

# TX Text Control

DLL Reference Manual

Version 7.0

TX Text Control 7.0

Information in this document is subject to change without notice and does not represent a commitment on the part of The Imaging Source Europe GmbH. The software described in this document is furnished under a license agreement. The software may only be used or copied in accordance with the terms of this agreement.

Copyright 1991-1999 The Imaging Source Europe GmbH. All rights reserved.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

# Contents

What's New in Version 7.0 since Version 6.0	6
New Features	6
Changes and Extensions	6
New and Extended Messages and Data Structures	7
What's New in Version 7.0 since Version 5.2	9
New Features	9
Changes and Extensions	10
New and Extended Messages	10
1. Introduction	12
1.1 What is TX Text Control	12
1.2 The Library Files	12
1.3 The Source Files	13
1.4 Creating and Programming a Text Control Window	14
1.5 Text Formatting and Views	15
1.6 Headers and Footers	19
1.7 Tables	21
1.8 Marked Text Fields	23
1.9 Image Processing	28
1.10 Integrating OLE Objects	28
1.11 Integrating External Windows	32
1.12 Using Chains of linked Windows	33
1.13 Using Metafiles	34
1.14 ANSI and Unicode	35
1.15 Resources	38
2. Mouse and Keyboard Interface	40
2.1 Mouse Assignment	40
2.2 Keyboard Assignment	40
3. Function Directory	42
4. Message Overview	44
4.1 Processed Windows Messages	44
4.2 Font Messages	46
4.3 Paragraph Messages	46
4.4 Text Manipulation Messages	47

4.5	Messages to Adapt Formatted Text to a new Output Device	48
4.6	Print Messages	48
4.7	File IO Messages	49
4.8	Information Messages	49
4.9	Messages to Process Images	51
4.10	Messages to Handle Chains of Linked Windows	51
4.11	Messages to Handle Marked Text Fields	51
4.12	Messages to Perform Undo Operations	52
4.13	Messages to Handle Integrated Objects	52
4.14	Messages to Handle the Internal Scroll Interface	53
4.15	Messages to Handle Find and Replace Features	53
4.16	Table Messages	53
4.17	Messages to Handle Headers and Footers	54
4.18	Additional Features	55
4.19	Notification Messages	56
4.20	Obsolete Messages	58
5.	Message Directory	60
6.	Notification Messages	174
7.	Data Structures	185
8.	Obsolete Messages and Functions	207
Appendix A		216
	The TX Text Control Text Format	216
	Format Example	224
Appendix B		227
	Group Codes	227
	Error Code Table (TX kernel module)	228
	Error Code Table (OLE module)	240
Appendix C		241
	Development of a Text Filter	241
Appendix D		249
	Status Bar Control	249
	Functions	250
	Messages	253
Appendix E		255

Button Bar Control	255
Functions	256
Messages	259
Appendix F	262
Ruler Control	262
Functions	263
Messages	265
Appendix G	267
Debugging Functions	267
Error Code Table (Expansion for Debugging functions)	268
Appendix H	271
Object Window Messages	271

## IC Image Control 275

1. Introduction	283
1.1 What is IC	283
1.2 The Library Files (16 bit version)	283
1.3 The Library Files (32 bit version)	283
1.4 The Source Files	284
1.5 How to use the Image Control	284
1.6 How to use Images with a Text Control	285
1.7 Filter Selection	285
1.8 ANSI and Unicode	286
2. Function Directory	287
3. Message Overview	289
4. Message Directory	291
5. Notification Messages	301
Appendix A	303
Error Code Table (Image Control module)	303
Error Code Table (graphics import filter)	304
Appendix B	306
The IC Image Control File Format	306

# *What's New in Version 7.0 since Version 6.0*

## *New Features*

Text Control supports headers and footers. Several messages have been added for this feature. See chapter 1.6 "*Headers and Footers*" for more information about these messages and how headers and footers can be used and programmed.

Text Control supports several special types of marked text fields like source and destination fields for hypertext links or fields that display the current page number. The new messages TX\_FIELD\_GETTYPE and TX\_FIELD\_SETTYPE set or retrieve the type for a marked text field including additional data depending on this type.

Text Control offers an additional page view that centers the document in the window's client area and displays three-dimensional pages with shadows. This mode can be set with the TF\_EXTPAGEVIEW setting of the TX\_SETTEXTAREA message.

The new message TX\_SETLINEANDCOL can be used to specify a new text input position through a page, line and column number.

The new message TX\_RESETCONTENTS can be used to delete the complete contents of a Text Control.

The new message TX\_INPUTPOSFROMPOINT can be used to calculate a text position belonging to a certain geometric position.

The new message TX\_OBJ\_GETDISPINTERFACE can be used to get a pointer to the dispatch interface of an inserted object. This pointer can be used to set an object's properties or to call an object's methods.

## *Changes and Extensions*

The Text Control text format has been changed to support headers and footers and special types of marked text fields. The text format is fully compatible to prior formats. Furthermore all prior formats can be loaded. The new format version number is 700. More information about how headers and footers are integrated can be found in Appendix A of this manual.

## ***New and Extended Messages and Data Structures***

<b><u>Message</u></b>	<b><u>Description</u></b>
TX_FIELD_FROMCARETPOS	Returns the field identifier of the field containing the current input position.
TX_FIELD_GETNEXT	Additionally supports the special field types.
TX_FIELD_GETTYPE	Returns the type of a marked text field.
TX_FIELD_GOTO	Sets the current input position to the beginning of a marked text field.
TX_FIELD_SETTYPE	Defines a marked text field of a special type.
TX_GET/SETTEXTAREA	Supports an additional page view that centers the document in the Text Control's client area and displays three-dimensional pages with shadows.
TX_HF_ACTIVATE	Activates or deactivates a header or footer.
TX_HF_DISABLE	Disables the usage of headers and/or footers.
TX_HF_ENABLE	Enables the usage of headers and/or footers.
TX_HF_GETENABLED	Returns the currently enabled headers and/or footers.
TX_HF_GETPOSITION	Returns a header's or footer's position on the page.
TX_HF_SELECT	Selects a header or footer for message routing.
TX_HF_SETPOSITION	Sets a header's or footer's position on the page.
TX_INPUTPOSFROMPOINT	Returns the text input position belonging to a given point in the Text Control window's visible area.
TX_OBJ_GETDISPINTERFACE	Returns a pointer to an object's dispatch interface.
TX_RESETCONTENTS	Deletes the complete contents of the Text Control.

---

TX_SETLINEANDCOL	Sets a new text input position, given through a page, line and column number.
<b><u>Notification</u></b>	<b><u>Description</u></b>
TN_FIELD_LINKCLICKED	Occurs when a field has been clicked on that is the source of a hypertext link.
TN_HF_ACTIVATED	Informs about header or footer activation.
TX_HF_DEACTIVATED	Informs about header or footer deactivation.
<b><u>Data Structure</u></b>	<b><u>Description</u></b>
FIELDADATA	The <i>lReserved</i> parameter has been changed. One byte now indicates the type of the field, the remaining three bytes are still reserved.
FONTBLOCK	The <i>reserved</i> parameter has been changed to <i>nFieldType</i> . It stores the type of the marked text field.
TXFILTERIO	The <i>lpImagePath</i> member has been changed to <i>lpAbsPath</i> . The <i>lpreserved</i> member has been changed to <i>lpBasePath</i> . The member <i>lpDocPath</i> has been added. The <i>reserved[256]</i> has been changed to <i>reserved[252]</i> .

# *What's New in Version 7.0 since Version 5.2*

## *New Features*

The 32 bit version of Text Control has been extended to support Unicode, the character set for all languages. All messages and functions with string parameters have been implemented twice, one version for ANSI and one version for Unicode. See the new chapter 1.14 "*ANSI and Unicode*" for more information and a complete list of the extended messages and functions and how to use them. Unicode support is available on Windows NT and Windows 95/98.

The 32 bit version of Text Control now supports Far Eastern writing systems (Input Method Editors) and can process double-byte character sets. Internal dialog boxes and user messages are available in Japanese. This also is supported on Windows NT and Windows 95/98.

The TX\_DATAIN and TX\_DATAOUT messages support loading and storing Unicode text either as text only or integrated in the Text Control's text format.

The TX\_FINDTEXT message has been extended so that text can be searched for without using dialog boxes.

The new messages TX\_TABLE\_GETATTROFCELL and TX\_TABLE\_SETATTROFCELL can be used to handle attributes of table cells like border widths, text distances and background color.

The new message TX\_TABLE\_GETCELLPOSITION retrieves the indexes of the first and the last character in a table cell.

The new message TX\_TABLE\_GETNEXT returns the identifier of a table which follows a specified table. It can be used to construct a loop of all tables a Text Control contains.

The TX\_SETLANGUAGE message accepts the file name of a resource library which Text Control loads when resources are needed. This can be used to display information strings and dialog boxes in other than the built-in languages. See the new chapter 1.15 "*Resources*" for more information.

A new Tabulator type has been implemented. This type acts like a right-aligned tabulator but its position is always the rightmost text position. This tabulator type can only be set with the TX\_SETTABS message.

## Changes and Extensions

In table cells with a single decimal tabulator text is automatically formatted. It is not necessary to type a tabulator character.

The Text Control's text format has been extended to support Unicode. The following extensions have been made:

1. Text is stored either in ANSI or in Unicode format depending on the TX\_DATAOUT message settings. The *dwTextSize* member of the textblock structure stores the Unicode character size of 2 bytes in its high-order word when the following text is in Unicode format. International applications should store the text in Unicode format.
2. The *fAddData* member of the textblock structure indicates additional data following the *FieldData* byte array.
3. An array of TXLOGFONTEX data structures follows the *FieldData* byte array. These structures contain additional font data.

Several integral data types of the message and function parameters and data structure members have been changed to support Unicode. See the new chapter 1.14 "ANSI and Unicode" for a complete list. All changes are fully compatible to prior versions of Text Control.

User-developed 32 bit filters must be extended to process Unicode. The 32 bit version of Text Control 6.0 works only with filters that can handle Text Control's text format with text formatted as Unicode. Filters must provide the additional function **GetFilterInfo** that informs about the capabilities of a text filter. See appendix C for more information about this function.

## New and Extended Messages

<u>Message</u>	<u>Description</u>
TX_DATAIN / TX_DATAOUT	Provides the new format identifier TF_FORMAT_UNICODE.
TX_FINDTEXT	Can now be used without displaying a dialogbox.
TX_GETTABS / TX_SETTABS	Provides the new tabulator type RIGHTBORDERTAB.

---

TX_SETLANGUAGE	Accepts the filename of a resource library.
TX_TABLE_GETATTROFCELL	Retrieves information about the attributes of one or more table cells.
TX_TABLE_GETCELLPOSITION	Retrieves the indexes of the first and the last character in a table cell.
TX_TABLE_GETNEXT	Returns the identifier of a table following a specified table.
TX_TABLE_SETATTROFCELL	Alters one or more attributes of one or more table cells.

# ***1. Introduction***

## ***1.1 What is TX Text Control***

TX Text Control is an efficient programming tool for Microsoft Windows. It enables the software developer to realize software projects with sophisticated text processing features, in a fraction of the time normally required.

A complete object oriented approach is realized. All of the following features are accessible through messages:

- Keyboard interface.
- Mouse interface.
- File I/O.
- Clipboard operations with formatted text.
- Text formatting with all the fonts that are supported by the output device.
- Paragraph formatting which includes tabs and adjustment of line setting.
- Dialog boxes for convenient selection of fonts and adjustment of line spacing.
- Image processing.
- table processing.
- OLE support.

Even for actions like printing or file manipulation knowledge about the internal storage structures is not required.

## ***1.2 The Library Files***

TX32.DLL	The dynamic link library of the Text Control module.
TX.H	The Text Control's include file for your application.
TX32.LIB	The import library file to be linked with your object files.
TX_RTF32.DLL	The text import/export filter for Rich Text Format.

TX_HTM32.DLL	The text import/export filter for HTML Format (Hypertext Markup Language).
TX_WORD.DLL	The text import/export filter for Microsoft WORD.
TXTLS32.DLL	The dynamic link library containing the Text Control tools, such as status bar, ruler and button bar.
TXTLS32.LIB	The import library file for the Text Control tools.
TXTOOLS.H	The include file for the Text Control tools.
WNDTLS32.DLL	The dynamic link library for moving and sizing objects.
WNDTLS32.LIB	The import library file for the Text Control tools.
TXOBJ32.DLL	The dynamic link library for OLE support.
TX.HLP	This manual's Windows Help file.
WMXMSG.H	An additional include file for handling external objects.

### 1.3 *The Source Files*

If your version contains the source code, the following directories contain the various source and make files:

\TX	Contains the source files for the Text Control module.
\RTF	Contains the source files for the import/export filter TX_RTF.DLL.
\TX_HTML	Contains the source files for the import/export filter TX_HTML.DLL.
\TX_WORD	Contains the source files for the import/export filter TX_WORD.DLL.
\TXTOOLS	Contains the source files for the Text Control tools.
\WNDTOOLS	Contains the source files for the move and size tool.
\TXOBJ	Contains the source files for the OLE module.
\INC	Contains all include files.
\LIB	Contains all import library files.
\DLL	Contains all executable files.

Both versions, 16 and 32 bit, can be compiled from the same source files. For the 32 bit versions each directory contains a Visual C++ 4.0 project file \*.MDP. For the 16 bit versions makefiles are available for the MAKE utility from Turbo C and for the NMAKE utility from Microsoft C. Each directory contains four subdirectories to receive the object files as debug and retail version. The executable files are automatically be copied in the \DLL directory, the include files in the \INC directory and the import library files in the \LIB directory.

## 1.4 *Creating and Programming a Text Control Window*

First include the file TX.H into the source code of your application. The object files must be linked with TX.LIB. The dynamic link libraries must be in the current directory or in a directory that is included in the PATH environment variable.

Now you can create a window by calling **CreateTextControl**. Windows of this type are referred to as Text Control in the following text. Save the returned window handle because it is needed for communication with the Text Control. The communication is completely handled with functions like **PostMessage**, **SendMessage** or the window manager functions. All features of the Text Control are accessible through messages. The Text Control is established as a child of the controlling window. More than one Text Control can be established at different positions on the parent window. This can be used to simulate a page, where only certain areas can show or receive text. If the background mode is set to transparent, then even the background of one Text Control can be another Text Control .

After its creation, a text control has to be made visible with a call to **ShowWindow**. The behaviour of a child window is further described in the SDK reference. If the Text Control is to be able to receive keyboard input every time the controlling window is activated, call **SetFocus** with the Text Control handle as a parameter when the application receives a WM\_ACTIVATE message.

When calling **RegisterClass** for the parent window, the STYLE field of the WNDCLASS structure should not contain CS\_HREDRAW and CS\_VREDRAW to avoid unnecessary redrawing.

Furthermore a Text Control can be used as part of a dialogbox. In the resource file the CONTROL statement can be used with the class field set to »TX«, the Text Control class name. Then the dialog box processing code can use the functions **GetDlgItemText**, **GetDlgItemInt**, **SetDlgItemText** and **SetDlgItemInt**.

The Text Control informs the parent window about special conditions with several notification messages. The parent window receives these messages which are described in chapter 6 through the *lParam* parameter of a WM\_COMMAND message. To avoid message queue overflow frequently used notifications messages like TN\_POSCHANGED are sent with **SendMessage** instead of **PostMessage**. An application can therefore use the notification messages to change its own data structures or to get information from the Text Control but must be careful when changing the Text Control's current contents. In this case it is safer when the application implements a delay of its own, by for example sending a message to itself with **PostMessage**. In this way it can be ensured that the Text Control has finished the process which causes the notification message.

## 1.5 Text Formatting and Views

Text Control offers several ways, in which text may be formatted and viewed as described in the following list:

### 1. Control view:

The area for text formatting is the Text Control window's client area. This is the default setting after creating a Text Control.

### 2. Control view with autoexpand:

The area for text formatting is the Text Control window's client area, but the Text Control's window size is automatically expanded when the text exceeds the window's size.

### 3. Control view and linked Text Controls:

The area for text formatting is the Text Control window's client area, but text automatically flows to a following linked Text Control when it exceeds the window's size.

### 4. Normal View:

The formatting width has been specified by the programmer and invisible text can be shown with a built-in scroll interface with or without scroll-bars. The vertical scrolling amount depends on the text.

### 5. Page View:

The formatting width and height has been specified by the programmer. Text Control has a built-in scroll interface and displays pages with gaps, borders, margins and a gray background. The number of pages depends on the text.

**6. Extended Page View:**

This view works in the same way as the page view, but Text Control displays three-dimensional pages which are centered in the window's client area.

## *Control View*

The formatting area, which is the area Text Control uses to perform line breaks, is the client area of the Text Control's window. This means that every time the window is resized, the text is newly formatted.

A line break is automatically performed when the current input position reaches the window's right border. When the window's right border is reached without having a break character in a line, no line break is performed and Text Control indicates overflowing text with a vertical mark. This mark is displayed at the window's left or right border, depending on the paragraph alignment setting. When the number of lines exceed the window's height, overflowing text is indicated with a plus mark at the bottom of the window. The current input position, indicated by the caret, can never leave the window's client area. If the user tries to move the caret to overflowing text, the Text Control beeps.

When the Text Control is zoomed, the current window's size is also zoomed to adapt the formatting area to the new zooming factor. Its position in relation to its parent window is also zoomed.

The formatting area must always be large enough to fully display the largest character currently contained in the text. Information about the dimensions of this minimum window size can be obtained by the minimum tracking size retrieved through the WM\_GETMINMAXINFO message.

Autoexpanding and window linking are possible only in this view. In autoexpanding mode the window size is automatically expanded to the current text amount. This mode can be set through the TX\_SETMODE message. After Text Control has expanded its window size it sends TN\_HEXEXPAND and/or TN\_VEXPAND notification messages. The window linking feature is described in chapter 1.12.

Because the control view has no built-in scroll interface, Text Control sends several notification messages, to enable a programmer to implement an external scroll interface. This can be important for example when linked Text Controls are used to form a document with several pages. The TN\_CARETOUTBOTTOM, TN\_CARETOUTLEFT, TN\_CARETOUTRIGHT and TN\_CARETOUTTOP

notifications are sent, when the current input position has been moved outside the Text Control window's visible part. The `TN_CARETOUT` notification is sent, when the current input position has been moved to a completely invisible Text Control and the `TN_AUTOSCROLL` notification is sent during extension of a text selection with the mouse. All of these notification messages are only possible in the control view.

## *Normal View*

With the `TX_SETTEXTAREA` message the formatting area can be changed by the programmer. This is necessary to realize a text editor that formats the text accordingly to a certain page width. The page width is specified through the low-order word of the *lParam* parameter, the high-order word must be set to -1 for this view, which indicates that the formatting height depends on the current amount of text.

When the formatting width and/or height is greater than the current window size and the caret reaches the Text Control window's border, scrolling is performed automatically to make the new input position visible. Additionally a horizontal and/or a vertical scroll-bar can be used to scroll to text parts outside of the client area without changing the current input position. Scroll-bars can be used only when `TF_HSCROLL` and/or `TF_VSCROLL` has been specified with `TX_SETTEXTAREA`.

If the Text Control's window is sized the text is not newly formatted and if the Text Control is zoomed, the window's size and position are not changed. If the window is resized visible scroll-bars are automatically hidden when they are no longer needed. Conversely previously hidden scroll-bars are automatically shown when the window's size becomes smaller. To get the information as to whether a specific scrollbar is visible or not, use the **GetWindowLong** function with the `GWL_STYLE` flag.

## *Page View*

A programmer can also specify a page height through the *lParam*'s high-order word when calling the `TX_SETTEXTAREA` message (instead of -1). Text Control then shows pages with margins and borders. The number of pages depends on the amount of text.

Text Control interprets the formatting area specified through the *lParam* parameter of the TX\_SETTEXTAREA message as the area that can be filled with text. Page margins are added to these values to form a page. Page margins have a default value of 2 cm and can be changed through the TX\_SETPAGEMARGINS message.

## ***Extended Page View***

With the TF\_EXTPAGEVIEW setting of TX\_SETTEXTAREA, pages are displayed in three dimensions with borders and shadows and they are centered in the window's client area. The TF\_PAGEVIEW setting returns to the two-dimensional page view.

like for many other extension settings, Text Control uses twentieths of a point for defining the formatting area. This unit is often used in combination with text processing applications or fonts and is called TWIP. One TWIP is a 1/1440th of an inch.

In the normal view and both page views Text Control has a built-in scroll interface. Text Control sends TN\_HSCROLL and TN\_VSCROLL notification messages when it scrolls automatically. The current scroll positions can be obtained with the TX\_GETSCROLLPOS message or set with the TX\_SETSCROLLPOS message.

With the page view and the extended page view headers and footers can be used. For more information how to handle headers and footers see "*Headers and Footers*".

## ***Mixed Views***

The control view and the normal view can be mixed, when either the formatting width or height is set to zero. For example when specifying a width of zero and a height of -1, the text is formatted depending on the current window's width but the text is not limited to the window's height.

## 1.6 Headers and Footers

### *Using Headers and Footers*

Headers and footers can only be used when a formatting area has been set with the TX\_SETTEXTAREA message. Headers and footers are only visible on the screen when page view or extended page view has been selected (see chapter 1.5 "*Text Formatting and Views*" for more information).

Headers and/or footers must be enabled with the TX\_HF\_ENABLE message. This message specifies whether headers and footers, only headers or only footers are to be used. Additionally special headers and/or footers for the first page can be specified. To edit an inserted header or footer, it must be activated either with the TX\_HF\_ACTIVATE message or with a built-in mouse interface. An activated header or footer gets the input focus and its border is shown with a dotted frame. When a header or footer is activated, the main text is displayed gray, otherwise a header's or footer's text is displayed gray. Text Control sends TN\_HF\_ACTIVATED and TN\_HF\_DEACTIVATED notification messages to inform its parent window about activation or deactivation of headers or footers.

The TX\_HF\_ENABLE message allows the following style settings:

1. Activation can be performed with mouse click and/or with mouse double-click.
2. The border of an activated header or footer can be solid, dotted or unframed.

The default style setting is a dotted frame and a mouse interface that activates a header or footer with double-clicks.

By default the top of a header has a distance of one centimeter from the top of the page and the bottom of a footer has a distance of one centimeter from the bottom of the page. With the TX\_HF\_GETPOSITION and TX\_HF\_SETPOSITION messages these values can be changed. The height of a header or footer depends on the header's or footer's current text.

When a document is loaded or converted from another format, contained headers and footers are automatically displayed. The TX\_HF\_GETENABLED message can be used to get the information about which headers and/or footers the current document contains.

To delete a header or footer or to disable a certain style setting, the TX\_HF\_DISABLE message can be used.

## *Programming Headers and Footers*

Headers and footers are separate text parts which are independent of the main text. When the user alters the text or the text format, for example with a connected button bar, Text Control uses the current input focus, to determine whether the text format of a header, a footer or the main text is changed. The same occurs when the text is manipulated from programming code. For example when a table is inserted from a menu with the TX\_TABLE\_INSERT message, the current input focus determines whether the table is inserted in a header's or footer's text or in the main text.

In addition to this default selection a programmer can use the TX\_HF\_SELECT message to use a certain message for a certain text part. For example the following code returns the size of a header's text:

```
LONG lTextSize;  
SendMessage(hwndTX, TX_HF_SELECT, 0, TF_HF_HEADER);  
lTextSize = SendMessage(hwndTX, TX_GETTEXTSIZE, 0, 0L);  
SendMessage(hwndTX, TX_HF_SELECT, 0, TF_HF_AUTO);
```

The first line selects the header, independent of the current input focus, the second line gets the size of the header's text and the third line returns to the default selection mode. There can be more than one message call between the two TX\_HF\_SELECT calls.

Almost all messages can be used in this way with some exceptions. The following is a complete list of these exceptions:

1. The following messages cannot be used with headers and footers:

- all messages that handle scrolling
- all messages that handle headers and footers
- all messages that handle printing
- all messages that handle chains of linked windows
- TX\_GETRECT
- all obsolete messages

2. The following messages are handled for all text parts (main text, headers and footers) in the same way, independent of the currently selected part. These messages are:

- TX\_DEVMODECHANGE
- TX\_GET/SETBKGNDCOLOR
- TX\_GET/SETLANGUAGE

- TX\_GET/SETCARETEXT
- TX\_GET/SETMODE
- TX\_GET/SETMODEEX
- TX\_GETSUPPORTEDFONTS
- TX\_GETSUPPORTEDSIZES
- TX\_GET/SETDEVICE
- TX\_GETZOOM/TX\_ZOOM
- TX\_SETWORDDDIVISION

3. The following messages can only be used with headers and footers in conjunction with the TX\_HF\_SELECT message:

- TX\_GETDATASIZE
- TX\_DATAIN
- TX\_DATAOUT
- TX\_COPYDATA
- TX\_LOAD
- TX\_PASTEDATA
- TX\_SAVE

## 1.7 Tables

### *Using Tables*

Tables can be inserted into a Text Control either with the message TX\_TABLE\_INSERT or as part of a document formatted with the RTF or HTML formats. Text Control treats a table as a number of cells organized in rows and columns. Each cell can have as many lines and paragraphs as required. Paragraph formatting is performed in relation to a cell's borders. Each cell has a position and an extension in the document, within this area a cell's frames and text are drawn along with its paragraph and character formatting attributes. There can be a distance between the frame and the text.

Text can be selected either within a single cell or in steps of complete cells or rows. When a selection is deleted inside a table only the text is deleted. To delete one or more complete rows use the TX\_TABLE\_DELETELINES message. Tables can be copied to the clipboard and pasted from the clipboard. When a table is inserted at the first position of another table or immediately behind another table and both

tables have the same number of columns they are combined into a single table. The insertion of one table inside another table is not possible.

A table's attributes are its frame width, distance between frame and formatted text, and background color. To alter the attributes of a table or part of a table, cells must be selected. Then either the messages `TX_TABLE_GETATTR` and `TX_TABLE_SETATTR` messages can be used or a built-in dialog box can be opened with the `TX_TABLE_ATTRDIALOG` message. When the selection extends over several tables or tables mixed with text, attributes cannot be changed. To get information about whether attributes can be changed or tables can be inserted or deleted, for example to implement a menu, the message `TX_TABLE_ISPOSSIBLE` can be used.

When the current input position is inside a table, the ruler shows the positions of all the cells in a table's row and the formatting attributes of the cell the input position belongs to. Then the cells' positions and extensions can be changed with a built-in mouse interface. This can also be performed with the messages `TX_TABLE_GETPOSITIONS` and `TX_TABLE_SETPOSITIONS`.

## ***Programming with Table Identifiers***

Like objects and marked text fields each table has a unique identifier which is set by Text Control. This identifier is returned from the `TX_TABLE_INSERT` message. A programmer can change this identifier with the `TX_TABLE_CHANGEID` message. Changing the identifier is not necessary but recommended when a table's text or attributes are to be changed from the program instead from an end-user. The user-defined identifier need not to be unique and remains valid if a table is saved and reloaded.

When a table or a part of a table is inserted inside another table the inserted table becomes a part of the existing table and the inserted table's identifier is lost.

When a table with a user-defined identifier is inserted outside of all existing tables a new table is created and the table's identifier remains valid. Text Control informs its parent window with a `TN_TABLE_CREATED` notification message that a new table has been created. The programmer can change the identifier sent with the notification by setting the return value of the notification message.

When a table is inserted from another application which means it cannot have an user-defined identifier, Text Control sends an own-selected identifier with the `TN_TABLE_CREATED` notification so that the program can change it.

When tables are imported with one of the data import messages TN\_TABLE\_CREATED notifications are sent only when text is inserted into an existing document or when an imported table has no user-defined identifier. Otherwise when a table with an user-defined identifier is saved and reloaded no notification is sent.

When a table is completely deleted Text Control informs its parent with a TN\_TABLE\_DELETED notification message.

The following messages can be used with table identifiers to get information or to set table attributes regardless whether the current input position is or is not inside this table:

<u>Message</u>	<u>Description</u>
TX_TABLE_CHANGEID	Changes a table's identifier.
TX_TABLE_GETROWSANDCOLS	Returns the number of rows and columns in a table.
TX_TABLE_GETTEXTTOFCELL	Gets a table cell's text.
TX_TABLE_SETTEXTTOFCELL	Changes a table cell's text.

When more than one table with a certain identifier exists, these messages perform the operation with the original inserted table. In chains of linked windows these messages can be sent to any window in the chain regardless of which window contains the table.

## 1.8 *Marked Text Fields*

### *Using Marked Text Fields*

A set of messages has been implemented to define areas in the text of a Text Control called marked text fields. These fields can be used to create hypertext features like those in the Windows Help application, to realize database embedding while text of different datasets can be included into the text or to combine several fields with formulas as in spreadsheet applications.

An application can use the TX\_FIELD\_INSERT message to define a marked text field. The whole communication works with unique numbers returned by this message. The current text can be changed or retrieved with the messages

TX\_FIELD\_CHANGETEXT and TX\_FIELD\_GETTEXT, the message TX\_FIELD\_GETPOSITION returns the current text position of a field. Special attributes can be set with the messages TX\_FIELD\_GETATTR and TX\_FIELD\_SETATTR. These attributes can prevent a field from being deleted or the text of a field from being changed. Further attributes which help the end-user to edit the field's contents are described in the next chapter.

With different notification messages Text Control informs the application about special conditions. The notification messages TN\_FIELD\_CLICKED and TN\_FIELD\_DBLCLICKED inform the application about mouse clicks; TN\_FIELD\_ENTERED and TN\_FIELD\_LEFT indicate whether the current input position has been moved into or from a marked text field. TN\_FIELD\_SETCURSOR can be used to define the cursor when it is moved over a field. The default cursor is the up-arrow cursor. The notification message TN\_FIELD\_CHANGED is sent if the text of a field has been altered, and the notification messages TN\_FIELD\_DELETED and TN\_FIELD\_CREATED are sent if fields have been deleted or created while inserting or deleting text with the keyboard or the clipboard. If the text and format data of a Text Control which contains marked text fields are saved and then reloaded all field identifiers remain the same.

## ***Editing Marked Text Fields***

When marked text fields are used in an editable Text Control and these fields are editable, the end-user can alter the contents of the field like any other text. Because it is not always unique whether the current input position is or is not inside a field, some field attributes have been implemented to help the end-user to edit fields. These attributes can be used in any combination and must be defined with the TX\_FIELD\_INSERT message or can be altered with the TX\_FIELD\_SETATTR message.

When the current input position is in front of or behind a field, the next inserted character can either belong to the field or to the text outside the field. In normal editing mode an inserted character has the attributes of its preceding character which means that inserted text just behind a field belongs to the field and inserted text in front of a field does belong to the text in front of the field. To solve these problems an extended edit mode can be defined for every field with the TF\_EXTEDITMODE setting that implements a second input position at the beginning and the end of the field. The end-user can switch between the two positions with the left and right arrow keys. This is especially important when a

marked text field is at the beginning or the end of the complete text. For example when a field is at the end of the text the end-user can press CTRL+END to reach the text end. When this position is also the end of a marked text field the right arrow key must be pressed first when the next inserted character should not belong to the field.

To help the end-user to find the correct position the TF\_USEFIELDCARET and TF\_SHOWCURFIELDGRAY attributes can be used either stand alone or in combination. TF\_USEFIELDCARET defines an attribute that changes the caret's width when it is inside a marked text field. This width can be defined with the TX\_SETCARETEXT message. TF\_SHOWCURFIELDGRAY defines an attribute that displays the complete text of a field with a gray background when the current input position is inside this field.

Each of the described attributes can be defined for a single field in any combination which means that different kinds of marked text fields can be implemented in a single Text Control.

## ***Relating data to a marked text field***

For each marked text field Text Control can store any data that can be set with the TX\_FIELD\_SETDATA message. For example, when a Text Control is used to show the contents of a database, a marked text field can be created for each database field. The database's field names can then be related to the Text Control's marked text fields using the TX\_FIELD\_SETDATA message.

Other parts of the program can use the TX\_FIELD\_GETDATA message to retrieve the name of the database field to which a marked text field is linked. For example when the user has clicked on a marked text field, the TX\_FIELD\_GETDATA message can be used with the field identifier, which has been sent with the TN\_FIELD\_CLICKED notification message. The message then retrieves the name of the database field the user has clicked on.

Data entries can also be numbers instead of strings. When a marked text field is copied via the clipboard or saved to a file the data belonging to the field is also copied or saved. The usage of TX\_FIELD\_SETDATA does not change the current text contents of a marked text field. When new data is set, all previously set data is overwritten independently of the kind of data involved.

## *Special Types of Marked Text Fields*

Text Control supports special types of marked text fields that can be defined with the TX\_FIELD\_SETTYPE message. The following field types are possible:

<b><u>Type</u></b>	<b><u>Description</u></b>
FT_EXTERNALLINK	This field defines the source of a hypertext link to a location outside of the document.
FT_INTERNALLINK	This field defines the source of a hypertext link to a location in the same document.
FT_LINKTARGET	This field defines the target of a hypertext link.
FT_PAGENUMBER	This field displays the current page number. It can only be used in headers or footers.
FT_HIGHLIGHT	This field defines a piece of text that can be highlighted.
FT_TOPIC	This field defines a position in a document that is the beginning of a topic.

All of these fields have the same general properties as standard marked text fields with the following exceptions: Fields of the type FT\_LINKTARGET or FT\_TOPIC define text positions in a document. Therefore as they have no visible text they cannot be edited and have no extended edit mode. Fields of the type FT\_PAGENUMBER can only be used in headers or footers.

For each of the special field types Text Control handles some additional data, called type-related data. These data can also be defined with the TY\_FIELD\_SETTYPE message and is accessible for a certain field through the TX\_FIELD\_GETTYPE message. For the types FT\_EXTERNALLINK and FT\_INTERNALLINK these data are the information to where the link points. This can be an address or a file name and/or the name of a target in a document. Targets in documents can be realized with marked text fields, which have the type FT\_LINKTARGET. These fields can have a name that is saved as type-related data. When the user clicks on a field of the type FT\_EXTERNALLINK or FT\_INTERNALLINK a TN\_FIELD\_LINKCLICKED notification is sent. The application can get the link information with the TX\_FIELD\_GETTYPE message and perform the necessary tasks. The TX\_FIELD\_GOTO message can be used to scroll to an internal link position and the TX\_FIELD\_GETNEXT message can be used to enumerate all fields of a certain type.

To insert a field of a special type from programming code, use the `TX_FIELD_INSERT` message first and then set the type and its data. The following C example inserts a field that represents a link to the Text Control homepage:

```
WORD wField;
TXFIELDSETTYPE fst;
TCHAR text[] = _T("visit the Text Control homepage");
TCHAR data[] = _T("http://www.textcontrol.com");

ZeroMemory(&fst, sizeof(TXFIELDSETTYPE));
fst.wStructSize = sizeof(TXFIELDSETTYPE);
fst.nFieldType = FT_EXTERNALLINK;
fst.dwTypeDataSize = (lstrlen(address)+1) * sizeof(TCHAR);
fst.lptypedata = data;

wField = (WORD)SendMessage(hwndTX, TX_FIELD_INSERT, 0,
(LPARAM)(LPTSTR)text);

SendMessage(hwndTX, TX_FIELD_SETTYPE, (WPARAM)wField, (LPARAM)&fst);
```

When a user clicks on this marked text field, a `TN_FIELD_LINKCLICKED` notification is sent. To insert a field of the type `FT_LINKTARGET`, the created field must not have text. Change the previous example to:

```
TCHAR text[] = _T("");
TCHAR data[] = _T("first target");
```

This creates a field with the name "first target". The `TX_FIELD_GOTO` message can be used to scroll to this target:

```
SendMessage(hwndTX, TX_FIELD_GOTO, FT_LINKTARGET, (LPARAM)data);
```

When HTML, RTF or Word documents are loaded with the `TX_DATAIN` message and the `dwUsageFlags` member of the `TXFILTERIO` data structure is set to `FIO_ENABLELINKS`, all source and target fields for hypertext links are automatically created.

Fields of the type `FT_HIGHLIGHT` can be used to mark pieces of text in a document that can be highlighted. This is useful, for instance, to highlight occurrences of a word found during a global search. The color of the highlight is stored as additional data for these fields. The `TX_FIELD_GOTO` message enables the programmer to scroll from highlight to highlight. When RTF documents are loaded with the `TX_DATAIN` message and the `dwUsageFlags` member of the `TXFILTERIO` data structure is set to `FIO_ENABLEHIGHLIGHTS`, all RTF `\cbN` keywords are automatically converted to fields of the type `FT_HIGHLIGHT`. *N* is the index of a color in the RTF color table.

Fields of the type FT\_TOPIC are text positions in a document which define the beginning of a topic. The TX\_FIELD\_GOTO message can be used to scroll to a topic with a certain number. When RTF documents are loaded with the TX\_DATAIN message and the *dwUsageFlags* member of the TXFILTERIO data structure is set to FIO\_ENABLETOPICS, all RTF '\sect' keywords are automatically converted to fields of the type FT\_TOPIC. These topics are numbered from 1 to n in the order they appear in the RTF document.

## 1.9 Image Processing

With the help of a second programming tool, the IC Image-Control, the Text Control can integrate images into the text. Sending a TX\_CREATEIMAGE message to an active Text Control inserts the given image into the text at the current caret position. The image can be selected by mouse-click or be included into a text selection. The messages TX\_GETIMAGEFORMAT and TX\_SETIMAGEFORMAT can be used to set several attributes like a fast display mode or a scaling factor.

Images are handled with the Text Control's file input/output functions just like the text they are embedded in. They can also be transferred via the clipboard. If the image is selected with a mouse click and copied to the clipboard, only the raw image data is copied. If the image is included into a text selection and copied to the clipboard, the data that describes its relation to the text like the horizontal position is copied, too.

To delete an image from the text use the BACKSPACE key from the next text position below the image or send a WM\_CLEAR message to the Text Control if an image has been selected.

## 1.10 Integrating OLE Objects

### *Insertion*

OLE objects can be inserted into a Text Control document like any other object with the TX\_OBJ\_EMBED message. The *wParam* parameter must be set to TF\_OBJ\_OLEOBJECT. This message opens the OLE built-in *Insert Object* dialog box where the user can select one of the system-registered OLE servers. Depending

on the embedding mode selected the new OLE object is inserted either at a fixed position or as a character and is immediately in-place activated.

The *Insert Object* dialog box allows the user to insert newly created or existing objects into a Text Control document. It also allows the user to choose to display the object as an icon and enables the *Change Icon* command button. The dialog box is normally displayed when the user chooses *Insert Object* from the *Edit* menu of a OLE container application. Because objects in Text Control can be inserted either at fixed positions or as characters it is useful to expand the *Edit* menu with a second entry, for example *Insert Object as character*.

## ***User Interface***

An inserted OLE object can be in any one of the following states:

### **1. Deselected state**

In this state the object's contents are displayed with a solid, thin border indicating an embedded object. The object has this state when either another object is selected or the Text Control has been clicked so that the user can enter text.

### **2. Selected state**

An object has the selected state after it has been clicked. In this mode resize handles are displayed so that it can be moved and resized. When the object is resized in this state its contents are scaled. A programmer can get the new scaling factors with the TX\_OBJ\_GETATTR message. When a scaled object is activated in-place it displays its contents either scaled or, when scaling is not supported, it shows scrollbars.

