USER MANUAL

T E DEVELOPER'S KIT

For

MS Windows

Version 4.5

SUB SYSTEMS, INC.

USER MANUAL

T E DEVELOPER'S KIT

For

WIN32

Version 4.5

SUB SYSTEMS, INC.

1990-95

TER EDITOR ROUTINE
(MS Windows Version)

Version 4.5
1990-95

SUB SYSTEMS, INC.
11 TIGER ROW
GEORGETOWN, MA 01833
(508) 352-9020

Software License Agreement

This license agreement allows the purchaser the right to use the software contained in this package. The purchaser has the right to modify and link the editor routine into their application. Such an application is free of distribution royalties with these conditions: the target application is 'larger' than the editor; 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; and the source code (or part) of the editor is not distributed in any form. Sub Systems, Inc. reserves the right to prosecute anybody found to be making illegal copies of this software or compiling the source alone with the express purpose of distributing it as a stand-alone editor thereafter.

Multideveloper Licenses: Each copy of the product is licensed to one developer. Multiple licenses are discounted as following:

2 to 4 Licenses:	20 percent discount
5 or more Licenses:	40 percent discount

Sub Systems, Inc. offers a 30 day money back guarantee with the product. The source code diskette envelope must remain sealed. Must call for an RMA number before returning the product.

Disclaimer

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.

TABLE OF CONTENTS

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
Native TER 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
Native TER format

File and buffer size 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 Feature:

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

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.

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.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_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 *include* 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.
TER1.H	The *include* 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.
TER.DLL	The DLL file (TER32.DLL for WIN32).
TER.LIB	Import library for the TER DLL (TER32.LIB for WIN32).

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	Compiled help file.

Visual Basic Interface Files :
(The VBX interface is not available for WIN32)

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.
TVB.VBX:	Contains the VBX interface for the editor. This file must be included by an application which uses the editor as a VBX.
TVB.C	Source for file for TVB.VBX file.
TVB.H	Contains the global variable definition for the TVB.C file.
TVB_PROT.H	Contains the function prototypes for the TVB.C file.
TVB.DEF	Linker definition file for TVB.VBX
TVB.RC	Resource file for TVB.VBX

TVB_*.BMP Bitmap files to display editor icon on the VB toolbar.

Visual Basic VBX Interface Demo:
(These files are included in the VBDEMO.ZIP file)

DMO_VBX.BAS: Contain the global variables for the internal use of the demo program.
DMO_VBX.FRM: Main form.
DMO_VBX.FRM1: Form that contains the editor control.
DMO_VBX.MAK: Make file for this demo program.

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 MAKE-BC	Compiles and links the TER routines using Borland 'C'.
MAKE-TVB.BAT MAKE-TVB	Compiles and links the editor VBX using Microsoft ‘C’.

Linking the Editor to Your Application

The package contains the TER.LIB (TER32.LIB for WIN32) file which must be linked to your application. Modify the link statement of your application to include the TER.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.

Creating an Editor Window

There are three methods of creating a TER editor window:

1. **By using the 'CreateTerWindow' function**
2. **By using the Dialog Editor and Dialog Manager (Custom Control)**
3. **By using the 'CreateWindow' SDK function (Custom Control).**

Here, we will discuss each one of these methods to help you choose one that is appropriate for your application.

The 'CreateTerWindow' function method

To create the editor window using this method, your application calls the 'CreateTerWindow' function (see Application Interface Functions) to create the editor window. The parameters are passed to this function using the 'arg_list' structure (see Parameter Structure).

This is the most flexible method of creating the editor window. Your application can control the window style and editing parameters. The initial data can be supplied in a global memory buffer or by a text file name.

This method can be used by any application that can interface with a Windows DLL, including the Visual Basic programs.

Refer to the source code of the DEMO.EXE program for an example of creating the TER window using the 'CreateTerWindow' function. Please also refer to the 'Analysis of the Demo Program' chapter for further information.

Dialog Editor and Dialog Manager Method

This method allows you to incorporate the TER editor control into a dialog template. To use this method, select the 'Custom Control' option from the Dialog Editor menu and specify 'TER.DLL' (TER32.DLL for WIN32) as a custom control. Now you will be able to select the custom control from the Dialog Editor tool bar and insert it into you dialog template. You can modify the control styles by double clicking on the control template. Please refer to the next section for the description of the TER control styles.

Your program that makes use of a dialog template containing the TER custom control must call the 'LoadTerControl' function to load the TER library.

Within the dialog procedure of your application, 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.

Refer to the source code of the CTRL.EXE program to an example of using the TER editor as a control.

The 'CreateWindow' SDK function method

This method is appropriate if your application must create the window itself by using the SDK 'CreateWindow' function. You can specify the window style by using any combination of the following styles with predefined styles such as WS_CHILD, WS_BORDER etc:

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_HSCROLL	Enable the horizontal scroll bar.
TER_VSCROLL	Enable the vertical scroll bar.
TER_SHOW_STATUS	Show the status ribbon
TER_SHOW_RULER	Show the 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.

Include the TER.H file to define these constants.

The window class should be specified as "TerClass" ("Ter32Class" for WIN32). 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.

Parameter Structure

The TER editor can be used as a Custom Control or directly as an editor DLL. When using it as a Custom Control, the editor window is created either by the dialog manager or by your application directly. When using the DLL as an editor, the editor window is created by calling the 'CreateTerWindow' function. This section describes the argument structure to be passed to create the TER window using the 'CreateTerWindow' function.

