\_ter control in your application.



## Initialize the OCX control:

When you run your application, the `u_ter` object may not fully initialize to the point of gaining focus and displaying a cursor. The `setfocus()` function of PowerBuilder will not perform this function. Place the following code in the Activate event of the window containing your `u_ter` object to simulate a user mouse click. This will make sure that the TER object has proper focus when your window opens.

```
// #define WM_LBUTTONDOWN 0x0201, = 513
```

```
post send( ole_rtext.il_hWnd, 513, 0, 0 )
```

```
// #define WM_LBUTTONUP 0x0202, = 514
```

```
post send( ole_rtext.il_hWnd, 514, 0, 0 )
```

If you place `u_ter` on a sheet window, you will need to resize the window to the size of the sheet. When you design the sheet window, position the `u_ter` object in the extreme upper left hand corner of the sheet. Then, place the following code in the resize event of the sheet window.

```
int li_ret
```

```
li_ret = u_ter.resize( WorkspaceWidth() - 10, WorkspaceHeight() - 10
```



## OCX Events, Properties and Methods:

Here is more information on `u_ter` events, properties and methods. Open the `u_ter` object in PowerBuilder. Right click on the `u_ter` user object and select Script. Click on the event dropdown and you will see a combined list of PowerBuilder events, `u_ter` OCX events and any user defined events. The PowerBuilder events do not have any function for an OCX because the OCX events take precedence. You should become familiar with the names of the TER OCX events. When you view the list of user events in the user object painter prior to adding any user events, the PowerBuilder events are grayed out and the TER OCX events are white. In addition, user defined events can be added to the `u_ter` object in the user object painter. If you use the naming convention of `ue_eventname` to name your events, you will always know which of the events are user added from within PowerBuilder. Remember that user events that are added from within PowerBuilder must be triggered by your PowerBuilder code. They are not connected to the TER OCX.

You can use the PowerBuilder object browser to see a list of the TER properties that are exposed by the TER OCX and you can find the property definitions in this manual. Use either of the following syntax to get or set an OCX property value. You must make sure that you use the correct type variable in the following since the PowerBuilder compiler does not do type checking. Also, the PowerBuilder compiler does not check that the property name used is valid. The compiler ignores everything after the ".object." part of the syntax.

To set a property value, use: (for example)

**u\_ter.object.PageMode = True**

To get a property value, use: (for example)

**lb\_pagemode = t\_ter.object.PageMode**

If you are going to use the command property to issue menu commands, you need to define instance variables in your u\_ter user object. The command values are found in the source code .h files.

OCX functions are accessed in much the same way as OCX properties. You need to know the function arguments and the type of the return value.

To get the about box, use the following syntax:

**u\_ter.object.AboutBox()**



## Using DLL Functions with the OCX:

Much of the TER functionality is accessed through the dll functions. To access a dll function, the function must be declared. The following are examples of local external function declarations for u\_ter.

**private function int HtsInitialize( long hWnd, long hParentWnd ) library "HTS15.dll"**

**private function int GetTerFields( long hWnd, ref s\_rtext\_parms\_32 parms ) library "ter32.dll"**

**private function int GetTerFieldsAlt( long hWnd, ref s\_parms parms, long LineNo ) library "ter32.dll"**

**private function int InsertTerText( long hWnd, ref blob aBlob, int repaint ) library "ter32.dll"**

**function long HandleToStr( ref blob aBlob, long length, long hMem ) library "ter32.dll"**

**private function int TerInsertRtfFile( long hWnd, ref string rtf\_file, long line, int column, int repaint ) library "ter32.dll"**

Once a function is declared, it can be used in the same manner as a PowerBuilder function. The PowerBuilder compiler performs error checking against the function declarations. If the function declarations are not correct, you will get a run-time error.

Note that integers are passed instead of booleans for arguments. This is because PowerBuilder does not handle booleans well in external function calls. You will need to test the integer values that represent TER booleans in a user function. For example:

**Lb\_boolean = ( HtsInitialize( il\_hWnd, hParentWnd ) <> 0 )**

Some DLL functions require passing integer and boolean values in structures. In Win32 PowerBuilder applications you must create a structure that passes only longs. Your code must convert or interpret the values passed back. The length of boolean and integer values in a structures does not correctly correspond to the win32 values produced by the

Visual C++ compiler.



## Borland OWL Interface

### In This Chapter

[TTreeView Class Interface:](#)



### TTreeView Class Interface:

The TTreeView class interface is provided by the TER\_OWL.V.H and TER\_OWL.V.CPP files. These files are included in the OWL.ZIP (OWL32.ZIP for Win32) file. These files are available from our FTP site: [ftp.subsystems.com](http://ftp.subsystems.com). The OWL.ZIP file also includes a demo program which utilizes this class. Your application module that uses the TTreeView class object should include the TER\_OWL.V.H file. Your application project should include the TER\_OWL.V.CPP and TER.LIB (ter30BC.LIB for Win32) files. Also the ter32.dll and txm12.dll files must be available in the current directory or a directory included in the path statement.

TTreeView is a class derived from the TView and TWindow classes provided by Borland's OWL library. Your application's view class should be derived from the TTreeView class.

The TTreeView class provides functions to invoke the editor menu commands. The editor command ids are included in the TER\_CMD.H file. Another file, TER\_OWL.V.RCH, includes the ids to create the menu items in the Borland resource workshop. The TTreeView class also provides the serialization functions.

Other than invoking menu commands, your application uses the DLL functions described in the '*Application Interface Functions*' to communicate with the editor.

### Class Member Function Description

#### CTerView:

Initialize internal variables.

```
CTerView();
```

#### EditStyles:

Returns the editor attribute bits.

```
WORD EditStyles();
```

Remarks: This function is called by the 'TTreeView::PerformCreate' function to modify the editor window style bits. By default, the function returns TER\_WORD\_WRAP, TER\_VSCROLL, and TER\_BORDER\_MARGIN styles. You can override this function to

provide other styles. Please refer to the 'Create Window' section in the 'Getting Started' chapter for a list of control style constants.

### **MenuEnable:**

Returns TRUE if a menu item should be enabled.

BOOL MenuEnable(Menuitem);

int Menuitem: // menu item number to test. The Menuitem can be one of the constants defined in the TER\_CMD.H file.

Return Value: This function returns TRUE if the menu option is to be enabled.

### **MenuSelect:**

Returns TRUE if a menu item should be checked.

BOOL MenuSelect(Menuitem);

int Menuitem: // menu item number to test. The Menuitem can be one of the constants defined in the TER\_CMD.H file.

Return Value: This function returns TRUE if the menu option is to be checked.

### **TerMenu:**

Execute an editor menu command.

void TerMenu(Menuitem)

int Menuitem: // menu item number to test. The Menuitem can be one of the constants defined in the TER\_CMD.H file.



## **Borland Delphi Interface**

TE Developer's kit includes a OCX (toc30.ocx) which can be used with a 32 bit Delphi application. Please refer to the chapter on ActiveX Control for a complete description of the properties and events for this control. Every property except the 'data' property is accessible to a Delphi application. Instead, for data input and output, a Delphi application should use the SetTerBuffer and GetTerBuffer functions (described later in this function). In addition to these two functions, all other DLL functions are also available for use with a Delphi application.

The package contains two Delphi interface files: TER.PAS and TER\_PROT.PAS. The TER.PAS file contains the constant and record declarations. This file should be included in the interface section of a unit after the type declaration for the form. The TER\_PROT.PAS file contains the function declarations. This file should be included in the 'Implementation' section of your program unit.

Example:

```
Interface
```

```

uses ....
type ...
...
...
end;
{$I TER.PAS}
Implementation
{$I TER_PROT.PAS}

```

Please also refer to the demo application contained in the DELPHI.ZIP file.