### **3. In-place activated state**

An object is in-place activated after it has been double-clicked. In this mode the object can be edited. It is displayed with a hatched border including resize handles. When an object is resized or edited in this state the object's natural size can be changed. After editing and deactivating (selected or deselected) the Text Control adapts the object to its new natural size. Scaling factors remain the same in this case. Text Control does not support the exchanging of menus and controlbars.

### **4. Open state**

An object's server application is fully opened when the object is double-clicked whilst pressing the CTRL key. The object's contents are then overlaid with a hatched pattern. After the server has been closed the object is updated with the new contents and adapted to its new natural size.

## ***Clipboard***

When an OLE object is in selected state it can be copied to the clipboard in standard formats such as metafile, and in OLE formats. When an 'as character' inserted object is selected in combination with text it is integrated into the Text Control's text format. When an OLE object is pasted from the clipboard it is always inserted as a character at the current input position. If an object is being pasted while another object is selected the selected object is replaced.

To realize the menu and keyboard interface the Windows messages WM\_PASTE, WM\_CUT, WM\_COPY and WM\_CLEAR can be used. When the command routing features of MFC are used, these messages can be called up from the command handlers of the application's view class. If MFC is not used and the standard keyboard keys for clipboard actions CONTROL+V, CONTROL+C, CONTROL+X and DEL are not implemented via an accelerator table, the keyboard interface is automatically implemented by Text Control.

## ***Loading and Saving***

OLE objects are integrated into the Text Control's text format like any other objects. Therefore all messages that support loading, saving and general data exchange (like TX\_LOAD, TX\_SAVE, TX\_COPYDATA, TX\_PASTEDATA, TX\_IMPORTTEXT, TX\_EXPORTTEXT, TX\_DATAIN and TX\_DATAOUT) can be used without changes.

## ***Printing***

Text Control prints an object's current contents via its metafile. This metafile is "played" on the printer device context which is sent with the TX\_PRINT or TX\_PRINTPAGE message. Therefore no changes are necessary with these messages. In addition Text Control supports the WM\_PAINT message with the *wParam* parameter set to a metafile device context, and in the 32 bit version it supports the new messages WM\_PRINT and WM\_PRINTCLIENT.

## ***Zooming***

When a Text Control is zoomed integrated OLE objects are also zoomed. In the selected, deselected and open states, zooming is realized by stretching the object's metafile. When a zoomed object is in-place activated, whether its contents are zoomed or not depends on the object. When an object does not support zooming the smaller client site set by the Text Control makes it necessary to show scrollbars to indicate that the content's natural size is larger than the object's client site.

## ***Undo***

When an OLE object is part of a block of text, the undo function is fully supported as with any other object. When an object has been selected stand alone and is then deleted or replaced, undo is not supported.

## ***Creating a OLE container application with MFC 4.0***

The following describes how to create an OLE container application with Text Control. The sample program TXCon.exe was created in this way.

1. Use application wizzard to create a new application. Use all default settings in Steps 1 to 6, Do not select *OLE container* because this is implemented in Text Control.
2. In the View class replace **CView** with **CCtrlView**.
3. In the view constructor initialize the base-class constructor.
4. In the view's **PreCreateWindow** function initialize the Text Control window class.
5. Include the header file TX.H.
6. In the project *Settings* dialog box for *Link...Object/Library modules* add TX32.LIB.
7. Copy all Text Control library files to the project's directory containing your executable file.
8. In the view class add a handler for WM\_CREATE and initialize Text Control.

9. Add two menu entries *Insert Object...* and *Insert Object as character...* including their handlers, and send the TX\_OBJ\_EMBED message to create new OLE objects.
10. Add handlers for clipboard, delete, and undo handling and send the appropriate Text Control messages.
11. Add an accelerator for the ESCAPE key including its handler, and send the TX\_OBJ\_OLE\_CANCEL message.
12. Add code to the document's **Serialize** function (send TX\_LOAD and TX\_SAVE) to realize loading and saving.

## 1.11 Integrating External Windows

Like images, each object represented as an externally created child window, e.g. a pushbutton, can be integrated into the text. The message TX\_OBJ\_EMBED connects such an object with a Text Control which then becomes the parent of the external window. This object can be either inserted into the text, handled like a single character, or fixed at a specified position relative to the top left corner of the text.

In some special situations the Text Control sends messages to the object. If the object does not respond to the message the Text Control performs a default action, for example if a Text Control is zoomed it sends a WMX\_ZOOM message to the object. If the object does not react Text Control continues to show the object with its old dimensions. If the object is a predefined control, e.g. a pushbutton, the handling of these messages can be implemented by sub-classing.

The following shows a list of possible messages a Text Control can send to an object window. These messages have the prefix WMX\_ and are defined in the header file "wmxmsg.h":

<b><u>Message</u></b>	<b><u>Purpose</u></b>
WMX_COPYDATA	Sent if the object contains data that is to be stored in a file or in memory. A Text Control sends this message if it receives a TX_SAVE or a TX_COPYDATA message.
WMX_GETDATASIZE	Sent to get the buffer size needed for the WMX_COPYDATA message.

WMX_GETWINDOW	This message is not sent to the object but to the Text Control's parent window. If a Text Control stores its objects it cannot also save the window handles representing the objects. Instead it stores the child window identifier chosen by the object's creator. During a loading process the Text Control sends a WMX_GETWINDOW message with the <i>wParam</i> parameter set to the child window identifier so that its parent window, e.g. the application's main window, can create the window.
WMX_GETZOOM	Requests for the object's zooming factor.
WMX_PASTEDATA	Sent to reload the data stored with the WMX_COPYDATA message. A Text Control sends this message if it receives a TX_LOAD or a TX_PASTEDATA message.
WMX_PRINT	A Text Control sends this message whilst its contents are being printed.
WMX_ZOOM	A Text Control sends this message when it is zoomed.

For more information about the messages' parameters see appendix H.

## ***1.12 Using Chains of linked Windows***

Text Controls can be connected to enable text to flow from one window to the next. The message TX\_SETLINKWND connects two Text Controls, and TX\_GETLINKWND returns information about the relationship of linked windows in a chain.

If text overflows in a Text Control that has a successor, the overflowing text is automatically copied to that window. If text is deleted, the cleared area at the bottom of a control is automatically filled with text from a successor. When the blinking caret reaches the bottom of a Text Control, it jumps to the following window. When it reaches the top of a Text Control, it jumps to the preceding window regardless of whether it is moved with the keyboard or the mouse interface.

In a chain of linked windows, each Text Control can contain up to 64 kB of text while the amount of text in a chain is only limited by the system resources. Text

selections can extend over several controls. All messages that refer to a text selection must only be sent to the Text Control that has the input focus. TX passes these messages on to every Text Control that belongs to the current selection. This applies to the clipboard messages and to all messages for changing text attributes like font and paragraph settings.

Other messages that refer to the whole contents of a Text Control, like setting modes, saving data to or loading data from a file, or printing have to be sent to each Text Control in a chain. This makes it possible to print, for example, a single page if the chain of controls realizes a text processing application with several pages.

The TX\_REPLACESEL message can be used to import an ASCII string larger than 64 kB. To save the contents of a chain the TX\_SAVE or TX\_COPYDATA message has to be sent to every Text Control. To reload the text first create a chain of controls and then send the TX\_LOAD or TX\_PASTEDATA message to every Text Control. The messages TX\_IMPORTTEXTBUFFER and TX\_IMPORTTEXTFILE support the insertion of a formatted text greater 64 kB.

To maximize working speed for a chain of linked Text Controls, the window size should be designed so that a single Text Control does not contain more than 32 kB of text.

## 1.13 Using Metafiles

Metafiles can be used with the message WM\_PAINT in 16 and 32 bit applications and additionally with the messages WM\_PRINT and WM\_PRINTCLIENT in 32 bit applications. The caller must create a metafile device context and send it as the *wParam* parameter of these messages. Text Control fills the metafile with function calls that represent only the visible portion of a Text Control window. By default the complete client area of the Text Control window is mapped to one inch of the current output device which is the standard printer or another device set with the TX\_SETDEVICE message. This means when the output device has a resolution of 300 x 300 dpi the metafile bounding rectangle is 0, 0, 300, 300 and the Text Control's client area is mapped to this rectangle.

With the TX\_SETDEVICE message the caller can define a metafile target device independent of the current output device. The *wParam* parameter must be set to TF\_METAFILETARGET and the *lParam* parameter must specify a valid device context handle identifying the metafile target device. The viewport settings of the target device context are used as metafile bounding rectangle. The following code example maps the client area of a Text Control to a bounding rectangle on an

output device specified through *hdcTarget*. The coordinates of the bounding rectangle must be in pixels of the output device:

```
HDC hdcTarget, hdcMetafile;
RECT rcBounds;

// calculate bounding rectangle and create metafile and target device
// context:
...

// prepare the target device context:
SetMapMode(hdcTarget, MM_ANISOTROPIC);

SetViewportOrgEx(hdcTarget, rcBounds.left, rcBounds.top, 0L);
SetViewportExtEx(hdcTarget, rcBounds.right-rcBounds.left,
                  rcBounds.bottom-rcBounds.top, 0L);

// inform TX about the target device:
SendMessage(hwndTX, TX_SETDEVICE, TF_METAFILETARGET,
            (LPARAM)hdcTarget);

// call the TX paint routine:
SendMessage(hwndTX, WM_PAINT, (WPARAM)hdcMetafile, 0L);

...
```

## 1.14 ANSI and Unicode

The Text Control 32 bit DLL can be used either from an ANSI or a Unicode application. The Text Control include file TX.H defines an ANSI and a Unicode version for each message and function which has string or character parameters. Like in the Windows 32 bit SDK the letter A (ANSI) or W (wide char) has been appended to the message or function name, for example the TX\_SETFONT message is available as TX\_SETFONTA and as TX\_SETFONTW. Depending on whether UNICODE is defined or not TX\_SETFONT is defined as TX\_SETFONTA or as TX\_SETFONTW:

```
#ifdef UNICODE
#define TX_SETFONT TX_SETFONTW
#else
#define TX_SETFONT TX_SETFONTA
#endif
```

To build a Unicode application using Text Control, simply define UNICODE before including TX.H and call TX\_SETFONT. In this case TX\_SETFONTW is sent to the Text Control and must contain a font name formatted as a Unicode

string. Because the A-versions have the same numbers as the corresponding messages in prior Text Control versions the new version is fully compatible.

The following is a complete list of all messages and functions that have two implementations. The type of the string pointers has been changed from LPSTR to LPTSTR which is the character format independent form of LPSTR. LPTSTR is a pointer to a Unicode string (LPWSTR), when UNICODE is defined and a pointer to an ANSI string (LPSTR), when UNICODE is not defined. ANSI strings can contain characters from double-byte character sets.

**Message:**

**Changes:**

TX\_CREATEIMAGE

*lParam* changed from **LPSTR** to **LPTSTR**

TX\_DATAIN/OUT

The *lpFilterName* member of the TXDATAIO data structure has been changed from **LPSTR** to **LPCTSTR**.

The *lpImagePath* member of the TXFILTERIO data structure has been changed from **LPSTR** to **LPCTSTR** and the *lfDefFont/lfMonoFont* members have been changed from **char** to **TCHAR**. The buffer identified through *hDocTitle* contains either a Unicode or an ANSI string.

TX\_EXPORTTEXT

*lParam* changed from **LPSTR** to **LPTSTR**

TX\_FIELD\_CHANGETEXT

*lParam* changed from **LPSTR** to **LPTSTR**

TX\_FIELD\_GETTEXT

*lParam* changed from **LPSTR** to **LPTSTR**

TX\_FIELD\_INSERT

*lParam* changed from **LPSTR** to **LPTSTR**

TX\_GETDEVICE

*lParam* changed from **LPSTR** to **LPTSTR**

TX\_GETFONT

*lParam* changed from **LPSTR** to **LPTSTR**

TX\_GETIMAGE

*lParam* changed from **LPSTR** to **LPTSTR**

TX\_GETIMAGEFILTERS

The returned data buffer contains Unicode or ANSI strings.

TX\_GETSUPPORTEDFONTS

The returned data buffer contains Unicode or ANSI strings.

TX_GETSUPPORTEDSIZES	<i>lParam</i> changed from <b>LPSTR</b> to <b>LPTSTR</b> . The returned data buffer contains Unicode or ANSI strings.
TX_GETTEXT	<i>lParam</i> changed from <b>LPSTR</b> to <b>LPTSTR</b>
TX_IMPORTTEXTBUFFER	<i>lParam</i> changed from <b>LPSTR</b> to <b>LPTSTR</b>
TX_IMPORTTEXTFILE	<i>lParam</i> changed from <b>LPSTR</b> to <b>LPTSTR</b>
TX_OBJ_XXX	The <i>lpFileName</i> member of the TXOBJECT data structure has been changed from <b>LPSTR</b> to <b>LPTSTR</b> .
TX_REPLACESEL	<i>lParam</i> changed from <b>LPSTR</b> to <b>LPTSTR</b>
TX_SETDEVICE	<i>lParam</i> changed from <b>LPSTR</b> to <b>LPTSTR</b>
TX_SETFONT	<i>lParam</i> changed from <b>LPSTR</b> to <b>LPTSTR</b>
TX_SETIMAGE	<i>lParam</i> changed from <b>LPSTR</b> to <b>LPTSTR</b>
TX_TABLE_GETTEXTTOFCELL	The returned data buffer contains a Unicode or a ANSI string.
TX_TABLE_SETTEXTTOFCELL	The <i>lpcText</i> member of the SETTEXTTOFCELL data structure has been changed from <b>LPCSTR</b> to <b>LPCTSTR</b> .

**Function:**

CreateTextControl

**changes:**

The *lpLogFont* parameter can point to a LOGFONTA or a LOGFONTW data structure both defined through the Windows 32 bit SDK.

## 1.15 Resources

Text Control has several built-in resources like information strings, error messages and dialog boxes. These resources are available in different languages. When a new control is created Text Control selects the current set system language as the default one. With the TX\_SETLANGUAGE message this setting can be altered independent of the system language. The description of the TX\_SETLANGUAGE message lists all currently available built-in languages. To alter the language of the Button

Bar and Status Bar the appropriate messages `BBM_SETLANGUAGE` and `STB_SETLANGUAGE` must be used. See the descriptions of these messages in the appendixes D and E for a list of available languages.

To display resources in additional languages external resource libraries can be built and then set with the `TX_SETLANGUAGE` message through its file name. A resource library is a dynamic link library that only contains resources and an entry point. The `SAMPLES\TXRES` subdirectory contains the basic files to create such a DLL file. The following is a list of these files:

<code>TXRES.C</code>	Contains the DLL's entry point.
<code>TXRES.RC</code>	Contains Text Control's resources in English.
<code>TXRES.H</code>	Contains the definitions of all resource identifiers.

Furthermore Microsoft Visual C++ project files are contained that can be used to build the resource library.

The `TXRES.RC` file has the following contents:

Dialog boxes	Dialog box templates for the built-in dialog boxes which can be displayed with the <code>TX_FONTDIALOG</code> , <code>TX_PARAGRAPHDIALOG</code> and <code>TX_TABLE_ATTRDIALOG</code> messages.
String tables	The string tables contain information strings and error messages and the status strings of the status bar. Strings must not be larger than 255 characters.
Bitmaps	Bitmaps for the bold, italic and underline buttons of the button bar. The bitmap files are in the <code>TXRES\BMP</code> subdirectory.

To avoid conflicts with other programs that also uses own resources or with future versions of Text Control the following points are important:

1. The resource library should have a unique file name. The `TXRES` sample builds a DLL file named `TXRES.DLL`. This name should be changed.
2. The resource library should be placed in the same directory as the final application. Get the full path name of the application's executable file at run time and send the file name of the resource library including this path with the `TX_SETLANGUAGE` message.

At runtime Text Control determines resources in the following way:

1. When the TX\_SETLANGUAGE message is not used Text Control uses the system default language. If the system language is not built-in, Text Control displays English resources.
2. When TX\_SETLANGUAGE has been sent with an identifier of a built-in language Text Control displays resources in this language independent of the system language.
3. When TX\_SETLANGUAGE has been sent with a file name of a resource library Text Control tries to load the resources from this library. Previously sent language identifiers are ignored. When the resource library does not contain a needed resource or when the specified file could not be found Text Control displays English resources without reporting an error.
4. Setting a resource library for a Text Control does not automatically set the same library for a connected Button Bar or Status Bar. This must be performed with the appropriate messages of these windows.

## 2. *Mouse and Keyboard Interface*

### 2.1 *Mouse Assignment*

<u>Mouse Action</u>	<u>Reaction of Text Control</u>
Click	Moves cursor to point of click or selects an image.
Shift+Click	Extends the selection to the point of click.
Double-click	Selects the word that is clicked on or opens a modal dialog box to select an image alignment.
Drag	Selects text from point of button down to point where button is released.
Double-click and drag	Extends the selection from word to word.
Triple-click and drag	Extends the selection from row to row.

### 2.2 *Keyboard Assignment*

Moving the input position while SHIFT is pressed extends the current selection to the new caret position.

<u>Key type</u>	<u>Reaction of Text Control</u>
DEL	Deletes selected text.
END	Moves the caret to the end of the line.
HOME	Moves the caret to the beginning of the line.
PAGEUP	Scrolls the height of the current client area upwards and moves the caret to the same screen position. This key is only available once an internal scroll-interface has been set with the TX_SETTEXTAREA message.

---

PAGEDOWN	Scrolls the height of the current client area downwards and moves the caret to the same screen position. This key is only available once an internal scroll-interface has been set with the TX_SETTEXTAREA message.
(Left Arrow)	Moves the caret one character to the left.
(Right Arrow)	Moves the caret one character to the right.
(Up Arrow)	Moves the caret one line up.
(Down Arrow)	Moves the caret one line down.
CTRL+(Left Arrow)	Moves the caret to the beginning of the current word.
CTRL+(Right Arrow)	Moves the caret to the beginning of the next word.
CTRL+HOME	Moves the caret to start of text.
CTRL+END	Moves the caret to end of text.
CTRL+ENTER	Inserts a new page.
CTRL+(-)	Inserts an end-of-line hyphen.
CTRL+INS	Copies selected text to the clipboard.
CTRL+(Backspace)	Deletes the previous word.
SHIFT+ENTER	Creates a line feed.
SHIFT+DEL	Copies selected text to the clipboard and deletes the selection.
SHIFT+INS	Inserts text from the clipboard.
CTRL+SHIFT+(Spacebar)	Inserts a non-breaking space.
CTRL+'C'	Copies selected text to the clipboard.
CTRL+'V'	Inserts text from the clipboard.
CTRL+'X'	Copies selected text to the clipboard and deletes the selection.
CTRL+'Z'	Undos the last operation.

## 3. *Function Directory*

---

### CreateTextControl

**Syntax**      **HWND CreateTextControl**(*hWndParent*, *wChildID*, *lpRect*, *lpLogFont*)

This function creates a Text Control child window.

<u>Parameter</u>	<u>Type/Discription</u>
<i>hWndParent</i>	<b>HWND</b> Identifies the parent window of the Text Control window being created.
<i>wChildID</i>	<b>WORD</b> Is the child window identifier.
<i>lpRect</i>	<b>LPRECT</b> Points to a <b>RECT</b> data structure that contains the position and size of the Text Control window in client area coordinates of the parent window.
<i>lpLogFont</i>	<b>LPLOGFONT</b> Points to a <b>LOGFONT</b> data structure which defines the logical font the Text Control will use. The Text Control matches this font with an existing physical font of the currently selected printer. If no printer is present, the nearest physical screen font is used. If <i>lpLogFont</i> is 0L, the Text Control uses a default font.  If <i>lpLogFont</i> specifies a valid TX window handle, the function uses the font and paragraph attributes and all mode settings of that window for initialization. The TX window handle must be placed in the low-order word of <i>lpLogFont</i> , the high order word must be zero.

**Return Value**    The return value identifies the new Text Control window. It is zero if an error has occurred.

**Comments** Height and width values in the **LOGFONT** data structure must be specified in terms of twentieths of a point.

---

## TXGetErrorCode

**Syntax** **LONG TXGetErrorCode(void)**

This function returns an internal error code, and can be called if the parent window has received a **TN\_ERRCODE** notification or after a Text Control function has failed.

**Return Value** The return value contains an error number in the low-order word and a module number and a group code in the high-order word. The module number is 1 for the programming tool described in this manual, but it can also be the number of other modules the Text Control uses for special purposes.

The error numbers and group codes belonging to the Text Control kernel module are described in the error code table in appendix B. For a description of error codes that belong to other module numbers, see the corresponding reference manuals of these modules.

**Comments** To split the error code use the following syntax:

```
lResult = TXGetErrorCode();
ErrorNumber = LOWORD(lResult);
Module = LOBYTE(HIWORD(lResult));
GroupCode = HIBYTE(HIWORD(lResult));
```

---

## TXGetVersion

**Syntax** **LONG TXGetVersion(void)**

This function returns the current version number. The version number can be different from the text format number explained in appendix A.

**Return Value** The return value contains the version number in its low-order word. For example, for the release TX Text Control 4.0 the version number is 400.

## 4. Message Overview

### 4.1 Processed Windows Messages

<u>Message</u>	<u>Description</u>
WM_CHAR	Evaluates keyboard input.
WM_CLEAR	Clears the selection or a selected image.
WM_COMMAND	Processes notifications messages sent by Image-Control windows.
WM_COPY	Copies text and format data of the current selection to the clipboard.
WM_CREATE	Initialises internal data structures.
WM_CUT	Cuts selected text including format data to the clipboard.
WM_DESTROY	Deletes the internal data structures.
WM_DESTROYCLIPBOARD	Deletes internally saved data.
WM_ERASEBKGND	Prevents default background painting.
WM_GETDLGCODE	Returns a code which makes it possible to use a Text Control as a dialog box control.
WM_GETMINMAXINFO	Retrieves minimum and maximum values for the Text Control's window size.
WM_GETTEXT	Copies the text contained by the Text Control. If <i>wParam</i> contains zero the Text Control copies the whole text to the buffer <i>lParam</i> points to. In this case the buffer must be of the size returned by WM_GETTEXTLENGTH plus one byte for the terminating zero.
WM_GETTEXTLENGTH	Retrieves the amount of text (in bytes) contained by the Text Control.

---

WM_HSCROLL	Scrolls the Text Control's client area horizontally.
WM_KEYDOWN	Evaluates keyboard input.
WM_KILLFOCUS	Frees the caret and hides a current selection.
WM_LBUTTONDOWNBLCLK	Handles special features of the mouse interface.
WM_LBUTTONDOWN	Sets the focus and starts the mouse interface.
WM_LBUTTONUP	Ends the mouse interface.
WM_MOUSEMOVE	Handles special features of the mouse interface.
WM_MOVE	Moves the window to a new position.
WM_NCPAINT	Prevents default frame painting.
WM_PASTE	Inserts data from the clipboard.
WM_RENDERFORMAT	Copies text in RTF format to the clipboard.
WM_RENDERALLFORMATS	Copies text in RTF format to the clipboard.
WM_SETFOCUS	Shows the caret and displays a hidden selection.
WM_SETTEXT	Sets the text of a Text Control. The return value is TR_ERR if an error has occurred.
WM_SIZE	Formats the text for the new window size.
WM_TIMER	Handles special features of the mouse interface.
WM_VSCROLL	Scrolls the Text Control's client area vertically.
WM_WINDOWPOSCHANGING	Prevents the Text Control's formatting area from becoming too small to show at least one character.

All other windows messages are handled by **DefWindowProc**.

Additional messages have been implemented to access the Text Control's special features. An overview of these messages is presented in the following chapters.

## 4.2 *Font Messages*

<u>Message</u>	<u>Description</u>
TX_ENLARGEFONT	Enlarges or reduces the pointsize of all fonts in the current selection.
TX_FONTDIALOG	Opens a modal dialog box to select a font, its size and its attributes for the current selection. The fonts and sizes that belong to the current standard output device are displayed.
TX_GETBASELINE	Returns the current baseline alignment value.
TX_GETFONT	Retrieves the common typeface and pointsize of all currently selected fonts.
TX_GETFONTATTR	Returns a bit mask that contains font attribute information for the current selection. Can be used to place checkmarks for menu items concerning font attributes.
TX_GETTEXTCOLOR	Returns the current text color value.
TX_SETBASELINE	Sets a new baseline alignment.
TX_SETFONT	Sets a new typeface and pointsize for all currently selected fonts.
TX_SETFONTATTR	Sets new font attributes.
TX_SETTEXTCOLOR	Sets a new text color for the currently selected text.

## 4.3 *Paragraph Messages*

<u>Message</u>	<u>Description</u>
TX_GETFORMAT	Returns the text alignment of the current selection. Can be used to place checkmarks for menu items concerning paragraph attributes.
TX_GETINDENTS	Retrieves indent values.
TX_GETLINESPACING	Returns the linespacing value.

---

TX_GETPARAFORMATFLAGS	Returns a bit-field indicating special paragraph formats.
TX_GETPGFRAME	Returns paragraph frame attributes.
TX_GETTABS	Retrieves common tab types and positions of all selected paragraphs.
TX_PARAGRAPHDIALOG	Opens a modal dialog box for setting paragraph attributes.
TX_SETFORMAT	Sets the text alignment of all selected paragraphs.
TX_SETINDENTS	Sets new indent values for all selected paragraphs.
TX_SETLINESPACING	Sets a new line spacing value for all selected paragraphs.
TX_SETPARAFORMATFLAGS	Sets special paragraph formats.
TX_SETPGFRAME	Defines frame attributes for all selected paragraphs.
TX_SETTABS	Sets a new tablist for all selected paragraphs.

## 4.4 *Text Manipulation Messages*

These messages can be used to manipulate the text and format data of a text control.

<b><u>Message</u></b>	<b><u>Description</u></b>
TX_CANCOPY	Returns TRUE when something can be copied to the clipboard.
TX_CANPASTE	Returns TRUE when something can be pasted from the clipboard.
TX_COPYDATA	Copies the data of a Text Control into a buffer.
TX_GETDATASIZE	Returns the size of all formatting and text data. This is needed to allocate the buffer before sending the TX_COPYDATA message.

---

TX_GETTEXT	Retrieves text in the Text Control's text format without formatting information.
TX_GETTEXTSIZE	Returns the size of the current text in the Text Control's text format.
TX_PASTEDATA	Sets the data of a Text Control that was previously saved with the TX_COPYDATA message.
TX_REPLACESEL	Replaces the currently selected text with text out of a global buffer.

## 4.5 *Messages to Adapt Formatted Text to a new Output Device*

<u>Message</u>	<u>Description</u>
TX_DEVMODECHANGE	Adjusts the font information of a Text Control. This message has to be sent to all existing Text Controls as a reaction to receiving a WM_DEVMODECHANGE or a WM_WININICHANGE message.
TX_GETDEVICE	Retrieves the name of the currently set device for which the text has been formatted.
TX_SETDEVICE	Registers a new device for which the text will be formatted.

## 4.6 *Print Messages*

<u>Message</u>	<u>Description</u>
TX_GETPAGECOUNT	Returns the current number of pages.
TX_PRINT	Prints a specified portion of the text.
TX_PRINTPAGE	Prints a single page.

## 4.7 File IO Messages

<u>Message</u>	<u>Description</u>
TX_DATAIN	Inserts text data into a Text Control.
TX_DATAOUT	Retrieves text data from a Text Control.
TX_LOAD	Fills the buffer of an empty Text Control with formatted text out of a file. The text must have previously been saved by sending a TX_SAVE message.
TX_RESETCONTENTS	Deletes the complete contents of the Text Control.
TX_SAVE	Stores all the formatted text of a Text Control to a file.

## 4.8 Information Messages

<u>Message</u>	<u>Description</u>
TX_GETBASELINEPOS	Returns the position of the baseline for a specified line number.
TX_GETBKGNDCOLOR	Returns the background color.
TX_GETCARETEXT	Returns the caret size.
TX_GETCARETPOS	Returns the caret position. Together with the caret notifications it can be used to implement auto scrolling.
TX_GETLANGUAGE	Returns a language identifier.
TX_GETLINEANDCOL	Returns the line and the column of the current caret position.
TX_GETLINECOUNT	Returns the number of text lines.
TX_GETLINERECT	Retrieves the size and alignment of the line part, covered with text.
TX_GETMODE	Returns the mode flags.

TX_GETMODEEX	Returns the expanded mode flags.
TX_GETRECT	Returns the size of the Text Control window regardless of the zoom factor.
TX_GETSEL	Returns the selection.
TX_GETSUPPORTEDFONTS	Returns information about all supported font families.
TX_GETSUPPORTEDSIZES	Returns information about all supported font sizes for a specified font family.
TX_GETTEXTTEXTENT	Returns the size of the smallest bounding rectangle for the text inside the Text Control. Can be used to adjust the size of the Text Control to the text that it contains.
TX_GETTEXTHEIGHT	Returns the height of the text.
TX_GETTEXTWIDTH	Returns the Text Control's longest line width.
TX_GETZOOM	Returns the zooming factor.
TX_INPUTPOSFROMPOINT	Returns the text input position belonging to a given point in the Text Control window's visible area.
TX_LINEFROMCHAR	Returns the line number of the line which contains the character whose position (indexed from the beginning of the text) is specified by the wParam parameter.
TX_LINEFROMPOINT	Returns the line number of the line which contains the given point. The point must specify pixel coordinates with an origin at the left top corner of the window.
TX_LINEINDEX	Returns the number of characters that precede the first character in a given line.

## 4.9 Messages to Process Images

<u>Message</u>	<u>Description</u>
TX_CREATEIMAGE	Inserts a new image at the current caret position.
TX_GETIMAGE	Retrieves the DOS pathname for the image file registered by the currently selected Image-Control window.
TX_GETIMAGEFILTERS	Returns a buffer containing information about the available image filters.
TX_SETIMAGE	Sets a new image for the currently selected Image-Control window.

## 4.10 Messages to Handle Chains of Linked Windows

<u>Message</u>	<u>Description</u>
TX_GETLINKWND	Searches for a handle of a window that is part of linked Text Control windows.
TX_SETLINKWND	Sets a new following window for a Text Control window.

## 4.11 Messages to Handle Marked Text Fields

<u>Message</u>	<u>Description</u>
TX_FIELD_CHANGETEXT	Alters the text of a marked text field.
TX_FIELD_DELETE	Deletes a marked text field.
TX_FIELD_FROMCARETPOS	Returns the field identifier of the field containing the current input position.
TX_FIELD_GETATTR	Returns the attributes of a marked text field.
TX_FIELD_GETCURRENT	Returns a field identifier.

TX_FIELD_GETDATA	Retrieves the data related to a marked text field with the TX_FIELD_SETDATA message.
TX_FIELD_GETNEXT	Returns the identifier of the next field from a given field.
TX_FIELD_GETPOSITION	Returns the position of a marked text field.
TX_FIELD_GETTEXT	Retrieves the text of a marked text field.
TX_FIELD_GETTYPE	Returns the type of a marked text field.
TX_FIELD_GOTO	Sets the current input position to the beginning of a marked text field.
TX_FIELD_INSERT	Creates a marked text field.
TX_FIELD_SETATTR	Sets the attributes of a marked text field.
TX_FIELD_SETTYPE	Defines a marked text field of a special type.
TX_FIELD_SETDATA	Relates any data to a marked text field.

## ***4.12 Messages to Perform Undo Operations***

<b><u>Message</u></b>	<b><u>Description</u></b>
TX_CANUNDO	Returns the information about whether an operation can be undone or redone.
TX_EMPTYUNDOBUFFER	Resets the undo flag.
TX_REDO	Restores the last undone operation.
TX_UNDO	Undoes the last Text Control operation.

## ***4.13 Messages to Handle Integrated Objects***

<b><u>Message</u></b>	<b><u>Description</u></b>
TX_OBJ_DELETE	Deletes an integrated object.
TX_OBJ_EMBED	Integrates a new object in the text.
TX_OBJ_GETATTR	Retrieves attributes of an integrated object.

TX_OBJ_GETDISPINTERFACE	Returns a pointer to an object's dispatch interface.
TX_OBJ_GETNEXT	Returns the identifier of the next object in the object list.
TX_OBJ_OLE_CANCEL	Deactivates an OLE Object.
TX_OBJ_SETATTR	Sets an object's attributes.

#### ***4.14 Messages to Handle the Internal Scroll Interface***

<b><u>Message</u></b>	<b><u>Description</u></b>
TX_GETPAGEMARGINS	Retrieves the page margins.
TX_GETSCROLLPOS	Returns the current scroll position.
TX_GETTEXTAREA	Retrieves the Text Control's formatting area.
TX_SETPAGEMARGINS	Sets new page margins.
TX_SETSCROLLPOS	Sets a new scroll position.
TX_SETTEXTAREA	Sets the text formatting area and/or a scroll interface.

#### ***4.15 Messages to Handle Find and Replace Features***

<b><u>Message</u></b>	<b><u>Description</u></b>
TX_FINDTEXT	Finds a text string.
TX_REPLACETEXT	Replaces a text string.

#### ***4.16 Table Messages***

<b><u>Message</u></b>	<b><u>Description</u></b>
TX_TABLE_ATTRDIALOG	Opens a dialog box for setting new attributes for the cells of a table.
TX_TABLE_CHANGEID	Changes a table's identifier.

TX_TABLE_DELETELINES	Deletes the currently selected table lines.
TX_TABLE_FROMCARETPOS	Retrieves the table identifier and the number of row and column for the current input position.
TX_TABLE_GETATTROFCELL	Retrieves information about the attributes of one or more table cells.
TX_TABLE_GETCELLATTR	Provides information about the attributes of all selected table cells.
TX_TABLE_GETCELLPOSITION	Retrieves the indexes of the first and the last character in a table cell.
TX_TABLE_GETCOLPOSITIONS	Provides information about the positions of table columns.
TX_TABLE_GETNEXT	Returns the identifier of a table following a specified table.
TX_TABLE_GETROWSANDCOLS	Returns the number of rows and columns in a table.
TX_TABLE_GETTEXTTOFCELL	Gets a table cell's text.
TX_TABLE_INSERT	Inserts a new table into the text.
TX_TABLE_ISPOSSIBLE	Provides information about whether certain table operations are possible.
TX_TABLE_SETATTROFCELL	Alters one or more attributes of one or more table cells.
TX_TABLE_SETCELLATTR	Sets new attributes for all selected table cells.
TX_TABLE_SETCOLPOSITIONS	Sets new column positions within a table.
TX_TABLE_SETTEXTTOFCELL	Changes a table cell's text.

## ***4.17 Messages to Handle Headers and Footers***

<b><u>Message</u></b>	<b><u>Description</u></b>
TX_HF_ACTIVATE	Activates or deactivates a header or footer.

---

TX_HF_DISABLE	Disables the usage of headers and/or footers.
TX_HF_ENABLE	Enables the usage of headers and/or footers.
TX_HF_GETENABLED	Returns the currently enabled headers and/or footers.
TX_HF_GETPOSITION	Returns a header's or footer's position on the page.
TX_HF_SELECT	Selects a header or footer for message routing.
TX_HF_SETPOSITION	Sets a header's or footer's position on the page.

## 4.18 Additional Features

<u>Message</u>	<u>Description</u>
TX_LIMITLINE	Limits the number of characters that can be entered into a single line.
TX_LIMITTEXT	Limits the number of characters the user can enter.
TX_SELTEST	Checks whether a selection is visible.
TX_SETBKGNDCOLOR	Sets a new background color.
TX_SETCARETEXT	Defines the width of the caret.
TX_SETLANGUAGE	Defines the language for a Text Control.
TX_SETLINEANDCOL	Sets a new text input position, given through a page, line and column number.
TX_SETMODE	Sets different mode flags like the background mode or the character insertion mode.
TX_SETMODEEX	Sets expanded mode flags.
TX_SETSEL	Sets and displays a new text selection.
TX_SETWORDDIVISION	Informs TX that an application-supplied function should be used to perform word-division.
TX_ZOOM	Sets a new zooming factor.

## 4.19 Notification Messages

A Text Control sends the following notification messages through a WM\_COMMAND message to its parent window to inform the application about special conditions.

<b><u>Message</u></b>	<b><u>Description</u></b>
TN_AUTOLINK	Notifies the parent window that a chain of linked windows have to be expanded.
TN_AUTOSCROLL	Can be used to scroll in special cases.
TN_CARETOUTBOTTOM	Caret is moved down out of the visible area.
TN_CARETOUTLEFT	Caret is moved left out of the visible area.
TN_CARETOUTRIGHT	Caret is moved right out of the visible area.
TN_CARETOUTTOP	Caret is moved up out of the visible area.
TN_CARETOUT	Caret is moved to a Text Control that is completely outside the visible area.
TN_CHANGED	Notifies the parent window that the user has modified the text or the formatting.
TN_CHARFORMATCHANGED	Notifies the parent window that formatting attributes of the selected characters have been changed.
TN_DOUBLECLICKED	Notifies the parent window that a word has been doubleclicked and selected.
TN_ERRCODE	Notifies the parent window that an error has occurred.
TN_FIELD_CHANGED	The text of a marked text field has been changed.
TN_FIELD_CLICKED	The user has clicked on a marked text field.
TN_FIELD_CREATED	A new marked text field has been created.
TN_FIELD_DBLCLICKED	The user has doubleclicked on a marked text field.
TN_FIELD_DELETED	A marked text field has been deleted.

---

TN_FIELD_ENTERED	The current input position has been moved to a marked text field.
TN_FIELD_LEFT	The current input position has been moved from a marked text field.
TN_FIELD_LINKCLICKED	Occurs when a field has been clicked on that is the source of a hypertext link.
TN_FIELD_SETCURSOR	Enables the application to select its own cursor to indicate marked text fields.
TN_FORCEUPDATE	Is sent when Text Control's tool bars should update their contents and must be passed on to connected toolbars.
TN_HEXPAND	The Text Control window has been expanded horizontally.
TN_HF_ACTIVATED	Informs about header or footer activation.
TX_HF_DEACTIVATED	Informs about header or footer deactivation.
TN_HMOVED	The Text Control window has been moved horizontally.
TN_HSCROLL	The Text Control's client area has been scrolled horizontally.
TN_IMAGECLICKED	An Image-Control window has been clicked and selected.
TN_KEYSTATECHANGED	Informs the parent window that the user has pressed special control characters.
TN_KILLFOCUS	The Text Control has lost the input focus.
TN_OBJ_CLICKED	Informs the parent window that an embedded object has been clicked.
TN_OBJ_CREATED	Informs the parent window that an embedded object has been created.
TN_OBJ_DBLCLICKED	Informs the parent window that an embedded object has been doubleclicked.
TN_OBJ_DELETED	Informs the parent window that an embedded object has been deleted.

---

TN_OBJ_MOVED	Informs the parent window that an embedded object has been moved.
TN_OBJ_SIZED	Informs the parent window that an embedded object has been sized.
TN_PAGEFORMATCHANGED	Informs the parent window that the page format settings have been altered.
TN_PGCHANGED	Informs the parent window that the character input position has been moved to another paragraph.
TN_PGFORMATCHANGED	Informs the parent window that paragraph formatting attributes of the selected paragraphs have been changed.
TN_POSCHANGED	Informs the parent window that the character input position has been changed.
TN_SETFOCUS	The Text Control has obtained the input focus.
TN_TABLE_CREATED	Sent when a new table has been created.
TN_TABLE_DELETED	Sent when a new table has been deleted.
TN_VEXPAND	The Text Control window has been expanded vertically.
TN_VSCROLL	The Text Control's client area has been scrolled vertically.
TN_ZOOMED	The Text Control has been zoomed.