The arguments are passed to the function using the 'arg_list' structure. Some of the variables within the structure are used for the window creation itself. The TER routine simply passes these variables to the CreateWindow function of MS Windows. The structure also contains variables to customize the TER window environment. Finally, there are some variables to control the I/O from the TER routine.

struct arg_list {

int	x	Initial X position of the editing window. You may specify CW_USEDEFAULT here to use the default value.
int	y	Initial Y position of the editing window. You may specify CW_USEDEFAULT here to use the default value.
int	width	Window width. You may specify CW_USEDEFAULT here to use the default value.

int	height	Window Height. You may specify CW_USEDEFAULT here to use the default value.
int	LineLimit	Maximum number of lines allowed for text editing. Set this field to 0 to allow unlimited number of lines.
BOOL	WordWrap	If set to TRUE, this flag enables the word wrapping feature.
BOOL	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.
BOOL	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.
BOOL	ShowStatus	If set to TRUE, the editor will display a status line describing the current cursor location, and the insert/overtype indicator.
BOOL	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.
BOOL	ShowHorBar	If set to TRUE, the editor displays the horizontal scroll bar.
BOOL	ShowVerBar	If set to TRUE, the editor displays the vertical scroll bar.
BOOL	ruler	Set this variable to TRUE to show a ruler with tab stops.
BOOL	ToolBar	Set this variable to TRUE to show the tool bar.
BOOL	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 *CloseTer* function.
BOOL	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.
BOOL	ReadOnly	This is a browser mode which does not allow text modifications.
int	InitLine	The first line number to position on when the TER routine is called.
char	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'.
char	file[129]	If the InputType field is set to 'F', this field specifies the full path of the input file.
HANDLE	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 *GetTerBuffer* prior to closing the TER window.
long	BufferLen	When the InputType field is set to 'B', the BufferLen field specifies the length of the input buffer.
char	delim	When the InputType field is set to 'B', the 'delim' field specifies a line delimiter character. This field must be used to delimit the individual lines within the text buffer. You have an option of using a NULL, a carriage return or any other character of your choice. We recommend a NULL character as the delimiter.
int	SaveFormat	This field determines the format of the output file or buffer: SAVE_DEFAULT: Save in the format of the input file. SAVE_TER: Save in the native TER format. SAVE_RTF: Save in the Rich Text Format. SAVE_TEXT: Save in ASCII format. SAVE_TEXT_LINES: Save in ASCII format with line breaks.
HANDLE	hInst	Handle for the current instance of the application.
HWND	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).
HWND	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.
DWORD	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.
LPCATCHBUF	EndTer	The location to jump to if a fatal error is encountered by the editor.
BOOL	open	The editor set this variable to a TRUE value after initializing a TER window.
BOOL	modified	This variable is reserved for internal use.

}

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:

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: ClearTab, SetTab

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

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.

See Also: CreateTerWindow, GetTerBuffer

CreateTerWindow

Open a TER Window:

BOOL 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 types of parameters:

Initial window position and size
Input buffer (global) or input file name
Window style parameters.
Miscellaneous parameters.

Please refer to the previous chapter for a complete description of the arg_list structure.

Description: This function opens a TER window.

Return Value: If successful, the function returns a TRUE value. The handle to the newly opened window is 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.

See Also: CloseTer

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, ReadTerFile, SaveTerFile, CloseTer, HandleToStr

---

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

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.

See Also: SetTerLine, GetFontInfo.

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

GetFontInfo

Retrieve information about an editor font id.

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

HWND	hWnd;	// Handle of the window to be accessed
int	FontId;	// The editor font id to retrieve. This value must be between 0 and TotalFonts - 1.
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.
LPWORD	styles:	// This word 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~~
SUPSCR: Superscript
SUBSCR: Subscript
HIDDEN: Hidden Text
PROTECT: Protected Text

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

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: GetTerLine, GetTerFields

GetTerCursorPos

Retrieve the current cursor position.

BOOL GetTerCursorPos(hWnd, CurLine, CurCol)

HWND	hWnd;	// Handle of the window 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.
```

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

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	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 *SetTerFields* 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)
long	BeginLineW	First line number in the window. The editor ensures that CurRow is always equal to CurLine minus BeginLine.
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 text
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.
BOOL	ReclaimesourcesW	TRUE to reuse unused font and paragraph ids.
BOOL	ModifyProtectColorW	TRUE to show the protected text in a lighter shade.
BOOL	LinkDblClickW	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.
COLORREF	LinkColorW	The color of the hyperlink phrase.
char	text[MAX_WIDTH]	Text data for the current line (NULL terminated).
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.
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.

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
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	WinWidthW	Current window width in character columns
int	WinHeightW	Number of lines displayed in the window.
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         CurSectW            Current 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
};
```

Return Value: A TRUE value indicates success.

See Also: SetTerFields, GetTerFieldsAlt, GetFontInfo

GetTerFieldsAlt

Retrieve window variables for a specified line number:

BOOL GetTerFieldsAlt(hWnd,field,LineNo)

```
HWND                hWnd;       // Window handle
struct StrTerField *field;      // information buffer, see below
long                LineNo;     // line number to get the information for
```

Description: This function returns various operational parameters for the current TER window. This function is similar to the GetTerFields function except that it takes an additional argument (LineNo). The line depended information in the 'field' structure pertains to the specified line number.

Please refer to the GetTerFields function for the descriptions of the member variables for the StrTerFields structure.