**Inserting text into the editor control:** As mentioned earlier, the SetTerBuffer function is used to insert text into the control. The following example reads the data from a disk file and inserts into the control:

**var**

```

InFile : file;
FileLength: Integer;
BytesRead: Integer;
hMem: THandle;
pMem: PChar;
begin
  { open a TER data file}
  AssignFile(InFile, 'DATA.TER');
  Reset(InFile,1);
  FileLength:=FileSize(InFile);
  {allocate space for the input buffer}
  hMem:=GlobalAlloc(GMEM_MOVEABLE,FileLength+1);
  pMem:=GlobalLock(hMem);
  {Read the file into the input buffer.
   The pointer is passed as pMem[0] because
   the buffer argument is passed by reference}
  BlockRead(InFile,pMem[0],FileLength,BytesRead);
  CloseFile(InFile);
  {unlock the buffer and transfer to TER control}
  GlobalUnlock(hMem);
  SetTerBuffer(ter1.handle,hMem,FileLength,' ',True);
end;

```

**Retrieving text from the edit control:** Your application uses GetTerBuffer to retrieve text and formatting information from the control. Example:

**var**

```

hMem: THandle;
TextLength: LongInt;
begin
    {retrieve the text}
    hMem:=GetTerBuffer(ter1.handle,TextLength);
end;

```



## Visual FoxPro Interface

Please note that this product has been tested in 32 bit environment under Visual FoxPro 6.0. The package contains a Visual FoxPro sample application in the foxpro.zip file. To use this product move toc30.ocx, ter32.dll and txml2.dll files to the system directory or to a directory available during runtime. Please refer to the chapter on *ActiveX Control* for a complete description of the properties and the events available through the OCX. In addition to the properties and events exported by the OCX, functions exported by ter32.dll are also available to a Visual ForPro application. These functions have been described in this manual under [Application Interface Functions](#) chapter. An API function must be declared in your application before it can be used. The sample application declares a few functions in the dmo\_fp.prg file.

This file also includes ter\_fp.h file which defines the constants used by various API functions.



## Mail/Merge Support

The product supports two kinds of mail-merge fields.

### **RTF type mail/merge fields:**

First, the RTF type of mail-merge is created using the TerInsertField function, and populated using the TerChangeField function for each field in the document. The TerLocateField function is used to locate the RTF type of fields in the document. The advantage of using the RTF type of mail-merge field is that the field-name is stored separately from the field-data. Therefore, when the data is applied to the field using the TerChangeField field, you can still use the field-name to locate the field again. Click here for the detail description of these [Mail-merge](#) functions.

### **Double-underline type mail/merge fields:**

The second type of fields are created by simply typing the field name and then underlining it. Please refer to the merge.rtf file for an example. The advantage of using



this type of fields is that you can merge all fields in one call to the TerMergeFields function. The second advantage is that you are able to do merge and print as one step without opening a TE window using the TerMergePrint or TerMergePrintVB functions. In this topic we will explore this simple type of mail merge operation.

The simple mail/merge method consists of two components. The first component involves the user who creates a document containing the data field names. The second component is your application which calls the mail/merge print API to print a mail/merge document replacing the field names with field data.

**Creating a Mail/Merge Document:** A mail merge document is very similar to an ordinary document. To insert a field name in a document, do the following:

- Input the field name using the keyboard.
- Highlight the text for the field name.
- Select the 'double underline' option from the font menu to apply the double underline style to the field name. The 'double underline' style is used to indicate the field names.

**Printing a Mail/Merge Document:** A mail merge document must be printed within your program's control. (The 'Print' option in the 'File' menu can not be used to print a mail/merge document).

Your program initiates a mail/merge printing by using the 'TerMergePrint' function. Please refer to the 'Application Interface Functions' chapter for the complete description of this function.

Your application passes the print specification to the 'TerMergePrint' function using the 'StrPrint' structure variable. The 'TerMergePrint' function is called for each record that you wish to merge and print in the document. The following two member variables within the 'StrPrint' structure are used for supplying data for the field names:

*MergeFields:* This field specifies the pointer to a list of mail merge field names. Each field name must be separated by a '|' character. The list must be terminated by a NULL character. If you do not wish to merge field data, set this field to NULL.

*MergeData:* This field specifies the pointer to a list of mail merge data strings. Each data string must be separated by a '|' character. The number of data elements in the 'MergeData' array MUST be the same as the number of elements in the 'MergeFields' array. The list must be terminated by a NULL character. If you do not wish to merge field data, set this field to NULL.

Example:

```
MergeFields="name|address|city|st|zip";  
MergeData="Jim|139 Main St|Springfield|MA|02371"
```

The 'TerMergePrint' function scans the document to extract the field names. If a field name is found in the 'MergeFields' array, the corresponding string in the 'MergeData' is used to replace the field name with the data string in the document.

If the field name is not found in the 'MergeFields' array, the 'TerMergePrint' function sends a TER\_MERGE message to the parent window. The 'lParam' argument for this message contains a far pointer to the field name. If your application processes this message, it should return a far pointer to the corresponding data string.

Example of processing the TER\_MERGE message:

```
case TER_MERGE:
```

```

if (lstrcmpi((LPSTR)lparam,"date")==0)
    return (long)(LPSTR)"Jan 1, 1995";
break;

```

This example returns 'Jan 1, 1995' as a data string for the 'date' field.



## Hyperlink Hooks

The editor provides the hooks to implement hyperlink facility.

**Activation** When the user double clicks on the text formatted with the double underline attribute, the editor sends a hyperlink message to the parent window. The hyperlink text format can be changed from double underline to any format of your choice by using the following code.

```
GetTerFields(hWnd,&field)
```

```
field.LinkStyle = style-constant
```

```
field.LinkColor = RGB-color
```

```
field.LinkDbClick = set to FALSE to activate the hyperlink on a single mouse click.
```

```
SetTerFields(hWnd,&field)
```

For a list of character styles, please refer to the SetTerCharStyle function. A special style-constant called HLINK is also available. This style does not have any visible attribute, but it allows the hyperlink text to have any mix of fonts and colors.

**Message** Message ID=TER\_LINK

```
wParam=0
```

```
lParam=(struct StrHyperLink far *)
```

The 'StrHyperLink' structure is defined  
in the TER.H file as following:

```

struct StrHyperLink {
    HWND hWnd;        // handle of the TER window
    BYTE code[MAX_WIDTH+2];    // link code
    BYTE text[MAX_WIDTH+2];    // link text
    BOOL DoubleClick;    // always TRUE
};

```

The 'text' member variable stores the text formatted with the double underline attribute. If the text is greater than MAX\_WIDTH, only the first MAX\_WIDTH

characters are stored.

The 'code' member variable stores the 'hidden' text found *immediately* before the link text. The format of the link code text can be changed from the 'hidden' attribute to any format of your choosing by editing the 'HYPER\_CODE' global constant.

Return Value: Your application should return a TRUE value if it processes this message. Otherwise it should return a FALSE value.



## Editing Modes

The TER editor offer these four editing modes:

### Text Mode:

The text mode is initiated when the editor is called with word wrapping turned *off*. This mode is most suitable for editing the text files such as computer programs and batch files. In this mode, the lines are not wrapped automatically. This mode does not offer the paragraph formatting features.

### Word Wrap Mode:

This mode is initiated when the routine is called with the word wrapping turned *on*. In this mode, the text in a window is automatically formatted to wrap at the end of the line. Therefore the complete line of text is always visible regardless of the window width. A special character 'ParaChar' is used to delimit a paragraph. This character is not displayed on the screen. Additionally, you have an option of suppressing this character when the file is written out to the disk.

This mode also allows the character and paragraph formatting features.

### Print View Mode:

This mode is initiated when the editor is called with both the Word Wrap and the Print View flags turned on. In this mode, the text lines are wrapped as they would be wrapped when printed to the selected printer. The horizontal scrolling is automatically provided when the text goes beyond the current width of the window. This mode offers all the features of the Word Wrap Mode. In addition, it provide automatic repagination. This mode also allows for sections with multiple columns.