## ***4.20 Obsolete Messages***

TX_ADJUSTCLIPBOARD	Adjusts the clipboard's font information.
TX_EXPORTTEXT	Converts text to an external format with the help of a filter and copies it to a file or to a buffer.
TX_GETASCIITEXT	Retrieves text in a Windows compatible text format.

---

TX_GETASCIITEXTSIZE	Returns the size of the current text in a Windows compatible text format.
TX_GETHANDLE	Returns the global handle of a buffer that holds the text and the format attributes of the Text Control. Is used to destroy the old buffer of a Text Control after sending the TX_SETHANDLE message.
TX_GETIMAGEFORMAT	Retrieves formatting parameters for an image.
TX_IMPORTTEXTBUFFER	Imports externally formatted text from a buffer, with the help of a filter.
TX_IMPORTTEXTFILE	Imports externally formatted text from a file, with the help of a filter.
TX_SETHANDLE	Sets a new data handle for the Text Control. It has to be a global handle.
TX_SETIMAGEFORMAT	Sets new formatting parameters for an image.

---

## 5. Message Directory

---

### TX\_CANCOPY

This message can be sent to determine whether part of a Text Control's document has been selected and can be copied to the clipboard.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is nonzero if something can be copied to the clipboard. Otherwise it is zero.

---

### TX\_CANPASTE

This message can be sent to determine whether the clipboard contains a format that can be pasted into a Text Control's document.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is nonzero if something can be pasted. Otherwise it is zero.

---

### TX\_CANUNDO

This message returns whether a Text Control operation can be undone or whether an undone operation can be restored.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.

*lParam* Is not used.

**Return Value** The low-order word of the return value is zero if there is no Text Control operation that can be undone. Otherwise it is one of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
UNDO_INSERT	The next undo operation deletes inserted text.
UNDO_DELETE	The next undo operation inserts deleted text.
UNDO_FORMAT	The next undo operation resets the last formatting operation.

The high-order word of the return value is zero if there is no undone operation that can be restored. Otherwise it is one of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
REDO_INSERT	The next redo operation restores inserted text.
REDO_DELETE	The next redo operation deletes restored text.
REDO_FORMAT	The next redo operation restores the last formatting operation.

---

## TX\_COPYDATA

This message is used to copy the complete text and all the format information of a Text Control window to a buffer pointed to by *lParam*.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a buffer that is to receive the data.

**Return Value** The return value is a pointer to the next free position behind the copied data. It is zero if an error has occurred.

**Comments** This message is implemented to save the data of a Text Control to a global buffer. For more information on how to restore the data from memory, see the description of the TX\_PASTEDATA message. To save the data to a file, use the TX\_SAVE message. The required buffer size can be obtained with the TX\_GETDATASIZE message. For other applications its format is described in appendix A.

---

## TX\_CREATEIMAGE

This message inserts a new image at the current text input position.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies an image filter as an index into the buffer returned by the TX_GETIMAGEFILTERS message. The first pair of strings has an index value of 1. If the buffer returned by TX_GETIMAGEFILTERS is used to initialize the <i>lpstrFilter</i> member of an <b>OPENFILENAME</b> structure, another member of that structure, <i>nFilterIndex</i> , can be used to initialize this parameter. See the Windows SDK for more information about the <b>OPENFILENAME</b> structure. If <i>wParam</i> is set to 0, the Text Control automatically tries to select a filter.
<i>lParam</i>	Points to a null-terminated string that is a full DOS path name for the file containing the new image.

**Return Value** The low-order word of the return value is zero if an error has occurred. Otherwise it is nonzero. If an error has occurred, the high-order word contains an error code value. For more information about the meaning of this value see the description of the IC\_SETIMAGE message in the IC Image-Control reference.

**Comments** The IC Image-Control programming tool is needed for this message.

---

## TX\_DATAIN

This message inserts text data into a Text Control.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to the TXDATAIO structure that has been defined for general data exchange with a Text Control. The following comments section describes this structure in more detail.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** The TXDATAIO structure is defined as follows:

```
typedef struct tagTXDATAIO {
    WORD        wTXVersion;
    HFILE       hFile;
    LPVOID      lpreserved;
    LPVOID      hpInBuffer;
    HGLOBAL     hOutBuffer;
    WORD        wFormat;
    LPCTSTR     lpFilterName;
    BOOL        bCurSelection;
    POINT       ptMinSize;
    DWORD       dwBytesReadWritten;
    LONG        lReserved;
    LPTXFILTERIO lpFilterIO;
} TXDATAIO;
```

The TXDATAIO structure has the following fields:

<u>Field</u>	<u>Description</u>
<i>wTXVersion</i>	Specifies the Text Control's current version number in the same format returned by the <i>TXGetVersion</i> function, e.g. 500. Set this parameter as a number and not as a <i>TXGetVersion</i> function call.
<i>hFile</i>	Identifies a file where the data is to be read from or written to. The file pointer is moved, and the number of copied bytes is stored to the <i>dwBytesReadWritten</i> member. When this member is set to <i>HFILE_ERROR</i> , a pointer to a memory buffer must be specified with the <i>hpInBuffer</i> member for data input. For data output the Text Control allocates a buffer and copies the handle to the <i>hOutBuffer</i> member.
<i>lpreserved</i>	Reserved for future use. This member must be set to zero.
<i>hpInBuffer</i>	For data input only: points to a buffer containing incoming data

and should be set to zero when *hFile* is specified. The number of bytes read is copied to the *dwBytesReadWritten* member.

*hOutBuffer*

For data output only: when *hFile* is not specified Text Control creates a buffer with the outgoing data and copies its global data handle to this variable. The user must use the **GlobalFree** function after using the buffer to free it. The amount of data, in bytes, is copied to the *dwBytesReadWritten* member.

*wFormat*

Specifies a format identifier. This member is only used when the *lpFilterName* member is set to zero. The identifiers TF\_FORMAT\_TEXT and TF\_FORMAT\_TX are implemented as ANSI (TF\_FORMAT\_TEXTA and TF\_FORMAT\_TXA) and Unicode versions (TF\_FORMAT\_TEXTW and TF\_FORMAT\_TXW). Depending on whether Unicode is defined or not either the A- or the W-version is used. The following values are possible:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TF_FORMAT_ANSI	Text only in ANSI format (Windows compatible).
TF_FORMAT_UNICODE	Text only in Unicode format (Windows compatible).
TF_FORMAT_TEXT	Text only in ANSI or Unicode format (Text Control compatible), depending on whether UNICODE is defined or not before TX.H is included. To enforce a certain format use TF_FORMAT_TEXTA or TF_FORMAT_TEXTW explicitly.
TF_FORMAT_TX	Text and formatting attributes using Text Control's text format. Text is stored in ANSI or Unicode format, depending on whether UNICODE is defined or not before TX.H is included. To enforce a certain format use TF_FORMAT_TXA or TF_FORMAT_TXW explicitly.

---

	TF_FORMAT_HTML	HTML (Hypertext Markup Language)
	TF_FORMAT_RTF	RTF (Rich Text Format)
	TF_FORMAT_WORD	Microsoft WORD format.
	TF_FORMAT_USER	The <i>lpFilterName</i> parameter must contain the name of a user-developed filter. User-developed filters must be able to process Unicode. The development of a text filter is described in appendix C.
<i>lpFilterName</i>		Points to a null-terminated character string that specifies the name of the filter library that is to be used to convert the data. If the filter library is not in the same directory as the TX library, the character string must contain the complete path. This parameter is only used when <i>wFormat</i> contains TF_FORMAT_USER.
<i>bCurSelection</i>		When set to TRUE the current selection is replaced or saved. Otherwise the complete contents of the Text Control are replaced or saved. When the selection is replaced the new current input position is behind the inserted data, otherwise it is at the beginning of the text.
<i>ptMinSize</i>		For data input only: the Text Control copies the minimum window size (in pixels) to this buffer when the control is too small to display a minimum amount of the text. In this case the message returns zero to indicate an error. If a text area has been defined with the TX_SETTEXTAREA message the new minimum text area is copied in twentieths of a point. The size is calculated from the font information of the loaded text. If both values are 0, an error has occurred.
<i>dwBytesReadWritten</i>		Text Control fills this member with the number of read or written bytes. This value is set to zero when an error has occurred and set to -1 when the value could not be calculated, for example when a filter does not support this information.
<i>lReserved</i>		This member is for future use and should be set to zero.
<i>lpFilterIO</i>		Points to a TXFILTERIO structure when additional data is to be exchanged with the filter. This member is ignored and should be

set to zero when no filter is necessary or the additional data is not needed. A complete description of all members and their usage can be found in chapter 7 "*Data Structures*".

Text only is available either Windows compatible or Text Control compatible. To exchange text with other Windows programs that do not support RTF or HTML the Windows compatible text only format should be used. Otherwise to work with selections and lines the Text Control compatible text only format should be used. Control characters used with the Text Control compatible text only format are listed in appendix A.

---

## TX\_DATAOUT

This message retrieves text data from a Text Control in a certain format.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a TXDATAIO structure that has been defined for general data exchange with a Text Control. The Comments section of the TX_DATAIN message describes this structure in more detail.

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

---

## TX\_DEVMODECHANGE

This message adjusts the font information of a Text Control window when the user changes the standard printer. It must be sent to a Text Control window every time the application receives a WM\_DEVMODECHANGE message from Windows. If another device as the standard printer has been set with the TX\_SETDEVICE message sending this message is not necessary.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a POINT data structure. The Text Control copies the new minimum window size (in pixels) to this buffer.

This size is calculated from the adjusted fonts. If the values copied are both 0, an error has occurred.

**Return Value** The return value is zero if an error has occurred or if a particular printer has been set with the TX\_SETDEVICE message. Otherwise it is nonzero.

**Comments** The return value is also zero if the window size is too small for the new fonts. After resizing the window with the minimum values given by *lParam*, this message can be sent again.

---

## TX\_EMPTYUNDOBUFFER

This message clears the undo flag of a Text Control. The undo flag is set whenever an operation within the Text Control can be undone.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** This message does not return a value.

**Comments** The undo flag is automatically cleared whenever the Text Control receives one of the following messages: TX\_LIMITTEXT, TX\_LIMITLINE, TX\_LOAD, TX\_PASTEDATA, TX\_PRINT, TX\_PRINTPAGE, TX\_SAVE, TX\_SETDEVICE, TX\_SETHANDLE, TX\_SETLINKWND and WM\_SETTEXT.

---

## TX\_ENLARGEFONT

Enlarges or reduces the pointsizes of all fonts in the current selection. After modifying the fonts, the Text Control is correctly updated.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	contains one of the following values:

	<b><u>Value</u></b>	<b><u>Meaning</u></b>
	TF_REDUCE	If the fonts are to be reduced by one pointsize.
	TF_ENLARGE	If the fonts are to be enlarged by one pointsize.
<i>lParam</i>	Points to a POINT data structure. The Text Control copies the new minimum window size (in pixels) to this buffer. If a text area has been defined with the TX_SETTEXTAREA message the new minimum text area is copied in TWIPS. This size is calculated from the enlarged or reduced fonts. If the values copied are both 0, an error has occurred.	

***Return Value*** The return value is:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TR_ERR	If an error has occurred.
TR_UNCHANGED	If no font could be changed because the maximum or minimum sizes have been reached.
TR_CHANGED	Otherwise.

***Comments*** The return value also evaluates TR\_ERR if the window size is too small for the enlarged fonts. After resizing the window with the minimum values given by *lParam*, this message can be sent again.

The Text Control updates the modified selection if the window is visible.

---

## **TX\_FIELD\_CHANGETEXT**

This message alters the text of a marked text field.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a field identifier.
<i>lParam</i>	Points to a null-terminated string which is the altered text.

**Return Value** The return value is zero if an error has occurred or if the specified field identifier does not exist. Otherwise it is nonzero.

---

## TX\_FIELD\_DELETE

This message deletes a marked text field. The field is deleted independent of its attributes.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a field identifier.
<i>lParam</i>	If lParam is zero this message removes only the field property. If lParam is nonzero the field including its text is deleted.

**Return Value** The return value is zero if an error has occurred or if the specified field identifier does not exist. Otherwise it is nonzero.

**Comments** If a marked text field is deleted with this message, the Text Control does not send a TN\_FIELD\_DELETED notification message.

---

## TX\_FIELD\_FROMCARETPOS

This message returns the field identifier of the field containing the current input position.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is the identifier of the field containing the input position. Zero is returned when the input position is not inside a field.

## TX\_FIELD\_GETATTR

This message returns the attributes of a marked text field. These attributes are described in more detail in chapter 1.8 "*Marked Text Fields*".

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a field identifier.
<i>lParam</i>	Is not used.

### ***Return Value***

The return value is zero if the specified field does not exist. Otherwise it is a combination of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TF_DELETEABLE	Set if the marked text field can be deleted.
TF_UNDELETEABLE	Set if the marked text field cannot be deleted.
TF_CHANGEABLE	Set if the text of the specified marked text field can be changed.
TF_UNCHANGEABLE	Set if the text of the specified marked text field cannot be changed.
TF_EXTEEDITMODE	Set if the specified marked text field can be edited with a second input position at the beginning and the end of a field.
TF_NORMALEEDITMODE	Set if the specified marked text field is edited in normal mode.
TF_SHOWCURFIELDGRAY	Set if the specified marked text field is displayed with a gray background when it contains the current character input position.
TF_SHOWCURFIELDNORMAL	Set if the specified marked text field is not displayed with a gray background.
TF_USEFIELDCARET	Set if the caret for marked text fields is used in the specified field. This caret can be defined with the TX_SETCARETEXT message.

---

TF_USETEXTCARET	Set if the normal text caret is used in the specified field.
TF_ENABLEDBLCLICKS	Set if normal double-click processing is performed inside marked text fields, which starts a wordwise selection.
TF_DISABLEDBLECLICKS	Set if the normal double-click processing inside marked text fields is disabled.

---

## TX\_FIELD\_GETCURRENT

This message returns the identifier of a marked text field, when the parent window has received a notification message for that field.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is the identifier of the marked text field. It is zero if no such field exists.

**Comments** For chains of linked windows this message must be sent to the window that has sent the notification message.

---

## TX\_FIELD\_GETDATA

This message retrieves the data related to a marked text field with the TX\_FIELD\_SETDATA message.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a field identifier.
<i>lParam</i>	Points to a TXFIELDSETDATA data structure which is used to retrieve the data. See the description of the TX_FIELD_SETDATA message for more information about this data structure.

**Return Value** The return value is zero if the specified field does not exist. Otherwise it is non-zero. When the specified field exists but no data has been related to the field, the return value is non-zero; but all members of the TXFIELDSETDATA structure to which *lParam* points are set to zero.

---

## TX\_FIELD\_GETNEXT

This message returns the identifier of a marked text field that follows the specified field in the Text Control's current text. In a list of linked Text Controls the search is performed in all windows.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a field's identifier. If this parameter is zero, the first field's identifier is returned.
<i>lParam</i>	The low-order word specifies the group of fields. It can be a combination of any of the values described in the following Comments section. If <i>lParam</i> is zero the identifiers of all the fields are returned.

**Return Value** The return value is the identifier of the field which follows the specified field in the Text Control's text. It is zero if no following fields exist.

**Comments** The low-order word of the *lParam* parameter can be a combination of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
FGN_CHANGEANDDELETEABLEONLY	Returns only identifiers of fields which are changeable and deleteable.
FGN_UNCHANGEABLEONLY	Returns only identifiers of fields which are unchangeable.
FGN_UNDELETEABLEONLY	Returns only identifiers of fields which are undeleteable.
FGN_EXTERNALLINK	Returns only identifiers of fields that have the type FT_EXTERNALLINK.

FGN_HIGHLIGHT	Returns only identifiers of fields that have the type FT_HIGHLIGHT.
FGN_INTERNALLINK	Returns only identifiers of fields that have the type FT_INTERNALLINK.
FGN_LINKTARGET	Returns only identifiers of fields that have the type FT_LINKTARGET.
FGN_PAGENUMBER	Returns only identifiers of fields that have the type FT_PAGENUMBER.
FGN_TOPIC	Returns only identifiers of fields that have the type FT_TOPIC.

**Example**

The following code example uses the TX\_FIELD\_GETNEXT message to get the character positions of all the fields a Text Control contains:

```
DWORD dwPosition[2];
WORD wID = 0;

while (wID = (WORD)SendMessage(hwndTX, TX_FIELD_GETNEXT, wID, 0L)) {
    SendMessage(hwndTX, TX_FIELD_GETPOSITION,
        (WPARAM)wID, (LPARAM)(LPDWORD)dwPosition);
}
```

**TX\_FIELD\_GETPOSITION**

This message retrieves the start and end character positions of a marked text field.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a field identifier.
<i>lParam</i>	Points to an array of two DWORD variables. The first variable receives the start position and the second variable receives the end position of the field.

**Return Value** The return value is zero if an error has occurred or if the specified field identifier does not exist, otherwise it is nonzero.

**Comments** The start position is the one-based character position of the first character associated with the field. The end position is the one-based character position of the last character associated with the field. If a marked text field contains no text the end position is one less than the start position.

If marked text fields are used in chains of linked windows the position values are relative to the beginning of the text that is the first character in the first window of the chain. To get the window which contains the field and the alignment of the field in that window use the TX\_GETLINKWND message with the GWTX\_FROMOFFSET and the GWTX\_GETOFFSET option.

---

## TX\_FIELD\_GETTEXT

This message retrieves the text of a marked text field.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a field identifier.
<i>lParam</i>	Points to the buffer that is to receive the text. The size of the buffer must be previously calculated with the TX_FIELD_GETPOSITION message.

**Return Value** The return value is zero if an error has occurred or if the specified field identifier does not exist. Otherwise, it is nonzero.

**Comments** The buffer which *lParam* points to must be large enough to add a terminating zero. It can be larger than 64 kB.

**Unicode** The TX\_FIELD\_GETPOSITION message retrieves character positions. For Unicode and double-byte processing applications the character based calculation must be doubled to get the buffer size in bytes.

---

## TX\_FIELD\_GETTYPE

This message retrieves the type-related data belonging to a marked text field of a special type.

---

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a field identifier.
<i>lParam</i>	Points to a TXFIELDSETTYPE data structure. This structure retrieves the type of the field and its type-related data. See the description of the TX_FIELD_SETTYPE message for more information about this data structure.

**Return Value** The return value is zero if the specified field does not exist. Otherwise it is non-zero.

---

## TX\_FIELD\_GOTO

This message sets the current input position at the beginning of the specified marked text field and scrolls the text so that this position is at the top of the window's client area.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies the type of the field. The description of the <i>nFieldType</i> member of the TXFIELDSETTYPE data structure (TX_FIELD_SETTYPE message) lists all possible values.
<i>lParam</i>	Identifies the marked text field to which should be scrolled. It must be a valid field identifier. For fields of the type FT_LINKTARGET this parameter can also be the name of the field. For fields of the type FT_TOPIC this parameter can also be a valid topic number.

**Return Value** The return value is zero if the specified field does not exist. Otherwise it is non-zero.

---

## TX\_FIELD\_INSERT

This message inserts a new marked text field at the current input position or defines selected text as a marked text field.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies attributes for the new field. See the TX_FIELD_SETATTR message for more information.
<i>lParam</i>	Points to a null-terminated string which is the text to be inserted. If <i>lParam</i> is zero the currently selected text is defined as the new field. If selected text exists this parameter is ignored.

**Return Value** The return value is an identifier for the newly created field. It is zero if an error has occurred. The return value is also zero if the given string is empty or no selection exists.

**Comments** The returned identifier can be used to manipulate the text and the formatting attributes of the marked text field. The parent window is informed through notification messages when the user has clicked or doubleclicked a marked text field or if text is changed or deleted. If the cursor is moved over a field, it changes to a vertical arrow cursor.

If *lParam* is zero and the current selection covers an existing field, a new field cannot be created and the return value is zero.

For chains of linked Text Control windows this message must be sent to the window containing the current input position.

If the field should be visible for the user, additional attributes like colors or font attributes can be set. The attributes must be set before sending this message. If *lParam* is nonzero the attributes must also be reset after sending this message.

**Example** The following example inserts a green colored marked text field:

```
DWORD dwColor;  
LONG lReturn;  
lReturn = SendMessage(hWndTX, TX_GETTEXTCOLOR, 0,  
    (LPARAM)(LPDWORD)&dwColor);  
SendMessage(hWndTX, TX_SETTEXTCOLOR, 0, RGB(0, 255, 0));  
wField = (WORD)SendMessage(hWndTX, TX_FIELD_INSERT, 0,  
    (LPARAM)(LPSTR)buf);  
SendMessage(hWndTX, TX_SETTEXTCOLOR, (WPARAM)(lReturn == 1),  
    dwColor);
```

## TX\_FIELD\_SETATTR

This message sets attributes for the specified marked text field. Changing one attribute does not alter other attributes. These attributes are described in more detail in chapter 1.8 "*Marked Text Fields*".

<b><u>Parameter</u></b>	<b><u>Description</u></b>																		
<i>wParam</i>	Specifies a field identifier.																		
<i>lParam</i>	Specifies new attributes. It can be a combination of the following values:																		
	<table border="0"> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>TF_DELETEABLE</td> <td>If the specified field is to be deleteable.</td> </tr> <tr> <td>TF_UNDELETEABLE</td> <td>If the specified field is not to be deleteable.</td> </tr> <tr> <td>TF_CHANGEABLE</td> <td>If the text of the specified field may be altered.</td> </tr> <tr> <td>TF_UNCHANGEABLE</td> <td>If the text of the specified field may not be altered.</td> </tr> <tr> <td>TF_EXTEDITMODE</td> <td>If for the specified field a second character position is to be implemented at the beginning and the end of a field.</td> </tr> <tr> <td>TF_NORMALEEDITMODE</td> <td>If the specified field is to be edited in normal edit mode.</td> </tr> <tr> <td>TF_SHOWCURFIELDGRAY</td> <td>If the specified field is to be displayed with a gray background when it contains the current character input position.</td> </tr> <tr> <td>TF_SHOWCURFIELDNORMAL</td> <td>If the specified field is not to be displayed with a gray background.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	TF_DELETEABLE	If the specified field is to be deleteable.	TF_UNDELETEABLE	If the specified field is not to be deleteable.	TF_CHANGEABLE	If the text of the specified field may be altered.	TF_UNCHANGEABLE	If the text of the specified field may not be altered.	TF_EXTEDITMODE	If for the specified field a second character position is to be implemented at the beginning and the end of a field.	TF_NORMALEEDITMODE	If the specified field is to be edited in normal edit mode.	TF_SHOWCURFIELDGRAY	If the specified field is to be displayed with a gray background when it contains the current character input position.	TF_SHOWCURFIELDNORMAL	If the specified field is not to be displayed with a gray background.
<b><u>Value</u></b>	<b><u>Meaning</u></b>																		
TF_DELETEABLE	If the specified field is to be deleteable.																		
TF_UNDELETEABLE	If the specified field is not to be deleteable.																		
TF_CHANGEABLE	If the text of the specified field may be altered.																		
TF_UNCHANGEABLE	If the text of the specified field may not be altered.																		
TF_EXTEDITMODE	If for the specified field a second character position is to be implemented at the beginning and the end of a field.																		
TF_NORMALEEDITMODE	If the specified field is to be edited in normal edit mode.																		
TF_SHOWCURFIELDGRAY	If the specified field is to be displayed with a gray background when it contains the current character input position.																		
TF_SHOWCURFIELDNORMAL	If the specified field is not to be displayed with a gray background.																		

**TF\_USEFIELDCARET** If the caret for marked text fields is to be used in the specified field. This caret can be defined with the **TX\_SETCARETEXT** message.

**TF\_USETEXTCARET** If the normal text caret is to be used in the specified field.

**TF\_ENABLEDBLCLICKS** If normal double-click processing is to be performed inside marked text fields, which starts a wordwise selection.

**TF\_DISABLEDBLCLICKS**  
If the normal double-click processing inside marked text fields is to be disabled. This is useful when the **TN\_FIELD\_DBLCLICKED** notification is processed.

The bitwise OR operator can be used to specify more than one value.

**Return Value** The return value is zero if the new attributes could not be set or if the specified field identifier does not exist. Otherwise it is non-zero.

**Comments** The attributes are grouped. The following attributes cannot be used together:

**TF\_DELETEABLE** and **TF\_UNDELETEABLE**

**TF\_CHANGEABLE** and **TF\_UNCHANGEABLE**

**TF\_NORMALEDITMODE** and **TF\_EXTEEDITMODE**

**TF\_SHOWCURFIELDNORMAL** and **TF\_SHOWCURFIELDGRAY**

**TF\_USETEXTCARET** and **TF\_USEFIELDCARET**

**TF\_DISABLEDBLCLICKS** and **TF\_ENABLEDBLCLICKS**

The default attributes for a newly created field with the **TX\_FIELD\_INSERT** message are **TF\_DELETEABLE**, **TF\_CHANGEABLE**,

**TF\_NORMALEDITMODE**, **TF\_SHOWCURFIELDNORMAL**,

**TF\_USETEXTCARET** and **TF\_DISABLEDBLCLICKS**

If a field is undeleteable or unchangeable and the user tries to delete or to change that field, the Text Control beeps.

If a Text Control is destroyed or the text is completely exchanged, the field

attributes are ignored and all fields are deleted. In this case TN\_FIELD\_DELETED notifications are not sent.

## TX\_FIELD\_SETDATA

This message can be used to relate any data to a marked text field. The data is stored independently of its contents.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the identifier of a field in which the data is to be stored.
<i>lParam</i>	Points to a TXFIELDSETDATA data structure that is defined as follows:

```
typedef struct tagTXFIELDSETDATA {
    LONG        lReserved;
    DWORD       dwData;
    DWORD       dwDataSize;
    LPVOID      lpdata;
    HGLOBAL     hData;
} TXFIELDSETDATA;
typedef TXFIELDSETDATA *PTXFIELDSETDATA;
typedef TXFIELDSETDATA FAR *LPTXFIELDSETDATA;
```

The TXFIELDSETDATA data structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>lReserved</i>	A 4 byte value for future use. This value must be set to zero.
<i>dwData</i>	Is a 4-byte value that should be used when the amount of the data to be stored is 1 to 4 bytes. When this member is used all other members of this structure must be set to zero.
<i>dwDataSize</i>	Specifies the size of the data when the amount is larger than 4 bytes. In this case the <i>lpdata</i> member must be used with TX_FIELD_SETDATA and the <i>hData</i> member is used with TX_FIELD_GETDATA.
<i>lpdata</i>	This member is for TX_FIELD_SETDATA only. It specifies the memory address of the data that is to

be stored. Text Control copies the data to an internally created buffer.

*hData* This member is for TX\_FIELD\_GETDATA only. Text Control creates this buffer when the amount of data to be stored is more than 4 bytes. The caller of TX\_FIELD\_GETDATA must free it after using.

To delete all data of a field, all members of this structure must be set to zero.

**Return Value** The return value is zero if the specified field does not exist or when the data could not be stored. Otherwise it is non-zero.

---

## TX\_FIELD\_SETTYPE

This message defines a marked text field of a special type. Text Control supports several special types of marked text fields like source and destination fields for hypertext links or fields that display the current page number. Additional data for these fields - for example the link information - can also be specified through this message. See the chapter 1.8 "*Marked Text Fields - Special Types of Marked Text Fields*" for more information about these fields and their type-related data.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a field identifier.
<i>lParam</i>	Points to a TXFIELDSETTYPE data structure that is defined as follows:

```
typedef struct tagTXFIELDSETTYPE {
    WORD        wStructSize;
    LONG        lReserved;
    BYTE        nFieldType;
    DWORD       dwTypeData;
    DWORD       dwTypeDataSize;
    LPVOID      ltypedata;
    HGLOBAL     hTypeData;
} TXFIELDSETTYPE;
```

The following Comments section describes the members of this structure.

**Return Value** The return value is nonzero if the field type could be set. Otherwise it is zero.

**Comments** The following describes the TXFIELDSETTYPE data structure. This structure is used only for the messages TX\_FIELD\_SETTYPE and TX\_FIELD\_GETTYPE. The *lptypedata* member is for TX\_FIELD\_SETTYPE only and the *hTypeData* member is for TX\_FIELD\_GETTYPE only.

<u>Field</u>	<u>Description</u>																
<i>wStructSize</i>	Specifies the size of this data structure, in bytes.																
<i>lReserved</i>	A 4 byte value for future use. This value must be set to zero.																
<i>nFieldType</i>	Defines the type of the field. It can be anyone of the following values:																
	<table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left;"><u>Type</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>FT_EXTERNALLINK</td> <td>Defines the source of a hypertext link to a location outside of the document.</td> </tr> <tr> <td>FT_INTERNALLINK</td> <td>Defines the source of a hypertext link to a location in the same document.</td> </tr> <tr> <td>FT_LINKTARGET</td> <td>Defines a position in a document which is the target of a hypertext link.</td> </tr> <tr> <td>FT_PAGENUMBER</td> <td>Displays the current page number.</td> </tr> <tr> <td>FT_HIGHLIGHT</td> <td>Defines a piece of text that can be highlighted.</td> </tr> <tr> <td>FT_TOPIC</td> <td>Defines a position in a document that is the beginning of a new topic.</td> </tr> <tr> <td>FT_STANDARD</td> <td>Defines a standard marked text field without a special type. For this value all of the following structure members must be set to zero. This value resets a special field to a standard field and deletes all additional data belonging to the special type.</td> </tr> </tbody> </table>	<u>Type</u>	<u>Description</u>	FT_EXTERNALLINK	Defines the source of a hypertext link to a location outside of the document.	FT_INTERNALLINK	Defines the source of a hypertext link to a location in the same document.	FT_LINKTARGET	Defines a position in a document which is the target of a hypertext link.	FT_PAGENUMBER	Displays the current page number.	FT_HIGHLIGHT	Defines a piece of text that can be highlighted.	FT_TOPIC	Defines a position in a document that is the beginning of a new topic.	FT_STANDARD	Defines a standard marked text field without a special type. For this value all of the following structure members must be set to zero. This value resets a special field to a standard field and deletes all additional data belonging to the special type.
<u>Type</u>	<u>Description</u>																
FT_EXTERNALLINK	Defines the source of a hypertext link to a location outside of the document.																
FT_INTERNALLINK	Defines the source of a hypertext link to a location in the same document.																
FT_LINKTARGET	Defines a position in a document which is the target of a hypertext link.																
FT_PAGENUMBER	Displays the current page number.																
FT_HIGHLIGHT	Defines a piece of text that can be highlighted.																
FT_TOPIC	Defines a position in a document that is the beginning of a new topic.																
FT_STANDARD	Defines a standard marked text field without a special type. For this value all of the following structure members must be set to zero. This value resets a special field to a standard field and deletes all additional data belonging to the special type.																
<i>dwTypeData</i>	Specifies the topic number for fields of the type FT_TOPIC and																

the color of the highlight for fields of the type FT\_HIGHLIGHT. For all other fields this member must be zero.

*dwTypeDataSize* Specifies the size of the buffer, either *lptypedata* points to or that is identified through *hTypeData*.

*lptypedata* This member is for TX\_FIELD\_SETTYPE only. It points to a buffer containing additional data for the field depending on its type. In all cases *lptypedata* must point to a zero-terminated string:

FT\_EXTERNALLINK      *lptypedata* specifies the location to where the hypertext link points. This can be an address or a file name.

FT\_INTERNALLINK      Specifies the name of the field to where the hypertext link points. This must be a field of the type FT\_LINKTARGET.

FT\_LINKTARGET      Specifies the field's name.

For all other types this member must be zero. It must also be zero when this structure is used with the TX\_FIELD\_GETTYPE message.

*hTypeData* This member is for TX\_FIELD\_GETTYPE only. Text Control creates a buffer and copies the type-related data for the field to this buffer depending on the field's type. The caller of TX\_FIELD\_GETTYPE must free it with the **GlobalFree** function after using. This buffer is only created for fields of the types FT\_EXTERNALLINK, FT\_INTERNALLINK and FT\_LINKTARGET. For all other fields it is set to zero.

---

## TX\_FINDTEXT

This message searches for a specified text string or opens the system-defined modeless dialog box which makes it possible for the user to find text within a Text Control's contents.

<u>Parameter</u>	<u>Description</u>
------------------	--------------------

<i>wParam</i>	Is not used.
---------------	--------------

*lParam* When this parameter is zero the system-defined modeless dialog box is opened. Otherwise this parameter must be point to a TXFINDTEXT structure. See the following comments section for a complete description.

**Return Value** The return value has no meaning when the system-defined dialogbox is used. Otherwise the return value is the index of the first character of the match if the text searched for is found. If the specified text is not found, the return value is -1.

**Comments** The TXFINDTEXT structure is defined as follows:

```
typedef struct tagTXFINDTEXT {
    DWORD      lStructSize;
    DWORD      Flags;
    LONG       lStart;
    LPCTSTR    lpstrFindWhat;
} TXFINDTEXT;
```

The TXFINDTEXT structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>										
<i>lStructSize</i>	Specifies the size, in bytes, of this data structure.										
<i>Flags</i>	Specifies a combination of the following flags:										
	<table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left;"><b><u>Value:</u></b></th> <th style="text-align: left;"><b><u>Description:</u></b></th> </tr> </thead> <tbody> <tr> <td>TXFR_SEARCHUP</td> <td>Determines the direction of searches through a document. If this flag is used, the search direction is up; if the flag is not used, the search direction is down.</td> </tr> <tr> <td>TXFR_MATCHCASE</td> <td>Indicates case-sensitive searches.</td> </tr> <tr> <td>TXFR_NOHIGHLIGHT</td> <td>Determines if a match appears highlighted.</td> </tr> <tr> <td>TXFR_NOMESSAGEBOX</td> <td>Suppresses the built-in message boxes which inform the user that a match could not be found.</td> </tr> </tbody> </table>	<b><u>Value:</u></b>	<b><u>Description:</u></b>	TXFR_SEARCHUP	Determines the direction of searches through a document. If this flag is used, the search direction is up; if the flag is not used, the search direction is down.	TXFR_MATCHCASE	Indicates case-sensitive searches.	TXFR_NOHIGHLIGHT	Determines if a match appears highlighted.	TXFR_NOMESSAGEBOX	Suppresses the built-in message boxes which inform the user that a match could not be found.
<b><u>Value:</u></b>	<b><u>Description:</u></b>										
TXFR_SEARCHUP	Determines the direction of searches through a document. If this flag is used, the search direction is up; if the flag is not used, the search direction is down.										
TXFR_MATCHCASE	Indicates case-sensitive searches.										
TXFR_NOHIGHLIGHT	Determines if a match appears highlighted.										
TXFR_NOMESSAGEBOX	Suppresses the built-in message boxes which inform the user that a match could not be found.										
<i>lStart</i>	Specifies a character index that determines where to begin the										

search. The first character of text in the control has an index of 0. When this parameter is set to -1, the search begins at the current input position.

*lpstrFindWhat* Specifies the string to search for.

---

## TX\_FONTDIALOG

This message opens a modal dialog box which contains all available fonts and point sizes for the currently selected printer. Font attributes and values for subscript and superscript can also be set.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a POINT data structure. The Text Control copies the new minimum window size (in pixels) to this buffer. If a text area has been defined with the TX_SETTEXTAREA message the new minimum text area is copied as twentieths of a point. This size is calculated from the new fonts and attributes set by this message. If the values copied are both 0, an error has occurred.

***Return Value*** The return value is:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TR_ERR	If an error has occurred.
TR_UNCHANGED	If the user leaves the dialog box with the CANCEL button.
TR_CHANGED	If the user leaves the dialog box with the OK button.

***Comments*** The return value also evaluates TR\_ERR if the window size is too small for the changed fonts. After resizing the window with the minimum values given by *lParam*, this message can be sent again.

The Text Control updates the modified selection if the window is visible.

## TX\_GETBASELINE

This message returns the baseline alignment value of the currently selected text.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value contains one of the following values in the low-order word:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
FA_NOCOMMONS	The current selection contains different subscript and superscript values
FA_STANDARD	The common baseline alignment value is zero
FA_SUPERSCRIPT	The common baseline alignment is superscript
FA_SUBSCRIPT	The common baseline alignment is subscript

The high-order word of the return value contains the baseline align value in twentieths of a point.

**Comments** If the current selection contains different baseline alignment values but all are subscript, then the low-order word of the return value is FA\_SUBSCRIPT and the high-order word is zero.

---

## TX\_GETBASELINEPOS

This message returns the baseline position of the specified line. The dimensions are given in twentieths of a point with an origin at the upper left corner of the text. The relationship between the upper left corner of the text and the upper left corner of the Text Control's client area can be obtained with the TX\_GETSCROLLPOS message.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Specifies the index of the line for which the baseline

position is to be returned. The index of the first line is zero.

**Return Value** The return value specifies the requested baseline position in twentieths of a point.

---

## TX\_GETBKGNDCOLOR

This message retrieves an RGB value for the background color of the Text Control.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a long variable that is to receive an RGB value for the background color.

**Return Value** The return value is 1 if the retrieved value is the system color for the window background. Otherwise the return value is 2. The return value is zero, if an error has occurred.

---

## TX\_GETCARETEXT

This message returns the current extension of the caret in pixels.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value contains the caret extension. The width is in the low-order word, the height is in the high-order word.

---

## TX\_GETCARETPOS

This message returns the current caret position as screen coordinates.

---

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value contains the caret position. The x-coordinate is in the low-order word, the y-coordinate is in the high-order word.

---

## TX\_GETDATASIZE

This message returns the size of all the data the Text Control currently contains, including the text.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value contains the datasize. It is TR\_ERR if an error has occurred.

**Comments** This message is necessary to calculate the size of the buffer for the TX\_COPYDATA message.

---

## TX\_GETDEVICE

This message retrieves the name of the device for which the text is currently formatted.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies the size of the buffer <i>lParam</i> points to.
<i>lParam</i>	Points to a buffer, of a length specified by <i>wParam</i> , which is to receive the name of the device. This name is only be copied if the return value is TF_PRINTER.

**Return Value** The return value is one of the following values:

<u>Value</u>	<u>Meaning</u>
TF_SCREEN	The device is the screen.
TF_STANDARD	The device is the standard device, specified in the [windows] section of the WIN.INI file.
TF_PRINTER	The device is a printer.

**Comments** The name of the device is copied in the same format as that used in the WIN.INI file, for example:

```
PostScript Printer, PSCRIPT, LPT1:
```

**Unicode** The size specified through *wParam* is in characters, when UNICODE is defined, otherwise it is in bytes.

---

## TX\_GETFONT

This message retrieves the common typeface and pointsize of all currently selected fonts.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies the dimensions of the returned pointsize. If the high-order bit is set, the pointsize is returned in twentieths of a point. Otherwise it is returned in points.
<i>lParam</i>	Points to a buffer of length LF_FACESIZE which is to receive the typeface string. The string is set to an empty string if no common typeface exists.