Return Value: A TRUE value indicates success.

See Also: SetTerFields, GetTerFields, GetFontInfo

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

InsertRtfBuf

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

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

```
HWND            hWnd;   // Handle of the window to be accessed
BYTE huge *buffer;      // pointer to the buffer containing RTF data
long length;            // length of the RTF buffer
long line;              // line location (base 0) where the text will be inserted.
int column;             // column location (base 0) where the text will be inserted
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.

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

Return Value: This function returns TRUE if successful.

See Also: InsertTerText, SetTerBuffer

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. A new paragraph is initiated by inserting the paragraph break character (ASCII 182) before the CR/LF pair of the previous paragraph.

The buffer must be terminated using a NULL character.

Return Value: This function returns TRUE if successful.

See Also: InsertRtfBuffer, TerAppendText, SetTerBuffer, SetTerCursorPos

---

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.

Return Value: This function always returns TRUE.

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 handing 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

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

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

---

ParaRightIndent

Increment or decrement the right indents.

BOOL ParaRightIndent(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 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, ParaRightIndent, ParaNormal

ReadTerFile

Load a file into the editor window

BOOL ReadTerFile(hWnd, FileName)

HWND	hWnd;	// Handle of the window to be accessed
LPBYTE	FileName;	// Name of the disk file to read.

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)

HWND	hWnd;	// Handle of the window to be accessed
LPBYTE	FileName;	// Name of the disk file to save text.

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

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 the absolute term, 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 the absolute term, 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

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.

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: ClearTab, ClearAllTabs

SetTerBkColor

Set the background color for the text.

BOOL SetTerBkColor(hWnd,color,repaint)

HWND hWnd; // Window handle to access
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

SetTerBuffer

Set Window Text:

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

HWND	hWnd;	// Handle of the window
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 exising 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, ReadTerFile, SaveTerFile

---

SetTerCharStyle

Set or reset the character styles

BOOL SetTerCharStyle(hWnd, styles, OnOff, repaint)

HWND	hWnd;	// Handle of the window to be accessed
WORD	styles;	// Character style to set or reset:

BOLD: **Bold**
ULINE: Underline
ULINED: Double Underline

ITALIC:	*Italic*
STRIKE:	~~Strikethrough~~
SUPSCR:	Superscript
SUBSCR:	Subscript
HIDDEN:	Hidden Text
PROTECT:	Protected Text

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

SetTerColor

Set text color

BOOL SetTerColor(hWnd, COLORREF color, repaint)

HWND	hWnd;	// 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:

SelectSetText(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

SetTerCursorPos

Set the cursor position

BOOL SetTerCursorPos(hWnd, line, column, repaint)

HWND	hWnd;	// Handle of the window to be accessed
long	line;	// new line position of the cursor
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

SetTerDefaultFont

Set the default font and pointsize

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

```
HWND        hWnd;       // Handle of the window to be accessed
LPBYTE      typeface;   // typeface of the default font
int         PointSize;  //  Point size for the default font
WORD        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 a control or a VBX 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:

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

See Also: SetTerFont

SetTerFields

Set Window Variables:

BOOL SetTerFields(hWnd,field)

```
HWND            hWnd;       // Window handle
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: GetTerFields

SetTerFont

Set font typeface for the text

BOOL SetTerFont(hWnd, typeface, repaint)

HWND	hWnd;	// 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:

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

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

See Also: SelectTerText, SetTerPointSize, SetTerCharStyle, SetTerDefaultFont

---

SetTerLine

Set the text and font ids for a line

BOOL SetTerLine(hWnd, LineNo, text, font)

HWND	hWnd;	// 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 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.

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

See Also: GetTerLine, GetFontInfo

SetTerParaFmt

Set paragraph styles

BOOL SetTerParaFmt(hWnd, styles, 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
		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:

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

See Also: ParaNormal

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).
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:

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

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

See Also: SelectTerText, SetTerFont, SetTerCharStyles

StrToHandle

Convert a Visual Basic string to a global memory handle.

HANDLE StrToHandle(string, length)