### Page Mode:

This mode is initiated when the editor is called with both the Word Wrap and the Page Mode flags turned on. As in the Print View mode, the text lines are wrapped as they would be wrapped when printed to the selected printer. In this mode, however, the editor displays one page at a time. This mode is most useful for the documents containing multiple columns, as the columns are displayed side by side. In addition, this mode provides all the features of the Print View mode

## FittedView:

This is a special case of the page mode in which the text wraps to the window width and the soft page breaks are not displayed



## Message Communication

Your application can communicate with the TER editor using the following three methods:

*Application Interface Functions*

*Process messages from the editor*

*Send messages to the editor*

The chapter on *Application Interface Functions* describes the application interface functions. This chapter describes the remaining two methods of communication.

### Process Messages from the Editor

The TER editor sends certain messages to your application window. To receive these messages, you must have assigned a valid window handle to the *hParentWnd* variable within the parameter structure (see Getting Started).

TER_MODIFIED	This message is sent to your application window when the editor data is modified the first time. The wParam contains the handle of the TER window. The lParam variable is not used. Your application window can grab this message and take any necessary action.
TER_CLOSE	This message is sent to your application window before a TER window is closed. The wParam contains the handle of the TER window. The lParam variable is not used. Upon receiving this message, your application can retrieve the modified text buffer by using the <i>GetTerBuffer</i> function.
TER_DRAG	This message is sent before text drag/drop is begun from an external application, or from within the editor.
TER_BERORE_DROP	This message is sent immediately before the text is dropped from an external application, or from within the editor.
TER_DROP	This message is sent when text is dropped from an external application, or from within the editor.
TER_NOT_SAVED	Your application received this message when the user

	exits the editor without saving the modifications. This message is always followed by the 'TER_CLOSE' message.
TER_MERGE	Please refer to the 'Mail/Merge Support' chapter for the description of this message.
TER_UPDATE_TOOLBAR	This message notifies your application to update any external toolbar the application might be using.
TER_UPDATE_STATUSBAR	This message notifies your application to update any external status bar the application might be using.
TER_ACTION:	<p>This message is sent after an user initiated action is <i>completed</i>. This message uses the following parameters:</p> <p>wParam (or ActionType): This parameter can be one of the following:</p>
ACTION_COMMAND:	This action indicates any of the menu or the accelerator key generated commands. The actual command id is given by the lParam (or ActionId) argument. For a list of command ids, please refer to the 'command' property in the Visual Basic Interface chapter.
ACTION_VSCROLL:	This action message is sent when the vertical scrollbar is clicked. The 'lParam' argument for this message identifies the actual scrollbar operation and is given by the SB_XXXXX SDK constants.
ACTION_HSCROLL:	This action message is sent when the horizontal scrollbar is clicked. The 'lParam' argument for this message identifies the actual scrollbar operation and is given by the SB_XXXXX SDK constants.
ACTION_CHAR:	This action message is sent when the editor processes a WM_CHAR message. The 'lParam' argument for this message indicates the virtual key code for the key.
ACTION_CHECKBOX	<p>All message posted by Windows to the checkbox input field is sent to your application using ACTION_CHECKBOX. The high 16 bits of the lParam parameter provides the check-box control id, and the low 16 bits of the lParam provides the windows message id (such as WM_CHAR, WM_LBUTTONDOWN, etc).</p> <p>The control-id returned by this message can be used with the check-box input field APIs to retrieve or set new information for the check-box.</p>

lParam (or ActionId): This value is specific to the action type as described above.

TER\_PREPROCESS: This message is sent *before* a message is processed by the editor. This message uses the following parameters:

wParam (or ActionType): This parameter can be one of the following:

*ACTION\_COMMAND*: See TER\_ACTION message for description.

*ACTION\_VSCROLL*: See TER\_ACTION message for description.

*ACTION\_HSCROLL*: See TER\_ACTION message for description.

*ACTION\_CHAR*: See TER\_ACTION message for description.

*ACTION\_LBUTTONDOWN*  
*N*: Left mouse button down. The ActionId contains the x and y mouse position in the pixel units. The x position is given by the low 16 bits and the y position is given by the high 16 bits.

*ACTION\_RBUTTONDOWN*  
*N*: Right mouse button down. The ActionId parameter holds the mouse position as described for the ACTION\_LBUTTONDOWN message.

*ACTION\_LBUTTONUP*: Left mouse button up. The ActionId parameter holds the mouse position as described for the ACTION\_LBUTTONDOWN message.

*ACTION\_RBUTTONUP*: Right mouse button up. The ActionId parameter holds the mouse position as described for the ACTION\_LBUTTONDOWN message.

*ACTION\_LBUTTONDBLCLICK*: Left mouse double click. The ActionId parameter holds the mouse position as described for the ACTION\_LBUTTONDOWN message.

*ACTION\_RBUTTONDBLCLICK*: Right mouse double click. The ActionId parameter holds the mouse position as described for the ACTION\_LBUTTONDOWN message.

*ACTION\_MOUSEMOVE*: Mouse move. The ActionId contains the x and y mouse position in the pixel units. The x position is given by the low 16 bits and the y position is given by the high 16 bits.

*ACTION\_SIZE*: Window being resized.

*ACTION\_SETFOCUS*: Ter window receiving focus.

**ACTION\_KILLFOCUS:** Ter window loosing focus.

**ACTION\_QUERYENDSESSION** This message is sent before asking the user to end the editing session.

IParam (or ActionId): This value is specific to the action type as described above.

See Also: TerIgnoreCommand

**TER\_PAGE\_SIZE\_CHANGING:** **This message is sent before TE adjusts the page size.**

The wParam parameter contains the handle of the TE control window. The lParam parameter contains the pointer to the suggested new page size.

This event is fired only when the TFLAG5\_VARIABLE\_PAGE\_SIZE flag is set. When this flag is set, TE calculates the new page size to contain the entire content of the control. Then the control fires this event to allow your application to override or modified the suggested page size. The page size is provided by the 'NewPageSize' parameter in the twips unit (lParam contains the pointer to new page size variable).

You can set the 'NewPageSize' parameter to 0 to disable the current page adjustment. You can also set it to another value to make the page bigger than the suggested size. However, you can not set this parameter to a lower value because TE needs to display the entire content on one page.

## Send Messages to the Editor

Your application can initiate an editor action by sending the appropriate message. This method is especially useful when the editor menu is disabled, or the TER editor window is used as a child window (WS\_CHILD style).

Your application module that will be communicating with the editor should include the TER\_CMD.H file. This file contains the message constants.

The messages are sent using the WM\_COMMAND message. The TerWndProc function in the TER.C file processes these messages. You can refer to this function also for a brief description of each message.

Example:

```
#include "TER_CMD.H"

SendMessage(hTextWnd, WM_COMMAND, ID_SAVE, 0L);
```

This message instructs the TER window (hTerWnd) to save (ID\_SAVE) the current file. The ID\_SAVE constant is defined in the TER\_CMD.H file.



## Memory Allocation For The Text

The memory allocation for the text is facilitated by two arrays. The first being the array of memory handles. This array is dimensioned to the maximum number of lines (startup parameter) allowed in the editor. The size of this array is increased as required. Each element of this array contain a memory handle for a memory block that stores a line of text. The second arrays contain the length for each text line.

We prefer this dual array scheme over other prevalent algorithms because of the following reasons:

- A. The memory overhead per line is minimum; 11 bytes per line compared with up to 32 bytes when using a linked list approach.
- B. Each line of text is directly addressable unlike some techniques where multiple lines are stored in a single buffer.
- C. Positioning at a line number is simplest using this approach.
- D. The underlying programming required to manage the memory is simplest in this approach.

Four additional arrays are also used. The first one (cfmt) stores the formatting information for each line. Each array element can store 2 types of information. When all characters in a text line use uniform format value, the array element stores the value of the uniform format. However, if the text line has more than one format, the array elements contain a handle to the buffer that contains the format value for each character in the text line. This scheme provides huge memory savings in a typical situation where most text lines contain only one format.