**Return Value** The return value identifies the common pointsize in points or twentieths of a point. It is zero if no common pointsize exists.

**Comments** The pointsize is calculated from the values of the TEXTMETRIC data structure retrieved through the **EnumFonts** function. The following formula is used to calculate the pointsize:

$$\text{pointsize} = (\text{tm.tmHeight} - \text{tm.tmInternalLeading} + 10) / 20$$

The TEXTMETRIC values are given in twentieths of a point.

**Unicode** The buffer length is in characters, when UNICODE is defined, otherwise it is in bytes.

---

## TX\_GETFONTATTR

This message returns a bitmask of the font attributes for all fonts in the current selection.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is one or more of the following values, indicating the common attributes:

<u>Value</u>	<u>Meaning</u>
FA_NOCOMMONS	No common font attribute.
FA_BOLD	Each font is bold.
FA_STANDARD	Each font is normal.
FA_ITALIC	Each font is italic.
FA_UNDERLINE	Each font is underlined.
FA_STRIKEOUT	Each font is struck out.
FA_UL_DOUBLE	Each font is double underlined.
FA_UL_WORDSONLY	Words are underlined, word gaps are omitted.

The return value is TR\_ERR if an error has occurred.

**Comments** This message can be used to set the correct checkmarks concerning menu items for font attributes. It allocates memory and initializes the font manager, so the return value should always be checked.

---

## TX\_GETFORMAT

This message returns the paragraph format value of the currently selected paragraphs.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is one of the following:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TF_LEFT	Text is left aligned
TF_RIGHT	Text is right aligned.
TF_CENTER	Text is centered.
TF_BLOCK	Text is block formatted.
TF_NOCOMMONS	No common text alignment.

The return value is TR\_ERR if an error occurred.

---

## TX\_GETIMAGE

If an image has been selected, then this message retrieves the full DOS path name of the file that contains the currently registered image of the Image-Control window.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the size of the buffer which <i>lParam</i> points to, including the null-terminating character.
<i>lParam</i>	Points to a buffer which is to receive the path name.

**Return Value** The return value is the number of bytes copied. It is zero if an error has occurred or no image is currently selected or registered.

**Comments** The IC Image-Control programming tool is needed for this message.

**Unicode** The size specified through *wParam* is in characters, when UNICODE is defined, otherwise it is in bytes.

---

## TX\_GETIMAGEFILTERS

This message returns a buffer containing pairs of null-terminated strings specifying image filters. The format of the buffer has the same form as described for the *lpstrFilter* member of an **OPENFILENAME** structure and can be used to initialize the **GetOpenFileName** dialog box. See the description of the **OPENFILENAME** structure in Windows SDK for more information.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is a global memory handle identifying the buffer that holds the strings. Each string ends with a terminating zero, the buffer itself ends with two terminating zeros. The return value is zero if an error has occurred. If an application has finished using the buffer it must free it with the **GlobalFree** function.

**Comments** The IC Image-Control programming tool is needed for this message.

---

## TX\_GETIMAGEFORMAT

If an image has been selected this message retrieves information about the formatting of an image. The Text Control sends a **TN\_IMAGECLICKED** notification message if an image has been selected.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an <b>IMAGEFORMAT</b> data structure which is defined as follows:

```
typedef struct tagIMAGEFORMAT {
    POINT ptPosition;
    POINT ptSize;
    POINT ptMaxPosition;
    WORD wScale;
    WORD wFlags;
} IMAGEFORMAT;
```

The IMAGEFORMAT structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>ptPosition</i>	Specifies the image's horizontal position in twentieths of a point. The y-coordinate of this POINT structure is not used.
<i>ptSize</i>	Specifies the image's unscaled horizontal and vertical dimensions in twentieths of a point.
<i>ptMaxPosition</i>	Specifies the maximum horizontal position in twentieths of a point. The y-coordinate of this POINT structure is not used.
<i>wScale</i>	Specifies the image's scaling factor in percent. This is a value between 10 and 250.
<i>wFlags</i>	Can contain a combination of the following flags:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
ICF_GRAYED	The image is displayed in fast mode.
ICF_SAVEASDATA	The Text Control saves the image by its data instead of its filename.

***Return Value*** The return value is zero if an error has occurred or if no image is currently selected or registered. Otherwise it is nonzero.

***Comments*** The IC Image-Control programming tool is needed for this message.

---

## TX\_GETINDENTS

This message retrieves the indent values of the currently selected paragraphs.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an array of five integers which is to receive the indent values in twentieths of a point. The values are in the order: left indent, right indent, additional indent of the first line, top indent and bottom indent. The third value, the additional indent of the first line, is the only one that can be negative. If a value contains TR_IGNORED, no common value of this indent exists for the selected paragraphs.

***Return Value*** The return value is zero if an error has occurred. Otherwise it contains maximum values for a combination of new indents that can be set with the TX\_SETINDENTS message. The maximum x-value is in the low-order word and the maximum y-value is in the high-order word.

The x-value is the maximum value for the sum of the left indent, the right indent and the additional indent of the first line. The y-value is the maximum value for the top indent and the bottom indent.

These values become invalid if the size of the Text Control is changed, or a new font or pointsize is set.

---

## TX\_GETLANGUAGE

This message returns the current language identifier for the language which the Text Control is using to display information strings, warnings or dialog boxes.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

---

**Return Value** The return value is the language identifier. See the TX\_SETLANGUAGE message for possible values.

---

## TX\_GETLINEANDCOL

This message retrieves the line and the column of the current caret position. The first line and the first column both have the number one.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an array of two DWORD variables. The first variable receives the line number and the second variable receives the column number of the current input position.

**Return Value** The return value is zero if an error has occurred, otherwise it is a page number if a page height has been set with the TX\_SETTEXTAREA message. In this case the retrieved line number is relative to the top of this page.

---

## TX\_GETLINECOUNT

This message returns the number of text lines in the Text Control.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is the number of text lines in the Text Control.

---

## TX\_GETLINERECT

This message retrieves the rectangular area covered by a line of text. The rectangle does not include the external leading area, additional linespacing or indents. The dimensions are given in twentieths of a point with an origin at the upper left corner of the line. The alignment of the origin of the line is given by the return value.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a LINERECT data structure which is defined as followed:

```
typedef struct tagLINERECT {
    LONG lLineIndex;
    RECT rLine;
} LINERECT;
```

The LINERECT structure has the following fields:

<u>Field</u>	<u>Description</u>
<i>lLineIndex</i>	Specifies the index of the line whose rectangle is to be retrieved. The index of the first line is zero.
<i>rLine</i>	Specifies a RECT data structure that is to receive the text rectangle. If an error has occurred, the rectangle is set to empty.

**Return Value** The return value specifies the offset of the line in twentieths of a point with an origin at the upper left corner of the text. The relationship between the upper left corner of the text and the upper left corner of the Text Control's client area can be obtained with the TX\_GETSCROLLPOS message.

---

## TX\_GETLINESPACING

This message returns the linespacing value of the currently selected paragraphs.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value contains the linespacing value as a percentage of the size of the currently used font in the low-order word and the linespacing value in twentieths of a point in the high-order word. The low-order word is zero if no common percentage value exists. The high-order word is zero if no common absolute linespacing value exists.

---

## TX\_GETLINKWND

This message searches for the handle of a window that is part of a Text Control window's chain. This message can be sent to any window that belongs to the chain.

<b><u>Parameter</u></b>	<b><u>Description</u></b>																		
<i>wParam</i>	Specifies the relationship between the original window and the returned window. It can be one of the following values:																		
	<table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left;"><b><u>Value</u></b></th> <th style="text-align: left;"><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td style="vertical-align: top;">GWTX_HWNDFIRST</td> <td>Returns the first window of a chain of linked windows.</td> </tr> <tr> <td style="vertical-align: top;">GWTX_HWNDLAST</td> <td>Returns the last window of a chain of linked windows.</td> </tr> <tr> <td style="vertical-align: top;">GWTX_HWNDNEXT</td> <td>Returns the window that follows the specified window.</td> </tr> <tr> <td style="vertical-align: top;">GWTX_HWNDPREV</td> <td>Returns the previous window of a chain of linked windows.</td> </tr> <tr> <td style="vertical-align: top;">GWTX_HWNDFIRSTSEL</td> <td>Returns the first window of a chain of linked windows that contains selected text.</td> </tr> <tr> <td style="vertical-align: top;">GWTX_HWNDLASTSEL</td> <td>Returns the last window of a chain of linked windows that contains selected text.</td> </tr> <tr> <td style="vertical-align: top;">GWTX_FROMOFFSET</td> <td>Returns the window of a chain that contains the one-based character offset specified by <i>lParam</i>.</td> </tr> <tr> <td style="vertical-align: top;">GWTX_GETCOUNT</td> <td>Returns the total number of</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	GWTX_HWNDFIRST	Returns the first window of a chain of linked windows.	GWTX_HWNDLAST	Returns the last window of a chain of linked windows.	GWTX_HWNDNEXT	Returns the window that follows the specified window.	GWTX_HWNDPREV	Returns the previous window of a chain of linked windows.	GWTX_HWNDFIRSTSEL	Returns the first window of a chain of linked windows that contains selected text.	GWTX_HWNDLASTSEL	Returns the last window of a chain of linked windows that contains selected text.	GWTX_FROMOFFSET	Returns the window of a chain that contains the one-based character offset specified by <i>lParam</i> .	GWTX_GETCOUNT	Returns the total number of
<b><u>Value</u></b>	<b><u>Meaning</u></b>																		
GWTX_HWNDFIRST	Returns the first window of a chain of linked windows.																		
GWTX_HWNDLAST	Returns the last window of a chain of linked windows.																		
GWTX_HWNDNEXT	Returns the window that follows the specified window.																		
GWTX_HWNDPREV	Returns the previous window of a chain of linked windows.																		
GWTX_HWNDFIRSTSEL	Returns the first window of a chain of linked windows that contains selected text.																		
GWTX_HWNDLASTSEL	Returns the last window of a chain of linked windows that contains selected text.																		
GWTX_FROMOFFSET	Returns the window of a chain that contains the one-based character offset specified by <i>lParam</i> .																		
GWTX_GETCOUNT	Returns the total number of																		

windows that belong to a chain of linked windows.

**GWTX\_GETNUMBER** Returns the position of a window within its chain. The first window is assigned position one.

**GWTX\_GETOFFSET** Returns the one-based character offset of the first character of the window that receives this message relative to the beginning of the text.

*lParam* Depends on the value of *wParam*. If *wParam* is **GWTX\_FROMOFFSET** *lParam* must specify a one-based character offset. Otherwise *lParam* is not used.

**Return Value** The return value is a value which depends on the *wParam* parameter. It is zero if the value could not be found.

## TX\_GETMODE

This message returns information about different modes of the Text Control, like the current background mode or the current character insertion mode.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a POINT structure which becomes filled with values for the maximum window size (in pixels) when the <b>TF_AUTOEXPAND</b> mode is set. The width is in the x-coordinate and the height is in the y-coordinate. If these values are not needed, <i>lParam</i> can be zero.

**Return Value** The return value is a combination of the following values:

<u>Value</u>	<u>Meaning</u>
<b>TF_AUTOEXPAND</b>	Set if the Text Control window is automatically expanded to prevent text overflows.

---

TF_FIXED	Set if the Text Control window has a fixed size.
TF_FRAMED	Set if a borderline is drawn on the screen.
TF_HIDESELNA	Set if the text selection is hidden in the inactive window state.
TF_HIDEWHITESPACE	Set if whitespace characters are hidden on the screen.
TF_INSERT	Set if the character insertion mode is insert.
TF_KEESEL	Set if currently selected text is not deleted before text insertion.
TF_NOTFRAMED	Set if the borderline is hidden on the screen.
TF_OPAQUE	Set if the background mode is opaque.
TF_OVERWRITE	Set if the character insertion mode is overwrite.
TF_REPLACESEL	Set if currently selected text is deleted before text insertion.
TF_SHOWSELNA	Set if the text selection remains visible in the inactive window state.
TF_SHOWWHITESPACE	Set if whitespace characters are shown on the screen.
TF_TRANSPARENT	Set if the background mode is transparent.

---

## TX\_GETMODEEX

This message returns modes set with the TX\_SETMODEEX message.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

***Return Value*** The return value is a combination of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TF_DISPLAY	Set if the Text Control can only display text.

---

TF_EDIT	Set if the Text Control can display and edit text.
TF_NOWAITCURSOR	Set if the Text Control does not change the cursor during long term operations.
TF_WAITCURSOR	Set if the Text Control changes the cursor to an hourglass during long term operations.
TF_TOPINDENTFIRSTPG	Set if the Text Control allows a top indent for the first paragraph.
TF_NOTOPINDENTFIRSTPG	Set if the Text Control suppresses a top indent of the first paragraph.
TF_ERRORBOXES	Set when in some important cases error boxes are shown.
TF_NOERRORBOXES	Set when the all error message boxes are suppressed.
TF_SHOWGRIDLINES	Set when grid lines in tables are shown.
TF_HIDEGRIDLINES	Set when grid lines in tables are hidden.

---

## TX\_GETPAGECOUNT

This message returns the current number of pages.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Specifies the height of the page in twentieths of a point.

**Return Value** The return value is the number of pages with the height specified by *lParam*.

---

## TX\_GETPAGEMARGINS

This message retrieves the current page margins.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a RECT data structure. The Text Control copies the current margin values in twentieths of a point to this structure.

**Return Value** The return value is zero if an error has occurred. Otherwise it is either 1L if the page margins are currently not shown or 2L if the margins are shown on the screen. Margins are only shown if a text area has been set via the TX\_SETTEXTAREA message with a height value not set to -1. See the chapter 1.8 "*Text Area and Coordinates*" for more information about the text area.

**Comments** If no margins have been set with the TX\_SETPAGEMARGINS message, the default margins of 20mm (1134 TWIPS) are retrieved.

---

## TX\_GETPARAFORMATFLAGS

This message returns special formats of the paragraphs currently selected.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is a bit-field and can contain the following values:

<u>Value</u>	<u>Meaning</u>
TF_ATLEASTLINEPACING	This value specifies that absolute linespacing values set with the TX_SETLINEPACING message are to be used as minimal values. If a line contains characters or images which would be cropped with this setting, the linespacing is enlarged accordingly.
TF_EXACTLINEPACING	This value specifies that absolute linespacing values set with the

TX\_SETLINESPACING message are to be used as exact values, regardless of whether larger characters or images are being cropped.

TF\_PAGEBREAKNOTALLOWED A page break is not allowed within a paragraph.

TF\_PAGEBREAKALLOWED Page breaks are allowed within a paragraph.

If no value is set the selected paragraphs are differently formatted.

**Comments** For future extensions use the bitwise OR operator to get the information from the return value.

---

## TX\_GETPGFRAME

This message returns the appearance, style and line width for the frames of all selected paragraphs.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a DWORD variable that receives the line width in the low-order word and the distance of the frame from the text in the high-order word. Both values are in twentieths of a point. The low-order word contains zero if the selected paragraphs have different width values and the high-order word contains -1 if the selected paragraphs have different distance values.

**Return Value** The return value contains a combination of frame appearance and style flags. It can be a combination of any of the following values:

<u>Value</u>	<u>Meaning</u>
BF_LEFTLINE	Set if the frame has a left line.
BF_RIGHTLINE	Set if the frame has a right line.
BF_TOPLINE	Set if the frame has a top line.

---

BF_BOTTOMLINE	Set if the frame has a bottom line.
BF_BOX	Set if the frame is a complete box.
BF_TABLINES	Set if the frame includes vertical lines at each tabulator position.
BF_TABLE	Set if the frame is a complete box including vertical lines at each tabulator position.
BF_SINGLE	Set if the lines are single lines.
BF_DOUBLE	Set if the lines are doubled lines.
BF_BOXCONNECT	Set if the frame is connected to its neighbours.

---

## TX\_GETRECT

This message retrieves the size of the window rectangle for the Text Control regardless of the zoom factor. A value of 100 percent is assumed.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a RECT data structure. The Text Control copies the window rectangle to this structure.

**Comments** The dimensions of the rectangle are in client area coordinates of the windows parent window.

---

## TX\_GETSCROLLPOS

This message returns the current scroll position.

<u>Parameter</u>	<u>Description</u>						
<i>wParam</i>	Specifies the direction. It can be one of the following values:						
	<table> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>TF_HSCROLL</td> <td>Returns the horizontal scroll position.</td> </tr> <tr> <td>TF_VSCROLL</td> <td>Returns the vertical scroll position.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	TF_HSCROLL	Returns the horizontal scroll position.	TF_VSCROLL	Returns the vertical scroll position.
<u>Value</u>	<u>Meaning</u>						
TF_HSCROLL	Returns the horizontal scroll position.						
TF_VSCROLL	Returns the vertical scroll position.						

---

*lParam* Is not used.

**Return Value** The return value is the current scroll position of the client area's upper left corner in twentieths of a point.

---

## TX\_GETSEL

This message retrieves the start and end character positions of the currently selected text.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an array of two DWORD variables. The first variable receives the start position and the second variable receives the end position of the selection.

**Return Value** The return value is zero if an error has occurred, otherwise it is nonzero.

**Comments** The start position is always the same as the position of the blinking caret.

---

## TX\_GETSUPPORTEDFONTS

This message returns a buffer containing strings of all the font family names which are supported by the currently selected output device.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is a global memory handle identifying the buffer that holds the strings. Each string ends with a terminating zero, the buffer itself ends with two terminating zeros. The return value is zero if an error has occurred. If an application has finished using the buffer it must free it with the **GlobalFree** function.

---

## TX\_GETSUPPORTEDSIZES

This message returns a buffer containing strings of all point sizes which are supported for a specified font family by the currently selected output device.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a buffer of length LF_FACESIZE containing the name of the font family.

***Return Value*** The return value is a global memory handle identifying the buffer that holds the point sizes as strings. Each string ends with a terminating zero, the buffer itself ends with two terminating zeros. The return value is zero if an error has occurred. If an application has finished using the buffer it must free it with the **GlobalFree** function.

---

## TX\_GETTABS

This message retrieves common tab positions and tab types for all selected paragraphs.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the kind of units of the position values. It can be one of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
UF_PIXEL	The tab position values are retrieved in pixels.
UF_TWIPS	The tab position values are retrieved in terms of twentieths of a point.

*lParam* Points to an array of type TABSCT and size NTABS. The TABSCT data structure is defined as followed:

```
typedef struct tagTABSCT {
    BYTE nTabFlag;
    WORD wTabPos;
} TABSCT;
```

The TABSCT structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>												
<i>nTabFlag</i>	Specifies the type of tabstop. It can be any one of the following values:												
	<table> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>LEFTTAB</td> <td>The tab position is at the left side of the text.</td> </tr> <tr> <td>RIGHTTAB</td> <td>The tab position is at the right side of the text.</td> </tr> <tr> <td>RIGHTBORDERTAB</td> <td>The tab position is at the rightmost text position. All tabs in a list which follow a RIGHTBORDERTAB are ignored.</td> </tr> <tr> <td>CENTERTAB</td> <td>The text is centered at the tab position.</td> </tr> <tr> <td>DECIMALTAB</td> <td>The decimal sign installed in Windows is located at the tab position.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	LEFTTAB	The tab position is at the left side of the text.	RIGHTTAB	The tab position is at the right side of the text.	RIGHTBORDERTAB	The tab position is at the rightmost text position. All tabs in a list which follow a RIGHTBORDERTAB are ignored.	CENTERTAB	The text is centered at the tab position.	DECIMALTAB	The decimal sign installed in Windows is located at the tab position.
<b><u>Value</u></b>	<b><u>Meaning</u></b>												
LEFTTAB	The tab position is at the left side of the text.												
RIGHTTAB	The tab position is at the right side of the text.												
RIGHTBORDERTAB	The tab position is at the rightmost text position. All tabs in a list which follow a RIGHTBORDERTAB are ignored.												
CENTERTAB	The text is centered at the tab position.												
DECIMALTAB	The decimal sign installed in Windows is located at the tab position.												
<i>wTabPos</i>	Specifies the x-coordinate of the tab position. When the <i>nTabFlag</i> member of a tab is set to RIGHTBORDERTAB this value has no meaning and should be set to zero.												

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

***Comments*** If the number of retrieved tabs is less than NTABS, the values of the rest of the tab list are set to zero.

---

## TX\_GETTEXT

This message copies the text belonging to the Text Control into a buffer provided by the caller. To calculate the size of the buffer required the TX\_GETTEXTSIZE

message can be used. The buffer must be large enough to accommodate a terminating zero character.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the maximum number of characters to be copied, including the terminating zero character. If <i>wParam</i> is zero the complete text contained by the Text Control is copied.
<i>lParam</i>	Points to the buffer that is to receive the text.

***Return Value*** The return value is the number of characters (Unicode) or bytes (otherwise) copied.

***Comments*** The copied text is in the Text Control's text only format and contains only control characters listed in appendix A. This message should be used to work with other Text Control messages that use indices like TX\_SETSEL or TX\_LINEFROMCHAR. To receive the text in a Windows compatible text format for example to exchange it with a Windows Edit-Control the message TX\_DATAOUT with the format identifier set to TF\_FORMAT\_ANSI or TF\_FORMAT\_UNICODE should be used.

***16 bit*** This message supports buffers larger than 64 kB of text.

---

## **TX\_GETTEXTAREA**

This message retrieves the text area previously set with the TX\_SETTEXTAREA message.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a LONG variable which retrieves the size of the text area. The low-order word specifies the width, the high-order word specifies the height. A height value of -1 means a variable height depending on the text. A width or height value of zero specifies that this value is always equal to the window size.

**Return Value** The return value is a combination of flags. See the *wParam* parameter description of the TX\_SETTEXTAREA message for more information about possible values.

---

## TX\_GETTEXTCOLOR

This message retrieves RGB values for the text color and the text background color of the currently selected text.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	If <i>wParam</i> is zero <i>lParam</i> retrieves the RGB value for only the text color. If <i>wParam</i> is nonzero, <i>lParam</i> retrieves values for the text and the text background colors.
<i>lParam</i>	Points to an array of two variables of type COLORREF which is to receive RGB values for the text color and the text background color. The text color is in the first variable and the background color is in the second variable.

**Return Value** The low-order word of the return value specifies the type of text color. It can contain one of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
CV_UNDEFINED	If the current selection contains more than one text color.
CV_TEXTDEFAULT	If the retrieved text color represents the system color for the window text.
CV_TEXTUSER	If the retrieved text color represents a user-defined value.

The high-order word of the return value specifies the type of text background color. It can contain one of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
CV_UNDEFINED	If the current selection contains more than one color for the text background.
CV_BKDEFAULT	If the retrieved text background color represents the system color for the window background.
CV_BKCONTROL	If the retrieved text background color represents the Text

	Control's background color, set with the TX_SETBKGNDCOLOR message.
CV_BKUSER	If the retrieved background color represents a user-defined value.

---

## TX\_GETTEXTTEXTENT

This message returns the dimensions of the text contained in the Text Control. It can only be used if the Text Control does not have a built-in scroll-interface specified with the TX\_SETTEXTAREA message. In this event the TX\_GETTEXTWIDTH or the TX\_GETTEXTHEIGHT messages can be used.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value contains the text dimensions in pixels. The width is in the low-order word and the height is in the high-order word. If the text is larger than the window size, the window size is returned. The return value is zero if an error has occurred or if a built-in scroll-interface is set.

**Comments** This message can be used to create a Text Control with a user-defined width and a height that will be large enough to show all of a specified text. To achieve this, create a window with a large height, e.g. 32000 pixels. Load the text and calculate its dimensions with the TX\_GETTEXTTEXTENT message. After that reduce the window's size to the returned values.

---

## TX\_GETTEXTHEIGHT

This message returns the height of the text currently contained in the Text Control in twentieths of a point. The value is in 100 percent, independent of the zoom factor currently set.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

---

**Return Value**     The return value is the height value.

---

## TX\_GETTEXTSIZE

This message returns the length of the text (in characters) associated with the Text Control.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value**     The return value is the length of the text.

**Comments**        For more information when to use this message see the comments section of the TX\_GETTEXT message.

---

## TX\_GETTEXTWIDTH

This message returns the largest line width the Text Control currently contains in twentieths of a point. The value is in 100 percent, independent of the zoom factor currently set.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value**     The return value is the width value.

---

## TX\_GETZOOM

This message returns the current zooming factor in percent.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value specifies the current zooming factor in percent.

---

## TX\_HF\_ACTIVATE

This message activates or deactivates a header or a footer. During activation the current input focus is set in the header or footer area, so that the user can alter the text and/or the format. During deactivation the input focus is set back to the main text.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	When this parameter is zero a currently activated header or footer is deactivated. Otherwise it specifies the header or footer to activate and can be one of the following values:

<u>Value:</u>	<u>Description:</u>
TF_HF_HEADER	Activates the header area.
TF_HF_1STHEADER	Activates the header area for the first page.
TF_HF_FOOTER	Activates the footer area.
TF_HF_1STFOOTER	Activates the footer area for the first page.

**Return Value** The return value is nonzero if the header or footer could be activated. Otherwise it is zero.

---

## TX\_HF\_DISABLE

This message disables certain parts of the header and footer functionality.

<u>Parameter</u>	<u>Description</u>																		
<i>wParam</i>	Is not used.																		
<i>lParam</i>	When this parameter is zero, all currently enabled header and footer functionality is disabled and all allocated memory is freed. Otherwise it can be a combination of the following values:																		
	<table border="0"> <thead> <tr> <th><u>Value:</u></th> <th><u>Description:</u></th> </tr> </thead> <tbody> <tr> <td>TF_HF_HEADER</td> <td>Disables headers.</td> </tr> <tr> <td>TF_HF_1STHEADER</td> <td>Disables a special header for the first page.</td> </tr> <tr> <td>TF_HF_FOOTER</td> <td>Disables footers.</td> </tr> <tr> <td>TF_HF_1STFOOTER</td> <td>Disables a special footer for the first page.</td> </tr> <tr> <td>TF_HF_MOUSECLICK</td> <td>Disables activation through single mouse clicks.</td> </tr> <tr> <td>TF_HF_NOMOUSEDCLICK</td> <td>Enables activation through mouse double-clicks.</td> </tr> <tr> <td>TF_HF_SOLIDFRAME</td> <td>Suppresses a solid border.</td> </tr> <tr> <td>TF_HF_UNFRAMED</td> <td>Resets the border to framed.</td> </tr> </tbody> </table>	<u>Value:</u>	<u>Description:</u>	TF_HF_HEADER	Disables headers.	TF_HF_1STHEADER	Disables a special header for the first page.	TF_HF_FOOTER	Disables footers.	TF_HF_1STFOOTER	Disables a special footer for the first page.	TF_HF_MOUSECLICK	Disables activation through single mouse clicks.	TF_HF_NOMOUSEDCLICK	Enables activation through mouse double-clicks.	TF_HF_SOLIDFRAME	Suppresses a solid border.	TF_HF_UNFRAMED	Resets the border to framed.
<u>Value:</u>	<u>Description:</u>																		
TF_HF_HEADER	Disables headers.																		
TF_HF_1STHEADER	Disables a special header for the first page.																		
TF_HF_FOOTER	Disables footers.																		
TF_HF_1STFOOTER	Disables a special footer for the first page.																		
TF_HF_MOUSECLICK	Disables activation through single mouse clicks.																		
TF_HF_NOMOUSEDCLICK	Enables activation through mouse double-clicks.																		
TF_HF_SOLIDFRAME	Suppresses a solid border.																		
TF_HF_UNFRAMED	Resets the border to framed.																		

**Return Value**     The return value is nonzero if at least one header, footer or style setting has been disabled. Otherwise it is zero.

## **TX\_HF\_ENABLE**

This message enables the usage of headers and footers. Headers and footers can only be used when a user-defined formatting area has been set previously with the TX\_SETTEXTAREA message.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.

*lParam* Specifies what to enable. It can be a combination of the following values:

<b><u>Value:</u></b>	<b><u>Description:</u></b>
TF_HF_STANDARD	Enables headers and footers with a special header and footer on the first page. Headers and footers can be activated through mouse double-clicks. An activated header or footer has a dotted border to indicate its size.
TF_HF_HEADER	Enables headers only.
TF_HF_1STHEADER	Enables only a special header for the first page.
TF_HF_FOOTER	Enables footers only.
TF_HF_1STFOOTER	Enables only a special footer for the first page.
TF_HF_MOUSECLICK	Headers and footers can be activated through single mouse clicks.
TF_HF_NOMOUSEDLCLK	Headers and footer cannot be activated through mouse double-clicks.
TF_HF_SOLIDFRAME	An activated header or footer has a solid border to indicate its size.
TF_HF_UNFRAMED	An activated header or footer has no border.

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

***Comments*** This message can only be used to add a certain header or footer or a certain style setting. To disable a certain functionality, the TF\_HF\_DISABLE message must be used. For example when activation with mouse clicks is enabled, a message call with *lParam* set to TF\_HF\_SOLIDFRAME, displays an activated header or footer with a solid frame. Activation with mouse clicks remains active.

## TX\_HF\_GETENABLED

This message returns which headers and/or footers are enabled for the current document.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is a bitwise or combination of one or more of the following values:

<b><u>Value:</u></b>	<b><u>Description:</u></b>
TF_HF_HEADER	Headers are enabled.
TF_HF_1STHEADER	A special header for the first page is enabled.
TF_HF_FOOTER	Footers are enabled.
TF_HF_1STFOOTER	A special footer for the first page is enabled.

## TX\_HF\_GETPOSITION

This message returns a header's or footer's position. For headers the position value is the distance between the top of the header and the top of the page. For footers the position value is the distance between the bottom of the footer and the bottom of the page. All values are in twips. The default value is 567 twips = 1 cm.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the header or footer the position of which is requested. It can be one of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TF_HF_HEADER	Returns the header's position.
TF_HF_1STHEADER	Returns the position of the special header for the first page.
TF_HF_FOOTER	Returns the footer's position.

	TF_HF_1STFOOTER	Returns the position of the special footer for the first page.
<i>lParam</i>	Is not used.	

**Return Value** The return value is the requested position in twips. It is -1, if an error has occurred.

---

## TX\_HF\_SELECT

This message defines, whether a certain Text Control message is sent to a header, to a footer or to the main text.

<b><u>Parameter</u></b>	<b><u>Description</u></b>															
<i>wParam</i>	Is not used.															
<i>lParam</i>	Specifies the header or footer that is to be selected. It can be one of the following values:															
		<table> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>TF_HF_HEADER</td> <td>Selects the header.</td> </tr> <tr> <td>TF_HF_1STHEADER</td> <td>Selects the special header for the first page.</td> </tr> <tr> <td>TF_HF_FOOTER</td> <td>Selects the footer.</td> </tr> <tr> <td>TF_HF_1STFOOTER</td> <td>Enables the special footer for the first page.</td> </tr> <tr> <td>TF_HF_AUTO</td> <td>Selects the automatic mode. A message is sent to the text part with the current input position. This is the default setting.</td> </tr> <tr> <td>TF_HF_MAINTTEXT</td> <td>Selects the main text.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	TF_HF_HEADER	Selects the header.	TF_HF_1STHEADER	Selects the special header for the first page.	TF_HF_FOOTER	Selects the footer.	TF_HF_1STFOOTER	Enables the special footer for the first page.	TF_HF_AUTO	Selects the automatic mode. A message is sent to the text part with the current input position. This is the default setting.	TF_HF_MAINTTEXT	Selects the main text.
<b><u>Value</u></b>	<b><u>Meaning</u></b>															
TF_HF_HEADER	Selects the header.															
TF_HF_1STHEADER	Selects the special header for the first page.															
TF_HF_FOOTER	Selects the footer.															
TF_HF_1STFOOTER	Enables the special footer for the first page.															
TF_HF_AUTO	Selects the automatic mode. A message is sent to the text part with the current input position. This is the default setting.															
TF_HF_MAINTTEXT	Selects the main text.															

**Return Value** The return value is nonzero if the selection was successful. Otherwise, it is zero.

**Comments** The Text Control's button bar, ruler and status bar need the default automatic mode for correct working. Therefore when a selection is not longer needed it should be reset to the default mode.

**Example** The following message sequence gets the size of the current header text:

```
LONG lTextSize;
SendMessage(hwndTX, TX_HF_SELECT, 0, TF_HF_HEADER);
lTextSize = SendMessage(hwndTX, TX_GETTEXTSIZE, 0, 0L);
SendMessage(hwndTX, TX_HF_SELECT, 0, TF_HF_AUTO);
```

---

## TX\_HF\_SETPOSITION

This message sets a new position for a header or footer. For headers the position value is the distance between the top of the header and the top of the page. For footers the position value is the distance between the bottom of the footer and the bottom of the page. All values are in twips. The default value is 567 twips = 1 cm.

<u>Parameter</u>	<u>Description</u>										
<i>wParam</i>	Specifies the header or footer the position of which is to be set. It can be one of the following values:										
	<table> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>TF_HF_HEADER</td> <td>Sets the header's position.</td> </tr> <tr> <td>TF_HF_1STHEADER</td> <td>Sets the position of the special header for the first page.</td> </tr> <tr> <td>TF_HF_FOOTER</td> <td>Sets the footer's position.</td> </tr> <tr> <td>TF_HF_1STFOOTER</td> <td>Sets the position of the special footer for the first page.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	TF_HF_HEADER	Sets the header's position.	TF_HF_1STHEADER	Sets the position of the special header for the first page.	TF_HF_FOOTER	Sets the footer's position.	TF_HF_1STFOOTER	Sets the position of the special footer for the first page.
<u>Value</u>	<u>Meaning</u>										
TF_HF_HEADER	Sets the header's position.										
TF_HF_1STHEADER	Sets the position of the special header for the first page.										
TF_HF_FOOTER	Sets the footer's position.										
TF_HF_1STFOOTER	Sets the position of the special footer for the first page.										
<i>lParam</i>	Specifies the new position.										

**Return Value** The return value is non-zero if the position could be set, otherwise it is zero.

## TX\_INPUTPOSFROMPOINT

This message returns the text input position belonging to a certain geometric position. The text input position is relative to the beginning of the text and the geometric position is a position in the visible part of the text.

<b><u>Parameter</u></b>	<b><u>Description</u></b>						
<i>wParam</i>	Specifies the coordinate system for the position values. It can be one of the following values:						
	<table> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>UF_PIXEL</td> <td>The position values are in pixels.</td> </tr> <tr> <td>UF_TWIPS</td> <td>The position values are in terms of twentieths of a point.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	UF_PIXEL	The position values are in pixels.	UF_TWIPS	The position values are in terms of twentieths of a point.
<b><u>Value</u></b>	<b><u>Meaning</u></b>						
UF_PIXEL	The position values are in pixels.						
UF_TWIPS	The position values are in terms of twentieths of a point.						
<i>lParam</i>	Points to a POINT variable containing the position values.						

***Return Value*** The return value specifies the text input position beginning with zero for the position in front of the first character. The return value is -1, if a text position could not be found.

---

## TX\_LIMITLINE

This message limits the number of characters one line can contain.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains the new limit value. The value can vary between 1 and 0xffff.
<i>lParam</i>	Specifies whether or not the Text Control is to be updated. If <i>lParam</i> is zero, the Text Control will not be updated. If <i>lParam</i> is nonzero, the Text Control will be updated.

***Comments*** The default line limit value is 0xffff - 1.

## TX\_LIMITTEXT

This message limits the number of characters which may be entered by the user.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Contains the new limit value which can vary between 1 and 0xffff. If <i>wParam</i> contains zero the amount of text is unlimited.
<i>lParam</i>	Is not used.

**Comments** The Text Control beeps if the user tries to enter more text than the given limit.  
If a Text Control has a following window in a chain of linked windows the amount of text is always unlimited.

---

## TX\_LINEFROMCHAR

This message returns the line number of the line which contains the character whose position (indexed from the beginning of the text) is specified by the *lParam* parameter.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Contains the zero-based index value for the desired character in the text.

**Return Value** The return value is a line number. The first line has the number 1. The return value is zero if an error has occurred.

---

## TX\_LINEFROMPOINT

This message returns the number of the line which contains a given point. The point must be specified in pixels with an origin at the top left corner of the window's client area.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a POINT data structure which identifies the point.

**Return Value** The return value contains the line number in the low-order word. The first line has the number 1. The return value is zero if an error has occurred.

---

## TX\_LINEINDEX

This message receives a line number. It returns the number of characters in the Text Control that precede the first character in that line.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is the given line number. The line number of the first line is zero.

**Return Value** The return value is the index of the first character of the specified line. The character index of the first line is always zero. If the line specified is not the first line and the return value is zero an error has occurred.

---

## TX\_LOAD

This message fills the internal data buffers of the Text Control with data stored in a file. The data must have been previously stored with the TX\_SAVE message.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Contains a DOS file handle. The file pointer must be positioned at the head of the data.
<i>lParam</i>	Points to a POINT data structure. The Text Control copies the minimum window size (in pixels) to this buffer. If a text area has been defined with the TX_SETTEXTAREA message the new minimum text area is copied in

twentieths of a point. The size is calculated from the font information of the loaded text. If both values are 0, an error has occurred.

- Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.
- Comments** The return value is also zero if the window size is too small to show a minimum of the text. After resizing the window with the minimum values given by *lParam* the message can be sent again.
- The file pointer is only moved if the return value is nonzero.
- Unicode** This message does not support Unicode. To load text previously saved in Unicode format use the TX\_DATAIN message.

---

## TX\_OBJ\_DELETE

This message deletes an embedded object.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a unique identifier returned by the TX_OBJ_EMBED message. If this parameter is zero the currently selected object is deleted.
<i>lParam</i>	Is not used.

- Return Value** The return value is zero if an error has occurred or if either an invalid identifier is specified or no object is currently selected. Otherwise it is nonzero.
- Comments** This message generates a TN\_OBJ\_DELETED notification message.

---

## TX\_OBJ\_EMBED

This message embeds a new object or image. Depending on the TXOBJECT structure member *wEmbedMode*, the object is either handled like a single character

in the text or is fixed at a specified position in the text area and the text then flows around the object.