```
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

TerAbsToRowCol

Convert the given character position to the row and column position

void TerAbsToRowCol(hWnd, abs, row, col)

```
HWND hWnd;    // the window handle to access
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

TerAppendText

Append specified text at the end of the buffer.

BOOL TerAppendText(hWnd, text, FontId, ParaId, 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.
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 function after making a series of calls to this function.

Return Value: This function returns TRUE when successful.

See Also: TerInsertText, TerCreateFont, TerCreateParaId

TerColBreak

Create a column break.

BOOL TerColBreak(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 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.

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

TerCreateFont

Create a font id.

BOOL TerCreateFont(hWnd, ReuseId, shared, typeface, pointsize, style, color, BkColor, FieldId, AuxId)
HWND hWnd; // The handle of the window to be accessed
int ReuseId; // 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 ReuseId field.
LPBYTE typeface; // Typeface of the new font
int PointSize; // Point size of the new font
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 0xFFFFFF 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.

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 'ReuseId' argument, otherwise set the 'ReuseId' 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: TerCreateParaId, TerAppendText

TerCreateParaId

Create a paragraph id.

BOOL TerCreateParaId(hWnd, ReuseId, shared, LeftIndent, RightIndent, FirstIndent, TabId, CellId, AuxId, Shading, FrameId, SpaceBefore, SpaceAfter, SpaceBetween, flags);

HWND hWnd; // The handle of the window to be accessed
int ReuseId; // 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 ReuseId field.
int LeftIndent; // Left indentation (specified in twips). Use 0 for default.
int RigthIndent; // Right indentation (specified in twips). Use 0 for default.
int FirstIndent; // Indentation for the first line (specified in twips). Use 0 for default.
int TabId; // Tab id. Use 0 for default.
int CellId; // Cell id. Use 0 for default. Must use the current cell id when using the resulting paragraph id to insert a text inside a table.
int AuxId; // 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).
int FrameId; // Frame id. Must use the current frame id when using the resulting paragraph id to insert the text inside a text frame.
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.

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 'ReuseId' argument, otherwise set the 'ReuseId' 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

TerCreateTable

Create a text table.

BOOL TerCreateTable(hWnd, row, col, 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 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 on the first line after the table.

Return Value: This function returns TRUE if successful.

See Also: None.

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.

TerGetAppMemory

Retrieve the application memory block.

LPVOID TerGetAppMemory(hWnd)

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

Return Value: This function returns the pointer to the application memory block allocated by the 'TerSetAppMemory' function.

See Also: TerSetAppMemory

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 (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.

Return Value: This function returns TRUE when successful.

See Also: TerSetPictInfo, TerPastePicture, TerPictureFromFile

---

TerGetTextColor

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

BOOL TerGetTextColor(hWnd, FontId, TextColor, TextBkColor)

HWND hWnd; // The windows handle to access

int FontId; // The font id to extract the colors for
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

---

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.

See Also: None

TerEnableSpeedKey

Enable or disable a speed key.

BOOL TerEnableSpeedKey(hWnd, CommandId, enable)

HWND hWnd; // Editor window to access
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.

Return Value: The function returns the previous status of the speed key..

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.

TerInsertParaFrame

Insert a paragraph frame.

int TerInsertParaFrame(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.

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 return 0.

See Also: TerMoveParaFrame

TerInsertText

Insert text at the cursor position.

BOOL TerInsertText(hWnd, text, FontId, ParaId, 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.
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.

See Also: TerAppendText, InsertTerText, TerCreateFont, TerCreateParaId

---

TerLocateStyle

Locate text with the given character style.

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

HWND	hWnd;	// The handle of the window to be accessed
WORD	style;	// Style bits: BOLD: **Bold** ULINE: Underline ULINED: Double Underline ITALIC:*Italic* STRIKE: ~~Strikethrough~~ SUPSCR: Superscript SUBSCR: Subscript HIDDEN: Hidden Text PROTECT: Protected Text 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: SetTerCharStyle

TerMenuEnable

Get menu item 'enable' status

UINT TerMenuEnable(hWnd, MenuId)

HWND hWnd; // Editor window to access
int MenuId; // menu id can be any of the constants defined in the ter_dlg.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 function returns one of the following constants:

MF_ENABLED = Enable menu item
MF_GRAYED = Gray out the menu item

Example:

```
#include “ter_dlg.h”
int status;

status = TerMenuEnable(hWnd,ID_CUT);

// The ‘status’ variable will be equal to MF_ENABLED it a text block is highlighted to be
copied to the clipboard.
```

See Also: TerMenuSelect

TerMenuSelect

Get menu item selection status

```
UINT  TerMenuSelect(hWnd, MenuId)

HWND  hWnd;   // Editor window to access
int  MenuId;       // menu id can be any of the constants defined in the ter_dlg.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 function returns one of the following constants:

MF_CHECKED = Check the menu item
MF_UNCHECKED = Uncheck the menu item

Example:

```
#include “ter_dlg.h”
int status;

status = TerMenuSelect(hWnd,ID_BOLD);

// The ‘status’ variable will be equal to MF_CHECKED if the current character has the bold
style.
```

See Also: TerMenuEnable

TerMergeFields

Replace field names with field data strings.

```
BOOL TerMergeFields(hWnd,names,data,repaint)

HWND        hWnd;       // The 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

TerMergePrint

Print a document without invoking the editing session.

BOOL TerMergePrint(prt)

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;
}
```

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 carraige-return (ASCII 13), or any other character of your choice. This field is not used when the buffer contains RTF data.

hDC: The device context of the printer or the window to print the text. The device context MUST be mapped to use the pixel units. If this field is set to NULL, the editor opens a new device context to the current printer.

rect: This field contains a far pointer to a printing rectangle. The rectangle co-ordinates MUST be specified in the millimeter (mm) units.

StartPos: This field specifies the character position to begin the 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.

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

See Also: TerPrint, TerMergeFields

TerMoveParaFrame

Move or resize the current paragraph frame.

BOOL TerMoveParaFrame(hWnd, ParaFID, x, y, width, height)

```
HWND hWnd;    // The 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

TerPageBreak

Create a hard page break.

BOOL TerPageBreak(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 page. The page break is created before the current line.

A page break is indicated by a solid line.

Return Value: This function returns TRUE if successful.

See Also: TerColBreak, TerSectBreak

TerPastePicture

Paste a picture from the buffer or clipboard.

int TerPastePicture(hWnd, format, hData, align, insert)

HWND hWnd; // Window handle to access
UINT format; // Picture format:
CF_DIB: Device independent bitmap format
CF_BITMAP: Bitmap format
CF_METAFILEPICT: Metafile picture format
HGLOBAL hData; // picture data
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: TerPictureFromFile, TerInsertObjectId, TerGetPictInfo, TerSetPictInfo

---

TerPictureFromFile

Paste a picture from a disk bitmap file.

BOOL TerPictureFromFile(hWnd, FileName, align, insert)

```
HWND    hWnd;       // Window handle to access
LPBYTE  FileName;   // Name of the disk bitmap 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, TerInsertObjectId

TerPrint

Print the current document.