The second array (TerFont) stores the font table.

The third array stores the paragraph id for each line in the document. The paragraph id is an index into the paragraph table. The fourth array stores the paragraph information for each paragraph id in use.



## Program Data Structures

TE Edit control employs the following major data structures:

- |                           |  |
|---------------------------|--|
| <b>Line Handle Array:</b> | (hLine) This array stores the data handle for each text line. The size of this array is equal to 4 times the maximum number of lines allowed.  |
| <b>Line Length Array:</b> | (LineLen) This array stores the length of each text line in the integer format. The size of this array is 2 times the maximum number of lines. |



<b>Format Data Array:</b>	<p>(cfmt) This array stores the formatting information about a text line. The size of this array is equal to 4 times the maximum number of lines.</p> <p>If all the characters in a text line use the same format, then the corresponding array element simply stores the value (index in font table) of the format. Otherwise, the corresponding array element stores the handle to the memory object that contains the format byte for each character of the line.</p>
<b>Font Structure Array:</b>	<p>(TerFont) This structure array stores the information about the fonts and picture bitmaps used by the document. The size of this array is specified by the MAX_FONTS constant.</p>
<b>Paragraph Array:</b>	<p>(pfmt) This array stores an integer size paragraph id for each line in the document. The paragraph id is actually an index into the paragraph table.</p>
<b>Paragraph Table:</b>	<p>(PfmtId) This table stores the paragraph information for each paragraph id in use. For example, if a document uses 5 different combinations of paragraph formats, this table will then contain 5 items. The maximum size of the paragraph table is governed by the MAX_PFMT global constant.</p>
<b>Tab Table:</b>	<p>(TerTab) This table stores the tab information for each tab id in use. For example, if a document uses 5 different combinations of tab stops, this table will then contain 5 items. Each tab item stores up to 20 tab stops. The maximum size of the tab table is governed by the MAX_TABS global constant.</p>



## Incorporating A New Feature

The mechanism of adding a new feature is fairly simple. In this section we will describe the steps to add a new feature to the editor. As an example, we will add the spell checking facility to the editor. Follow these steps to add this new function to the editor:

Define a constant that represents the command id for the new command. This constant is defined in the TER\_CMD.H include file. Select the next sequential number as the command id. Example:

```
#define ID_SPELL_CHECK 670
```

Select a unique accelerator key for the command. We will select the F5 function key for this command. Insert a statement in the accelerator resource section of the TER.RC file as following:

```
VK_F5, ID_SPELL_CHECK, VIRTKEY, NOINVERT
```

Edit the menu resource statement in the TER.RC file to create a menu option for the spell check command. We will create a menu option in the 'Other' submenu. Insert the following line in the 'Other' pop-up menu section:

```
MENUITEM "Spell &Check\tF5", ID_SPELL_CHECK
```

Modify the 'TerWndProc' routine in the TER.C source file. Insert the following 'case' statement in the 'Miscellaneous Section' section under the WM\_COMMAND processing:

```
case ID_SPELL_CHECK: /* spell check routine */  
    SpellCheck(w);break;
```

These statements will cause the editor to call the SpellCheck routine whenever the user hits the F5 function key, or selects the option from the 'Other' submenu.

. Now select a source code file to write the SpellCheck routine. We will include the SpellCheck routine in the TER\_MISC.C source file.

```
SpellCheck(PTERWND w)  
{  
    .../* write your programming code here */  
    ...  
    ...  
}
```



## Analysis of the Demo Program

This section analyses the DEMO program to describe a step by step method of incorporating the editor window routine into your application. For the sake of drawing a parallel with your application, assume that demo.c represents your application.

**DEMO.H:** (A section of your program that contains the global variables)

You can ignore most of the 'define' statements and the function prototypes in DEMO.H file. These are required by the code that merely accepts the user parameters.

The only 'define' statement that is important to note here is 'MAX\_WINDS'. The demo application allows up to 10 simultaneous TER windows. This define statement equates 'MAX\_WINDS' to 11. This variable is used by the demo program so that no more than 'MAX\_WINDS' windows are open at a time.

The second section defines an array variable 'arg' of the structure type 'arg\_list'. This structure is defined in the TER.H file. One element of the 'arg' array is required to open each TER window. If you plan to open only one TER window at a time, then the 'arg' variable does not have to be an array. The demo program allows up to 'MAX\_WINDS' TER windows, thus the 'arg' array variable has a dimension of 'MAX\_WINDS'.

The CurWnd variable is used internally by the demo program to store the current window number.

The 'AbortProgram' defines the error location to jump to if a fatal error is

encountered. Your program passes a pointer to this variable to the editor using the structure 'arg\_list'. You can pass a NULL for this argument to skip any fatal error trapping.  
The 'PrevFile' is used by the demo program for its internal use.

**DEMO.C:** The source file that contains the main message loop and a call to the editor routine.

Include statement for the TER.H file. This file defines the argument structure (arg\_list). It also contains the prototypes for the TER API functions.  
The remaining part of DEMO.C program accepts user parameters to stuff into the structure 'arg\_list'. In your application, you can either prompt the user for this information, or you may define them without the user's assistance. The 'InitInstance' function of the demo program defines the default value for each element of the structure. The 'DemoParam', the dialog box routine accepts any modification from the user. The 'MakeBuffer' function, allocates a global memory block and reads a text file into this location. The handle to this location is passed in the 'arg\_list' structure. The 'CallTer' function actually calls the TER editor.

Syntax:

```
int CreateTerWindow(struct arg_list far *);
```

The '*CreateTerWindow*' function opens a TER window and returns immediately.

**DEMO.RC:** This resource file contain the menu and dialog template for the internal use of the demo program.



## Text Editor Commands

This chapter describes the editor commands by menu groups.

### In This Chapter

- [How To Scroll Through The Text](#)
- [File and Print Commands](#)
- [Line Edit Commands](#)
- [Block Edit Commands](#)
- [Clipboard Commands](#)
- [Picture and Object Import Commands](#)
- [Character Formatting Commands](#)
- [Paragraph Formatting Commands](#)
- [Paragraph Spacing, Borders and Shading](#)
- [Tab Support](#)
- [Page Break and Repagination](#)
- [Page Header/Footer, Bookmark and Footnote Commands](#)
- [Table Commands](#)
- [Section and Columns](#)

[Stylesheet and Table-of-contents](#)  
[Text/Picture Frame and Drawing Objects](#)  
[View Options](#)  
[Navigation Commands](#)  
[Search/Replace Commands](#)  
[Highlighting Commands](#)



## How To Scroll Through The Text

### Keyboard:

Use *Up*, *Down*, *Left* and *Right* arrow keys to scroll up or down a line, or left or right one character.

. Hit the *Home* key to position at the beginning of the current line.

Hit the *End* key to position at the end of the current line.

Hit *Ctrl-PgUp* to position at the beginning of a file.

Hit *Ctrl-PgDn* to position at the end of a file.

Hit *PgUp* to display the previous page.

Hit *PgDn* to display the next page.

Hit *Ctrl* - *Left* arrow key to position on the next word.

Hit *Ctrl* - *Right* arrow key to position on the previous word.

Hit *Ctrl* - *Up* arrow key to position at the first column of the current line (if not already on the first column) or at the first column of the previous line.

Hit *Ctrl* - *Down* arrow key to position at the first column of the next line.

Hit the *F10* key and type in the line number to jump to. This function is also available from the Navigation menu.

### Mouse:

You can click mouse on the vertical and horizontal scroll bar to accomplish various scrolling function. These functions are available only if the horizontal or the vertical bar has been enabled by the startup parameters:

**Vertical Scroll Bar:** Click the mouse on the arrows on either end to scroll the screen up or down by one line. Click the mouse above the elevator to scroll the screen up by one page. Similarly, click the mouse below the elevator to scroll the screen down by one page. You may also drag the elevator to any position in the bar. As the elevator is dragged, the editor will scroll the screen up or down accordingly to maintain the correct cursor position.