<b><u>Parameter</u></b>	<b><u>Description</u></b>														
<i>wParam</i>	Specifies the kind of object to be integrated. It can be any one of the following values:														
	<table border="0"> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>TF_OBJ_IMAGECONTROL</td> <td>Creates an Image-Control window and handles this window internally.</td> </tr> <tr> <td>TF_OBJ_OLEOBJECT</td> <td>Creates an OLE object. The type of object can be selected with the system embedded <i>OLE Insert</i> dialog box.</td> </tr> <tr> <td>TF_OBJ_OLEPROGID</td> <td>Creates a newly created OLE object. In this case the <i>Filename</i> parameter must specify a string which is the programmatic identifier of the OLE object to insert. The programmatic identifier is stored under the <i>ProgID</i> key in the registration database. For example the programmatic identifier of a Text Control 5.0 is TX.TextControl.110.</td> </tr> <tr> <td>TF_OBJ_OLEEMBEDFILE</td> <td>Inserts a newly created embedded OLE object from a file. In this case the <i>Filename</i> parameter must specify a valid filename.</td> </tr> <tr> <td>TF_OBJ_OLELINKFILE</td> <td>Inserts a newly created linked OLE object from a file. In this case the <i>Filename</i> parameter must specify a valid filename.</td> </tr> <tr> <td>Otherwise</td> <td>Specifies an externally created window like a Windows button that represents the object to be inserted.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	TF_OBJ_IMAGECONTROL	Creates an Image-Control window and handles this window internally.	TF_OBJ_OLEOBJECT	Creates an OLE object. The type of object can be selected with the system embedded <i>OLE Insert</i> dialog box.	TF_OBJ_OLEPROGID	Creates a newly created OLE object. In this case the <i>Filename</i> parameter must specify a string which is the programmatic identifier of the OLE object to insert. The programmatic identifier is stored under the <i>ProgID</i> key in the registration database. For example the programmatic identifier of a Text Control 5.0 is TX.TextControl.110.	TF_OBJ_OLEEMBEDFILE	Inserts a newly created embedded OLE object from a file. In this case the <i>Filename</i> parameter must specify a valid filename.	TF_OBJ_OLELINKFILE	Inserts a newly created linked OLE object from a file. In this case the <i>Filename</i> parameter must specify a valid filename.	Otherwise	Specifies an externally created window like a Windows button that represents the object to be inserted.
<b><u>Value</u></b>	<b><u>Meaning</u></b>														
TF_OBJ_IMAGECONTROL	Creates an Image-Control window and handles this window internally.														
TF_OBJ_OLEOBJECT	Creates an OLE object. The type of object can be selected with the system embedded <i>OLE Insert</i> dialog box.														
TF_OBJ_OLEPROGID	Creates a newly created OLE object. In this case the <i>Filename</i> parameter must specify a string which is the programmatic identifier of the OLE object to insert. The programmatic identifier is stored under the <i>ProgID</i> key in the registration database. For example the programmatic identifier of a Text Control 5.0 is TX.TextControl.110.														
TF_OBJ_OLEEMBEDFILE	Inserts a newly created embedded OLE object from a file. In this case the <i>Filename</i> parameter must specify a valid filename.														
TF_OBJ_OLELINKFILE	Inserts a newly created linked OLE object from a file. In this case the <i>Filename</i> parameter must specify a valid filename.														
Otherwise	Specifies an externally created window like a Windows button that represents the object to be inserted.														

The child identifier for external windows must be no larger than 0x7FFF.

*lParam* Points to an TXOBJECT data structure which defines the object's attributes. A complete description of this data structure can be found in chapter 7 "*Data Structures*".

**Return Value** The low-order word of the return value is zero if an error has occurred. Otherwise it is a unique identifier that can be used by an application to change the object's attributes later. See the TX\_OBJ\_GETATTR and the TX\_OBJ\_SETATTR messages for more information.

If an error has occurred, the high-order word contains an error code value if the inserted object is an Image-Control. For more information about the meaning of this value see the description of the IC\_SETIMAGE message in the IC Image-Control reference.

**Comments** The IC Image-Control programming tool is needed for this message if the object is an image.

If an externally created window is embedded the Text Control automatically becomes the parent window and sends messages to this window. For more information see chapter 1.9 "*Integrating External Windows*" or the message descriptions in appendix H.

A fixed-positioned object can be moved and sized with an internal mouse interface. To activate the interface the object must be clicked whilst pressing the ALT key.

If an error has occurred the Text Control destroys the external window.

---

## TX\_OBJ\_GETATTR

This message retrieves information about the attributes of an embedded object like embedding mode, position or scaling factor.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a unique identifier returned by the TX_OBJ_EMBED message. If this parameter is zero information about the currently selected object is retrieved.

*lParam* Points to an TXOBJECT data structure whose members are filled with the specified object's attributes. A complete description of this data structure can be found in chapter 7 "Data Structures". The member *lTextPos* is only filled for inserted objects (mode EOM\_INSERT) and the members *xPosition* and *lyPosition* are only filled for fixed-positioned objects (other modes). The following structure's members must be filled by the caller before sending this message:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wTXVersion</i>	Must be set to the current Text Control's version number.
<i>lpFileName</i>	Must point to a buffer of length <i>nBufLength</i> . This member can be set to zero if the object is represented by an externally created window.
<i>nBufLength</i>	Must specify the length, in bytes, of the buffer <i>lpFileName</i> points to. This member can be set to zero if the object is represented by an externally created window.
<i>nFilterIndex</i>	This member is not used.

**Return Value** The return value is zero if an error has occurred or if either an invalid identifier is specified or no object is currently selected. Otherwise it is nonzero.

**Comments** The IC Image-Control programming tool is needed for this message if the specified object is an image.

---

## TX\_OBJ\_GETDISPINTERFACE

This message returns a pointer to an object's dispatch interface. It can be used to call properties and methods for an object.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a unique identifier returned by the TX_OBJ_EMBED message.
<i>lParam</i>	Is not used.

**Return Value** The return value points to an object's dispatch interface. It is zero if such an interface could not be found.

## TX\_OBJ\_GETNEXT

This message returns the identifier of the embedded object that follows the specified object in the Text Control's internal list of objects.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a unique identifier returned by the TX_OBJ_EMBED message. If this parameter is zero the first object's identifier is returned.
<i>lParam</i>	The low-order word specifies the kind of object. It can be a combination of any of the following values:

<u>Value</u>	<u>Meaning</u>
OGN_FIXEDONLY	Returns only identifiers of fixed positioned objects.
OGN_ASCHARONLY	Returns only identifiers of objects that act as single characters.
OGN_IMAGESONLY	Returns only identifiers of objects which are internally created by the Text Control using the Image-Control module.
OGN_EXTERNALONLY	Returns only identifiers of objects which are externally created by the application.
OGN_OLEOBJECTONLY	Returns only identifiers of OLE objects.

If *lParam* is zero the identifiers of all the objects are returned.

**Return Value** The return value is the identifier of the object which follows the specified object. It is zero if no following object exists.

**Example**

The following code example uses the TX\_OBJ\_GETNEXT message to delete all the objects a Text Control contains:

```
WORD wID;

while (wID = (WORD)SendMessage(hTX, TX_OBJ_GETNEXT, 0, 0L)) {
    SendMessage(hTX, TX_OBJ_DELETE, (WPARAM)wID, 0L);
}
```

---

**TX\_OBJ\_OLE\_CANCEL**

This message deactivates an OLE object and changes its state from in-place activated to selected. This message can be used to implement the standard action for the ESCAPE key in a OLE container application.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

---

**TX\_OBJ\_SETATTR**

This message changes the attributes of an embedded object.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a unique identifier returned by the TX_OBJ_EMBED message. If this parameter is zero the attributes of the currently selected object are changed.
<i>lParam</i>	Points to an TXOBJECT data structure. The following structure's members can be used to change attributes:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wxScale</i>	Sets the object's horizontal scaling factor as a percentage. This must be a value between 10 and 250. If <i>wxScale</i> is zero this parameter is ignored.
<i>wyScale</i>	Sets the object's vertical scaling factor as a percentage. This must be a value between 10

---

	and 250. If <i>wyScale</i> is zero this parameter is ignored.
<i>Distances</i>	Specifies the distances between the object and the text. For objects inserted as a single character this parameter is ignored.
<i>wFlags</i>	Specifies a combination of flags. See the description of the TX_OBJ_EMBED message for more information about possible values. The ICF_BKGNDIMAGE flag cannot be set with this message.
<i>lpFileName</i>	If the specified object is an image this parameter can contain the filename of a new image. If this parameter contains zero it is ignored.
<i>nBufLength</i>	Specifies the length, in bytes, of the buffer <i>lpFileName</i> points to.
<i>nFilterIndex</i>	Specifies an image filter for loading the data contained in the file specified by <i>lpFileName</i> . See the description of the TX_OBJ_EMBED message for more information.

***Return Value*** The low-order word of the return value is zero, if an error has occurred or if either an invalid identifier is specified or no object is currently selected. It is also zero if a scaling factor has been specified such that the object's window becomes too large for the Text Control's window size. Otherwise the low-order word is either TR\_UNCHANGED if nothing could be changed or TR\_CHANGED if the new values were set and updated.

If an error occurred during the loading of a new image, the high-order word contains an error code value. For more information about the meaning of this value see the description of the IC\_SETIMAGE message in the IC Image-Control reference.

***Comments*** The IC Image-Control programming tool is needed for this message if the specified object is an image.

A complete description of the TXOBJECT data structure can be found in chapter 7 "Data Structures".

---

## TX\_PARAGRAPHDIALOG

This message opens a modal dialog box which can be used to set attributes for all currently selected paragraphs. The attributes are linespacing, alignment, indents and the distance to the previous and the following paragraphs. All values are set in millimeters, the value for linespacing can be set either in millimeters or as a multiple of the font size.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

***Return Value*** The return value is:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TR_ERR	If an error has occurred.
TR_UNCHANGED	If the user leaves the dialog box with the CANCEL button.
TR_CHANGED	If the user leaves the dialog box with the OK button.

***Comments*** The Text Control updates the modified selection if the window is visible.

---

## TX\_PASTEDATA

This message sets the data of a Text Control previously obtained with the TX\_COPYDATA message. The data contains all the text and format information the Text Control needs. This message can be used only to set the contents of an empty Text Control. If a control contains text, this text is deleted.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a buffer which contains the data.

<b><i>Return Value</i></b>	The return value points to the position in the buffer behind the pasted data. It is zero if an error has occurred.
<b><i>Comments</i></b>	<p>The font information contained by the data is adjusted for the currently selected printer. The Text Control returns 0L if the window is too small for at least the first character.</p> <p>The Text Control formats the new text but does not update its client area.</p> <p>Text can be formatted externally and then pasted with this message. The data must be formatted according to appendix A.</p>

---

## TX\_PRINT

This message can be sent to print the contents of the Text Control with the device identified by the printer-DC.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains a printer device context.
<i>lParam</i>	Points to a data structure of type PRINTSCT that is defined as followed:

```
typedef struct tagPRINTSCT {
    RECT    rPage;
    RECT    rBand;
    WORD    wOptions;
    WORD    wZoom;
} PRINTSCT;
typedef PRINTSCT *PPRINTSCT;
typedef PRINTSCT FAR *LPPRINTSCT;
```

The PRINTSCT data structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>rPage</i>	Specifies the part of the Text Control window that covers the page currently printed. The components of the rectangle must be in millimeters with the origin at the upper left corner of the Text Control window.
<i>rBand</i>	Specifies the part of the Text Control window that covers the band currently printed. The coordinates

must also be in mm with the origin at the upper left corner of the Text Control window. If banding is not used, *rBand* should contain the same values as *rPage*.

*wOptions* Specifies print options. It can be a combination of the following values:

**Value:**

**Meaning:**

TF\_PRINTCOLORS

Specifies that text colors are to be printed. If this flag is not set the text is printed in black.

TF\_PRINTSCALED

Specifies that text to be printed will be scaled. The scaling factor is given through the structure member *wZoom*.

*wZoom* Specifies a scaling factor in percent. This value is only be used if the TF\_PRINTSCALED option is set. The value can differ from 10 to 400 percent.

If *lParam* is zero, the contents of the Text Control is printed with the same offset, the Text Control window has relative to the client area of its parent window. This can be used if one page contains more than one Text Control.

If *lParam* is nonzero the SetViewPortOrg( ) function can be used to realize an additional top or left offset. This offset is added to the printer offset retrieved by the GETPRINTINGOFFSET Escape function. To avoid clipping, the size of the TX window and the dimension given by the *rPage* rectangle of the PRINTSCT data structure must be calculated again.

***Return Value***

The low-order word of the return value contains the position next to the last printed line. This value can be less than the bottom value of the *rPage* rectangle. The return value should be used as the top value of the next page.

The high order word is zero if an error has occurred. Otherwise it is nonzero.

**Comments**

This message cannot be used if a text formatting area has been set with the TX\_SETTEXTAREA message. In this case the TX\_PRINTPAGE message has to be used.

**Example**

The following code demonstrates the use of this message for printing one or more pages:

```
#define TwipsToMM(val) ((int)((val)*254L/14400L)
#define INCH_INTWIPS      1440
#define DINA4_WIDTH      11906
#define DINA4_HEIGHT     16837

static DOCINFO    DocInfo;
PRINTSCT         printsct;
LONG             lReturn, lHeight;
int              nTextHeight, nPageHeight;

// get the height of the text and convert it to mm:
lHeight = SendMessage(hTXWnd, TX_GETTEXTHEIGHT, 0, 0L);
nTextHeight = TwipsToMM(lHeight);
nPageHeight = TwipsToMM(DINA4_HEIGHT-2*INCH_INTWIPS);

// init the PRINTSCT data structure:
SetRect(    &printsct.rPage,
            0, 0,
            TwipsToMM(DINA4_WIDTH-2*INCH_INTWIPS),
            nPageHeight);
printsct.wOptions = TF_PRINTCOLORS;
printsct.wZoom = 100;

ZeroMemory(&DocInfo, sizeof(DOCINFO));
DocInfo.cbSize = sizeof(DOCINFO);
DocInfo.lpszDocName = (LPTSTR)"Test Application";

if (StartDoc(hPrintDC, &DocInfo) == SP_ERROR) {
    goto print_end;
}

// print the pages:
lReturn = 0L;
while ((int)LOWORD(lReturn) < nTextHeight) {

    if (StartPage(hPrintDC) <= 0) {
        AbortDoc(hPrintDC);
        goto print_end;
    }
}
```

```

    }

    // fill the page relative values of the PRINTSCT structure:
    printsct.rPage.top = (int)LOWORD(lReturn);
    printsct.rPage.bottom = printsct.rPage.top + nPageHeight;
    CopyRect(&printsct.rBand, &printsct.rPage);

    // print the page:
    lReturn = SendMessage(hTXWnd, TX_PRINT, (WPARAM)hPrintDC,
        (LONG)(LPPRINTSCT)&printsct);
    if (HIWORD(lReturn) == 0) {
        AbortDoc(hPrintDC);
        goto print_end;
    }

    if (EndPage(hPrintDC) < 0) {
        AbortDoc(hPrintDC);
        goto print_end;
    }
}

EndDoc(hPrintDC);

print_end:
...

```

---

## TX\_PRINTPAGE

This message prints one page of the Text Control's current contents.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a printer device context.
<i>lParam</i>	Points to a PRINTPAGE data structure that is defined as follows:

```

typedef struct tagPRINTPAGE {
    WORD  wStructSize;
    WORD  wPageNo;
    WORD  wZoom;
    RECT  rPage;
    RECT  rBand;
    WORD  wOptions;
} PRINTPAGE;
typedef PRINTPAGE *PPRINTPAGE;
typedef PRINTPAGE FAR *LPPRINTPAGE;

```

The PRINTPAGE data structure has the following fields:

<u>Field</u>	<u>Description</u>
<i>wStructSize</i>	Specifies the size of the PRINTPAGE data structure in bytes.
<i>wPageNo</i>	Specifies the number of the page to print. The first page has the number one.
<i>wZoom</i>	Specifies a scaling factor in percent. This value can range from 10 to 400.
<i>rPage</i>	Specifies the page alignment on the current printer's paper. The top left corner specifies the printing offset. The rectangle's width specifies the formatting width of the text. To get WYSIWYG, this width must be the same as that specified with the TX_SETTEXTAREA message. The rectangle's height should be the same as specified for the TX_GETPAGECOUNT message. All values must be given in twentieths of a point.
<i>rBand</i>	Specifies the alignment of the current printed band. If banding is not used, <i>rBand</i> should contain the same values as <i>rPage</i> .
<i>wOptions</i>	Specifies print options. It must contain TF_PRINTCOLORS if text colors are to be printed. If <i>wOption</i> contains zero, text is printed in black.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Example** The following code demonstrates the use of this message for printing one or more pages:

```
#define INCH_INTWIPS      1440
#define DINA4_WIDTH      11906
#define DINA4_HEIGHT     16837

static DOCINFO DocInfo;
WORD wPages, wNum;
PRINTPAGE prpage;
```

```
// set the dimensions of the page and get number of pages:
SetRect(&prpage.rPage, INCH_INTWIPS, INCH_INTWIPS,
        DINA4_WIDTH-INCH_INTWIPS,
        DINA4_HEIGHT-INCH_INTWIPS);

wPages = (WORD)SendMessage(hWnd, TX_GETPAGECOUNT, 0,
                           prpage.rPage.bottom-prpage.rPage.top);

prpage.wStructSize = sizeof(PRINTPAGE);
prpage.wZoom = 100;
prpage.wOptions = 0;
CopyRect(&prpage.rBand, &prpage.rPage);

ZeroMemory(&DocInfo, sizeof(DOCINFO));
DocInfo.cbSize = sizeof(DOCINFO);
DocInfo.lpszDocName = (LPTSTR)"Test Application";

if (StartDoc(hPrintDC, &DocInfo) == SP_ERROR) {
    goto print_end;
}

for (wNum=1; wNum<=wPages; wNum++) {

    if (StartPage(hPrintDC) <= 0) {
        AbortDoc(hPrintDC);
        goto print_end;
    }

    prpage.wPageNo = wNum;
    if (! SendMessage(hTXWnd, TX_PRINTPAGE,
                     (WPARAM)hPrintDC, (LPARAM)(LPPRINTPAGE)&prpage)) {
        AbortDoc(hPrintDC);
        goto print_end;
    }
    if (EndPage(hPrintDC) < 0) {
        AbortDoc(hPrintDC);
        goto print_end;
    }
}

EndDoc(hPrintDC);

print_end:
...
```

## TX\_REDO

This message restores the last undone Text Control operation.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is zero if the redo operation fails. Otherwise it is nonzero.

**Comments** The TX\_CANUNDO message can be used to determine whether an undone Text Control operation can be itself be undone. If this message is sent despite there being no undone operation the Text Control beeps.

---

## TX\_REPLACESEL

This message replaces the currently selected text with new text.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a null-terminated character string which is to replace the currently selected text.

**Return Value** The return value is:

<u>Value</u>	<u>Meaning:</u>
TR_ERR,	If an error has occurred.
TR_CHANGED	Otherwise

---

## TX\_REPLACETEXT

This message opens the system-defined modeless dialog box which makes it possible for the user to find and replace text within the Text Control's contents.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

---

## TX\_RESETCONTENTS

This message deletes the complete contents of a Text Control including tables, objects, marked text fields and headers and footers.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is non-zero if everything could be deleted. Otherwise it is zero.

---

## TX\_SAVE

This message saves all internal text and formatting information, that the Text Control needs for displaying the text, to a file.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Contains a DOS file handle.
<i>lParam</i>	Is not used.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** The Text Control does not save the size of its window.  
The Text Control does not save modes set with the TX\_SETMODE message.

**Unicode** This message does not support Unicode. To save text in Unicode format use the TX\_DATAOUT message.

## TX\_SELTEST

This message can be used to check whether a selection is visible and which part is invisible.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	This parameter must be nonzero if <i>lParam</i> points to an array of two DWORD variables.
<i>lParam</i>	Points to an array of two DWORD variables specifying the selection to be checked. The start position is in the first variable and the end position is in the second variable. If <i>wParam</i> contains zero, <i>lParam</i> can contain the selection directly packed in its low-order and high-order words. The start position is in the low-order word and the end position is in the high-order word.

***Return Value*** The return value is -1 if an error has occurred. Otherwise it contains one of the following numbers:

The low-order word contains

- 0 If the selection would be completely invisible.
- 1 If the end position would be invisible.
- 2 If the start position would be invisible.
- 3 If both positions would be visible.

The high-order word contains

- 0 If the selection would be below the window area.
- 1 If the selection would be above the window area.

***Comments*** This message can be used to find information about the alignment of a given string for implementing search and replace features.

---

## TX\_SETBASELINE

This message sets a new baseline alignment value for the currently selected text.

<b><u>Parameter</u></b>	<b><u>Description</u></b>								
<i>wParam</i>	Contains one of the following values: <table border="0" style="margin-left: 2em;"> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>FA_STANDARD</td> <td>The new baseline alignment is set to zero</td> </tr> <tr> <td>FA_SUPERSCRIPT</td> <td>The new baseline alignment is superscript</td> </tr> <tr> <td>FA_SUBSCRIPT</td> <td>The new baseline alignment is subscript</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	FA_STANDARD	The new baseline alignment is set to zero	FA_SUPERSCRIPT	The new baseline alignment is superscript	FA_SUBSCRIPT	The new baseline alignment is subscript
<b><u>Value</u></b>	<b><u>Meaning</u></b>								
FA_STANDARD	The new baseline alignment is set to zero								
FA_SUPERSCRIPT	The new baseline alignment is superscript								
FA_SUBSCRIPT	The new baseline alignment is subscript								
<i>lParam</i>	Contains the new baseline alignment value. The value must be in twentieths of a point. For example, to set a baseline alignment value of 3 pt subscript, <i>lParam</i> must contain 60. If <i>wParam</i> contains FA_STANDARD, <i>lParam</i> is not used.								

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

***Comments*** The baseline alignment value is limited to 48 pt.  
The Text Control updates the modified selection if the window is visible.

---

## **TX\_SETBKGNDCOLOR**

This message sets a new background color. The Text Control uses this color to paint the background in TF\_OPAQUE mode. The default value for the background color is the system color for the window background.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	If <i>wParam</i> is nonzero, the new background color is the system color for the window background. If <i>wParam</i> is zero, the new background color value is given by <i>lParam</i> .
<i>lParam</i>	Specifies an RGB value that identifies the new background color.

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

*Comments*      The Text Control is refreshed if the window is visible.

---

## TX\_SETCARETEXT

This message sets the width of the caret. The caret's height depends on the current font.

<u>Parameter</u>	<u>Description</u>						
<i>wParam</i>	Contains one of the following values: <table> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>TF_TEXTCARET</td> <td>Sets the caret's width inside standard text sections.</td> </tr> <tr> <td>TF_FIELDCARET</td> <td>Sets the caret's width inside marked text fields.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	TF_TEXTCARET	Sets the caret's width inside standard text sections.	TF_FIELDCARET	Sets the caret's width inside marked text fields.
<u>Value</u>	<u>Meaning</u>						
TF_TEXTCARET	Sets the caret's width inside standard text sections.						
TF_FIELDCARET	Sets the caret's width inside marked text fields.						
<i>lParam</i>	Specifies the caret's new width in pixels. This value must be in the low-order word and the high-order word must be set to zero. A value of zero resets the width to its default value which is the system-defined window-border width in standard text sections and 2 pixels in marked text fields. The maximum width is 255 pixels.						

*Return Value*      The return value is zero if an error has occurred. Otherwise it is non-zero.

---

## TX\_SETDEVICE

This message sets a new device to which the text of the Text Control is formatted. If the text is formatted with fonts not supported by the new device these fonts are adapted.

<u>Parameter</u>	<u>Description</u>						
<i>wParam</i>	Contains one of the following values: <table> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>TF_SCREEN</td> <td>The new device is the screen.</td> </tr> <tr> <td>TF_STANDARD</td> <td>The new device is the standard</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	TF_SCREEN	The new device is the screen.	TF_STANDARD	The new device is the standard
<u>Value</u>	<u>Meaning</u>						
TF_SCREEN	The new device is the screen.						
TF_STANDARD	The new device is the standard						

device specified in the [windows] section of the WIN.INI file.

TF\_PRINTER The new device is a printer specified by the string which *lParam* points to.

TF\_METAFILETARGET Sets a target device for metafile recording. In this case *lParam* must specify a valid device context handle.

*lParam* Depends on the *wParam* parameter:  
If *wParam* is TF\_PRINTER *lParam* must point to a buffer containing a zero-terminated string, which represents the name of the new device. This name must be specified in the same format as that used in the WIN.INI file, for example:

```
PostScript Printer, PSCRIPT, LPT1:
```

If *wParam* contains TF\_SCREEN or TF\_STANDARD this parameter can be zero.

If *wParam* contains TF\_METAFILETARGET this parameter must specify a valid device context handle. See chapter 1.13 "Using metafiles" for more information.

**Return Value** The return value is:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TR_ERR	An error has occurred.
TR_UNCHANGED	The specified device is the same as the current device.
TR_CHANGED	The new device has been set.
Otherwise	The new device could not be set because the Text Control's client area was too small to display the text with the adapted fonts. The return value is the minimum window size the Text Control needs to format the text for the specified device. The low-order word specifies the minimum width and the high-order word specifies the minimum height. The values are in twentieths of a point if a text-area has been set with the TX_SETTEXTAREA message, otherwise the values are in pixels.

**Comments** If the return value specifies a new minimum window size the application can enlarge the window and send this message again.

The device given through *lParam* must be one of the devices specified in the [devices] section of the WIN.INI file.

The Text Control is updated if the new device was able to be set.

---

## TX\_SETFONT

This message sets a new typeface and a new pointsize for all currently selected fonts.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies the new pointsize in the lower 15 bits. If the high-order bit is set this pointsize must be specified in twentieths of a point. If the high-order bit is not set, it must be specified in points. If <i>wParam</i> contains zero it is ignored and only the typeface will be set.
<i>lParam</i>	Points to a buffer of length LF_FACESIZE which contains the new typeface string. If <i>lParam</i> is zero, it is ignored and only the pointsize will be set.

**Return Value** The return value contains the new minimum window size (in pixels) that is calculated by the new font description. If a text area has been defined with the TX\_SETTEXTAREA message, the new minimum text area is returned in TWIPS. The width is in the low-order word and the height is in the high-order word. The return value is zero if an error has occurred.

**Comments** The Text Control does not accept the new font if its window size is too small to display at least one character of that font. After resizing the window with the minimum values returned, this message can be sent again.

The Text Control uses only fonts which are supported by the output device currently selected. If the given typeface or pointsize does not match any supported font, the Text Control adjusts the given values.

The Text Control does not support GDI's vector fonts.

The Text Control updates the modified selection if the window is visible.

**Unicode** The buffer length is in characters, when UNICODE is defined, otherwise it is in bytes.

---

## TX\_SETFONTATTR

This message adds the specified font attributes to all the fonts which the current selection contains.

<b><u>Parameter</u></b>	<b><u>Description</u></b>																												
<i>wParam</i>	Contains one or more of the following values:																												
	<table> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>FA_STANDARD</td> <td>Each font is set to normal.</td> </tr> <tr> <td>FA_BOLD</td> <td>Each font is set to bold.</td> </tr> <tr> <td>FA_ITALIC</td> <td>Each font is set to italic.</td> </tr> <tr> <td>FA_UNDERLINE</td> <td>Each font is set to underline.</td> </tr> <tr> <td>FA_STRIKEOUT</td> <td>Each font is set to strike out.</td> </tr> <tr> <td>FA_NOBOLD</td> <td>Resets each bold font.</td> </tr> <tr> <td>FA_NOITALIC</td> <td>Resets each italic font.</td> </tr> <tr> <td>FA_NOUNDERLINE</td> <td>Resets each underlined font.</td> </tr> <tr> <td>FA_NOSTRIKEOUT</td> <td>Resets each struck out font.</td> </tr> <tr> <td>FA_UL_DOUBLE</td> <td>Each font is set to double underline.</td> </tr> <tr> <td>FA_UL_WORDSONLY</td> <td>Words are underlined, word gaps are omitted. This value can only be used in combination with FA_UNDERLINE or FA_UL_DOUBLE.</td> </tr> <tr> <td>FA_UL_NODOUBLE</td> <td>Resets each double underlined font.</td> </tr> <tr> <td>FA_UL_NOWORDSONLY</td> <td>Resets each font that has the words</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	FA_STANDARD	Each font is set to normal.	FA_BOLD	Each font is set to bold.	FA_ITALIC	Each font is set to italic.	FA_UNDERLINE	Each font is set to underline.	FA_STRIKEOUT	Each font is set to strike out.	FA_NOBOLD	Resets each bold font.	FA_NOITALIC	Resets each italic font.	FA_NOUNDERLINE	Resets each underlined font.	FA_NOSTRIKEOUT	Resets each struck out font.	FA_UL_DOUBLE	Each font is set to double underline.	FA_UL_WORDSONLY	Words are underlined, word gaps are omitted. This value can only be used in combination with FA_UNDERLINE or FA_UL_DOUBLE.	FA_UL_NODOUBLE	Resets each double underlined font.	FA_UL_NOWORDSONLY	Resets each font that has the words
<b><u>Value</u></b>	<b><u>Meaning</u></b>																												
FA_STANDARD	Each font is set to normal.																												
FA_BOLD	Each font is set to bold.																												
FA_ITALIC	Each font is set to italic.																												
FA_UNDERLINE	Each font is set to underline.																												
FA_STRIKEOUT	Each font is set to strike out.																												
FA_NOBOLD	Resets each bold font.																												
FA_NOITALIC	Resets each italic font.																												
FA_NOUNDERLINE	Resets each underlined font.																												
FA_NOSTRIKEOUT	Resets each struck out font.																												
FA_UL_DOUBLE	Each font is set to double underline.																												
FA_UL_WORDSONLY	Words are underlined, word gaps are omitted. This value can only be used in combination with FA_UNDERLINE or FA_UL_DOUBLE.																												
FA_UL_NODOUBLE	Resets each double underlined font.																												
FA_UL_NOWORDSONLY	Resets each font that has the words																												

only bit. This value can only be used in combination with FA\_NOUNDERLINE or FA\_UL\_NODOUBLE.

FA\_TOGGLE Toggles the specified attributes instead of adding them.

The bitwise OR operator can be used to specify more than one value.

*lParam* Points to a POINT data structure. The Text Control copies the new minimum window size (in pixels) to this buffer. If a text area has been defined with the TX\_SETTEXTAREA message the new minimum text area is copied in TWIPS. This size is calculated from the new font attributes. If the values copied are both 0, an error has occurred.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** The FA\_TOGGLE flag can be set with any combination. The toggling of an attribute results in the deletion of this attribute if *wParam* contains the same value as all fonts of the current selection. The return value also evaluates TR\_ERR if the window size is too small for the fonts changed by this message. After resizing the window with the minimum values given by *lParam*, this message can be sent again. The Text Control updates the modified selection if the window is visible.

## TX\_SETFORMAT

This message sets the paragraph format value for the currently selected paragraphs.

<u>Parameter</u>	<u>Description</u>				
<i>wParam</i>	contains one of the following values:				
	<table border="1"> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>TF_LEFT</td> <td>Paragraphs are set to left aligned</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	TF_LEFT	Paragraphs are set to left aligned
<u>Value</u>	<u>Meaning</u>				
TF_LEFT	Paragraphs are set to left aligned				

	TF_RIGHT	Paragraphs are set to right aligned
	TF_CENTER	Paragraphs are set to centered text
	TF_BLOCK	Paragraphs are set to block format
<i>lParam</i>	Is not used.	

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** The Text Control updates the modified selection if the window is visible.

---

## TX\_SETIMAGE

This message sets a new image for the currently selected Image-Control window. If no Image-Control window is selected, the message is ignored.

### **Parameter**

### **Description**

*wParam*

Specifies an image filter as an index into the buffer returned by the TX\_GETIMAGEFILTERS message. See the description of the TX\_CREATEIMAGE message for more information.

*lParam*

Points to a null-terminated string that is a full DOS path name for the file containing the new image.

**Return Value** The low-order word of the return value is zero if an error has occurred. Otherwise it is nonzero. If an error has occurred, the high-order word contains an error code value. For more information about the meaning of this value see the description of the IC\_SETIMAGE message in the IC Image-Control reference.

**Comments** The IC Image-Control programming tool is needed for this message.

---

## TX\_SETINDENTS

This message sets new indent values for all currently selected paragraphs.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an array of five integers which contain the new indent values in twentieths of a point. The values are in the order: left indent, right indent, additional indent of the first line, top indent and bottom indent. The third value, the additional indent of the first line is the only one that may be negative. If one or more of the values are to be ignored, they must contain TR_IGNORED.
<b>Return Value</b>	The return value is:
TR_ERR	if an error has occurred.
TR_UNCHANGED	if the Text Control has not accepted the new values.
TR_CHANGED	if the new values have been set.
<b>Comments</b>	<p>The Text Control does not accept a combination of values that does not fit within the Text Control window rectangle. The maximum values are returned by the TX_GETINDENTS message. For more information see the description of this message.</p> <p>The Text Control updates the modified selection if the window is visible.</p>

---

## TX\_SETLANGUAGE

This message sets the language which the Text Control uses to display information strings, warnings or dialog boxes. The language is specified either through an identifier or through the filename of a resource library.

<u>Parameter</u>	<u>Description</u>				
<i>wParam</i>	Specifies a language identifier. Languages are identified through the same numbers used in the WIN.INI file. The default language is the current WIN.INI setting when the Text Control is created. Supported languages are:				
	<table> <thead> <tr> <th><u>Language</u></th> <th><u>Identifier</u></th> </tr> </thead> <tbody> <tr> <td>English</td> <td>01</td> </tr> </tbody> </table>	<u>Language</u>	<u>Identifier</u>	English	01
<u>Language</u>	<u>Identifier</u>				
English	01				

French	33
Spanish	34
Italian	39
German (Switzerland)	41
German (Austria)	43
German	49
Japanese	81 (32 bit only)

*lParam* Points to a zero-terminated character string that specifies the file name including its full path of a resource library. See the new chapter 1.15 "*Resources*" for more information about creating a resource library. When this parameter is zero, Text Control uses the built-in language specified through *wParam*, when this parameter is non-zero the *wParam* parameter is ignored.

**Return Value** The return value is zero if an error has occurred or if the specified language has already been set, otherwise it is nonzero.

---

## TX\_SETLINEANDCOL

This message sets a new text input position from a page, line and column number.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an array of three DWORD variables. The first variable specifies the page number, the second specifies the line number and the third the column number. When Text Control does not display pages, the first variable is ignored and should be set to zero.

**Return Value** The return value is non-zero if the specified input position could be set. Otherwise it is zero.

## TX\_SETLINESPACING

This message sets a new linespacing value for the currently selected paragraphs.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Contains the new linespacing value
<i>lParam</i>	If <i>lParam</i> is zero, <i>wParam</i> must contain a value (in percent) of the currently used font. If <i>lParam</i> is nonzero, <i>wParam</i> must contain a value in twips (twentieths of a point). Before setting the linespacing in twips, a TX_SETPARAFORMATFLAGS message can be sent to specify whether the linespacing is used as a minimum, or as an absolute value.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** To realize double linespacing, *wParam* must contain 200 and *lParam* must be zero. The Text Control updates the modified selection if the window is visible. Minimum and maximum values are: 10% to 400% for relative values, 57 to 5669 TWIPS (1 to 100 mm) for absolute values.

---

## TX\_SETLINKWND

This message informs the given window about a window that is to be its successor in a chain of linked windows. The Text Control sends overflowing text to that window or fills deleted text with text from that window. The caret moves to the next window if it reaches the bottom of a Text Control. It moves to the previous window if the top of a Text Control is reached.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a handle to the window that should be the following window. This handle must be a valid Text Control window handle returned by <b>CreateTextControl</b> . To disconnect an existing link, <i>wParam</i> can be set to zero.

*lParam* Is not used.

**Return Value** The return value is zero if the windows could not be linked. Otherwise it is nonzero.

**Comments** If a window becomes part of a chain of linked windows, the modes TF\_HIDESELNA and TF\_AUTOEXPAND are changed to TF\_SHOWSELNA and TF\_FIXED respectively.

Linked Text Control windows can have different parent windows.

While loading a chain of linked windows send this message after sending TX\_LOAD and before showing and updating the window.

---

## TX\_SETMODE

This message sets several different modes of the Text Control. Changing one mode does not alter the other mode settings.

<b><u>Parameter</u></b>	<b><u>Description</u></b>										
<i>wParam</i>	Is the new mode value. It can be a combination of the following values:										
	<table border="1"> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>TF_AUTOEXPAND</td> <td>If the Text Control window size is to be expanded automatically when the text insertion or format changes results in text that does not fit into the Text Control anymore.</td> </tr> <tr> <td>TF_FIXED</td> <td>If the Text Control window size is to be fixed.</td> </tr> <tr> <td>TF_FRAMED</td> <td>If a borderline is to be drawn on the screen.</td> </tr> <tr> <td>TF_HIDESELNA</td> <td>If the text selection is to be hidden after the Text Control has lost the input focus.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	TF_AUTOEXPAND	If the Text Control window size is to be expanded automatically when the text insertion or format changes results in text that does not fit into the Text Control anymore.	TF_FIXED	If the Text Control window size is to be fixed.	TF_FRAMED	If a borderline is to be drawn on the screen.	TF_HIDESELNA	If the text selection is to be hidden after the Text Control has lost the input focus.
<b><u>Value</u></b>	<b><u>Meaning</u></b>										
TF_AUTOEXPAND	If the Text Control window size is to be expanded automatically when the text insertion or format changes results in text that does not fit into the Text Control anymore.										
TF_FIXED	If the Text Control window size is to be fixed.										
TF_FRAMED	If a borderline is to be drawn on the screen.										
TF_HIDESELNA	If the text selection is to be hidden after the Text Control has lost the input focus.										

---

TF_HIDEWHITESPACE	If the space and paragraph end characters are to be hidden.
TF_INSERT	If the character insertion mode is to be set to insert.
TF_KEESEL	If the selected text is not to be deleted before character insertion.
TF_NOTFRAMED	If the borderline is to be hidden on the screen.
TF_OPAQUE	If the background mode is to be set to opaque.
TF_OVERWRITE	If the character insertion mode is to be set to overwrite.
TF_REPLACESEL	If selected text is to be deleted before character insertion.
TF_SHOWSELNA	If the text selection is to be visible in the inactive window state.
TF_SHOWWHITESPACE	If the space and paragraph end characters are to be made visible.
TF_TRANSPARENT	If the background mode is to be set to transparent.

The bitwise OR operator can be used to specify more than one value.

*lParam* Contains maximum values for the Text Control window size (in pixels) if the TF\_AUTOEXPAND flag is set. If the window is expanded and these values are reached, the automatic expansion stops. The maximum width is in the low-order word and the maximum height is in the high-order word.

**Return Value** The return value is zero if one of the new modes could not be set. Otherwise it is nonzero.

**Comments**

The Text Control is completely updated if a new background mode is set or the white space-characters are shown or hidden. White space characters and the window frame are only shown on the screen, they are not printed.

The mode flags are grouped. The following flags must not be used together:

TF\_TRANSPARENT and TF\_OPAQUE

TF\_OVERWRITE and TF\_INSERT

TF\_SHOWWHITESPACE and TF\_HIDEWHITESPACE

TF\_SHOWSELNA and TF\_HIDESELNA

TF\_FRAMED and TF\_NOTFRAMED

TF\_AUTOEXPAND and TF\_FIXED

TF\_KEEPESEL and TF\_REPLACESEL

The default values are TF\_TRANSPARENT, TF\_INSERT, TF\_HIDEWHITESPACE, TF\_HIDESELNA, TF\_NOTFRAMED, TF\_FIXED and TF\_REPLACESEL.

The Text Control sends TN\_HEXEXPAND and TN\_VEXPAND notification messages to the parent window if the TF\_AUTOEXPAND mode is set and the window size is changed.

Linked Text Control windows are always set to TF\_FIXED and TF\_SHOWSELNA. This modes can not be changed for linked windows.

---

**TX\_SETMODEEX**

This message is an expansion of the TX\_SETMODE message for providing more mode settings. Changing one mode does not alter other mode settings.

<b><u>Parameter</u></b>	<b><u>Description</u></b>	
<i>wParam</i>	Is the new mode value. It can be one or more of the following values:	
	<b><u>Value</u></b>	<b><u>Meaning</u></b>
	TF_DISPLAY	This mode can be used to display text only. Text input or selecting text with the mouse or the keyboard is not possible. The cursor is the standard arrow cursor.