BOOL TerPrint(hWnd,dialog)

```
HWND        hWnd;       // The handle of the window to be accessed
BOOL        dialog;     // Set to TRUE to show a dialog box to the user.
```

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. When the 'dialog' parameter is FALSE, the editor prints the selected text only if a text block has been highlighted.

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 successful.

See Also: TerMergePrint

TerPrintPreview

Preview the specified page in the current document.

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

```
HWND        hWnd;       // The handle of the text 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.
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

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:

TerRowColToAbs

Convert the given line/row position to the character position.

long TerRowColToAbs(hWnd, row, col)

HWND hWnd;	// Window handle to access	
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

TerSearchReplace

Search, replace or retrieve text.

BOOL TerSearchReplace(hWnd, search, replace, flags, StartPos, EndPos, BufSize)

```
HWND  hWnd;          // Window handle to access
LPBYTE search;       // search text string
LPBYTE replace;      // replace text string
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: 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.

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, its absolute character position is returned via the 'EndPos' long pointer.

Replace Mode: To initiate this mode, set the SRCH_REPLACE flag in the 'flags' parameter. This function replaces the text between the 'StartPos' and 'EndPos' 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.

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

See Also:

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.

A section break is indicated by a double solid line.

Return Value: This function returns TRUE if successful.

See Also: TerPageBreak, TerColBreak

TerSetAppMemory

Reserve a memory block for your applications use.

```
BOOL TerSetAppMemory(hWnd, size)
HWND hWnd;    // The handle of the window to be accessed
long size;    // size of the block
```

Description: The editor allocates a block of the specified size. This block is automatically freed by the editor on closing. Your application should retrieve the pointer to this block by using the 'TerGetAppMemory' function.

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

See Also: TerGetAppMemory

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.

TerSetMargin

Set the margin values for the current section in the document.

```
BOOL TerSetMargin(hWnd,left,right,top,bottom,repaint)

HWND hWnd;    // The handle of the window to be accessed
int left;     // Left margin value in twip units
int right;    // right margin value in twip units
int top;      // top margin value in twip units
int bottom;   // bottom margin value in twip units
BOOL repaint; // set to TRUE to repaint the screen after this operation
```

Description: This function can be used by your application to modify the current values for the left, right, top and bottom margins for the current section in the document. Specify a -1 value for the margin that you wish to leave unchanged.

Return Value: This function returns TRUE if successful.

See Also: None

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: None

TerSetOutputFormat

Set the format of the output data.

BOOL TerSetOutputFormat(hWnd,format)

HWND hWnd; // The handle of the window to be accessed
BOOL format; // The output format can be set to one of the following:

SAVE_TER: Save in the TER native format
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_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.

See Also: None

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.

See Also: None

TerSetParaSpacing

Set the spacing parameters for the current paragraph.