**Horizontal Scroll Bar:** Click the mouse on the arrows on either end to scroll the screen left or right by one line. Click the mouse on either side of the elevator to scroll the screen left or right by 1/2 screen. You may also drag the elevator to any position in the bar. As the elevator is dragged, the editor will scroll the screen left or right accordingly to maintain the correct cursor position.



## File and Print Commands

<b>New File</b>	<p>This function is used to clear the existing text from the edit window and start an empty, unnamed document. The user is prompted to save any modification to the previous document.</p>
<b>Open File</b>	<p>This function is used to clear the exiting text from the edit window and open a new document. The user is prompted to save any modification to the previous document.</p>
<b>Save File</b>	<p>Use this selection to save the text to the current file name. If a file is not yet specified, the editor will prompt you for a file name. If a file with the same name already exists on the disk, the editor will save the previous file with a backup extension (.TE).</p> <p>If the I/O is conducted through a buffer rather than a disk file, the editor creates a new buffer with the updated text.</p> <p>You can invoke this function by hitting the F3 function key (or select the option from the menu).</p>
<b>Save File As..</b>	<p>This selection is similar to <i>Save File</i>. In addition, it allows you to specify a new file name for saving the text.</p> <p>This option is not available when the I/O is conducted through a buffer rather than a disk file.</p> <p>You can invoke this function by hitting the Shift F3 function keys together (or select the option from the menu)</p>
<b>Exit</b>	<p>Use this function to exit from the editor session. If the current file is modified, you will have an option to save the modifications.</p> <p>You can invoke this function by hitting the Ctrl F3 function keys together (or select the option from the menu).</p>
<b>Print</b>	<p>Use this option to print the contents of the current file. You may also choose to print only the selected part of the file. To print a block of text, the desired text must be highlighted before invoking the print function. This command supports these highlighted blocks:</p> <p>Line Block</p> <p>Character Block</p> <p>The print function will print on a default printer selected from the Windows' control panel. You can alter the <i>printer setup</i> or <i>Page Layout</i> prior to invoking the print option.</p> <p>You can invoke the printing function by hitting the F4 function key (or select the option from the menu). The editor will display a dialog box where you can select the scope of the printing.</p>

<b>Page Layout</b>	<p>Use this option before selecting the <i>Print</i> option to specify the page layout. You can specify margin (left, right, top and bottom) in inches.</p> <p>You can invoke this function by hitting the Ctrl F4 function keys together (or select the option from the menu).</p>
<b>Printer Setup</b>	<p>This option invokes a printer specific dialog box for the default printer (the default printer selection is made from the control panel of Windows) You select the parameters from a set of printer specific options. These options include page size, page orientation, resolution, fonts, etc.</p> <p>You can invoke this function by hitting the Shift F4 function keys together (or select the option from the menu).</p>
<b>Print Preview</b>	<p>This option is used to preview the document before printing. The editor displays up to 2 pages at a time. You can scroll to a different page by using the PgUp/PgDn or the scroll bar.</p> <p>By default the preview rectangle is sized to fit the current window. However, you can use the zoom option to enlarge or shrink the preview rectangle as you wish.</p>



## Line Edit Commands

<b>Insert After Current Line</b>	<p>In the <i>text mode</i> this function creates a blank line after the current line. Hit the F9 function key to insert a line after the current line.</p>
<b>Insert Before Current Line</b>	<p>In the <i>text mode</i> this function creates a blank line before the current line. Hit the Ctrl F5 keys together to insert a line before the current line.</p>
<b>Delete Line</b>	<p>Use this function to delete the current line. The remaining lines will be scrolled up by one line. Hit the Shift F9 keys together to delete the current line.</p>
<b>Join Lines</b>	<p>In the <i>text mode</i> this function joins the next line at the end of the current line. Hit the Alt J keys together to invoke this function.</p>
<b>Split Line</b>	<p>In the <i>text mode</i> this function splits the current line at the current cursor position. Hit the Alt S keys together to invoke this function.</p>



## Block Edit Commands

<b>Copy a Line Block</b>	<p>Use this command to <b>copy</b> a highlighted block of text lines from one location to another. This command provides a short alternative to using clipboard copy/paste functions.</p> <p>Highlight the lines of text to be copied, move the caret to the target</p>
--------------------------	---

location and hit Alt C (or select the option from the menu). This function does not delete the original block.

### Move a Line Block

Use this command to **move** a highlighted block of text lines from one location to another. This command provides a short alternative to using clipboard cut/paste functions.

Highlight a block of text to be moved, move the caret to the target location and hit Alt M (or select the option from the menu). This function deletes the original block.

### Undo Previous Edit

The editor remembers your last edit command. You can use this function to undo the last edit command.

You can invoke this function by hitting the Shift F8 keys together (or select the option from the menu). The editor will display a dialog box containing the information about the edit command to be undone. The dialog box displays the line number, column position, type of undo (delete/insert/edit) and the contents of the undo buffer. You may modify the target line number or column position. Confirm the operation by clicking on the OK button.

This undo feature is not available for column block edits, block move and replace string commands.

### Redo Previous Undo

This command reverses the previous undo operation.



## Clipboard Commands

### Cut/Copy Text To Clipboard

Use this command to **cut or copy** a highlighted block of text to the clipboard. This function also copies the associated formatting information using the RTF format and the native TER format.

Highlight a block of text to be copied to the clipboard and hit the Ctrl+X (cut) or Ctrl+C (copy) keys, or select the option from the menu.

### Paste Text From Clipboard

Use this command to paste the contents of the clipboard at the current caret location. The formatting information, if available, is also copied.

You can invoke this function by hitting the Ctrl+V keys together (or select the option from the menu).

### Paste Special Objects

This function displays the clipboard data in a number of available formats:

### Native Object Format

If available this is the first format in the list box. The data in this format can be later edited (by double clicking the object) using the *original* application. This data can be *embedded* into your application by using the Paste option, or you can create a *link* to the original file by using the Paste Link option.

<b>Formatted Text</b>	This is one of the text formats. This option offers the most suitable format if the data is pasted by another text output application as the font and formatting attributes are reproduced accurately.
<b>Unformatted Text</b>	This is another text format. This option pastes the text without retaining the formatting information.
<b>Picture Format</b>	The data is available in the Picture format. This object can be later edited (by double clicking the object) using the Microsofts MS Draw application. This format is preferred over the bitmap and the device independent bitmap formats.
<b>Device Independent and regular bitmap formats</b>	The data is available in the bitmap formats. The object can be later edited (by double clicking the object) using the Microsofts MS Draw application. The editor converts these formats into the Picture format before calling the drawing application.

:



## Picture and Object Import Commands

<b>Embed Picture</b>	Use this command to embed a picture bitmap or metafile from a disk file at the current caret location. The embedded picture is saved within the document.
<b>Link Picture</b>	Use this command to link a picture bitmap or metafile to the document. The linked picture appears at the current caret location. A linked picture data is not saved with the document, only its name is stored with the document.
<b>Edit Picture</b>	Use this command to change the width and height of a picture located at the current caret position. The width and height is specified in inches. This function also allows you to align (top, bottom, or middle) the picture relative to the base line of the text.
<b>Insert Object</b>	<p>This function is used to embed objects into the text. The list box shows the applications that are available to create the object. When you select an application, the editor launches the selected application. You can create the desired object using this application. When you save the application, the editor inserts an icon for the application. This icon indicates the inserted object. You can later edit the object by double clicking at it.</p> <p>Please note that you can also use the Paste Special function to import the OLE objects, provided the object is available in the clipboard.</p>
<b>Drag/Drop Function</b>	This is a method of inserting a file object into the text directly. To insert a file, open the Windows File Manager and locate the file to be inserted. Now click the mouse and keep the mouse button depressed as you move the mouse cursor to the editor window. Release the mouse button at the location where the object should be inserted. The editor shows an icon to



The object inserted using this method makes use of Microsofts Packager application to tie the file with the application that originally created it.