---

TF_READONLY	This mode can be used to display and select text. The cursor is the standard arrow cursor.
TF_EDIT	This mode can be used to edit and display text. The cursor is the text I-beam cursor.
TF_NOWAITCURSOR	This mode shows no wait cursor during a long duration operation.
TF_WAITCURSOR	This mode shows an hourglass wait cursor during a long duration operation.
TF_TOPINDENTFIRSTPG	This mode allows a top indent for the first paragraph.
TF_NOTOPINDENTFIRSTPG	This mode suppresses the top indent for the first paragraph.
TF_ERRORBOXES	The Text Control displays error message boxes. Error message boxes are only displayed in some important cases. Most errors are handled with the TN_ERRCODE notification message.
TF_NOERRORBOXES	The Text Control suppresses all error message boxes.
TF_SHOWGRIDLINES	This mode can be used to show the grid lines in tables.
TF_HIDEGRIDLINES	This mode can be used to hide the grid lines in tables.

The bitwise OR operator can be used to specify more than one value.

IParam

Is not used.

**Return Value** The return value is zero if the new mode could not be set. Otherwise it is nonzero.

**Comments** The mode flags are grouped. The following flags cannot be combined:

TF\_DISPLAY, TF\_READONLY and TF\_EDIT  
 TF\_WAITCURSOR and TF\_NOWAITCURSOR  
 TF\_TOPINDENTFIRSTPG and TF\_NOTOPINDENTFIRSTPG  
 TF\_ERRORBOXES and TF\_NOERRORBOXES  
 TF\_SHOWGRIDLINES and TF\_HIDEGRIDLINES

The default values are TF\_EDIT, TF\_WAITCURSOR,  
 TF\_NOTOPINDENTFIRSTPG, TF\_ERRORBOXES and  
 TF\_SHOWGRIDLINES.

If the Text Control's display mode is set to TF\_DISPLAY or TF\_READONLY the standard arrow cursor can be changed by processing the parent window's WM\_SETCURSOR message.

---

## TX\_SETPAGEMARGINS

This message sets the page margins.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	If this parameter is nonzero the Text Control reformats the complete text. Otherwise the text is not reformatted. If this message is sent in combination with a TX_SETTEXTAREA message this message should be sent first with <i>wParam</i> set to zero to avoid double reformatting.
<i>lParam</i>	Points to a RECT data structure containing the new margin values. The values must be specified in twentieths of a point.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** Page margins are only shown on the screen if a text area has been set via the TX\_SETTEXTAREA message with a height value not set to -1.

## TX\_SETPARAFORMATFLAGS

This message sets special formats for all paragraphs selected.

<u>Parameter</u>	<u>Description</u>										
<i>wParam</i>	Contains one of the following values:										
	<table border="1"> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>TF_ATLEASTLINESPACING</td> <td>This value specifies that absolute linespacing values set with the TX_SETLINESPACING message are to be used as minimal values. If a line contains characters or images which would be cropped with this setting, the linespacing is enlarged accordingly.</td> </tr> <tr> <td>TF_EXACTLINESPACING</td> <td>This value specifies that absolute linespacing values set with the TX_SETLINESPACING message are to be used as exact values, regardless of whether larger characters or images are being cropped.</td> </tr> <tr> <td>TF_PAGEBREAKNOTALLOWED</td> <td>Within a paragraph a page break is not allowed.</td> </tr> <tr> <td>TF_PAGEBREAKALLOWED</td> <td>Page breaks are allowed within a paragraph. This is the default value.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	TF_ATLEASTLINESPACING	This value specifies that absolute linespacing values set with the TX_SETLINESPACING message are to be used as minimal values. If a line contains characters or images which would be cropped with this setting, the linespacing is enlarged accordingly.	TF_EXACTLINESPACING	This value specifies that absolute linespacing values set with the TX_SETLINESPACING message are to be used as exact values, regardless of whether larger characters or images are being cropped.	TF_PAGEBREAKNOTALLOWED	Within a paragraph a page break is not allowed.	TF_PAGEBREAKALLOWED	Page breaks are allowed within a paragraph. This is the default value.
<u>Value</u>	<u>Meaning</u>										
TF_ATLEASTLINESPACING	This value specifies that absolute linespacing values set with the TX_SETLINESPACING message are to be used as minimal values. If a line contains characters or images which would be cropped with this setting, the linespacing is enlarged accordingly.										
TF_EXACTLINESPACING	This value specifies that absolute linespacing values set with the TX_SETLINESPACING message are to be used as exact values, regardless of whether larger characters or images are being cropped.										
TF_PAGEBREAKNOTALLOWED	Within a paragraph a page break is not allowed.										
TF_PAGEBREAKALLOWED	Page breaks are allowed within a paragraph. This is the default value.										
<i>lParam</i>	Is not used.										

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** The Text Control updates the modified selection when the window is visible.  
 Default values are TF\_ATLEASTLINESPACING and TF\_PAGEBREAKALLOWED.

---

## TX\_SETPGFRAME

This message draws a frame around each of the selected paragraphs.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the appearance and the style of the frame. It can be a combination of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
BF_LEFTLINE	Draws a left frame part.
BF_RIGHTLINE	Draws a right frame part.
BF_TOPLINE	Draws a top frame part.
BF_BOTTOMLINE	Draws a bottom frame part.
BF_BOX	Draws a complete box.
BF_TABLINES	Draws a vertical line at each tabulator position.
BF_TABLE	Draws a complete box including vertical lines at each tabulator position.
BF_SINGLE	Draws a single line.
BF_DOUBLE	Draws a doubled line.
BF_NOLEFTLINE	Resets an existing left part.
BF_NORIGHTLINE	Resets an existing right part.
BF_NOTOPLINE	Resets an existing top part.
BF_NOBOTTOMLINE	Resets an existing bottom part.
BF_NOTABLINES	Resets existing tabulator lines.

**BF\_BOXCONNECT** Connects two sequential boxes to form a single box.

*lParam* Specifies the width of the lines in the low-order word and the distance of the frame from the text in the high-order word. Both values must be in twentieths of a point. The value in the low-order word is ignored if it contains zero, whilst the value in the high-order word is ignored if it contains -1.

**Return Value** The return value is zero if an error has occurred. Otherwise, it is nonzero.

## TX\_SETSCROLLPOS

This message sets a new scroll position.

<b><u>Parameter</u></b>	<b><u>Description</u></b>	
<i>wParam</i>	Specifies the direction. It can be one of the following values:	
	<b><u>Value</u></b>	<b><u>Meaning</u></b>
	TF_HSCROLL	Sets the horizontal scroll position.
	TF_VSCROLL	Sets the vertical scroll position.
<i>lParam</i>	Specifies the new scroll position in twentieths of a point. The text associated with this position is displayed at the top of the Text Control's client area.	

**Return Value** The return value is nonzero if the scrolling operation was able to be performed. Otherwise it is zero.

## TX\_SETSEL

This message sets and displays a new text selection.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	This parameter must be nonzero if <i>lParam</i> points to an array of two DWORD variables.

*lParam* Points to an array of two DWORD variables specifying the selection to be set. The start position is in the first variable and the end position is in the second variable. If *wParam* contains zero, *lParam* can contain the selection directly packed in its low-order and high-order words. The start position is in the low-order word and the end position is in the high-order word.

**Return Value** The return value is zero if an error has occurred. Otherwise, it is nonzero.

**Comments** The start position is always the position of the blinking caret. If the start position is zero and the end position is -1 the entire text is selected.

## TX\_SETTABS

This message sets a new tab list for the paragraphs currently selected.

<b><u>Parameter</u></b>	<b><u>Description</u></b>	
<i>wParam</i>	Specifies the units of the position values. It can be one of the following:	
	<b><u>Value</u></b>	<b><u>Meaning</u></b>
	UF_PIXEL	The tab position values are given in pixels.
	UF_TWIPS	The tab position values are given in terms of twentieths of a point.
<i>lParam</i>	Points to an array of type TABSCT and size NTABS containing the new tablist. See the description of the TX_GETTABS message for more information about the TABSCT structure.	

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** The Text Control updates the modified paragraphs. If the number of tabs to set is less than NTABS, the values of the rest of the list must be set to zero. The first tab position must be a nonzero value.

## TX\_SETTEXTAREA

This message defines the text area which is used by the Text Control to perform line breaking. This area can be more or less than the window size specified by the **CreateTextControl** function. Furthermore, this message can be used to set scrollbars, scrolling options and view modes.

### Parameter

### Description

*wParam*

Specifies the appearance and the style of the scroll interface that is necessary for a text area larger than the Text Control's window size. It can be a combination of the following values:

### Value

### Meaning

TF\_HSCROLL

Displays a horizontal scroll bar if necessary.

TF\_NOHSCROLL

Displays no horizontal scroll bar.

TF\_VSCROLL

Displays a vertical scroll bar if necessary.

TF\_NOVSCROLL

Displays no vertical scroll bar.

TF\_THUMBTRACK

The Text Control updates its client area whilst moving the scrollbar's scroll box (thumb).

TF\_THUMBPOSITION

The Text Control updates its client area when the scrollbar's scroll box (thumb) has reached a new position.

TF\_PAGEVIEW

Text Control displays pages with margins, borders and a gray background. This setting has only effect when the high-order word of the *lParam* parameter is larger than zero.

TF\_EXTPAGEVIEW

Text Control displays three-dimensional pages which are centered in the windows visible area. This setting has only effect when the high-order

word of the *lParam* parameter is larger than zero.

The default values are TF\_NOHSCROLL, TF\_NOVSCROLL and TF\_THUMBPOSITION.

*lParam* Specifies the width and the height of the text area in twentieths of a point. The low-order word specifies the width, the high-order word specifies the height. A height value of -1 specifies a variable height depending on the text. A width or height value of zero specifies that this value is always equal to the window size.

**Return Value** The return value is zero if an error has occurred. Otherwise, it is nonzero.

**Comments** This message cannot be used in combination with the TF\_AUTOEXPAND mode set with the TX\_SETMODE message and in combination with linked windows.

With a height value of -1 the Text Control does not show page margins and page gaps. This mode can therefore be used to realize a normal display mode instead of a page display mode.

If the window is sized and the size becomes larger than the specified text width or height, visible scrollbars are automatically hidden. To get the information about whether a scrollbar is currently visible or not, use the **GetWindowLong** function with the GWL\_STYLE flag.

---

## TX\_SETTEXTCOLOR

This message sets the text color and the text background color of the currently selected text to new values.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	If <i>wParam</i> is zero, the new text color value is given by <i>lParam</i> . Otherwise it can contain a combination of the following values:

<u>Value</u>	<u>Meaning</u>
CV_TEXTDEFAULT	The new text color is the system color for the window text. If this value is specified the first variable <i>lParam</i> points to is ignored.

---

CV_TEXTUSER	The new text color is specified by the first variable <i>lParam</i> points to.
CV_BKDEFAULT	The new text background color is the system color for the window background. If this value is specified the second variable <i>lParam</i> points to is ignored.
CV_BKCONTROL	The new text background color is Text Control's background color. If this value is specified the second variable <i>lParam</i> points to is ignored.
CV_BKUSER	The new text color is specified by the second variable <i>lParam</i> points to.

*lParam* If *wParam* is zero *lParam* specifies a RGB value that identifies a new text color. Otherwise *lParam* must point to an array of two DWORD variables. The first variable contains a new text color value and the second variable contains a new value for the text background color.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** The Text Control updates the modified selection if the window is visible.  
 If the Text Control's background mode is TF\_TRANSPARENT and the *wParam* parameter contains CV\_BKCONTROL the text background color is not drawn.  
 The default text background color setting is CV\_BKCONTROL.

---

## TX\_SETWORDDIVISION

This message informs the Text Control that an application-supplied word division function should be used to perform end of line word division. The application-supplied function defines where TX should divide a word and inserts a hyphen.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	If <i>wParam</i> is nonzero, TX calls the word division function every time it must reformat a line break. If <i>wParam</i> is zero, TX only calls the word division function to undo the changes of a previous dividing process.
<i>lParam</i>	Is the procedure-instance address of the application-supplied word-division function. See the following 'Comments' section for details.

**Comments** The callback function address, passed as the *lParam* parameter, must be created by using the **MakeProcInstance** function.

The callback function must use the Pascal calling convention and must be declared FAR.

**Callback Function** **WORD FAR PASCAL** *WordDivisionFunc* (*lpWord*, *wDividePos*, *wWordLength*, *lpChangedWord*, *bRechange*)

**LPSTR** *lpWord*;  
**WORD** *wDividePos*;  
**WORD** *wWordLength*;  
**LPSTR** *lpChangedWord*;  
**BOOL** *bRechange*;

*WordDivisionFunc* is a placeholder for the application-supplied function name. The actual name must be exported by including it in an EXPORTS statement in the application's module definition file.

<u>Parameter</u>	<u>Description</u>
<i>lpWord</i>	Points to the word which is to be divided.
<i>wDividePos</i>	Specifies the last possible divide position so that the first word part, from the beginning of the word, to the character with the position <i>wDividePos</i> , fits into the current line of text. The first character has the position 1.
<i>wWordLength</i>	Specifies the number of bytes of the word <i>pointed to by lpWord</i> .
<i>lpChangedWord</i>	Points to a buffer of size $2 * wWordLength$ that can be used to inform TX that the word division has resulted in

changed characters. For example, in the German language in some cases consonants are doubled, and dividing a »ck« results in »k-k«. In such cases the whole word, including altered or additional characters, must be copied to the buffer pointed to by *lpChangedWord*. A terminating zero must be added at the end so that TX can calculate the new word length. In normal cases *lpChangedWord* is not used.

*bRechange*

If *bRechange* is TRUE, the function call is used to undo changes of a previous dividing process. In that case the buffer that *lpWord* points to, contains the changed word, and *wDividePos* specifies the position that has caused the change. The original unchanged word must be copied to the buffer *pointed to by lpChangedWord*.

**Return Value**

The return value specifies the character position TX should use to divide the word. The word is divided and a hyphen is inserted behind the returned position. The first character has the number 1. If *lpChangedWord* is used, the return value must specify a character position in that buffer. If *bRechange* is TRUE, the return value is ignored.

**TX\_TABLE\_ATTRDIALOG**

This message opens a built-in dialog box for setting table attributes such as frames and distances.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value**

The return value is:

<u>Value</u>	<u>Meaning</u>
TR_ERR	If an error has occurred.

TR_UNCHANGED	If the user has quit the dialog box with the CANCEL button.
TR_CHANGED	If the user has quit the dialog box with the OK button.

**Comments**

The return value also evaluates TR\_ERR if the selection contains no table, or more than one table, or if the selected table is mixed with other text. When this message is called as a reaction to a menu, the TX\_TABLE\_ISPOSSIBLE message can be used to get the information whether or not table attributes can be set.

The Text Control updates the modified selection if the window is visible.

---

**TX\_TABLE\_CHANGEID**

This message can be used to set a user-defined table identifier.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the current identifier. When previously no user-defined value has been set this value must be the identifier returned from the TX_TABLE_INSERT message.
<i>lParam</i>	Specifies a new identifier. It must be a value between 10 and 0x7FFF.

**Return Value** The return value is zero if the new identifier could not be set. Otherwise it is non-zero.

---

**TX\_TABLE\_FROMCARETPOS**

This message retrieves the table identifier and the number of row and column for the current input position. The retrieved values are set to zero when the input position is not inside a table or when more than one table cell is selected.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an array of two WORD variables that receive the

row and column number for the input position. The row number is in the first variable.

**Return Value** The return value is the table identifier of the table with the current input position.

---

## TX\_TABLE\_DELETELINES

This message deletes the currently selected table lines or the table line at the current input position.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is zero if an error has occurred, if no table line is selected, or if the current input position is not within a table. Otherwise it is nonzero.

---

## TX\_TABLE\_GETATTROFCELL

This message retrieves information about the attributes of one or more table cells.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a table identifier.
<i>lParam</i>	Points to a SETATTROFCELL data structure. See the TX_TABLE_SETATTROFCELL message for more information about this structure's members. Before calling this message the members <i>wStructSize</i> , <i>wRow</i> and <i>wColumn</i> must be initialized. Text Control fills all other members with information about these cells. When a common attribute could not be found the corresponding member is set to TF_TABLEATTRDEFAULT.

**TX\_TABLE\_GETCELLATTR**

This message returns information about the attributes of all the table cells currently selected.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used
<i>lParam</i>	Is not used.

***Return Value***

The return value is a global memory handle which identifies a buffer containing the attributes of the cells. All metric values are in twentieths of a point. The return value is zero when an error has occurred or when the selection is not completely within a single table.

The data contained by the buffer is structured in the following form:

```
WORD          wTXVersion;
WORD          wRows;
WORD          wColumns;
short        nMaxXValue;
CELLATTR cellattr[];
```

These fields have the following meaning:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wTXVersion</i>	Specifies the Text Control's current version number in the same format as that returned by the <i>TXGetVersion</i> function, e.g. 500. Set this parameter as a number and not as a <i>TXGetVersion</i> function call.
<i>wRows</i>	Specifies the number of selected rows.
<i>wColumns</i>	Specifies the number of selected columns.
<i>nMaxXValue</i>	Specifies a maximum value for frames and distances between frame and text in horizontal direction. The sum of the left frame width, the right frame width the left text distance and the right text distance must not be larger than <i>nMaxXValue</i> to avoid complete invisible text.
<i>cellattr[ ]</i>	Is an array of CELLATTR data structures containing the attributes of the cells. The array contains

*wRows*\**wColumns* structures. It starts with the cells of the first row and continues with the cells of the subsegment rows. The CELLATTR data structure is described in chapter 7 "Data Structures".

**Comments** If an application has finished using the buffer it must free it with the **GlobalFree** function.

---

## TX\_TABLE\_GETCELLPOSITION

This message retrieves the indexes (one-based) of the first and the last character in a table cell.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a table identifier.
<i>lParam</i>	Points to an array of four DWORD variables. The first variable specifies the row number and the second variable specifies the column number of the table cell. These variables must be filled by the caller. The third and fourth variable retrieves the character indexes. The third variable contains the index of the first character and the fourth variable contains the index of the last character. These variables are filled by Text Control.

**Return Value** The return value is zero if an error has occurred or if the specified table identifier does not exist, otherwise it is nonzero.

**Comments** If tables are used in chains of linked windows the position values are relative to the beginning of the text that is the first character in the first window of the chain. To get the window which contains the table and the alignment of the table in that window use the TX\_GETLINKWND message with the GWTX\_FROMOFFSET and the GWTX\_GETOFFSET option.

## TX\_TABLE\_GETCOLPOSITIONS

This message returns the horizontal beginning and end positions of the table columns and text columns for the currently selected table. Within a table the beginning and end positions of the text differ from the column positions because table columns can have border widths, and distances between borders and the text.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used
<i>lParam</i>	Is not used.

### ***Return Value***

The return value is zero when an error has occurred or when the selection does not lie completely inside of a single table. Otherwise it is a global memory handle identifying a buffer that contains the positions values. All position values are in twentieths of a point. The data the buffer contains is structured in the following form:

```
WORD        wNumOfColumns;
WORD        wInputColumn;
COLPOS      colpositions[];

typedef struct tagCOLPOS {
    LONG    lxColumnLeft;
    LONG    lxColumnRight;
    LONG    lxTextLeft;
    LONG    lxTextRight;
} COLPOS;
```

These fields have the following meaning:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wNumOfColumns</i>	Specifies the number of columns this buffer describes. This is also the number of COLPOS structures the <i>colpositions</i> array contains.
<i>wInputColumn</i>	Specifies the number of the column containing the current input position. The first column has the number 1.
<i>colpositions[ ]</i>	Is an array of COLPOS data structures containing the position values.
<i>lxColumnLeft</i>	Specifies the column's left border.

---

<i>lxColumnRight</i>	Specifies the column's right border. This value must be equal to the next column's left border.
<i>lxTextLeft</i>	Specifies the left-most position at which text can be placed. Left paragraph indents are not included.
<i>lxTextRight</i>	Specifies the right-most position at which text can be placed. Right paragraph indents are not included.

**Comments** If an application has finished using the buffer it must free it with the **GlobalFree** function.

---

## TX\_TABLE\_GETNEXT

This message returns the identifier of a table that follows the specified table in the Text Control's current text. In a list of linked Text Controls the search is performed in all windows.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a table's identifier. This identifier must not be a user-defined identifier because user-defined identifiers are not unique. It must be a Text Control identifier that was previously returned by the TX_TABLE_INSERT message. If this parameter is zero, the first table's identifier is returned.
<i>lParam</i>	Is not used.

**Return Value** The low-order word of the return value is the identifier of the table which follows the specified table in the Text Control's text. The high-order word is the corresponding user-defined identifier of this table or zero, if a user-defined identifier does not exist. The return value is zero if no following tables exist or if the specified table identifier was invalid.

**Example** The following code example uses the TX\_TABLE\_GETNEXT message to get the number of rows and columns of all the tables a Text Control contains:

```
WORD wRowsAndCols[2];  
WORD wID = 0;
```

```

while (wID = LOWORD(SendMessage(hwndTX, TX_TABLE_GETNEXT, wID, 0L)))
{
    SendMessage(hwndTX, TX_TABLE_GETROWSANDCOLS,
        (WPARAM)wID, (LPARAM)(LPWORD)wRowsAndCols);
}

```

---

## TX\_TABLE\_GETROWSANDCOLS

This message returns the number of rows and columns for the specified table.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
-------------------------	---------------------------

<i>wParam</i>	Specifies a table identifier. When this parameter is set to zero the table with the current input position is used.
---------------	---

<i>lParam</i>	Points to an array of two WORD variables to receive the number of rows and columns. The first variable receives the number of rows, the second receives the number of columns.
---------------	--

### ***Return Value***

The return value is the table's identifier. This is the same value as specified or the identifier of the table with the current input position when *wParam* has been set to zero. The identifier is either a user-defined value set with the TX\_TABLE\_CHANGEID message or a unique identifier selected by Text Control.

The return value is zero if an error has occurred or if the current input position is not inside a table when *wParam* has been set to zero.

---

## TX\_TABLE\_GETTEXTOFCELL

This message retrieves the text of a table cell.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
-------------------------	---------------------------

<i>wParam</i>	Specifies a table identifier.
---------------	-------------------------------

<i>lParam</i>	Specifies the row and column number of the cell which text is to be retrieved. The row number is in the low-order word and the column number is in the high-order word.
---------------	---

**Return Value** The return value is a global data handle containing the text of the cell as a zero-terminated character string. If the application has finished using the buffer it must free it with the **GlobalFree** function.

## TX\_TABLE\_INSERT

This message inserts a new table into the text.

<u>Parameter</u>	<u>Description</u>						
<i>wParam</i>	Specifies the table's description. It can be any one of the following values:						
	<table border="0"> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>TF_AUTOTABLE</td> <td>The table is inserted with default values. The <i>lParam</i> parameter contains the number of rows and columns in the table.</td> </tr> <tr> <td>TF_USERTABLE</td> <td>The <i>lParam</i> parameter contains a complete table description.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	TF_AUTOTABLE	The table is inserted with default values. The <i>lParam</i> parameter contains the number of rows and columns in the table.	TF_USERTABLE	The <i>lParam</i> parameter contains a complete table description.
<u>Value</u>	<u>Meaning</u>						
TF_AUTOTABLE	The table is inserted with default values. The <i>lParam</i> parameter contains the number of rows and columns in the table.						
TF_USERTABLE	The <i>lParam</i> parameter contains a complete table description.						
<i>lParam</i>	When <i>wParam</i> is set to TF_AUTOTABLE this parameter must contain the number of a table's rows in its low-order word and the number of columns in its high-order word. Otherwise when <i>wParam</i> is set to TF_USERTABLE <i>lParam</i> must point to a buffer containing a complete table description. The data contained by the buffer must be structured in the following form:						

```
WORD        wTXVersion;
LONG        lTextPos;
WORD        wRows;
WORD        wColumns;
CELLDATA    tablecells[][];
```

These fields have the following meaning:

<u>Field</u>	<u>Description</u>
<i>wTXVersion</i>	Specifies the Text Control's current version number in the same format as that returned by the <i>TXGetVersion</i> function, e.g. 500. Set this parameter as a number and not as a <i>TXGetVersion</i> function call.

<i>lTextPos</i>	This parameter specifies the text position where the table is to be inserted. If <i>lTextPos</i> is -1 the table is inserted at the current input position.
<i>wRows</i>	Specifies how many rows the table has.
<i>wColumns</i>	Specifies how many columns the table has.
<i>tablecells[][]</i>	Is an array of CELLDATA data structures containing one structure for each of the table's cells. The array first describes the cells of the first row and continues with the cells of the subsequent rows. The CELLDATA data structure is described in the following comments section.

**Return Value** The return value is:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TR_ERR	An error has occurred or the table could not be inserted. Tables cannot be inserted inside existing tables or when a section of text has been selected.
TR_UNCHANGED	The new table has been inserted at the top or at the bottom of an existing table and has been combined with this table.
otherwise	A unique table identifier. This identifier can be used with the TX_TABLE_CHANGEID message to set a user-defined identifier.

**Comments** The CELLDATA structure has the following form:

```
typedef struct tagCELLDATA {
    LONG        lxPos;
    LONG        lxExt;
    CELLATTR    cellattr;
} CELLDATA;
```

The CELLDATA structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>lxPos</i>	Specifies the horizontal cell position.

<i>lxExt</i>	Specifies the horizontal cell extension.
<i>cellattr</i>	Specifies the cell's attributes. The CELLATTR data structure is described in Chapter 7 "Data Structures".

## **TX\_TABLE\_ISPOSSIBLE**

This message returns TRUE when the specified action is possible.

<b><u>Parameter</u></b>	<b><u>Description</u></b>								
<i>wParam</i>	Specifies the action which is to be performed. It can be any one of the following values:								
	<table> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>TF_TABLE_CANINSERT</td> <td>A table can be inserted at the current input position. This message returns FALSE when a section of text has been selected or the current input position is inside a table.</td> </tr> <tr> <td>TF_TABLE_CANDELETELINES</td> <td>The selected table lines can be deleted. This message returns FALSE when no table line is selected or if the current input position is outside a table.</td> </tr> <tr> <td>TF_TABLE_CANCHANGEATTR</td> <td>The attributes of the selected table lines can be altered. This message returns FALSE when the selection is not completely within a single table.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	TF_TABLE_CANINSERT	A table can be inserted at the current input position. This message returns FALSE when a section of text has been selected or the current input position is inside a table.	TF_TABLE_CANDELETELINES	The selected table lines can be deleted. This message returns FALSE when no table line is selected or if the current input position is outside a table.	TF_TABLE_CANCHANGEATTR	The attributes of the selected table lines can be altered. This message returns FALSE when the selection is not completely within a single table.
<b><u>Value</u></b>	<b><u>Meaning</u></b>								
TF_TABLE_CANINSERT	A table can be inserted at the current input position. This message returns FALSE when a section of text has been selected or the current input position is inside a table.								
TF_TABLE_CANDELETELINES	The selected table lines can be deleted. This message returns FALSE when no table line is selected or if the current input position is outside a table.								
TF_TABLE_CANCHANGEATTR	The attributes of the selected table lines can be altered. This message returns FALSE when the selection is not completely within a single table.								
<i>lParam</i>	Is not used.								

**Return Value** The return value is TRUE when the specified action can be performed. Otherwise it is FALSE.

---

## TX\_TABLE\_SETATTROFCELL

This message alters one or more attributes of one or more table cells.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a table identifier.
<i>lParam</i>	Points to a SETATTROFCELL data structure which is defined as follows:

```
typedef struct tagSETATTROFCELL {
    WORD        wStructSize;
    WORD        wRow;
    WORD        wColumn;
    LONG        lxPos;
    LONG        lxExt;
    CELLATTR    cellattr;
} SETATTROFCELL;
```

The SETATTROFCELL structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wStructSize</i>	Specifies the size, in bytes, of this data structure.
<i>wRow</i>	Specifies the cell's row. The first row has the number one. To change the attributes of a complete column set this member to zero.
<i>wColumn</i>	Specifies the cell's column. The first column has the number one. To change the attributes of a complete row set this member to zero.
<i>lxPos</i>	Specifies the cell's horizontal position.
<i>lxExt</i>	Specifies the cell's horizontal extension.
<i>cellattr</i>	Specifies the cell's border and color attributes. The CELLATTR data structure is described in chapter 7 "Data Structures". Each member of this structure can be set to

TF\_TABLEATTRDEFAULT to indicate that this attribute is not to be changed.

**Return Value** The return value is zero if an error has occurred. Otherwise it is non-zero.

---

## TX\_TABLE\_SETCELLATTR

This message changes the attributes of all selected table cells.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a global data handle containing the cells' attributes. The data contained by the buffer is structured in the same way as that described for the TX_GETCELLATTR message.
<i>lParam</i>	Is not used.

**Return Value** The return value is zero if the new attributes could not be set. Otherwise it is nonzero.

---

## TX\_TABLE\_SETCOLPOSITIONS

This message changes the column positions of selected table rows.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the number of columns described by the buffer pointed to by <i>lParam</i> .
<i>lParam</i>	Points to an array of COLPOS data structures. The array size is specified by <i>wParam</i> . For a description of the COLPOS structure see TX_GETCOLPOSITIONS.

**Return Value** The return value is zero if the new column positions could not be set. Otherwise it is nonzero.

## TX\_TABLE\_SETTEXTOFCELL

This message alters the text of a table cell.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a table identifier.
<i>lParam</i>	Points to a SETTEXTOFCELL data structure which is defined as follows:

```
typedef struct tagSETTEXTOFCELL {
    WORD        wRow;
    WORD        wColumn;
    LPCTSTR     lpText;
} SETTEXTOFCELL;
```

The SETTEXTOFCELL structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wRow</i>	Specifies the cell's row.
<i>wColumn</i>	Specifies the cell's column.
<i>lpText</i>	points to a character string that is the new text. The string must be zero-terminated. This member can be set to zero to delete the cell's text.

***Return Value*** The return value is zero if an error has occurred. Otherwise it is non-zero.

---

## TX\_UNDO

This message undoes the last Text Control operation.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

***Return Value*** The return value is zero if the undo operation fails. Otherwise it is nonzero.

**Comments** The TX\_CANUNDO message can be used to determine whether a Text Control operation can be undone. If this message is sent although there is no operation that can be undone the Text Control beeps.

---

## TX\_ZOOM

This message sets a new zooming factor for the Text Control. This factor is given as a percentage. A value of 100 means the original size.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Contains the new zooming factor in percent. The value must be between 10 and 400.
<i>lParam</i>	Is not used.

**Return Value** The return value is zero if the window cannot be zoomed or if the specified zooming factor has already been set. Otherwise it is nonzero.

**Comments** The Text Control is not updated. The **InvalidateRect** function must be used to update the appropriate portion of the parent windows' client area.

## 6. Notification Messages

The notification messages notify a Text Control's parent window of actions that occur within the control via the WM\_COMMAND message. In 16 and 32 bit programs the parameters of the WM\_COMMAND message have different meanings.

For 16 bit programs the WM\_COMMAND message has the following parameters:

```
id = wParam;
hwndCtl = (HWND)LOWORD(lParam);
wNotifyCode = HIWORD(lParam);
```

For 32 bit programs the WM\_COMMAND message has the following parameters:

```
id = LOWORD(wParam);
hwndCtl = (HWND)lParam;
wNotifyCode = HIWORD(wParam);
```

Normally the *id* parameter specifies the control item identifier. For some Text Control notifications the *id* parameter has a different meaning. In this case the description contains a parameters section.

---

### TN\_AUTOLINK

This code specifies that text will be inserted into the last control in a chain of linked windows. The parent window can avoid a text overflow at the end of the chain if it responds to this notification by an expansion of the chain. This notification is sent before the text is inserted.

---

### TN\_AUTOSCROLL

This code is sent if the cursor leaves the visible portion of a Text Control's client area whilst a text selection is being expanded with the mouse. This code is only sent if the cursor movement does not result in a caret movement. This happens if the cursor is moved outside the client area or if the cursor is moved over parts which are not covered with text below the last line. In all cases where the cursor movement results in a caret movement, the Text Control sends TN\_CARETOUTxxx notification messages.

**TN\_CARETOUT**

This code specifies that the caret has been moved to or within a Text Control that is completely out of the visible client area.

---

**TN\_CARETOUTBOTTOM**

This code specifies that the caret has been moved down out of the visible area of the Text Control.

---

**TN\_CARETOUTLEFT**

This code specifies that the caret has been moved to the left out of the visible area of the Text Control.

---

**TN\_CARETOUTRIGHT**

This code specifies that the caret has been moved to the right out of the visible area of the Text Control.

---

**TN\_CARETOUTTOP**

This code specifies that the caret has been moved up out of the visible area of the Text Control.

---

**TN\_CHANGED**

This code specifies that the user has taken an action which may have altered text or format attributes.

## TN\_CHARFORMATCHANGED

This code specifies that the formatting attributes of the selected characters have been changed. Chapter 4.1 lists all messages for manipulating character formatting attributes. This notification is also sent if font settings have been changed because the Text Control has adapted fonts to a new output device.

---

## TN\_DOUBLECLICKED

This code is sent after a Text Control has been double-clicked. The message is sent after the double clicked word has been selected.

---

## TN\_ERRCODE

This code informs the parent window that an error has occurred. Further information can be obtained by calling **GetErrorCode**.

---

## TN\_FIELD\_CHANGED

This code specifies that the text of a marked text field has been changed.

<u>Parameter</u>	<u>Description</u>
<i>id</i>	Specifies the field identifier of the changed marked text field.

---

## TN\_FIELD\_CLICKED

This code specifies that the user has clicked on a marked text field.

<u>Parameter</u>	<u>Description</u>
<i>id</i>	Specifies the field identifier of the clicked marked text field.

---

## TN\_FIELD\_CREATED

This code specifies that a new marked text field has been created without using the TX\_FIELD\_INSERT message. This can happen if a marked text field is copied with the clipboard.

<u>Parameter</u>	<u>Description</u>
<i>id</i>	Specifies the field identifier of the created marked text field.

---

## TN\_FIELD\_DBLCLICKED

This code specifies that the user has double-clicked on a marked text field.

<u>Parameter</u>	<u>Description</u>
<i>id</i>	Specifies the field identifier of the marked text field on which the user has double-clicked.

---

## TN\_FIELD\_DELETED

This code specifies that a marked text field has been completely deleted. This notification is not sent if a marked text field is deleted because a Text Control has been destroyed or if the text of a Text Control was completely exchanged with the TX\_SETHANDLE message.

<u>Parameter</u>	<u>Description</u>
<i>id</i>	Specifies the field identifier of the deleted marked text field.

---

## TN\_FIELD\_ENTERED

This code specifies that the current input position indicated by the caret has been moved to a position that belongs to a marked text field. This notification is only sent if the caret has been moved using the keyboard. If the caret has been moved with a mouse click the parent window receives a TN\_FIELD\_CLICKED notification message.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies the field identifier of the marked text field which has been entered.

---

## **TN\_FIELD\_LEFT**

This code specifies that the current input position indicated by the caret has been moved to a position that does not belong to the marked text field at the previous input position.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies the field identifier of the left marked text field.

---

## **TN\_FIELD\_LINKCLICKED**

This code specifies that the user has clicked on a marked text field that represents the source of a hypertext link. The field must be of the type FT\_EXTERNALLINK or FT\_INTERNALLINK. The TX\_FIELD\_GETTYPE message can be used to get the information where the link points to.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies the field identifier of the clicked marked text field.

---

## **TN\_FIELD\_SETCURSOR**

This code specifies that the cursor is being moved over a marked text field. The parent window can set the cursor when it receives this notification, in this case it must return TRUE. If the parent window returns FALSE the Text Control sets the cursor to the vertical arrow cursor.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies the field identifier of the marked text field over which the cursor has been moved.

---

## TN\_FORCEUPDATE

This code specifies that connected tool bars must update their contents. It must be passed on to status bars, button bars and rulers that show or change the contents of a Text Control. See appendix D, E and F for more information how to pass on messages to tool bars.

---

## TN\_HEXPAND

This code specifies that the Text Control has changed its window size horizontally in autoexpanding mode.

---

## TN\_HF\_ACTIVATED

This code specifies that a header or footer has been activated.

<b><u>Parameter</u></b>	<b><u>Description</u></b>										
<i>id</i>	Specifies which kind of header or footer has been activated. It can be one of the following values:										
	<table><thead><tr><th><b><u>Value:</u></b></th><th><b><u>Description:</u></b></th></tr></thead><tbody><tr><td>TF_HF_HEADER</td><td>A header has been activated.</td></tr><tr><td>TF_HF_1STHEADER</td><td>The special header for the first page has been activated.</td></tr><tr><td>TF_HF_FOOTER</td><td>A footer has been activated.</td></tr><tr><td>TF_HF_1STFOOTER</td><td>The special footer for the first page has been activated.</td></tr></tbody></table>	<b><u>Value:</u></b>	<b><u>Description:</u></b>	TF_HF_HEADER	A header has been activated.	TF_HF_1STHEADER	The special header for the first page has been activated.	TF_HF_FOOTER	A footer has been activated.	TF_HF_1STFOOTER	The special footer for the first page has been activated.
<b><u>Value:</u></b>	<b><u>Description:</u></b>										
TF_HF_HEADER	A header has been activated.										
TF_HF_1STHEADER	The special header for the first page has been activated.										
TF_HF_FOOTER	A footer has been activated.										
TF_HF_1STFOOTER	The special footer for the first page has been activated.										

---

## TN\_HF\_DEACTIVATED

This code specifies that a header or footer has been deactivated.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies which kind of header or footer has been deactivated.

See the TN\_HF\_ACTIVATED notification message for possible values.

---

## **TN\_HMOVED**

This code specifies that the Text Control's window has been moved horizontally relative to its parent window.

---

## **TN\_HSCROLL**

This code is sent when the horizontal scroll position has been changed. The TX\_GETSCROLLPOS message can be used to get the new position.

---

## **TN\_IMAGECLICKED**

This code is sent if an Image-Control window has been clicked. The TX\_GETIMAGE message can be used to identify this image.

---

## **TN\_KEYSTATECHANGED**

This code is sent after the character insertion mode or when the state of the NUMLOCK or CAPSLOCK key has been changed.

---

## **TN\_KILLFOCUS**

This code specifies that the Text Control has lost the input focus.

---

## **TN\_OBJ\_CLICKED**

This code is sent after the user has clicked on an object contained in the Text Control. An Object can be any child window inserted or created with the TX\_OBJ\_EMBED or the TX\_CREATEIMAGE message.

---

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies an identifier of the object that has been clicked on. This is the same identifier returned from the TX_OBJ_EMBED message.

---

## TN\_OBJ\_CREATED

This code specifies that a new embedded object has been created without using the TX\_OBJ\_EMBED or TX\_CREATEIMAGE messages. This can happen if an object is copied using the clipboard.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies the identifier of an object that has been created.

---

## TN\_OBJ\_DBLCLICKED

This code is sent after the user has double-clicked on an object contained in the Text Control. An object can be any child window inserted, or created with the TX\_OBJ\_EMBED or TX\_CREATEIMAGE messages.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies the identifier of an object that has been double-clicked on. This is the same identifier returned from the TX_OBJ_EMBED message.