BOOL TerSetParaSpacing(hWnd, SpaceBefore, SpaceAfter, SpaceBetween, refresh)

```
HWND hWnd;     // The handle of the window to be accessed
int SpaceBefore;  // Space before the first line of the paragraph in twips
int SpaceAfter;   // Space after the last line of the paragraph in twips
int SpaceBetween;// Minimum space between the lines of the paragraph.
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: None

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, TerPastePicture, TerPictureFromFile

---

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: None

How To Scroll Through The Text

Keyboard:

a. Use *Up, Down, Left* and *Right* arrow keys to scroll up or down a line, or left or right one character.

b. Hit the *Home* key to position at the beginning of the current line.

c. Hit the *End* key to position at the end of the current line.

d. Hit *Ctrl-PgUp* to position at the beginning of a file.

e. Hit *Ctrl-PgDn* to position at the end of a file.

f. Hit *PgUp* to display the previous page.

g. Hit *PgDn* to display the next page.

h. Hit *Ctrl - Left* arrow key to position on the next word.

i. Hit *Ctrl - Right* arrow key to position on the previous word.

j. 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.

k. Hit *Ctrl - Down* arrow key to position at the first column of the next line.

l. 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.

Text Editor Commands

This chapter describes the editor commands by menu groups.

File and Print Commands

Save File

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).

If the I/O is conducted through a buffer rather than a disk file, the editor creates a new buffer with the updated text.

You can invoke this function by hitting the F3 function key (or select the option from the menu).

Save File As..

This selection is similar to *Save File*. In addition, it allows you to specify a new file name for saving the text.

This option is not available when the I/O is conducted through a buffer rather than a disk file.

You can invoke this function by hitting the Shift F3 function keys together (or select the option from the menu).

Exit

Use this function to exit from the editor session. If the current file is modified, you will have an option to save the modifications.

You can invoke this function by hitting the Ctrl F3 function keys together (or select the option from the menu).

Print

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:

Line Block
Character Block

The print function will print on a default printer selected from the Windows' control panel. You can alter the *printer setup* or *Page Layout* prior to invoking the print option.

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.

Page Layout

Use this option before selecting the *Print* option to specify the page layout. You can specify margin (left, right, top and bottom) in inches.

You can invoke this function by hitting the Ctrl F4 function keys together (or select the option from the menu).

Printer Setup

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.

You can invoke this function by hitting the Shift F4 function keys together (or select the option from the menu).

Print Preview

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.

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.

Line Edit Commands

Insert After Current Line

In the *text mode* this function creates a blank line after the current line. Hit the F9 function key to insert a line after the current line.

Insert Before Current Line

In the *text mode* this function creates a blank line before the current line. Hit the Ctrl F5 keys together to insert a line before the current line.

Delete Line

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.

Join Lines

In the *text mode* this function joins the next line at the end of the current line. Hit the Alt J keys together to invoke this function.

Split Line

In the *text mode* this function splits the current line at the current cursor position. Hit the Alt S keys together to invoke this function.

Block Edit Commands

Copy a Line Block

Use this command to **copy** a highlighted block of text lines from one location to another. This command provides a short alternative to using clipboard copy/paste functions.

Highlight the lines of text to be copied, move the caret to the target 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.

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.

Formatted Text: 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.

Unformatted Text: This is another text format. This option pastes the text without retaining the formatting information.

Picture Format: 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.

Device Independent and regular bitmap formats: 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

Import Picture From a Disk File

Use this command to read in a picture bitmap from a disk file at the current caret location.

You can invoke this function by hitting the Alt F8 function keys together (or select the option from the 'Insert' menu).

Edit Picture

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.

Insert Object

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.

Please note that you can also use the Paste Special function to import the OLE objects, provided the object is available in the clipboard.

Drag/Drop Function

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 indicate the inserted object. You can edit this object by double clicked at the icon.

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.

Character Formatting Commands

Character Styles

The following character style commands are available:

Command	Keystroke	Sample
Normal	Alt 0	ABC..123
Bold Formatting	Alt 1	**ABC..123**
Underlining	Alt 2	ABC..123
Italic	Alt 3	*ABC..123*
Superscript	Alt 4	ABC..123
Subscript	Alt 5	ABC..123
Strike	Alt 6	~~ABC..123~~

Character style options allows you to apply one or more style formats to the current character or to all characters in a highlighted block of text.

To apply a format to the current character, simply hit the appropriate keystroke (Alt 1 through Alt 6) (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 (Alt 1 through Alt 6), or select the option from the menu.

When you type in on the keyboard, the new characters automatically assume all the formatting characteristics of the preceding character.

TER allows multiple formats for a character. To apply more that one format, repeat the procedure described in the previous paragraphs.

To reset all character formats, highlight the characters and select the 'Normal' option from the menu, or hit the Alt 0 keystroke.

Fonts:

Use this option to change the font typeface and point size of the current character or of all characters in a highlighted block of text.

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).

Paragraph Justification

Use this selection to justify the text on both left and right margins.

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.

Paragraph Double Spacing

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.

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).

Paragraph Indentation (Left)

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.

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).

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.

Paragraph Indentation (Right)

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.

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).

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.

Paragraph Hanging Indentation

This option is similar to paragraph left indentation, except that the indentation is not applied to the first line of the paragraph.

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).

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.

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 mininum 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.

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.

Normally, a tab command is applicable to every line of the current paragraph. However, if you hightlight 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.

Page Header/Footer 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.

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 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.
Placement of the text on the next page.
Page Margins

The first two 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.

Text/Picture Frame

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.

The 'Frame' option from the 'Insert' menu is used to embed a frame into text. The new frame is inserted at the current text position.

To insert text into the frame, 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.

This option is available in the Page Mode only.

View Options

This menu allows you to turn on and off the following viewing options:

Page Mode

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.

Ruler

The ruler shows tab stops and paragraph indentation marks. The ruler can also be used to create or delete tab stops.

Tool Bar

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.

Show Status Ribbon

The status ribbon displays the current page number, line number, column number and row number. It also indicates the current insert/overtype mode.

Show Hidden Text

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.

Show Paragraph Mark

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.

Hyperlink Cursor

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.

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. This command supports these highlighted blocks:

Line Block
Character Block

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.

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 Developer's Kit employs the following major data structures:

Line Handle Array: (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.

Line Length Array: (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.

Format Data Array: (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.

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.

Font Structure Array: (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.

Paragraph Array: (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.

Paragraph Table: (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.

Tab Table: (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.

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 choosing by editing the 'HYPER_TEXT' global constant.

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 (300), 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.

Mail/Merge Support

The mail/merge support of the editor 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:

1. Input the field name using the keyboard.
2. Highlight the text for the field name.
3. 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.

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:

1. 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
   ```

2. 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
   ```

3. 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
   ```

4. 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.