Please note that a documented problem with the original Packager application may create errors during this function. Install the corrected version of the PACKAGER.EXE program for proper functioning.

**Background Picture** This option, available from the 'Other' menu, is used to set a background picture for the text. The background picture occupies the entire text area. The picture file can be a Windows' bitmap (.BMP) or Metafile (.WMF).



## Character Styles

Command		Keystroke	
Normal	Alt	0	
Bold Formatting	Ctrl	B	
Underlining	Ctrl	U	
Italic	Ctrl	I	
Superscript	Alt	4	
Subscript	Alt	5	
Strike	Alt	6	

To apply a format to the current character, simply hit the appropriate keystroke (or select the option from the menu). To apply this format on a block of characters, highlight a block using the Line Block or Character Block options. Now, hit the applicable keystroke, or select the option from the menu.

TER allows multiple formats for a character. To apply more than one format, repeat the procedure described in the previous paragraphs.

## Fonts

If you wish to change the font for a highlighted block of text, highlight the block using the Line or Character highlight function. If you wish to change the font of a single character, simply position the cursor on that character. Now select the font option from the menu or hit the Alt F10 keys together.

A dialog box will appear that shows the list of typefaces and point sizes to select from. Make the desired selection now.

### **Colors**

Use this selection to change the text color of the current character or of all characters in a highlighted block of text.

If you wish to change the color of a highlighted block of text, highlight the block using the Line or Character highlight function. If you wish to change the color of a single character, simply position the cursor on that character. Now select the color option from the menu. A dialog box will appear that shows the color selection. Make the desired selection now.

### **Hidden Text**

The text formatted with this attribute are treated as hidden text. Normally the hidden text, as the name implies, does not appear on the screen or printer. However you can display the hidden text by selecting the 'Show Hidden Text' option from the 'View' menu.

### **Protected Text**

The text formatted with this attribute are protected from the editing changes. The protected text appear with a light shade in the window. This function is available only when the 'protection lock' is turned off. The 'protection lock' can be turned off by using an option from the 'Other' menu.



## **Paragraph Formatting Commands**

### **Reset Paragraph Format**

Use this selection to reset all paragraph formats for the current paragraph or for all lines in a highlighted block of text.

To reset the paragraph formats for the current paragraph, simply hit the Alt P keys together (or select the option from the menu). To reset the formats for a block of lines, highlight a block and hit the Alt P Keys together (or select the option from the menu).

### **Paragraph Centering**

Use this selection to center all lines in the current paragraph or all lines in a highlighted block of text.

To center the current paragraph, simply hit the Alt 8 keys together (or select the option from the menu). To center a block of lines, highlight a block of text and hit the Alt 8 Keys together (or select the option from the menu).

### **Paragraph Right Justification**

Use this selection to right justify all lines in the current paragraph or all lines in a highlighted block of text.

To right justify the current paragraph, simply hit the Alt 9 keys together (or select the option from the menu). To right justify a block of lines, highlight a block of text and hit the Alt 9 Keys together (or select the option from the menu).

<b>Paragraph Justification</b>	<p>Use this selection to justify the text on both left and right margins.</p> <p>To justify the current paragraph, simply select the option from the paragraph menu. To justify a block of lines, highlight a block of text and then select this option from the menu.</p>
<b>Paragraph Double Spacing</b>	<p>Use this selection to double space all lines in the current paragraph or all lines in a highlighted block of text. A double spaced paragraph has a blank line between each text line.</p> <p>To double space the current paragraph, simply hit the Alt O keys together (or select the option from the menu). To double space a block of lines, highlight a block of text and hit the Alt O Keys together (or select the option from the menu)</p>
<b>Paragraph Indentation (Left)</b>	<p>Use this selection to create a left indentation for all lines in the current paragraph or for all lines in a highlighted block of text. The successive use of this option increases the amount of left indentation.</p> <p>To apply the left indentation to the current paragraph, simply hit the Alt L keys together (or select the option from the menu). To apply the left indentation to a block of lines, highlight a block of text and hit the Alt L Keys together (or select the option from the menu).</p> <p>To create the left indentation using the mouse, click the left mouse button on the indentation symbol on the lower left end of the ruler. While the mouse button is depressed, drag the mouse to the desired location and release the mouse button. The indentation created using this method is applicable to every line in the paragraph except the first line.</p>
<b>Paragraph Indentation (Right)</b>	<p>Use this selection to create a right indentation for all lines in the current paragraph or for all lines in a highlighted block of text. The successive use of this option increases the amount of right indentation.</p> <p>To apply the right indentation to the current paragraph, simply hit the Alt R keys together (or select the option from the menu). To apply the right indentation to a block of lines, highlight a block of text and hit the Alt R Keys together (or select the option from the menu).</p> <p>To create the right indentation using the mouse, click the left mouse button on the indentation symbol on the lower right end of the ruler. While the mouse button is depressed, drag the mouse to the desired location and release the mouse button.</p>
<b>Paragraph Hanging Indentation</b>	<p>This option is similar to paragraph left indentation, except that the indentation is not applied to the first line of the paragraph.</p> <p>To apply the hanging indentation to the current paragraph, simply hit the Alt T keys together (or select the option from the menu). To apply the left indentation to a block of lines, highlight a block of text and hit the Alt T Keys together (or select the option from the menu).</p> <p>To create the hanging indentation using the mouse, click the left mouse button on the indentation symbol on the upper left end of the ruler. While the mouse button is depressed, drag the mouse to the desired location and release the mouse button.</p>

### **Paragraph Keep Together**

When this attribute is turned on for a paragraph, the editor attempts to keep all lines within the paragraph on the same page.

### **Paragraph Keep with Next**

When this attribute is turned on for a paragraph, the editor attempts to keep the last line of the current paragraph and the first line of the next paragraph on the same page.

### **Widow/Orphan Control**

When this attribute is turned on for a paragraph, the editor attempts to avoid widow/orphan paragraphs. An 'orphan' paragraph results when the last line of the paragraph lies on the next page. A 'widow' paragraph results when the first line of the paragraph lies on the previous page



## **Paragraph Spacing, Borders and Shading**

This functionality is provided by two options in the paragraph menu, one to set the Border and Shading parameters and the other to set the spacing parameters for a paragraph.

The '**Paragraph Spacing**' menu option allows you to set the space before and after the paragraph. You can also specify the minimum space between the paragraph lines. All space parameters are specified in points.

The '**Border and Shading**' option in the paragraph menu allows you to create the paragraph borders and set the shading amount for the paragraph. You can draw all four sides of the border, or you can draw only the selected sides. Additional two options allow you to select a thick and double lined border.

When two or more contiguous paragraphs have identical paragraph formatting parameters, a single border is drawn to enclose all such contiguous paragraphs.

The top line of the border is placed beneath the top of the first line. The bottom line of the border is placed above the bottom of the last line. Create a blank line at the top and bottom if you need additional clearance at the top or bottom. The left line of the border is placed before the left indentation for the paragraph. Therefore, the left side may not be visible for the paragraph with no left indentation. The right line of the border is placed after the right indentation. Therefore, the right side may not be visible for the paragraph where the right margin extends up to or beyond the width of the window.



## **Tab Support**

TE Editor supports left, right, center, and decimal tab stops. The tab stops are very useful for creating columns and tables. A paragraph can have as many as 20 tab positions.

The 'left' tab stop begins the text following a tab character at the next tab position. To create a left tab stop, click the left mouse button at the specified location on the ruler. The

left tab stop is indicated on the ruler by an arrow with a tail toward the right.

The 'right' tab stop aligns the text at the current tab stop such that the text ends at the tab marker. To create a right tab stop, click the right mouse button at the specified location on the ruler. The right tab stop is indicated on the ruler by an arrow with a tail toward the left.