---

## TN\_OBJ\_DELETED

This code is sent after an embedded object has been deleted. This code is also called if the application has deleted the object using the TX\_OBJ\_DELETE message.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the identifier of an object that has been deleted. This is the same identifier returned by the

TX\_OBJ\_EMBED message. The identifier is thereafter invalid for subsequent message calls.

---

## TN\_OBJ\_MOVED

This code is sent after an embedded object has been moved via the integrated mouse interface.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies an identifier of the object that has been moved. This is the same identifier returned by the TX_OBJ_EMBED message.

---

## TN\_OBJ\_SIZED

This code is sent after an embedded object has been sized via the integrated mouse interface.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies an identifier of the object that has been sized. This is the same identifier returned from the TX_OBJ_EMBED message.

---

## TN\_PAGEFORMATCHANGED

This code specifies that the page format settings have been changed.

## TN\_PGCHANGED

This code indicates that the character input position has been moved to another paragraph.

---

## TN\_PGFORMATCHANGED

This code specifies that the paragraph attributes of the selected paragraphs have been changed. Chapter 4.2 lists all messages for manipulating paragraph attributes.

---

## TN\_POSCHANGED

This code specifies that the current character input position has been changed.

---

## TN\_SETFOCUS

This code specifies that the Text Control has obtained the input focus.

---

## TN\_TABLE\_CREATED

This code is sent after a new table has been created as a result of a text insertion via the clipboard. It is not sent when the table is inserted with the TX\_TABLE\_INSERT message or when a previously saved document is reloaded.

**Parameter****Description***id*

Specifies an identifier of the table. It is either a user-defined identifier set with the TX\_TABLE\_CHANGEID message or an identifier selected through Text Control.

*return value*

The return value specifies a new user-defined identifier for the table. It must be a value between 10 and 0x7FFF. It can be set to zero when the identifier should not be changed.

---

## TN\_TABLE\_DELETED

This code is sent after a table has been deleted.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>id</i>	Specifies an identifier of the table. It is either a user-defined identifier set with the TX_TABLE_CHANGEID message or an identifier selected through Text Control.

---

## **TN\_VEXPAND**

This code specifies that the Text Control has changed its window size vertically in autoexpanding mode.

---

## **TN\_VSCROLL**

This code is sent when the vertical scroll position has been changed. The TX\_GETSCROLLPOS message can be used to get the new position.

---

## **TN\_ZOOMED**

This code is sent after the Text Control has been zoomed.

---

## 7. Data Structures

---

### CELLATTR

The CELLATTR structure defines the attributes of a table cell. All geometric values are drawn inside the cell's extension given by the CELLPOSITION structure. All values are in twentieths of a point.

```
typedef struct tagCELLATTR {
    short leftBorder;
    short topBorder;
    short rightBorder;
    short bottomBorder;
    short leftTextDist;
    short topTextDist;
    short rightTextDist;
    short bottomTextDist;
    DWORD dwBkColor;
    BYTE reserved[8];
} CELLATTR;
```

The CELLATTR structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>leftBorder</i>	Specifies the width of the cell's left border.
<i>topBorder</i>	Specifies the height of the cell's top border.
<i>rightBorder</i>	Specifies the width of the cell's right border.
<i>bottomBorder</i>	Specifies the height of the cell's bottom border.
<i>leftTextDist</i>	Specifies the distance between the cell's left border and the text.
<i>topTextDist</i>	Specifies the distance between the cell's top border and the text.
<i>rightTextDist</i>	Specifies the distance between the cell's right border and the text.

<i>bottomTextDist</i>	Specifies the distance between the cell's bottom border and the text..
<i>dwBkColor</i>	Specifies the cell's background color. This value must be an RGB value returned by the RGB macro, 0x4000 0000 if the system color for the window background or 0x5000 0000 if the Text Control's background color is to be used.
<i>reserved</i> [8]	Reserved field. 8 bytes which are reserved for future use. They must be set to zero.

---

## CELLPOSITION

The CELLPOSITION structure defines the positions of a table cell. It is always used in combination with the TXTABLE structure and defines the general data of a table.

```
typedef struct tagCELLPOSITION {
    WORD    wCellStart;
    WORD    wCellStop;
    LONG    lxPos;
    LONG    lxExt;
    WORD    wAttrRefNum;
    BYTE    reserved[4];
} CELLPOSITION;
```

The CELLPOSITION structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wCellStart</i>	Specifies the one-based character position in the <i>szText</i> [ ] array of the cell's first character.
<i>wCellStop</i>	Specifies the one-based character position in the <i>szText</i> [ ] array of the cell's last character. This must be a paragraph end control character 0AH.
<i>lxPos</i>	Specifies, in twentieths of a point, the horizontal cell position.
<i>lxExt</i>	Specifies, in twentieths of a point, the horizontal cell extension.

<i>wAttrRefNum</i>	Specifies the reference number of the cell's attributes. The reference number is the position of the CELLATTR structure in the <i>caCellAttr</i> array beginning with the number one.
<i>reserved[4]</i>	Reserved field. 4 bytes which are reserved for future use. They must be set to zero.

---

## FIELDADATA

The FIELDADATA structure describes a field data entry in the Text Control's text format.

```
typedef struct tagFIELDADATA {
    WORD    wField;
    BYTE    nFieldType;
    BYTE    reserved[3];
    DWORD   dwData;
    DWORD   dwDataSize;
} FIELDADATA;
```

The FIELDADATA structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>						
<i>wField</i>	Specifies the identifier of the field to which this entry belongs.						
<i>nFieldType</i>	Specifies the type of the field to which this entry belongs. It can be anyone of the following values:						
	<table> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>FT_STANDARD</td> <td>The data this entry contains, has been set by the programmer or user with the TX_FIELD_SETDATA message.</td> </tr> <tr> <td>FT_EXTERNALLINK</td> <td>The data this entry contains, is the destination of a hypertext link to a position outside of the document.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	FT_STANDARD	The data this entry contains, has been set by the programmer or user with the TX_FIELD_SETDATA message.	FT_EXTERNALLINK	The data this entry contains, is the destination of a hypertext link to a position outside of the document.
<b><u>Value</u></b>	<b><u>Meaning</u></b>						
FT_STANDARD	The data this entry contains, has been set by the programmer or user with the TX_FIELD_SETDATA message.						
FT_EXTERNALLINK	The data this entry contains, is the destination of a hypertext link to a position outside of the document.						

	FT_INTERNALLINK	The data this entry contains, is the destination of a hypertext link to a position in this document. It is the name of a marked text field of the type FT_LINKTARGET.
	FT_LINKTARGET	The data this entry contains, is the name of the field. The field, this entry belongs to, is the target position of a hypertext link.
	FT_HIGHLIGHT	The data this entry contains, is the color of the highlight as an RGB value.
	FT_TOPIC	The data this entry contains, is the number of the topic.
<i>reserved[3]</i>		3 bytes for future use that must be set to zero.
<i>dwData</i>		A 4-byte value which is used when the amount of the data to store is 1 to 4 bytes. When this member is used the <i>dwDataSize</i> member must be zero. This parameter is used for FT_STANDARD, FT_HIGHLIGHT and FT_TOPIC entries.
<i>dwDataSize</i>		Specifies the size of the data when the amount is larger than 4 bytes. The data itself follows immediately after this structure. When this member is used the <i>dwData</i> member must be zero. This parameter is used for FT_STANDARD, FT_EXTERNALLINK, FT_INTERNALLINK and FT_LINKTARGET entries.

---

## FIXEDOBJECT

The FIXEDOBJECT structure is the header structure of a fixed positioned object in the *cFixedObjects[ ]* array. The data of the object follows this structure.

```
typedef struct tagFIXEDOBJECT {
    LONG reserved1;
    WORD wID;
    WORD wEmbedMode;
    short nxPosition;
    LONG lyPosition;
}
```

```

    short xSize;
    short ySize;
    WORD wxScale;
    WORD wFlags;
    DWORD dwData;
    WORD wyScale;
    short leftDist;
    short topDist;
    short rightDist;
    short bottomDist;
} FIXEDOBJECT

```

The FIXEDOBJECT structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>reserved1</i>	Reserved field used internally by TX.
<i>wID</i>	Specifies a child window identifier if the object is an externally created window. It contains zero if the object is an image.
<i>wEmbedMode</i>	Specifies the object's embedding mode. See the TX_OBJ_EMBED message for more information.
<i>nxPosition</i>	Specifies the object's horizontal position in twentieths of a point.
<i>lyPosition</i>	Specifies the object's vertical position in twentieths of a point.
<i>xSize</i>	Specifies the object's horizontal size in twentieths of a point.
<i>ySize</i>	Specifies the object's vertical size in twentieths of a point.
<i>wxScale</i>	Specifies the object's horizontal scaling factor as a percentage. It must be between 10 and 250.
<i>wFlags</i>	Specifies the object's flags. It can be a combination of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
ICF_NOMOVE	The object cannot be moved by the internal mouse interface.
ICF_NOSIZE	The object cannot be sized by the internal mouse interface.

	ICF_GRAYED	The object is an image and displayed in fast mode.
	ICF_SAVEASDATA	The object is an image which is saved by the Text Control using its data instead of its filename.
<i>dwData</i>		Specifies the size, in bytes, of the object's data that follows this structure. If the object is an image, this data is stored by the Image-Control programming tool. The format of this memory block is described in the IC Image-Control reference. Otherwise if the object is an externally created window the data depends on the load and save functionality of this window. See appendix H for more information about the load and save mechanism for external windows.
<i>wyScale</i>		Specifies the object's vertical scaling factor as a percentage. It must be between 10 and 250.
<i>leftDist</i>		Specifies the distance between the object's left side and the text.
<i>topDist</i>		Specifies the distance between the object's top side and the text.
<i>rightDist</i>		Specifies the distance between the object's right side and the text.
<i>bottomDist</i>		Specifies the distance between the object's bottom side and the text.

---

## FONTBLOCK

The FONTBLOCK structure defines part of a Text Control's text the contents of which are displayed with the same font, baseline align and color.

```
typedef struct tagFONTBLOCK {
    WORD    wFntNum;
    WORD    wBlkStart;
    WORD    wBlkLength;
    short   nBlAlign;
    DWORD   dwColor;
    WORD    wField;
```

```

        BYTE  fFieldFlags;
        BYTE  nFieldType;
        DWORD dwBkColor;
    } FONTBLOCK;

```

The FONTBLOCK structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wFntNum</i>	Specifies the reference number of the font used in this font block. The reference number is the position of the TXLOGFONT structure in the <i>lfLogFonts[ ]</i> array, beginning with the number one.
<i>wBlkStart</i>	Specifies the starting position of the font block in the text. The first character has the number one.
<i>wBlkLength</i>	Specifies the length of the font block (in bytes).
<i>nBlAlign</i>	Specifies the distance of the baseline from its normal position in twentieths of a point. The value is positive for superscript and negative for subscript and is limited to 48 pts.
<i>dwColor</i>	Specifies the text color of the font block. This value must be an RGB value returned by the RGB macro or 0x4000 0000 if the system color for window text is to be used.
<i>wField</i>	Specifies an identifier for a marked text field. The identifier must be unique.
<i>fFieldFlags</i>	Specifies the attributes of a marked text field. It can be a combination of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
TF_ENABLEDBLCLICKS	Inside the marked text field normal double-click processing is performed.
TF_EXTEDITMODE	The marked text field can be edited with a second input position at the beginning and the end of the field.

**TF\_HASUSERDATA** The marked text field has a user-defined data entry. This data can be accessed with the **TX\_FIELD\_GETDATA** and **TX\_FIELD\_SETDATA** messages. When this bit is set, the marked text field must have an entry in the *FieldData[ ]* array of the *cTextBlocks[ ]* data block.

**TF\_SHOWCURFIELDGRAY** The marked text field is displayed with a gray background when it contains the current input position.

**TF\_UNCHANGEABLE** The text of a marked text field cannot be changed.

**TF\_UNDELETEABLE** The marked text field cannot be deleted.

**TF\_USEFIELDPCARET** Inside the field the caret for marked text fields is used.

*nFieldType* Specifies the type of the marked text field. It can be anyone of the following values:

**Value**

**Meaning**

**FT\_STANDARD** Specifies a standard marked text field without special features.

**FT\_EXTERNALLINK** The marked text field is the source of a hypertext link to a location outside of the document.

**FT\_INTERNALLINK** The marked text field is the source of a hypertext link to another location in this document.

**FT\_PAGENUMBER** The marked text field displays page numbers.

	FT_LINKTARGET	The marked text field is a position in the document that is the target of a hypertext link.
	FT_HIGHLIGHT	The marked text field identifies a piece of text that can be highlighted.
	FT_TOPIC	The marked text field identifies the text position of a new topic.
<i>dwBkColor</i>		Specifies the text background color of the font block. This value must be an RGB value returned by the RGB macro, 0x4000 0000 if the system color for the window background or 0x5000 0000 if the Text Control's background color is to be used.

---

## IMAGEDATA

The IMAGEDATA structure is the header structure of an image in the *ImageData[]* array. The data of the image follows this structure.

```
typedef struct tagIMAGEDATA {
    LONG reserved1;
    WORD wFlags;
    WORD wxScale;
    WORD wyScale;
    short xSize;
    short ySize;
    WORD wScale;
    DWORD dwData;
} IMAGEDATA
```

The IMAGEDATA structure has the following fields:

<u>Field</u>	<u>Description</u>				
<i>reserved1</i>	Reserved field used internally by TX.				
<i>wFlags</i>	Specifies the alignment of the image and the display mode. It can be a combination of the following values:				
	<table> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>0x0001</td> <td>The image is updated in fast display mode.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	0x0001	The image is updated in fast display mode.
<u>Value</u>	<u>Meaning</u>				
0x0001	The image is updated in fast display mode.				

---

<i>wxScale</i>	Specifies the image's horizontal scaling factor as a percentage. It must be a value between 10 and 250.
<i>wyScale</i>	Specifies the image's vertical scaling factor as a percentage. It must be a value between 10 and 250.
<i>xSize</i>	Specifies the horizontal image size in twentieths of a point.
<i>ySize</i>	Specifies the vertical image size in twentieths of a point.
<i>wScale</i>	Specifies the image's scaling factor for both directions as a percentage. It must be between 10 and 250. If this value is zero the members <i>wxScale</i> and <i>wyScale</i> are used for scaling.
<i>dwData</i>	Specifies the size of the following image data (in bytes). This data is stored by the Image-Control programming tool. The format of this memory block is described in the IC Image-Control reference.

---

## PARAGRAPH

The PARAGRAPH structure defines the attributes of a paragraph.

```
typedef struct tagPARAGRAPH {
    WORD        wAlign;
    short       nLeftInd;
    short       nRightInd;
    short       nFirstInd;
    short       nTopInd;
    short       nBottomInd;
    WORD        wLSpace;
    WORD        wFrameStyles;
    WORD        wFrameWidth;
    WORD        wFrameDist;
    TABSCT      TabList[NTABS];
} PARAGRAPH;
```

The PARAGRAPH structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wAlign</i>	Specifies the text alignment of the paragraph in the upper two bytes. It can be any one of the following values:

<u>Value</u>	<u>Meaning</u>
0x0000	The text is left aligned
0x4000	The text is right aligned
0x8000	The text is centered
0xC000	The text is justified

The lower bytes contain a bitfield specifying special paragraph formats. This can be a combination of the following values:

<u>Value</u>	<u>Meaning</u>
TF_PAGEBREAKNOTALLOWED	A page break is not allowed within a paragraph.
TF_EXACTLINESPACING	The specified linespacing is used independent whether the paragraph contains characters or images with a larger height. This value is only used when <i>wLSpace</i> contains an absolute linespacing.

<i>nLeftInd</i>	Specifies a left indent.
<i>nRightInd</i>	Specifies a right indent.
<i>nFirstInd</i>	Specifies an additional left indent for the first line. This value can be negative.
<i>nTopInd</i>	Specifies an additional top indent for the first line of the paragraph.
<i>nBottomInd</i>	Specifies an additional bottom indent for the last line of the paragraph. All indent values are given in terms of a twentieth of a point.
<i>wLSpace</i>	Specifies the line spacing value of the paragraph. If the highest bit is 1, the lower 15 bits specify an absolute value in twentieths of a point. Otherwise the line spacing is given as a percentage of the height of the fonts which are currently being used.

*wFrameStyles* Specifies the appearance and styles of a paragraph frame. It can be a combination of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
BF_LEFTLINE	The frame has a left part.
BF_RIGHTLINE	The frame has a right part.
BF_TOPLINE	The frame has a top part.
BF_BOTTOMLINE	The frame has a bottom part.
BF_TABLINES	Vertical lines are drawn at each tabulator position.
BF_SINGLE	The frame is a single line.
BF_DOUBLE	The frame is a double line.
BF_BOXCONNECT	The frame is connected to its neighbours.

*wFrameWidth* Specifies the width of the frame lines in twentieths of a point.

*wFrameDist* Specifies the distance of the frame from the text in twentieths of a point.

*TabList*[NTABS] Is an array of TABSCT structures specifying type and position of tabulator positions.

---

## PGBLOCK

The PGBLOCK structure defines a text part of the Text Control which identifies a paragraph.

```
typedef struct tagPGBLOCK {
    WORD  wPgNum;
    WORD  wPgStart;
    WORD  wPgStop;
} PGBLOCK;
```

The PGBLOCK structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wPgNum</i>	Specifies the reference number for the paragraph used in this paragraph block. The reference number is the position of the PARAGRAPH structure in the <i>pgParagraphs[ ]</i> array, beginning with number one.
<i>wPgStart</i>	Specifies the starting position of the paragraph block in the text. The first character has the number one.
<i>wPgStop</i>	Specifies the end position of the paragraph block in the text. This is the position of the paragraph end character 0x0A or the terminating zero in the last paragraph.

---

## TABSCT

The TABSCT structure defines the attributes of a tab stop.

```
typedef struct tagTABSCT {
    BYTE  nTabFlag;
    WORD  wTabPos;
} TABSCT;
```

The TABSCT structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>										
<i>nTabFlag</i>	Specifies the type of the tabstop. It can be any one of the following values:										
	<table border="1"> <thead> <tr> <th><b><u>Value</u></b></th> <th><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td>LEFTTAB</td> <td>The tab position is at the left side of text.</td> </tr> <tr> <td>RIGHTTAB</td> <td>The tab position is at the right side of text.</td> </tr> <tr> <td>CENTERTAB</td> <td>The text is centered on the tab position.</td> </tr> <tr> <td>DECIMALTAB</td> <td>The decimal sign installed in “win.ini” is located at the tab position.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	LEFTTAB	The tab position is at the left side of text.	RIGHTTAB	The tab position is at the right side of text.	CENTERTAB	The text is centered on the tab position.	DECIMALTAB	The decimal sign installed in “win.ini” is located at the tab position.
<b><u>Value</u></b>	<b><u>Meaning</u></b>										
LEFTTAB	The tab position is at the left side of text.										
RIGHTTAB	The tab position is at the right side of text.										
CENTERTAB	The text is centered on the tab position.										
DECIMALTAB	The decimal sign installed in “win.ini” is located at the tab position.										

*wTabPos* Specifies the x-coordinate of the tab position in twentieths of a point.

---

## TXFILTERIO

The TXFILTERIO structure exchanges data with a text filter. It can be used with the TX\_DATAIN and the TX\_DATAOUT message. The meaning and usage of the structure members depend on whether the operation is a loading or a saving process. The user can set on function entry values which the filter can use for saving or as default values for loading. On function exit the filter sets values in a loading process which the user can then use for further processing.

```
typedef struct tagTXFILTERIO {
    WORD        wTXFileFormat;
    short       nMode;
    LPCTSTR     lpAbsPath;
    LPCTSTR     lpBasePath;
    DWORD       dwDocWidth;
    DWORD       dwDocHeight;
    short       nDocMarginLeft;
    short       nDocMarginTop;
    short       nDocMarginRight;
    short       nDocMarginBottom;
    HGLOBAL     hDocTitle;
    COLORREF    crDocBkGnd;
    LONG        lDocTextPos;
    COLORREF    crText;
    COLORREF    crTextBkGnd;
    COLORREF    crLink;
    WORD        wFontSize;
    TCHAR       lfDefFont[LF_FACESIZE];
    TCHAR       lfMonoFont[LF_FACESIZE];
    DWORD       dwUsageFlags;
    DWORD       dwOutDataSize;
    HGLOBAL     hOutData;

    LPCTSTR     lpDocPath;

    BYTE        reserved[252];
} TXFILTERIO;
```

The TXFILTERIO structure members have the following meanings:

---

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wTXFileFormat</i>	Is used internally by Text Control. It must be set to zero.
<i>nMode</i>	Is used internally by Text Control. It must be set to zero.
<i>lpAbsPath</i>	Points to a zero-terminated character string that is used to search for resources like images or destinations of hypertext links. When a document is loaded, the filter uses this path to locate a resource. It is only used for resources which are specified through an absolute location. In this case the absolute resource location is completely replaced through the path specified through this member. When a document is saved, this member is not used. This member can be zero, when it is not used.
<i>lpBasePath</i>	Points to a zero-terminated character string that is used to search for resources given through a relative location. When a document is loaded, the filter adds this path to the relative location of a resource. When a document is saved this member can only be a file path. The filter saves all files with a location relative to this path. This member can be zero, when it is not used.
<i>dwDocWidth</i>	Specifies the document's width in twentieths of a point including margins. When a document is loaded and this member is non-zero, it is used to convert relative width values, for example in percent, to absolute width values. The filter fills in a value when contained in the document or zero when the filter does not find a width. When a document is saved, the filter stores this value as part of the document when it is non-zero.
<i>dwDocHeight</i>	Specifies the document's height in twentieths of a point including margins. It is used in the same manner as <i>dwDocWidth</i> .
<i>nDocMarginxxx</i>	Specifies the document's margins in twentieths of a point. When a document is loaded, the filter fills in these values. A value is set to -1 if the document does not contain it. When a document is saved, the filter stores these values in the document except a value is set to -1.

---

<i>hDocTitle</i>	Specifies a document title. When a document is loaded, the filter allocates a global memory buffer and copies the document title to this buffer as a zero-terminated string. The user must free the buffer with the <b>GlobalFree</b> function after using. When a document is saved, the user must allocate this buffer when a title should be saved in the document and free it after calling the save operation. This structure member is only handled when the <i>dwUsageFlags</i> member contains the FIO_DOCTITLE setting.
<i>crDocBkGnd</i>	Specifies a document background color. When a document is loaded, the filter fills this member with a color value or with UNDEF_COLOR if it could not find a value. When a document is saved, the filter stores this value in the document except when it is set to UNDEF_COLOR.
<i>lDocTextPos</i>	Specifies a text position in the document to where the document's view should be scrolled. This value is provided by the filter after a document is loaded.
<i>crText</i>	Specifies a default text color. This member is only used as an initialization value before a document is loaded. Its usage depends on the settings of the <i>dwUsageFlags</i> member.
<i>crTextBkGnd</i>	Specifies a default text background color. This member is only used as an initialization value before a document is loaded. Its usage depends on the settings of the <i>dwUsageFlags</i> member.
<i>crLink</i>	Specifies a default text color for pieces of text that function as hypertext links. This member is only used as an initialization value before a document is loaded. Its usage depends on the settings of the <i>dwUsageFlags</i> member.
<i>wFontSize</i>	Specifies a base font size which is used to convert relative to absolute size values. This member is only used as an initialization value before a document is loaded.
<i>lfDefFont</i>	Specifies a default proportional font. This member is only used as an initialization value before a document is loaded. It can be an empty string.
<i>lfMonoFont</i>	Specifies a default mono-spaced font. This member is only used as an initialization value before a document is loaded. It can be an empty string.

*dwUsageFlags* Is a bit mask specifying the handling of some members. It can be a combination of any of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
FIO_TEXTCOLOR	The <i>crText</i> color is only used when a text color is not specified in the document.
FIO_FIXTEXTCOLOR	The <i>crText</i> color overwrites text colors specified in the document.
FIO_TEXTBKCOLOR	The <i>crTextBkGnd</i> color is only used when a text background color is not specified in the document.
FIO_FIXTEXTBKCOLOR	The <i>crTextBkGnd</i> color overwrites the text background colors specified in the document.
FIO_LINKCOLOR	The <i>crLink</i> color is only used when a text color for hypertext links is not specified in the document.
FIO_FIXLINKCOLOR	The <i>crLink</i> color overwrites text colors for hypertext links specified in the document.
FIO_LOADIMAGES	Images contained in the document are loaded.
FIO_UNDERLINELINKS	Text parts that identify hypertext links are underlined. This member is only used as an initialization value before a document is loaded.
FIO_DOCTITLE	The filter fills or saves the <i>hDocTitle</i> member.
FIO_ENABLELINKS	The filter converts source and target fields of hypertext links to appropriate marked text fields.
FIO_ENABLEHIGHLIGHTS	RTF only. The filter converts all "\cbN" keywords into marked text fields of the type FT_HIGHLIGHT.

---

	<b>FIO_ENABLETOPICS</b>	RTF only. The filter converts all '\sect' keywords into marked text fields of the type FT_TOPIC.
<i>dwOutDataSize</i>		Is used internally by Text Control.
<i>hOutData</i>		Is used internally by Text Control.
<i>lpDocPath</i>		Points to a zero-terminated character string specifying the file path of the loaded or saved document. When a document is loaded, the filter uses this path to search for resources.
<i>reserved[252]</i>		Reserved field. 252 bytes which are reserved for future use. They must be set to zero.

---

## TXLOGFONT

The TXLOGFONT structure defines the attributes of a font.

```
typedef struct tagTXLOGFONT {
    short lfHeight;
    short lfWidth;
    short lfEscapement;
    short lfOrientation;
    short lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    BYTE lfFaceName [LF_FACESIZE];
} TXLOGFONT;
```

The members of the TXLOGFONT structure have the same meanings as the members of the LOGFONT structure.

---

## TXLOGFONTEX

The TXLOGFONTEX structure contains additional font information.

```
typedef struct tagTXLOGFONTEX {
```

```

        BYTE    Reserved[16];
        WCHAR   lfUnicodeFaceName[LF_FACESIZE];
    } TXLOGFONTEX;

```

The TXLOGFONTEX structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>Reserved[16]</i>	A 16-byte array for future use. These values must be zero.
<i>lfUnicodeFaceName</i>	Specifies the name of the font formatted as a Unicode string.

---

## TXOBJECT

The TXOBJECT structure defines the attributes of an object integrated in a Text Control. This structure is used with the TX\_OBJ\_EMBED message.

```

typedef struct tagTXOBJECT {
    WORD        wTXVersion;
    WORD        wEmbedMode;
    LONG        lTextPos;
    int         xPosition;
    LONG        lyPosition;
    POINT       ptSize;
    WORD        wxScale;
    WORD        wyScale;
    RECT        Distances;
    WORD        wFlags;
    LPCTSTR     lpFileName;
    int         nBufLength;
    int         nFilterIndex;
} TXOBJECT;

```

The TXOBJECT structure members have the following meanings:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wTXVersion</i>	Specifies the Text Control's current version number in the same format returned by the <i>TXGetVersion</i> function, e.g. 420.
<i>wEmbedMode</i>	Specifies the embedding mode. It can be one of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
EOM_INSERT	The object is handled like a single character in the text.

	EOM_DISPLACELINE	The text flow stops at the top border of the object and continues at the bottom border. Empty areas on the left or right side of the object are not filled.
	EOM_DISPLACEWORD	Same as EOM_DISPLACELINE but empty areas on the left or right side of the object are filled with text so that a line's text is interrupted by the object.
<i>lTextPos</i>		This parameter is only used if <i>wEmbedMode</i> is set to EOM_INSERT. It specifies the text position where the object is to be inserted. If <i>lTextPos</i> is -1 the object is inserted at the current input position.
<i>xPosition</i>		Specifies the object's horizontal position in twentieths of a point. If <i>wEmbedMode</i> is set to EOM_INSERT this parameter is ignored.
<i>lyPosition</i>		Specifies the object's vertical position in twentieths of a point. If <i>wEmbedMode</i> is set to EOM_INSERT this parameter is ignored.
<i>ptSize</i>		Is not used for this message and should be set to zero.
<i>wxScale</i>		Specifies a horizontal scaling factor as a percentage. It must be a value between 10 and 250.
<i>wyScale</i>		Specifies a vertical scaling factor as a percentage. It must be a value between 10 and 250.
<i>Distances</i>		Specifies the distances between the object and the text. If <i>wEmbedMode</i> is set to EOM_INSERT this parameter is ignored.
<i>wFlags</i>		Specifies a combination of the following values:
	<b><u>Value</u></b>	<b><u>Meaning</u></b>
	ICF_NOMOVE	The object cannot be moved by the internal mouse interface.
	ICF_NOSIZE	The object cannot be sized by the internal mouse interface.

If the object is an image these values can additionally be combined with the following values:

<u>Value</u>	<u>Meaning</u>
ICF_GRAYED	The image is displayed in fast mode.
ICF_SAVEASDATA	The Text Control saves the image using its data instead of its filename.
ICF_BKGNDIMAGE	Inserts an image that can serve as a background for other sibling transparent controls.
<i>lpFileName</i>	Points to a buffer of length <i>nBufLength</i> specifying the full DOS path name of a file that contains an image. This parameter can be zero if <i>wParam</i> specifies an externally created window.
<i>nBufLength</i>	Specifies the length of the buffer <i>lpFileName</i> points to (Unicode: in characters, otherwise: in bytes).
<i>nFilterIndex</i>	Specifies an image filter as an index into the buffer returned by the TX_GETIMAGEFILTERS message. The first pair of strings has an index value of 1. If the buffer returned by TX_GETIMAGEFILTERS is used to initialize the <i>lpstrFilter</i> member of an <b>OPENFILENAME</b> structure, another member of that structure, <i>nFilterIndex</i> , can be used to initialize this parameter. See the Windows SDK for more information about the <b>OPENFILENAME</b> structure. If <i>nFilterIndex</i> is set to 0, the Text Control automatically tries to select a filter. This parameter can be zero if <i>wParam</i> specifies an externally created window.

---

## TXTABLE

The TXTABLE structure defines a table. It is always used in combination with CELLPOSITION structures defining the table cells' positions. One CELLPOSITION structure belongs to each cell, which means that *wRows\*wColumns* CELLPOSITION structures must follow each TXTABLE structure.

```
typedef struct tagTXTABLE {
```

```

        WORD   wRows;
        WORD   wColumns;
        WORD   wUserID;
        BYTE   reserved[6];
    } TXTABLE;

```

The TXTABLE structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wRows</i>	Specifies how many rows a table has.
<i>wColumns</i>	Specifies how many columns a table has.
<i>wUserID</i>	Specifies a user-defined identifier set with the TX_TABLE_CHANGEID message.
<i>reserved[6]</i>	Reserved field. 6 bytes which are reserved for future use. They must be set to zero.

---

## TXTEXTDATAEX

The TXTEXTDATAEX structure contains additional font information.

```

typedef struct tagTXTEXTDATAEX {
    BYTE   Reserved[16];
    LONG   CodePage;
} TXTEXTDATAEX;

```

The TXTEXTDATAEX structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>Reserved[16]</i>	A 16-byte array for future use. These values must be zero.
<i>CodePage</i>	Specifies the code page that belongs to the text of this text block when the text is saved in ANSI format.

## 8. *Obsolete Messages and Functions*

The following is a list of obsolete functions and messages. These functions and messages are provided for compatibility with earlier versions of Text Control. Newly developed applications should use the appropriate newer messages.

<u>Function</u>	<u>Replacement</u>
GetErrorCode	TXGetErrorCode
<u>Message</u>	<u>Replacement</u>
TX_ADJUSTCLIPBOARD	This message is no longer necessary. Therefore there is no appropriate newer message.
TX_EXPORTTEXT	TX_DATAOUT
TX_GETASCIITEXT	TX_DATAOUT
TX_GETASCIITEXTSIZE	-
TX_GETHANDLE	TX_DATAOUT
TX_GETIMAGEFORMAT	TX_OBJ_GETATTR
TX_IMPORTTEXTBUFFER	TX_DATAIN
TX_IMPORTTEXTFILE	TX_DATAIN
TX_SETHANDLE	TX_DATAIN
TX_SETIMAGEFORMAT	TX_OBJ_SETATTR

---

### GetErrorCode

#### *Syntax*

**LONG** GetErrorCode(*void*)

This function returns an internal error code, and can be called if the parent window has received a TN\_ERRCODE notification.

**Return Value** The return value contains an error number in the low-order word and a module number in the high-order word. The module number is 1 for the programming tool described in this manual but it can also be the number of other modules the Text Control uses for special purposes.

The error numbers belonging to the Text Control module are described in the error code table in appendix B. For a description of error codes that belong to other module numbers see the corresponding reference manuals of these modules.

---

## TX\_ADJUSTCLIPBOARD

This message adjusts the font information of the internal clipboard format when the user changes device-mode settings. It must be sent to the Text Control every time the application receives a WM\_DEVMODECHANGE message or a WM\_WININICHANGE message with the lParam parameter set to section “windows”.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** If the application contains more than one Text Control window, this message is sent to only one of them.

The application should send a TX\_DEVMODECHANGE message to all Text Control windows. For more information see the description of that message.

---

## TX\_EXPORTTEXT

This message exports the selected text of a Text Control in an external format.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains a DOS file handle. The converted data is written to that file at the current file position. If <i>wParam</i> contains

HFILE\_ERROR, the message returns a global data handle for a buffer containing the converted data.

*lParam* Points to a null-terminated character string that specifies the name of the filter library that shall be used to convert the data. If the filter library is not in the same directory as the TX library, the character string must contain the complete path. Otherwise it must contain the name of the filter, i.e. TX\_RTF.DLL.

**Return Value** The return value contains a global data handle if no file handle has been specified. Otherwise it contains a nonzero value. The return value contains zero if an error has occurred.

If no selection exists, the complete text is exported.

If *lParam* is set to zero, the return value is a global data handle for a buffer containing text formatted with the internal format described in appendix A.

The global data handle identifies a global memory block which contains the exported data. It must be freed with the **GlobalFree** function when it is not longer needed.

The selection can extend over several linked Text Controls.

The development of a filter is described in appendix C.

## TX\_GETASCIITEXT

This message copies the text belonging to the Text Control into a buffer provided by the caller. The text is copied in a Windows compatible text format for example to exchange it with a Windows Edit-Control. This message supports buffers larger than 64 kB of text. To calculate the size of the buffer required the TX\_GETASCIITEXTSIZE message should be used. The buffer must be large enough to accommodate a terminating zero character.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies the maximum number of characters to be copied, including the terminating zero character. If <i>wParam</i> is zero the complete text contained by the Text Control is copied.

*lParam* Points to the buffer that is to receive the text.

**Return Value** The return value is the number of bytes copied.

**Comments** For more information when to use this message see the comments section of the TX\_GETTEXT message.

32 bit: This message does not support Unicode.

---

## TX\_GETASCIITEXTSIZE

This message returns the length of the text (in bytes) associated with the Text Control in a Windows compatible text format.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is the length of the text.

**Comments** 32 bit: This message does not support Unicode.

---

## TX\_GETHANDLE

This message returns the data handle for the buffer that holds the text of the Text Control. It is always a global handle. This message is only supported if the Text Control contains only a small amount of text up to 10 kB. For larger amounts of text the WM\_GETTEXT message can be used.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value specifies the data handle. It is zero if the amount of text is too large.

**Comments** 32 bit: The character string identified through the returned data handle is in Unicode format.

## TX\_GETIMAGEFORMAT

If an image has been selected this message retrieves information about the formatting of an image. The Text Control sends a TN\_IMAGECLICKED notification message if an image has been selected.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an IMAGEFORMAT data structure which is defined as follows:

```
typedef struct tagIMAGEFORMAT {
    POINT ptPosition;
    POINT ptSize;
    POINT ptMaxPosition;
    WORD wScale;
    WORD wFlags;
} IMAGEFORMAT;
```

The IMAGEFORMAT structure has the following fields:

<u>Field</u>	<u>Description</u>
<i>ptPosition</i>	Specifies the image's horizontal position in twentieths of a point. The y-coordinate of this POINT structure is not used.
<i>ptSize</i>	Specifies the image's unscaled horizontal and vertical dimensions in twentieths of a point.
<i>ptMaxPosition</i>	Specifies the maximum horizontal position in twentieths of a point. The y-coordinate of this POINT structure is not used.
<i>wScale</i>	Specifies the image's scaling factor in percent. This is a value between 10 and 250.

<i>wFlags</i>	Can contain a combination of the following flags:						
	<table> <thead> <tr> <th style="text-align: left;"><u>Value</u></th> <th style="text-align: left;"><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>ICF_GRAYED</td> <td>The image is displayed in fast mode.</td> </tr> <tr> <td>ICF_SAVEASDATA</td> <td>The Text Control saves the image by its data instead of its filename.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	ICF_GRAYED	The image is displayed in fast mode.	ICF_SAVEASDATA	The Text Control saves the image by its data instead of its filename.
<u>Value</u>	<u>Meaning</u>						
ICF_GRAYED	The image is displayed in fast mode.						
ICF_SAVEASDATA	The Text Control saves the image by its data instead of its filename.						

**Return Value** The return value is zero if an error has occurred or if no image is currently selected or registered. Otherwise it is nonzero.

**Comments** The IC Image-Control programming tool is needed for this message.

---

## TX\_IMPORTTEXTBUFFER

This message imports text and format data which is given in an external format. The text is inserted at the current input position.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies a global data handle. The data to be imported will be read out of the memory block which is identified by the data handle.
<i>lParam</i>	Points to a null-terminated character string that specifies the name of the filter library that is to be used to convert the data. If the filter library is not in the same directory as the TX library, the character string must contain the complete path. Otherwise it must contain the name of the filter, i.e. TX_RTF.DLL. If <i>lParam</i> is zero, the buffer specified by <i>wParam</i> must contain text formatted with the Text Control's internal format according to appendix A.

**Return Value** The return value value is zero if an error has occurred, otherwise it is nonzero.

**Comments** The development of a filter is described in appendix C.

---

## TX\_IMPORTTEXTFILE

This message imports text and format data, given in an external format, from a file. The text is inserted at the current input position.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a DOS file handle. The data to be imported will be read from that file at the current file position.
<i>lParam</i>	Points to a null-terminated character string that specifies the name of the filter library which is to be used to convert the data. If the filter library is not in the same directory as the TX library, the character string must contain the complete path. Otherwise it must contain the name of the filter, i.e. TX_RTF.DLL.

***Return Value***     The return value value is zero if an error has occurred, otherwise it is nonzero.

***Comments***         The development of a filter is described in appendix C.

---

## TX\_SETHANDLE

This message sets a new data handle associated with the text contained in the Text Control. It has to be a global handle. The Text Control does not free the old data handle. This handle can be obtained using the TX\_GETHANDLE message. A new handle is only set if the TX\_GETHANDLE message returns a valid text handle.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains the new handle. The buffer identified by this handle must contain a null-terminated string. The size of the buffer is not limited.
<i>lParam</i>	Is not used.