5. 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 */
           ...
           ...
   }
   ```

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.

TER File Format

A TER disk file or the I/O buffer can have up to 5 sections. This chapter describes each component.

Text Lines:

This component stores the text lines. Each line is terminated by a pair of CR and Newline characters. In the document mode, the last line of each paragraph ends with a paragraph delimiter (ASCII 182), followed by a pair of CR and newline characters.

A TER IO buffer has a slightly different format for this section, in that the <CR> <NL> pair is replaced by a single delimiter. The value of this delimiter is specified within the parameter structure by the calling application.

Character Formatting Information

The character formatting information stores a font identifier for each character in the document. A special signature string at the end of the text lines identifies the beginning of the character formatting information. The contents of this string is as following:

~`!@#$%^&*()-TeCfmt

This string is terminated by a pair of CR and Newline characters.

The character formatting information is stored in the compressed format for the text lines which use a uniform font for every character in the line. The compressed format consists of two bytes. The first byte is equal to 1, and the second byte is equal to the value of the uniform font (index into the font table). The lines that have varying formatting information need the full length to store the information. For example if a text line consisting of 30 characters has varying formatting information, 31 bytes will be stored for this line. The first byte will hold the length of the line (30), and the following 30 bytes will store the font value (index into the font table) of each character of the line.

Font Table

This section stores the font table. It begins with a signature byte (0xBF) followed by a count byte to store the number of font records in the table.

The count byte is followed by a font record for each font in the table. Refer to the 'StrFontIO' structure in the TER1.H file for the description of the font record. If a font record actually represents a picture, the font record is followed by the picture bitmap or Windows metafile data.

Picture Height	WORD
Picture Width	WORD
Image Size	DWORD
Info Size	DWORD
Image	string of size Image Size
Info	string of size Info Size

Paragraph Formatting Information

The paragraph formatting information begins with a signature byte of the value 0xBF. It is followed by one byte for each text line. Each byte represents an index into the paragraph table which is stored immediately after the paragraph formatting information.

Paragraph Table

The paragraph table stores one paragraph record for each paragraph id used in the document. Refer to the 'StrParaIO' structure in the TER1.H file for the description of the paragraph record.

Tab Table

This block begins with a one byte signature (value 0xBF) followed by an integer that holds the number of entries in the tab table. This integer value is followed by the tab table entries. Each entry is stored as a structure of type 'StrTab' (see the TER1.H file).

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 *GetTerBuffer* function.

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.

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.

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)

1. 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 10. This variable is used by the demo program so that no more than 'MAX_WINDS' windows are open at a time.

2. 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'.

3. The CurWnd variable is used internally by the demo program to store the current window number.

4. 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.

5. 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.

1. 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.

2. 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.

Visual Basic Interface

Your Visual Basic application can use the TER editor either as a VBX or a DLL. Please note that the Visual Basic Interface is not available for WIN32 application).

Using the Editor as a VBX

The VBX method is preferable when a tight integration with the editor is desired.

To use the editor as a VBX, copy the **TVB.VBX** and **TER.DLL** files to your application's directory. Now include the TVB.VBX and TER.BAS files in your project. 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 VBX interface, refer to the **DMO_VBX** demo project included in the package.

The data in the edit control can be manipulated by using the properties and the API functions. The 'properties' can be used to manipulate the basic text attributes, whereas, the 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 the API functions. In this section, we will describe the control properties.

Properties:

This VBX includes a collection of design-time and run-time properties.

Design-time properties:

Word Wrap: Turn on word wrap.

Print View: 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).

Page Mode: Edit document one page at a time. This mode is useful when editing the documents containing multiple columns.

Vertical Scroll: Enable the vertical scroll bar.

Horizontal Scroll: Enable the horizontal scroll bar.

Show Status Bar: Show status bar indicating the cursor position.

Show Ruler: Show the ruler with tab stops and indentation indicators.

Show Toolbar: Enables tool bar.

Border Margin: Reserves a think blank area around the text box.

Read Only: The editor displays the text, but modifications are not allowed.

Output Rtf: 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 VBX control. This property allows you to access the menu commands indirectly.

Usage: control.Command=Command_id

The command id can be one of the following:

ID_BK_COLOR	Set background color
ID_BLOCK_COPY	Copy a highlighted block
ID_BLOCK_MOVE	Move a highlighted block
ID_BOLD_ON	Set bold on
ID_CENTER	Center the paragraph
ID_CHAR_NORMAL	Reset the character styles
ID_COL_BREAK	Insert a column break
ID_COLOR	Choose colors
ID_COPY	Copy text to clipboard
ID_CUT	Cut text to clipboard
ID_DEL	Delete the current character
ID_DEL_LINE	Delete the current line
ID_DOUBLE_SPACE	Double space the paragraph lines
ID_DOWN	Arrow down
ID_EDIT_PICT	Edit the picture size
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_HANGING_INDENT	Create or increment hanging indentation
ID_HELP	Show the help window
ID_HIDDEN_ON	Set the hidden attribute on
ID_HILIGHT_LINE	Highlight the line block
ID_INS_AFT	Insert a line after the current line
ID_INS_BEF	Insert a line before the current line
ID_INSERT	Toggle the insert mode
ID_INSERT_OBJECT	Insert an OLE object
ID_INSERT_PAGE_NUMBER	Insert page number string
ID_INSERT_PARA_FRAME	Insert a text frame
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_LEFT	Arrow left
ID_LEFT_INDENT	Create or increment left indentation
ID_LINE_BEGIN	Position at the beginning of the line
ID_LINE_END	Position at the end of the line
ID_NEXT_WORD	Position at the next word
ID_PAGE_BREAK	Insert a page break
ID_PAGE_MODE	Toggle page mode
ID_PAGE_OPTIONS	Set the page options
ID_PARA_BORDER	Create paragraph borders
ID_PARA_NORMAL	Reset the paragraph attributes
ID_PASTE	Paste text from the clipboard
ID_PASTE_SPEC	Paste special clipboard formats
ID_PGDN	Page down
ID_PGUP	Page up