The 'center' tab stop centers the text at the current tab position. To create a center tab stop, hold the shift key and click the left mouse button at the specified location on the ruler. The center tab stop is indicated on the ruler by a straight arrow.

The 'decimal' tab stop aligns the text at the decimal point. To create a decimal tab stop, hold the shift key and click the right mouse button at the specified location on the ruler. The decimal tab stop is indicated on the ruler by a dot under a straight arrow.

The tab stops can also be created by using the 'Set Tab' selection from the 'Paragraph' menu. This option allows you to specify the tab position, tab type (left, right, center, or decimal) and tab leader (dot, hyphen, underline, or none).

To move a tab position using the mouse, simply click the left mouse button on the tab symbol on the ruler. While the mouse button is depressed, drag the mouse to the desired location and release the mouse button.

To clear a tab position, simply click at the desired tab marker, or select the option from the menu. You can also clear all tab stops for the selected text by selecting 'Clear All Tabs' option from the menu.

The 'Snap To Grid' option in the 'Other' menu affects the movement of the tabs (and the paragraph indentation markers) on the ruler. When this option is checked, the movements of these markers are locked on to an invisible grid at an interval of 1/16 inch.

Normally, a tab command is applicable to every line of the current paragraph. However, if you highlight a block of text before initiating a tab command, the tab command is then applicable to all the lines in the highlighted block of text.



## Page Break and Repagination

A hard page break can be inserted in the document by pressing the Control and Enter keys together (or select the option from the menu: Edit->Break->Section Break). A hard page break places the text after the page break on the following page. A hard page break is indicated by a solid line in the editing window.

In the Print View editing mode, the editor also creates automatic page breaks when the text overflows a page. An automatic page break is indicated by a dotted line in the editing window. As the name implies, these page breaks are calculated automatically by the editor between the keystrokes. The repagination process is time consuming. Sometimes there may not be enough time for a large document to complete the repagination between the edits. Therefore, the menu also provides an option to provide complete repagination on demand.

**Inserting Page Number** The 'Page Number' selection from the 'Insert' menu allows you to insert

the page number into the document. The page number string is inserted at the current cursor position. This string is displayed using a gray color.

**Inserting Page Count** The 'Page Count' selection from the 'Insert' menu allows you to insert the total number of pages into the document. The page count string is inserted at the current cursor position. This string is displayed using a gray color.

**Show Page Border** When option is turned on, the editor displays the borders around the text on the screen. This option is available in the page mode only. The 'FittedView' option must be turned off.



## Page Header/Footer, Bookmark and Footnote Commands

The page header/footer functionality is available in the Page Mode only.

**Show Page Header/Footer** Normally, the editor does not show the header and footer for a page. You can use this option from the 'View' menu to display the page header and footer.

This option does not allow you to edit the text for the page header/footer. Every section in a document can have its own page header and footer. If a section does not have a page header/footer of its own, this option shows the header/footer from the preceding section for the pages in this section.

**Edit Page Header/Footer** The user can use this option to edit the text for the page header and footer. This option is available from the 'Edit' menu.

**Insert Footnote** This option allows you to insert a footnote at the current cursor location. The footnote is displayed at the bottom of the page

**Edit Footnote Text** This option displays the footnote text in-line with the regular text. It allows you to edit the footnote text. The modified footnote is displayed at the bottom of the page.

**Insert Bookmark** This dialog box is activated from the 'Insert' menu. It allows you to place a bookmark (new or existing) at the current text location. You can also position the cursor at a specified bookmark. It also allows you to delete an existing bookmark.



## Table Commands

The table menu is available in the Page mode or Print View modes only

(see Editing Modes). This menu contains the commands to create a new table or to edit table attributes.

#### **Insert Table**

Use this option to insert a new table in the document. This option prompts the user for the initial number of rows and columns in the table. The editor initially creates the cells of equal width. The user can, however, change the cell width by dragging the cell borders using the mouse.

In the Page Mode, the table cells are arranged by rows. In the Print View Mode, the table structure is not visible.

#### **Insert Table Row**

Use this option to insert a new row before the current table row. The new table row has the same number of columns as the current table row.

#### **Merge Table Cells**

Use this option to merge together the highlighted cells. The width of the resulting cells is equal to the sum of all merged cells. If the highlighted cells span more than one table row, this operation creates multiple merged cells each within its row.

#### **Split Table Cell**

Use this option to split the current table cell into two cells of equal width. The entire text of the original cell is assigned to the first cell. The second cell is created empty.

#### **Delete Table Cells**

Use this option to delete the selected cells from the table. A dialog box allows the user to select the cells for the deletion.

The dialog box has three options: cells, columns, and rows. The first option selects the current cell or all the cells in the highlighted block of text. The second option selects all the cells in the current column or the columns containing the cells in the highlighted block of text. The third option selects all the cells in the current row or the rows containing the cells in the highlighted block of text.

A table is automatically deleted when all its cells are deleted.

#### **Table Row Position**

Use this option to position the table or a selected table rows. A dialog box lets you position the table as left justified, centered, or right justified.

#### **Table Cell Border**

Use this option to create the borders around the selected cells. A dialog box allows the user to select the cells for this operation.

The dialog box has three options: cells, columns, and rows. The first option selects the current cell or all the cells in the highlighted block of text. The second option selects all the cells in the current column or the columns containing the cells in the highlighted block of text. The third option selects all the cells in the current row or the rows containing the cells in the highlighted block of text.

The user can specify the width of each border (top, bottom, left and right). The border width should be less than the cell text margin. The cell text margin is the distance from the left edge of the cell to the beginning of the text in the cell. The border width is specified in twips (1440 twips equal to one inch).

### Table Cell Shading

Use this option to shade the selected cells. A dialog box allows the user to select the cells for this operation.

The dialog box has three options: cells, columns, and rows. The first option selects the current cell or all the cells in the highlighted block of text. The second option selects all the cells in the current column or the columns containing the cells in the highlighted block of text. The third option selects all the cells in the current row or the rows containing the cells in the highlighted block of text.

The shading is specified in terms of the shading percentage. A value of 0 indicates a white background, where as the value of 100 indicates a black background. A value between 0 and 100 indicates the level of shading.

### Show Table Grid Lines

Use this option to enable or disable the display of the table grid lines. The table grid lines are for display purpose only, they are not drawn when printing to a printer



## Section and Columns

The editor allows you to divide a document into multiple sections. A multiple section document is useful when a) you need to vary the page margins from one page to another and b) you need to create multiple column text.

### Creating a New Section

To create a new section, select the 'Break' submenu option from the 'Edit' menu. A section break line (double solid line) is created before the current line. The new section begins at the text following the break line.

### Editing the Section Parameters

The following section parameters can be edited:

*Number of columns and column spacing.*

*Portrait or Landscape orientation.*

*Placement of the text on the next page.*

*Page Margins*

The first three parameters can be edited by selecting the 'Section Edit' option from the 'Edit' menu. The last parameter can be edited by selecting the 'Page Setup' option from the 'File' Menu.

### Deleting a section break line

To delete a section break line, simply position the cursor on the section break line and hit the <DEL> key.

### Multiple Column Editing

This option is available in the Print View and Page Modes only (See Editing Modes)

To create multiple columns for a section, select the 'Section Edit' option from the menu and specify the number of columns to create. You can also specify the space between the columns.



The text in the multiple column section wraps at the end of the column. When the text reaches the end of the page, or the end of a section, the new text is placed on the next column.

In the Print View mode, the multiple columns are not actually seen in the window. In the Page Mode, the columns are visible as they would be when the text is printed. Therefore, the Page Mode is useful when editing multiple column text.

### Column Break

Normally in a multiple column section, the text flows to the next column at the end of the current column. The column break option can be used to force the text to the next column before the current column is completely filled.

A column break can be inserted by selecting the option from the menu (Edit->Insert Break...). A column break is indicated by a line with a 'dot and dash' pattern. The text after the column break line is placed on the next column. To delete the column break line, simply position the cursor on the line and hit the <DEL> key



## Stylesheet and Table-of-contents

The editor supports the character and paragraph type stylesheet style items. The character stylesheet style constitutes a set of character formatting attributes and is applied to a character string. The paragraph stylesheet style constitutes not only a set of character formatting attributes, but also a set of paragraph formatting attributes. The paragraph style is applied to one or more paragraphs.

### Create and edit styles

A stylesheet style is created and modified using the 'Edit Style' menu option from the 'Edit' menu. This option displays a dialog box which allows you to choose between a character style or a paragraph style. You can select an existing style to modify from the list box or enter the name for the new style. Once you click the 'Ok' button, the recording of the stylesheet properties begins. You can use the ruler, toolbar, or the menu selections to modify the stylesheet items. The ruler, toolbar, and menu also reflect the currently selected properties for the stylesheet item. Please note that the paragraph properties are allowed only for the paragraph type of stylesheet item.

After you have selected the desired properties, terminate the stylesheet editing mode by either selecting the 'Edit Style' selection from the menu again or by clicking anywhere in the document. If the existing stylesheet item was modified, the document automatically reflects the updated stylesheet properties. If a new stylesheet item was created, your next step is to apply the style to the desired text by choosing the 'style' option from the 'Font' or the 'Paragraph' menu selection.

Apply character styles	The 'style' menu selection in the 'Font' menu allows you to apply a stylesheet style to the currently highlighted character string.
Apply paragraph styles	The 'style' menu selection in the 'Paragraph' menu allows you to apply a stylesheet style to the current paragraph. To apply a style to a range of paragraphs, highlight the paragraphs before selecting the 'style' menu option.
Table of Contents	<p>To insert a table of contents, first create the heading styles using the 'Edit Style' option from the 'Edit' menu. For example, if you wish to insert a three level deep table of contents, create heading styles 'heading 1', 'heading 2', and 'heading 3'. Then place the cursor at the heading lines and apply a suitable heading style using 'style' menu selection from the 'Paragraph' menu. The last step would be to position the cursor where you wish to insert the table of contents and select the 'Table of Contents' menu selection from the 'Insert' menu.</p> <p>The table-of-contents are automatically updated whenever repagination occurs.</p>



## Text/Picture Frame and Drawing Objects

A frame is a rectangular area on the page. A frame can contain both text and picture. The text outside the frame flows around the frame. A drawing object can be a text box, rectangle or a line. The drawing object overlays on top of the text

The 'Frame' or 'Drawing Object' option from the 'Insert' menu is used to embed a frame or a drawing object into the text. The new object is inserted at the current text position.

To insert text into the frame or a text box, click a mouse button inside the frame to select the frame. Now type the text at the cursor position.

To size a frame, click a mouse button inside the frame to select the frame. Now click the left mouse button on a sizing tab and move the mouse while the mouse button is depressed. Release the mouse when done. The text inside the frame is automatically rewrapped to adjust to the new width. If the new height of the frame is not enough to contain all text lines, the frame height is automatically adjusted to include all lines. If the frame contains only a picture, the picture size is automatically adjusted to fill the frame.

To move the frame, click a mouse button inside the frame to select the frame. Now move the mouse cursor just outside the frame until a plus shaped cursor appears. Click the left mouse button. While the mouse button is depressed, move the frame to the new location and release the mouse button.

To edit the base vertical position of the frame, select the 'Vertical Frame Base...' option from the edit->frame menu. The frame locked to the top of the page or the top of the margin retain their vertical position when the text is inserted before them.

To edit the border and the background of a drawing object, select the 'Edit Drawing Object' option from the edit->frame menu.

This option is available in the Page Mode only.



## View Options

This menu allows you to turn on and off the following viewing options:

<b>Page Mode</b>	In this mode, the editor displays one page at a time. This mode is available when the editor is called with both the Word Wrap and the Page Mode (or the PageView flag) flags turned on. This mode is most useful for the documents containing multiple columns, as the columns are displayed side by side. In addition, this mode provides all the features of the Print View mode.
<b>FittedView</b>	Special case of the page mode in which the text wraps to the window width and the soft page breaks are not displayed
<b>Ruler</b>	The ruler shows tab stops and paragraph indentation marks. The ruler can also be used to create or delete tab stops
<b>Tool Bar</b>	The tool bar provides a convenient method of selecting fonts, point sizes, character styles and paragraph properties. The tool bar also shows the current selection for font, point size and character styles.
<b>Show Status Ribbon</b>	The status ribbon displays the current page number, line number, column number and row number. It also indicates the current insert/overtyping mode.
<b>Show Hidden Text</b>	This option displays the text formatted with the hidden attribute (see Character Formatting Options) with a dotted underline. When this option is turned off, the hidden text is not visible.
<b>Show Paragraph Mark</b>	This option displays a symbol (an inverted 'P') at the end of each paragraph. This option may be useful when working with lines with many different heights
<b>Hyperlink Cursor</b>	This option is used to display the hyperlink cursor when the cursor is positioned on a hypertext phrase. The hyperlink cursor is an image of a hand with a finger pointing to the text.
<b>Zoom</b>	This feature allows you to compress or enlarge the display of the document text. The editor allows a zoom percentage between 25 and 200.



## Navigation Commands

### Jump

Use this function to position on a desired line number.

You can invoke this function by hitting the F10 function key (or select the option from the menu). The editor will then display a dialog box so that you can enter the line number to jump to.

See 'How to Scrolling Through the Text' section for other navigation functions.



## Search/Replace Commands

### Search a Text String

Use this function to locate a string of characters in the current file. The editor will search for the first instance of the given character string. To find the subsequent instances of the same character string, use *Search Forward* or *Search Backward* commands.

You can invoke this function by hitting the F5 function key (or select the option from the menu). The editor will display a dialog box where you enter the character string to locate. You can specify the search to be in the backward or the forward direction from the current cursor position or you can specify the search to take place from the beginning of the file. You can also force a non-casesensitive search, in which case the string is matched irrespective of the case of the letters in the string.

### Search Forward

Use this function to locate the next instance of a previously located string using the *Search Function*. If the Search Function is not yet invoked, this function will call the Search Function instead.

You can invoke this function by hitting the Control F Keys together (or select the option from the menu).

### Search Backward

Use this function to locate the previous instance of a previously located string using the *Search Function*. If the Search Function is not yet invoked, this function will call the Search Function instead.

You can invoke this function by hitting the Control Shift F Keys together (or select the option from the menu).

### Replace a Text String

Use this function to replace a character string with another character string.

You can invoke this function by hitting the F6 function key (or select the option from the menu). The editor will show a dialog box where you will enter the old and new character strings. You may also choose to conduct

the replace only within a selected part of the file. To choose such a block of text, the desired text must be highlighted before invoking the replace function.

The dialog box also offers you an option to force the editor to verify each replace.



## Highlighting Commands

### Highlight a Character Block

Use this function to highlight a block of characters.

Mouse: Position the mouse cursor on the first character of the block and depress the left button. While the left button is depressed, drag the mouse to the last character of the block and release the mouse.

Keystroke: Position the caret on the first character of the block and press the shift key. While the shift key is pressed, use the position keys to move the caret on the last character of the block and release the shift key. Normally, you can also use any position key in combination with the Shift key to create, expand, or shrink the text selection.

Normally, a function that utilizes a character block, also erases the highlighting. To explicitly erase the highlighting click a mouse button again or press any position key.

### Highlight a Line Block

Use this function to highlight a block of lines.

Mouse: Position the mouse cursor at any position on the first line of the block and depress the right button. While the right button is depressed, drag the mouse to the last line of the block and release the mouse.

Keystroke: Position the caret at any position on the first line of the block and hit the F8 function key. Use the Up and Down arrow keys to position the caret on the last line and hit F8 again.

Normally, a function that utilizes a line block, also erases the highlighting. To explicitly erase the highlighting click a mouse button again or press the F8 key again.

### Highlight a Word

Double click any mouse button on the desired word to highlight the word