ID_PICT_FROM_FILE	Import a bitmap from a disk file
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_ON	Set character protection on
ID_PROTECTION_LOCK	Toggle the protection lock for the document
ID_QUIT	Quit the editing session
ID_REPAGINATE	Repaginate now
ID_REPLACE	Replace text
ID_RIGHT	Arrow right
ID_RIGHT_INDENT	Create or increment right indentation
ID_RIGHT_JUSTIFY	Right justify the paragraph
ID_JUSTIFY	Justify paragraph on both margins
ID_RULER	Toggle ruler display
ID_SAVE	Save the current file
ID_SAVEAS	Save data to another file name
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_SELECT_ALL	Select the entire document
ID_SHOW_HIDDEN	Show the hidden characters
ID_SHOW_HYPERLINK_CUROSR	Show hyperlink cursor
ID_SHOW_PARA_MARK	Toggle the paragraph marker character display
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_CLEAR	Clear a tab stop position for a paragraph
ID_TAB_CLEAR_ALL	Clear all tab stop positions for a paragraph
ID_TABLE_CELL_BORDER	Edit table cell borders
ID_TABLE_CELL_SHADE	Set table cell shading
ID_TABLE_INSERT	Insert new table
ID_TABLE_DEL_CELLS	Delete table cells
ID_TABLE_INSERT_ROW	Insert new table row
ID_TABLE_MERGE_CELLS	Merge table cells
ID_TABLE_ROW_POS	Position table rows
ID_TABLE_SHOW_GRID	Show table grid lines
ID_TABLE_SPLIT_CELL	Split current table cell
ID_TOOL_BAR	Toggle the toolbar display
ID_ULINE_ON	Set underline on
ID_ULINED_ON	Set double underline attribute on
ID_UNDO	Undo previous edit
ID_UP	Arrow up

Example: control.Command=ID_FONTS

The above statement will invoke the font selection dialog box.

Data: (Not applicable to Delphi, see: Borland Delphi Interface)

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

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 propery 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.

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

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:

SAVE_TER:	Save in the control’s native format.
SAVE_TEXT:	Save in the text format. The formatting information is lost in the text format.
SAVE_TEXT_LINES:	Save in the text format with line breaks. The formatting information is lost in this format.
SAVE_RTF:	Save in the Rich Text Format.
SAVE_DEFAULT:	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

The following constants are available:

BOLD:	Bold
ULINE:	Underline
ULINED:	Double underline

ITALIC: Italic
STRIKE: Strikethrough
SUPSCR: Superscript
SUBSCR: Subscript
HIDDEN: Hidden text

Example: control.SetCharStyle=BOLD or ULINE.

See Also: ResetCharStyle

VBX Events:

Standard events: The VBX 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.

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).

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 independedly 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 most often 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.

GetTerBuffer

GetTerBuffer(hWnd as integer, BufferLen as long) as integer

Description: Returns the handle of a global memory block containing the data for the specified TER window . Please note that the contents of a global memory handle is not directly accessible by a Visual Basic program. However, you can use the 'HandleToStr' function to convert the global handle to the Visual Basic string data.

The length of the buffer is returned using the second argument.

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 he 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.

SetTerBuffer

SetTerBuffer(hWnd as integer, hMem as integer, BufferLen as long, NewTitle as string) as integer

Description: Use this function to set new text to an exiting TER window. The first argument specifies the handle to the TER window. The second argument specifies the global memory block containing the new text data. Please note that a Visual Basic program can not create a global memory block. However, you can use the 'StrToHandle' function to convert a Visual Basic string data to a global handle. The third argument specifies the length of the global memory block. The last argument specifies the new title for the TER window.

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 'StrTerFields' 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.

Borland Delphi Interface

The editor VBX (TVB.VBX) conforms to version 1.0, and is therefore compatible with Delphi. Please refer to the chapter on *Visual Basic Interface* for a complete description of the VBX. Every VBX 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 function, 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 C++ Interface

A Microsoft Visual C++ application can interface with the editor using one of the following three methods:

Use the editor control as a VBX
Use CTer class interface
Use CTerView class interface

Editor Control as a VBX:

This method is not available for WIN32 applications. This method allows you to incorporate the editor control within your dialog box using the dialog editor. Both the TVB.VBX and TER.DLL files must be available in the current directory or a directory included in the path statement.

To insert the editor icon in the dialog editor tool bar, select the 'Install Controls' option from the file menu and select the TVB.VBX file. You can then select the 'TE' icon to insert an editor control within your dialog box.

To set or retrieve data from the control, use the ::SetTerBuffer and ::GetTerBuffer functions. You can use any of the functions described in the 'Application Interface Function' chapter for manipulating the data in the edit control.

Please refer to the Microsoft Class Library User's Guide for further description on using a custom control with a Visual C++ application.

CTer Class Interface:

The CTer class interface is provide 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(ControlId, 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 is 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 ▭ // 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:

WS_CHILD	Child window
WS_VISIBLE	Window visible
WS_BORDER	Show window border
TER_WORD_WRAP	Turn on word wrapping
TER_VSCROLL	Show the vertical scroll bar
TER_BORDER_MARGIN	Leave a narrow area between the text rectangle and the window rectangle.

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 constant 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 is a menu item should be checked.

BOOL MenuSelect(MenuItem);

int MenuItem: // menu item number to test. The MenuItem can be one of the constant 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 is 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 is 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 after 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	Copy the selected block to the clipboard
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/overtype mode
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_REPAGINATE	Repaginate now
TID_SECT_BREAK	Insert a section break
TID_SECT_OPTIONS	Edit section options
TID_PAGE_MODE	Toggle page mode
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	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_INESRT_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 constant 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++ environment, follow these steps to create a Visual C++ project:

Files: TER*.C, TER.DEF and TER.RC

Executable Type: Windows DLL

Alignment (Compiler Option): 1 Byte

Remaining parameters should be left at their default values.