***Return Value***     The return value is:

<u>Value</u>	<u>Meaning</u>
TR_ERR,	An error has occurred.
TR_UNCHANGED,	The text is too long. The limit is the current text limit value set by the TX_LIMITTEXT message.
TR_CHANGED,	The new handle has been set.

**Comments**

With this message all format information gets lost. The Text Control uses the first font of the old text to display the new one. The Text Control is not refreshed.

If the return value is TR\_CHANGED the new handle belongs to the Text Control and can no longer be used by the calling application. The old handle no longer belongs to the Text Control and must be freed by the calling application.

32 bit: The character string identified through the data handle must be in Unicode format.

---

**TX\_SETIMAGEFORMAT**

If an image has been selected this message sets new formatting parameters for the image. The Text Control sends a TN\_IMAGECLICKED notification message if an image has been selected.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to an IMAGEFORMAT data structure. The structure members <i>wScale</i> and <i>wFlags</i> can be used to format the image. All other members are ignored. See the description of the TX_GETIMAGEFORMAT message for more information about the IMAGEFORMAT data structure.

**Return Value**

The return value is:

TR_ERR	if an error has occurred or if no image is currently selected or registered. It is also TR_ERR if a scaling factor is specified so that the image becomes too large for the Text Control's window size.
--------	---

TR\_UNCHANGED    if nothing could be changed.  
TR\_CHANGED        if the new values have been set and updated.

**Comments**        The IC Image-Control programming tool is needed for this message.

# Appendix A

## The TX Text Control Text Format

TX Text Control text format is described as follows. Text Control supports all prior versions of the format, but, to maximize working speed, the current format should be used. The data structures used with the text format are described in chapter 7 "Data Structures".

The text and formatting data is structured through several data blocks. The main data blocks are:

1. Text and formatting data structured in blocks of text
2. Fixed positioned objects
3. Document data

A certain format description must contain at least one block of text. Fixed positioned objects and document data are optional and exist only if the whole contents of a Text Control are saved.

The format has the following form:

```
WORD          wVersion;
WORD          wTextBlocks;
DWORD        dwTBOffset;
WORD          wFixedObjects;
DWORD        dwFxOffset;
BYTE         cTextBlocks[ ];
BYTE         cFixedObjects[ ];
LONG         lReserved;
DWORD        dwDocDataSize;
BYTE         cDocData[ ];
WORD         wDocSections;
BYTE         cDocSections[ ];
```

These fields have the following meanings:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wVersion</i>	Specifies the version number of the text format. Currently the number is 700.
<i>wTextBlocks</i>	Is the number of text blocks the <i>cTextBlocks</i> [ ] array

---

	contains.
<i>dwTBOffset</i>	Specifies the offset, in bytes, from the beginning of the format description to the beginning of the <i>cTextBlocks</i> [ ] data block.
<i>wFixedObjects</i>	Specifies the number of fixed positioned objects the <i>cFixedObjects</i> [ ] data block contains. This value is zero when the <i>cFixedObjects</i> [ ] data block is omitted.
<i>dwFxOffset</i>	Specifies the offset, in bytes, from the beginning of the format description to the beginning of the <i>cFixedObjects</i> [ ] data block. This value is zero when no <i>cFixedObjects</i> [ ] data block exists.
<i>cTextBlocks</i> [ ]	This data block contains the text and formatting data structured as blocks of text. The number of these blocks is specified by <i>wTextBlocks</i> . Further information about this data block's internal structure can be found later on in this chapter.
<i>cFixedObjects</i> [ ]	This data block contains FIXEDOBJECT data structures each of which is followed by the object's data. When no object exists this block is omitted.
<i>lReserved</i>	This value is for future use and must be zero. This value cannot be omitted.
<i>dwDocDataSize</i>	Specifies the size, in bytes, of the <i>cDocData</i> [ ] data block. This value cannot be omitted and must be zero when the <i>cDocData</i> [ ] data block does not exist.
<i>cDocData</i> [ ]	This data block contains general document data. Further information about this data block's internal structure can be found later in this chapter.
<i>wDocSections</i>	Is the number of document sections the <i>cDocSections</i> [ ] array contains. This value cannot be omitted and must be zero if the <i>cDocSections</i> [ ] data block does not exist.
<i>cDocSections</i> [ ]	This data block contains the document's section data, structured as an array of sections. The number of sections is specified by <i>wDocSections</i> . Further information about this data block's internal structure can be found later on in this chapter.

**cTextBlocks[ ]**

The *cTextBlocks[ ]* data block contains a number of blocks of text, each of which is structured in the following form:

```

DWORD      dwTextSize;
BYTE       szText[ ];
DWORD      dwListSize;
WORD       wFontBlocks;
WORD       wLogFonts;
WORD       wPgBlocks;
WORD       wParagraphs;
WORD       wImages;
WORD       wTables;
WORD       wCellAttrEntries;
WORD       wFieldDataEntries;
WORD       fAddData;
FONTBLOCK  fbFontBlocks[ ];
TXLOGFONT  lfLogFonts[ ];
PGBLOCK    pbPgBlocks[ ];
PARAGRAPH  pgParagraphs[ ];
BYTE       ImageData[ ];
BYTE       TableData[ ];
CELLATTR   caCellAttr[ ];
BYTE       FieldData[ ];
TXLOGFONTEX lfLogFontsEx[ ];
TXTEXTDATAEX TextDataEx;

```

This structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>dwTextSize</i>	<p>The low-order word specifies the size of the following text string (in characters when text is stored in Unicode format and in bytes when text is stored in ANSI format) without the terminating zero. To optimize the time for loading a text file, the size of the text string should be between 10000 and 20000 characters.</p> <p>The high-order word contains zero when the following text string is in ANSI format (single- or double-byte) and it contains 2 when <i>szText</i> is a Unicode string (2 bytes per character).</p> <p>When the text string contains double-byte characters all offsets in the following format lists are still offsets to characters and not byte-offsets in this string.</p>

*szText[]* Is a zero-terminated character string containing the text. The size of the string (Unicode: in characters, otherwise in bytes) including the terminating zero is  $\text{LOWORD}(dwTextSize) + 1$ . The format (Unicode or ANSI) is specified through the high-order word of the *dwTextSize* member.

TX uses the following special character codes:

<b><u>Code</u></b>	<b><u>Description</u></b>
01H	Image.
09H	Tabulator.
0AH	End of paragraph.
0BH	Forced end of line (no end of paragraph).
0CH	Forced end of page.
1EH	End of line hyphen included with a user-defined word-division function.
1FH	End of line hyphen included with the keyboard.
A0H	Non-breaking space.

*dwListSize* Specifies the size of the following format data (in bytes) including the beginning word values.

*wFontBlocks* Specifies the number of FONTBLOCK structures the *fbFontBlocks[]* array contains.

*wLogFonts* Specifies the number of TXLOGFONT structures the *lfLogFonts[]* array contains.

*wPgBlocks* Specifies the number of PGBLOCK structures the *pbPgBlocks[]* array contains.

*wParagraphs* Specifies the number of PARAGRAPH structures the *pgParagraphs[]* array contains.

*wImages* Specifies the number of images the *ImageData[]* array contains.

<i>wTables</i>	Specifies the number of TXTABLE structures contained by the <i>TableData</i> [ ] block.						
<i>wCellAttrEntries</i>	Specifies the number of CELLATTR structures contained by the <i>caCellAttr</i> [ ] array.						
<i>wFieldDataEntries</i>	Specifies the number of FIELDDATA structures contained by the <i>FieldData</i> [ ] block.						
<i>fAddData</i>	Is a flag field specifying additional data following the <i>FieldData</i> byte array. It can be a combination of the following values:						
	<table border="0"> <thead> <tr> <th style="text-align: left;"><u>Code</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>0001H</td> <td>When this bit is set the textblock structure contains an array of TXLOGFONTEX structures which follows the <i>FieldData</i>[ ] array.</td> </tr> <tr> <td>0002H</td> <td>When this bit is set the textblock structure contains a TXTEXTDATAEX structure which follows the <i>lfLogFontsEx</i>[ ] array.</td> </tr> </tbody> </table>	<u>Code</u>	<u>Description</u>	0001H	When this bit is set the textblock structure contains an array of TXLOGFONTEX structures which follows the <i>FieldData</i> [ ] array.	0002H	When this bit is set the textblock structure contains a TXTEXTDATAEX structure which follows the <i>lfLogFontsEx</i> [ ] array.
<u>Code</u>	<u>Description</u>						
0001H	When this bit is set the textblock structure contains an array of TXLOGFONTEX structures which follows the <i>FieldData</i> [ ] array.						
0002H	When this bit is set the textblock structure contains a TXTEXTDATAEX structure which follows the <i>lfLogFontsEx</i> [ ] array.						
<i>fbFontBlocks</i> [ ]	Is an array of FONTBLOCK structures.						
<i>lfLogFonts</i> [ ]	Is an array of TXLOGFONT structures.						
<i>pbPgBlocks</i> [ ]	Is an array of PGBLOCK structures.						
<i>pgParagraphs</i> [ ]	Is an array of PARAGRAPH structures.						
<i>ImageData</i> [ ]	Is an array of bytes, which contains an array of IMAGEDATA structures, each of which is followed by the image data itself. The first image in this list of images belongs to the first 01H control character in the <i>szText</i> array, the second image belongs to the second 01H character and so on.						
<i>TableData</i> [ ]	Is an array of bytes, which contains an array of TXTABLE structures, each of which is followed by CELLPOSITION structures describing the table cell's positions in the text and their geometric positions. The table's text is contained in the <i>szText</i> [ ] array. Each cell's text must end with a paragraph end control character (0AH).						

<i>caCellAttr</i> [ ]	Is an array of CELLATTR structures.
<i>FieldData</i> [ ]	Is an array of bytes which contains FIELDATA structures, each of which is followed by the field data itself. The entries in this array are in the following order: The array begins with all entries with user-defined data in the same order, the fields appear in the text. The array ends with all entries that contain data belonging to fields of a special type, for example link information. These entries are also in the same order as the fields appear in the text.
<i>lfLogFontsEx</i> [ ]	Is an array of TXLOGFONTEX structures. The size of this array is given through <i>wLogFonts</i> . This array exists only when it is specified through <i>fAddData</i> .
<i>TextDataEx</i>	Is a data structure containing additional text data. This structure exists only when it is specified through <i>fAddData</i> .

---

## **cDocData[ ]**

The *cDocData*[ ] data block contains general document data and is structured in the following form:

```

WORD          wDocHeaderSize;
WORD          wReserved;
DWORD        dwDocWidth;
DWORD        dwDocHeight;
short        nDocMarginLeft;
short        nDocMarginTop;
short        nDocMarginRight;
short        nDocMarginBottom;
COLORREF     crDocBkGnd;
LONG         lDocTextPos;
WORD         wDocTitleSize;
WORD         wDocBasePathSize;
BYTE         reserved[16];
BYTE         szDocTitle[];
BYTE         szDocBasePath[];

```

This structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wDocHeaderSize</i>	Specifies the size of this data block without the variable byte-arrays at the end.
<i>wReserved</i>	2 bytes reserved for future use. This member must be set to zero.
<i>dwDocWidth</i>	Specifies the width of a document's page in twentieths of a point including margins. This value is set to zero, if it is undefined.
<i>dwDocHeight</i>	Specifies the height of a document's page in twentieths of a point - including margins. This value is set to zero, if it is undefined.
<i>nDocMarginxxx</i>	Specifies the margins of a document's page in twentieths of a point. When a value is undefined it is set to -1.
<i>crDocBkGnd</i>	Specifies the document's background color. This value is set to UNDEF_COLOR, if it is undefined.
<i>lDocTextPos</i>	Specifies the document's current input position. The position zero is the position in front of the first character.
<i>wDocTitleSize</i>	Specifies the size, in bytes, of the <i>szDocTitle[ ]</i> string including the terminating zero character.
<i>wDocBasePathSize</i>	Specifies the size, in bytes, of the <i>szDocBasePath[ ]</i> string including the terminating zero character.
<i>reserved[16]</i>	16 bytes reserved for future use, which must be set to zero.
<i>szDocTitle[ ]</i>	Is a zero-terminated string specifying the document's title. This array exists only, if the <i>wDocTitleSize</i> member contains a non-zero value. This string is always in Unicode character format.
<i>szDocBasePath[ ]</i>	Is a zero-terminated string specifying the document's base path. This array exists only, if the <i>wDocBasePathSize</i> member contains a non-zero value. This string is always in Unicode character format.

**cDocSection[ ]**

The *cDocSection[ ]* data block contains a number of document sections, each of which is structured in the following form:

```

DWORD          dwSectionDataSize;
DWORD          dwSectionHeaderSize;
WORD           fElements;
DWORD          reserved;
short         nHeaderTop;
short         n1stHeaderTop;
short         nFooterBottom;
short         n1stFooterBottom;
BYTE          cHeaderTextBlock[ ];
BYTE          c1stHeaderTextBlock[ ];
BYTE          cFooterTextBlock[ ];
BYTE          c1stFooterTextBlock[ ];

```

This structure has the following fields:

<b><u>Field</u></b>	<b><u>Description</u></b>										
<i>dwSectionDataSize</i>	Specifies the size, in bytes, of this section's data including this member.										
<i>dwSectionHeaderSize</i>	Specifies the size, in bytes, of this section's header. The header starts with the <i>dwSectionDataSize</i> member and ends with the <i>n1stFooterBottom</i> member.										
<i>fElements</i>	Is a flag field specifying the elements, that this section contains. It can be a combination of the following values: <table border="0" style="margin-left: 2em;"> <thead> <tr> <th style="text-align: left;"><b><u>Code</u></b></th> <th style="text-align: left;"><b><u>Description</u></b></th> </tr> </thead> <tbody> <tr> <td>TF_HF_HEADER</td> <td>This section's data block has the <i>cHeaderTextBlock[ ]</i> array.</td> </tr> <tr> <td>TF_HF_1STHEADER</td> <td>This section's data block has the <i>c1stHeaderTextBlock[ ]</i> array.</td> </tr> <tr> <td>TF_HF_FOOTER</td> <td>This section's data block has the <i>cFooterTextBlock[ ]</i> array.</td> </tr> <tr> <td>TF_HF_1STFOOTER</td> <td>This section's data block has the <i>c1stFooterTextBlock[ ]</i> array.</td> </tr> </tbody> </table>	<b><u>Code</u></b>	<b><u>Description</u></b>	TF_HF_HEADER	This section's data block has the <i>cHeaderTextBlock[ ]</i> array.	TF_HF_1STHEADER	This section's data block has the <i>c1stHeaderTextBlock[ ]</i> array.	TF_HF_FOOTER	This section's data block has the <i>cFooterTextBlock[ ]</i> array.	TF_HF_1STFOOTER	This section's data block has the <i>c1stFooterTextBlock[ ]</i> array.
<b><u>Code</u></b>	<b><u>Description</u></b>										
TF_HF_HEADER	This section's data block has the <i>cHeaderTextBlock[ ]</i> array.										
TF_HF_1STHEADER	This section's data block has the <i>c1stHeaderTextBlock[ ]</i> array.										
TF_HF_FOOTER	This section's data block has the <i>cFooterTextBlock[ ]</i> array.										
TF_HF_1STFOOTER	This section's data block has the <i>c1stFooterTextBlock[ ]</i> array.										
<i>reserved</i>	4 bytes reserved for future use. This member must be set										

	to zero.
<i>nHeaderTop</i>	Specifies the distance between the top of the header and top of the page in twentieths of a point. If this value is undefined, it is set to -1.
<i>n1stHeaderTop</i>	Specifies the distance between the top of the first page's header and the top of the first page in twentieths of a point. If this value is undefined, it is set to -1.
<i>nFooterBottom</i>	Specifies the distance between the bottom of the footer and the bottom of the page in twentieths of a point. If this value is undefined, it is set to -1.
<i>n1stFooterBottom</i>	Specifies the distance between the bottom of the first page's footer and the bottom of the first page in twentieths of a point. If this value is undefined, it is set to -1.
<i>cHeaderTextBlock</i> [ ]	This data block contains the text and formatting data of this section's header. It is structured in the same form as one text block of the main text. This block exists only, if the appropriate bit in <i>fElements</i> is set.
<i>c1stHeaderTextBlock</i> [ ]	This data block contains the text and formatting data of this section's first page header. This block exists only, if the appropriate bit in <i>fElements</i> is set.
<i>cFooterTextBlock</i> [ ]	This data block contains the text and formatting data of this section's footer. This block exists only, if the appropriate bit in <i>fElements</i> is set.
<i>c1stFooterTextBlock</i> [ ]	This data block contains the text and formatting data of this section's first page footer. This block exists only, if the appropriate bit in <i>fElements</i> is set.

## Format Example

The following example shows a short formatted text and the text dump of the text format for this text. The text is written in Arial, character size 12, with a bold and an underlined part and has a left aligned and a centered paragraph. The text dump can be obtained with the TX\_DATAOUT or the TX\_COPYDATA message.

*Example* This simple **formatted text** contains  
two fonts and two paragraphs.

<i>Text Dump</i>	<u>Field</u>	<u>Value (hexadecimal)</u>
	<i>wVersion</i>	2EC
	<i>wTextBlocks</i>	1
	<i>dwTBOffset</i>	E
	<i>wFixedObjects</i>	0
	<i>dwFxOffset</i>	0
	<i>dwTextSize</i>	41
	<i>szText[ ]</i>	54 68 69 73 20 73 69 6D 70 6C 65 20 66 6F 72 6D 61 74 74 65 64 20 74 65 78 74 20 63 6F 6E 74 61 69 6E 73 0A 74 77 6F 20 66 6F 6E 74 73 20 61 6E 64 20 74 77 6F 20 70 61 72 61 67 72 61 70 68 73 2E 00
	<i>dwListSize</i>	12E
	<i>wFontBlocks</i>	3
	<i>wLogFonts</i>	2
	<i>wPgBlocks</i>	2
	<i>wParagraphs</i>	2
	<i>wImages</i>	0
	<i>wTables</i>	0
	<i>wCellAttrEntries</i>	0
	<i>wFieldDataEntries</i>	0
	<i>fAddData</i>	0
	<i>fbFontBlocks[3]</i>	1 1 C 0 40000000 0 0 0 2 D E 0 40000000 0 0 0

	1	1B	27	0	40000000	0	0	0
<i>lfLogFonts[2]</i>	FF10	0	0	0	190	0	0	0
	0	0	02	22	"Arial"			
	FF10	0	0	0	2BC	0	1	0
	0	0	02	22	"Arial"			
<i>pbPgBlocks[2]</i>	1	1	24					
	2	25	42					
<i>pgParagraphs[2]</i>	0	0	0	0	0	0	64	0
	0							
	...(42 bytes: the tablist of paragraph 1)							
	8000	0	0	0	0	0	64	0
	0							
	...(42 bytes: the tablist of paragraph 2)							
<i>lReserved</i>	0							
<i>dwDocDataSize</i>	0							
<i>wDocSections</i>	0							

## Appendix B

This appendix describes the error codes returned by the **GetErrorCode** or **TXGetErrorCode** function. The error codes described here belong to module number 1 which is the Text Control module number and module number 5 which is the module handling OLE objects. For information about error codes associated with other modules see the reference manuals for those modules.

Each code has a group code and a location code. The group code describes a general error condition for example when the program runs out of memory and the location code describes the location where the error has been occurred. In most cases it is sufficient for an application to handle the group code. **TXGetErrorCode** returns both the group and the location code while **GetErrorCode** only returns the location code.

### Group Codes

Code	Value	Description
<b>DBS_E_UNGROUPED</b>	<b>00</b>	A special error condition has occurred. See the description of the location code for more information.
<b>DBS_E_OUTOFMEMORY</b>	<b>01</b>	Not enough storage is available to complete the operation.
<b>DBS_E_NOMEMORYACCESS</b>	<b>02</b>	Invalid access to a memory location.
<b>DBS_E_UNEXPECTED</b>	<b>03</b>	Unexpected failure.
<b>DBS_E_FILEIO</b>	<b>04</b>	A file read/write operation cannot be performed.
<b>DBS_E_64K</b>	<b>05</b>	An operation cannot be performed because needed memory blocks must not be larger than 64 kByte.
<b>DBS_E_CLIPBOARD</b>	<b>06</b>	A clipboard operation cannot be performed. The clipboard cannot be opened or emptied or clipboard data cannot be accessed.
<b>DBS_E_DLLNOTLOADED</b>	<b>07</b>	An operation cannot be performed because a helper DLL or filter needed for the operation cannot be found or loaded.

<b>DBS_E_DLLINCOMPATIBLE</b>	<b>08</b>	An operation cannot be performed because a helper DLL or filter needed for the operation is too old.
<b>DBS_E_DLLOBSOLETE</b>	<b>09</b>	A helper DLL or filter needed for the operation is obsolete but can be used for the operation.
<b>DBS_E_INVALIDARG</b>	<b>0A</b>	One or more arguments are invalid.
<b>DBS_E_NOTIMPL</b>	<b>0B</b>	A needed feature is not implemented.
<b>DBS_E_INVALIDFORMAT</b>	<b>0C</b>	An operation cannot be performed because data has an invalid format.

## *Error Code Table (TX kernel module)*

The following location codes are sorted according to its group codes.

### **DBS\_E\_UNGROUPED**

<b>Code</b>	<b>Description</b>
<b>040C</b>	Data of fixed positioned object cannot be saved. The WMX_COPYDATA message failed.
<b>0801</b>	Unable to create a new Image-Control window because the requested window size is too large.
<b>0A00</b>	Error in <b>CreateTextControl</b> , the given window size is too small.
<b>0A01</b>	Error in <b>CreateTextControl</b> , the given window size is too large.
<b>0B06</b>	Unable to get image data from the Image-Control module.
<b>0C02</b>	Headers and footers are not initialized.
<b>0E00</b>	The requested window size is too small. Use the WM_GETMINMAXINFO message to obtain information about the minimum window size before resizing.
<b>0E01</b>	Unable to zoom the TX window. The window size cannot be zoomed further.
<b>100A</b>	The client area of the Text Control window is too small to display at least one line of text. Use the WM_GETMINMAXINFO message to obtain information about minimum sizes.
<b>1408</b>	Error while calling TX_SETDEVICE, device unknown.

<b>1901</b>	Text Control windows cannot be linked. The specified window handle is not a Text Control window handle.
<b>1902</b>	Text Control windows cannot be linked. The selected combination of connections is not possible.
<b>1903</b>	Full message queue. Text cannot be formatted.
<b>1904</b>	The following window cannot be connected because the specified Text Control has either a formatting area set with the TX_SETTEXTAREA message or the TF_AUTOEXPAND mode has been selected.
<b>1D01</b>	Filter error: the text data to be imported has the wrong text format.
<b>1D02</b>	Filter error: bad token in data.
<b>1D03</b>	Filter error: unable to allocate enough memory
<b>1D04</b>	Filter error: unable to read data.
<b>1D05</b>	Filter error: unable to write data.
<b>1D06</b>	Filter error: unable to open file.
<b>1D07</b>	Filter error: input file too big.
<b>1D08</b>	Filter error: unsupported part of data.
<b>1D09</b>	Filter error: internal filter error.
<b>1D0A</b>	Filter error: filter too old for the specified format.
<b>1F04</b>	Full message queue. The text operation was too complex. The <b>SetMessageQueue</b> function can be used to avoid this error.
<b>2800</b>	Zoom operation could not be performed. The Text Control becomes too large.
<b>2B00</b>	Full message queue. WM_CLEAR operation could not be performed.

### DBS\_E\_FILEIO

<b>Code</b>	<b>Description</b>
<b>0405</b>	File error, unable to read format data from a file.
<b>0407</b>	File error, unable to write format data into a file.
<b>040E</b>	Unexpected error. Invalid data size.
<b>0415</b>	Data cannot completely read from file.
<b>0416</b>	Data cannot completely read from buffer.
<b>3308</b>	Cannot read data formatted in the Text Control's text format from a specified file.
<b>330A</b>	Cannot write data to a specified file.

**DBS\_E\_64K**

<b>Code</b>	<b>Description</b>
<b>0400</b>	Unable to load or paste the given text. The size of the text is more than 64kB for a single Text Control or the format is unknown to TX.
<b>0B0E</b>	Format data part more than 64 kB. Use smaller blocks of text.
<b>1905</b>	The text size of linked windows is limited to 64 kByte per window.
<b>1907</b>	Text cannot be completely inserted. The text of a single Text Control overflows the 64 kB limit.

**DBS\_E\_CLIPBOARD**

<b>Code</b>	<b>Description</b>
<b>1403</b>	Unable to empty the clipboard during the TX_ADJUSTCLIPBOARD message.
<b>1405</b>	Clipboard data not available, <b>GlobalLock</b> failure.
<b>1407</b>	Unable to allocate enough memory to adjust the clipboard data to a new output device.
<b>1600</b>	Unable to empty the clipboard.
<b>1601</b>	<b>SetClipboardData</b> failed.
<b>1602</b>	<b>SetClipboardData</b> failed.
<b>1603</b>	<b>OpenClipboard</b> failed, the clipboard is still open.
<b>1604</b>	Clipboard data not available, <b>GlobalLock</b> failure.
<b>1611</b>	The clipboard contains an old TX format. Old clipboard formats are not supported.
<b>1612</b>	<b>OpenClipboard</b> failed.

**DBS\_E\_DLLNOTLOADED**

<b>Code</b>	<b>Description</b>
<b>0804</b>	The Image-Control module IC.DLL or IC32.DLL could not be loaded.
<b>1B00</b>	Filter not available, <b>LoadLibrary</b> failed.
<b>2C06</b>	The library file TXOBJ32.DLL cannot be found.
<b>2C0E</b>	The library file WNDTLS32.DLL or WNDTOOLS.DLL cannot be found.

**DBS\_E\_DLLINCOMPATIBLE**

<b>Code</b>	<b>Description</b>
<b>0803</b>	The Image-Control module IC.DLL or IC32.DLL must be a newer version.
<b>1B01</b>	Unknown filter or filter function not found.
<b>1B04</b>	The specified text filter does not contain the extended interface functions <b>TX_Ex_Export</b> and <b>TX_Ex_Import</b> .
<b>1B05</b>	Filter too old: GetFilterInfo not available.
<b>1B06</b>	The text filter cannot process Unicode strings.
<b>2C08</b>	The library file TXOBJ32.DLL must be a newer version.
<b>2C09</b>	Invalid library file TXOBJ32.DLL. Function not found.
<b>2C0B</b>	The library file WNDTLS32.DLL or WNDTOOLS.DLL must be a newer version.

**DBS\_E\_DLLOBSOLETE**

<b>Code</b>	<b>Description</b>
<b>0802</b>	The Image-Control module IC.DLL or IC32.DLL is obsolete but can be used.
<b>2C07</b>	The library file TXOBJ32.DLL is obsolete but can be used.
<b>2C0F</b>	The library file WNDTLS32.DLL or WNDTOOLS.DLL is obsolete but can be used.

**DBS\_E\_INVALIDARG**

<b>Code</b>	<b>Description</b>
<b>0300</b>	The TX_FIELD_SETDATA message has been used with invalid arguments. Either the <i>dwData</i> member or the <i>dwDataSize</i> and <i>lpdata</i> members of the FIELDSETDATA data structure must be set to zero.
<b>0308</b>	The TX_FIELD_SETDATA message has been used with invalid arguments. Either the <i>dwData</i> member or the <i>dwDataSize</i> and <i>lpdata</i> members of the FIELDSETDATA data structure must be set to zero.
<b>030D</b>	TX_FIELD_SETTYPE: invalid parameter. Data size not specified.
<b>030E</b>	TX_FIELD_SETTYPE: invalid parameter. Specified data not necessary.
<b>030F</b>	TX_FIELD_SETTYPE: invalid parameter. Specified data invalid for specified field type.
<b>0413</b>	No file or buffer specified.

---

<b>0900</b>	WM_GETTEXT/TX_GETTEXT messages: The specified buffer is too small.
<b>0901</b>	TX_SETLINEANDCOL message: Invalid page number.
<b>0902</b>	TX_SETLINEANDCOL message: Invalid line number.
<b>0903</b>	TX_SETLINEANDCOL message: Invalid column number.
<b>0C01</b>	The position value for the specified header or footer is invalid.
<b>1400</b>	Invalid new text handle sent with the TX_SETHANDLE message.
<b>1E00</b>	TX_FIELD_GETTEXT message: Invalid buffer length.
<b>1E01</b>	The field text cannot be set to an empty string, as this would result in two empty fields being at the same text input position.
<b>1E02</b>	Fields of the types FT_LINKTARGET and FT_TOPIC must not contain any text.
<b>1E03</b>	TX_FIELD_SETTYPE: The specified field could not be found.
<b>1E04</b>	TX_FIELD_SETTYPE: Fields of the types FT_LINKTARGET, FT_PAGENUMBER and FT_TOPIC must not contain any text.
<b>1E05</b>	TX_FIELD_SETTYPE: Fields of the types FT_PAGENUMBER can only be inserted in headers or footers.
<b>2100</b>	The specified page height for printing is too small. The height must be large enough for at least one line of text.
<b>2601</b>	The specified text area could not be set because the width is less than the minimum width.
<b>2602</b>	The specified text area could not be set because the height is less than the minimum height.
<b>2C01</b>	Fixed positioned object cannot be embedded. Its size is too large.
<b>2C02</b>	Fixed positioned object cannot be embedded. Its size is too large.
<b>2C03</b>	Fixed positioned object cannot be embedded. Its position is outside the Text Control's client area or page rectangle.
<b>2C04</b>	Fixed positioned object cannot be embedded. Invalid embedding mode.
<b>2C05</b>	External window cannot be inserted as external object. The window must be a child window.
<b>3300</b>	Invalid parameter specified with the TX_DATAIN message.
<b>3309</b>	Invalid parameter specified with the TX_DATAOUT message.
<b>330B</b>	This Text Control version does not support the specified format.

**DBS\_E\_NOTIMPL****Code Description**

<b>2101</b>	TX_PRINT cannot be used for Text Controls with a set text area. Use the TX_PRINTPAGE message.
<b>2102</b>	TX_PRINT cannot be used for metafiles. Use the WM_PAINT message and see chapter 1.13 for more information.
<b>2600</b>	Error while calling the TX_SETTEXTAREA message. The specified Text Control is either connected to a following window or the TF_AUTOEXPAND mode has been selected.
<b>2C0C</b>	OLE objects cannot be used with the 16 bit Text Control.

**DBS\_E\_INVALIDFORMAT****Code Description**

<b>0406</b>	Cannot convert old prior image format.
<b>0410</b>	The text format does not start with a valid version number.
<b>0414</b>	Document data part invalid.
<b>0B12</b>	TF_FORMAT_UNICODE is only supported through the Text Control 32 bit version.
<b>0B14</b>	A Unicode character string cannot be converted to ANSI.
<b>1B03</b>	TX_IMPORTTEXT* messages: Unable to convert old format, the format is too old.
<b>1C03</b>	Loading text in Unicode format is only supported through the Text Control 32 bit version.
<b>2001</b>	Image has no reference in the text buffer.
<b>2004</b>	The 16 bit Text Control does not support Unicode.
<b>2006</b>	A multibyte character string cannot be converted to Unicode.
<b>2009</b>	Format error, unknown format, use a valid TX format.
<b>200B</b>	Format error, unknown format. Use a valid TX format.
<b>200C</b>	Format error, text different from format data.
<b>3304</b>	Cannot insert data formatted in the Text Control's text format with the TX_DATAIN message. The format is too old. Use the TX_LOAD message for this format.

- 3305** Cannot insert data formatted in the Text Control's text format with the TX\_DATAIN message. The format contains an invalid text size. Use the **DebugTXFormat** function and see appendix G for more information.
- 3306** Cannot insert data formatted in the Text Control's text format with the TX\_DATAIN message. The format contains an invalid format data size. Use the **DebugTXFormat** function and see appendix G for more information.

### **DBS\_E\_OUTOFMEMORY**

<b>Code</b>	<b>Description</b>
<b>0100</b>	Unable to allocate enough local memory.
<b>0103</b>	Unable to allocate enough local memory.
<b>0202</b>	Unable to allocate enough memory to save the text.
<b>0204</b>	Unable to create any more paragraphs.
<b>0301</b>	Unable to allocate enough memory to store data for a marked text field.
<b>0304</b>	Unable to allocate enough memory to store data for a marked text field.
<b>030C</b>	Unable to allocate enough memory to store data for a marked text field.
<b>0401</b>	Unable to allocate enough memory to load the text from a file or to paste the text with the TX_PASTEDATA message.
<b>0403</b>	Unable to allocate enough memory to load the format data from a file or to paste the format data with the TX_PASTEDATA message.
<b>0408</b>	Unable to allocate enough memory to convert an earlier TX format.
<b>040A</b>	Unable to allocate enough memory to save fixed positioned objects.
<b>040D</b>	Unable to allocate enough memory to load fixed positioned objects.
<b>0412</b>	Unable to allocate enough memory for saving document data.
<b>0503</b>	Unable to allocate local memory for the font manager.
<b>0701</b>	Unable to get data from the font manager during the TX_SETFONT message.
<b>0703</b>	Unable to allocate enough memory to alter font information.
<b>0705</b>	Unable to allocate enough memory to initialize the font manager.
<b>0A03</b>	Unable to expand the global fontlist.
<b>0B00</b>	Unable to allocate enough memory to create data for the internal text format.
<b>0B09</b>	Unable to allocate enough memory.
<b>0B0D</b>	Unable to allocate enough memory.
<b>0B0F</b>	Unable to allocate enough memory.

---

<b>0B10</b>	Unable to allocate enough memory.
<b>0B13</b>	Unable to allocate enough memory to convert a Unicode string to ANSI.
<b>0C00</b>	Not enough memory available to enable headers and footers.
<b>0D01</b>	Unable to allocate enough memory.
<b>0D04</b>	Unable to allocate enough memory to expand internal data lists.
<b>0F00</b>	Unable to allocate enough memory to expand the list of line parameters.
<b>1002</b>	Unable to allocate enough memory.
<b>1003</b>	Unable to expand the line list buffer.
<b>1004</b>	Unable to allocate enough memory.
<b>1200</b>	Unexpected error, formatting process could not be performed.
<b>1300</b>	Unable to allocate enough local memory.
<b>1401</b>	Unexpected error, invalid local handles.
<b>1406</b>	Unable to allocate enough memory to get data from the clipboard.
<b>1409</b>	Unable to allocate enough local memory.
<b>1A00</b>	Cannot allocate enough memory to switch on word-division.
<b>1B02</b>	TX_IMPORTTEXT* messages: Unable to convert old format, not enough memory.
<b>1C00</b>	Unable to allocate enough memory to insert a large amount of text.
<b>1C01</b>	Unable to allocate enough memory to convert an ASCII string into the internal text format.
<b>1C02</b>	Unable to allocate enough memory to save the remainder of the text over 64 kB.
<b>2002</b>	Unable to allocate enough memory to insert format data.
<b>2003</b>	Unable to allocate enough memory.
<b>2005</b>	Unable to allocate enough memory to convert a multibyte character string to Unicode.
<b>2010</b>	Unable to allocate enough memory to insert format information.
<b>2300</b>	Unable to adjust the character spacing buffer. This can happen if the text is not formatted correctly.
<b>2303</b>	Unable to allocate enough local memory.
<b>2700</b>	Unable to allocate enough local memory.
<b>2705</b>	Unable to allocate enough memory.
<b>2706</b>	Unable to allocate enough memory.
<b>2707</b>	Unable to allocate enough memory.

<b>2708</b>	Unable to allocate enough memory.
<b>270A</b>	Unable to allocate enough memory.
<b>270B</b>	Unable to allocate enough memory.
<b>270C</b>	Unable to allocate enough memory.
<b>270D</b>	Unable to allocate enough memory.
<b>2710</b>	Unable to allocate enough memory.
<b>2711</b>	Unable to allocate enough memory.
<b>2712</b>	Unable to allocate enough memory.
<b>2713</b>	Unable to allocate enough memory.
<b>2715</b>	Unable to allocate enough memory.
<b>2717</b>	Unable to allocate enough memory.
<b>271A</b>	Cannot allocate enough memory to perform table operation.
<b>271C</b>	Cannot allocate enough memory to perform table operation.
<b>2A00</b>	Unable to allocate enough memory to copy TX data.
<b>2A01</b>	Unable to allocate enough memory to copy TX data.
<b>2A05</b>	Unable to allocate enough memory.
<b>2D05</b>	Unable to allocate enough memory to expand the object reference list.
<b>2F00</b>	Not enough memory available to perform search operation.
<b>2F01</b>	Not enough memory available to perform replace operation.
<b>3100</b>	Unable to allocate enough memory to add a new table.
<b>3200</b>	Cannot allocate enough local memory to insert a new table.
<b>3302</b>	Unable to allocate enough memory to read data from the specified text file.
<b>3307</b>	Cannot insert data formatted in the Text Control's text format with the TX_DATAIN message. Cannot allocate enough memory to copy the data.

### DBS\_E\_NOMEMORYACCESS

<b>Code</b>	<b>Description</b>
<b>0203</b>	Current text not available, <b>GlobalLock</b> failure.
<b>0205</b>	Current text not available, <b>GlobalLock</b> failure.
<b>0302</b>	Field data list not available, <b>GlobalLock</b> failure.
<b>0306</b>	Unexpected error. Field data list not available, <b>GlobalLock</b> failure.
<b>030A</b>	Unexpected error. Field data list not available, <b>GlobalLock</b> failure.

---

<b>0402</b>	Allocated memory block not available, <b>GlobalLock</b> failure.
<b>0404</b>	Allocated memory block not available, <b>GlobalLock</b> failure.
<b>040B</b>	Allocated memory block not available, <b>GlobalLock</b> failure.
<b>0500</b>	Font information not available, <b>GlobalLock</b> failure.
<b>0501</b>	Unexpected error, invalid local handle.
<b>0506</b>	Invalid image list.
<b>0B01</b>	Unexpected text format data not available, <b>GlobalLock</b> failure.
<b>0B08</b>	Current text not available, <b>GlobalLock</b> failure.
<b>0B0A</b>	Memory block not available, <b>GlobalLock</b> failure.
<b>0D02</b>	Allocated memory block not available, <b>GlobalLock</b> failure.
<b>0D03</b>	Unexpected data lists not available, <b>GlobalLock</b> failure.
<b>0D05</b>	Unexpected data lists not available, <b>GlobalLock</b> failure.
<b>0D08</b>	Unexpected data lists not available, <b>GlobalLock</b> failure.
<b>0F01</b>	Line list data not available. <b>GlobalLock</b> failure.
<b>0F02</b>	Line list data not available. <b>GlobalLock</b> failure.
<b>1000</b>	Unexpected display error, data not available.
<b>1001</b>	Unexpected display error, data not available.
<b>1007</b>	Unexpected display error, data not available.
<b>1008</b>	Unexpected display error, data not available.
<b>1009</b>	Unexpected display error, text data not available.
<b>1402</b>	New text not available, <b>GlobalLock</b> failure.
<b>2000</b>	New text to insert not available, <b>GlobalLock</b> failure.
<b>2007</b>	Unable to set format information, <b>GlobalLock</b> failure.
<b>2011</b>	Format information not available. <b>GlobalLock</b> failure.
<b>2301</b>	Character spacing information not available. <b>GlobalLock</b> failure.
<b>2714</b>	Unexpected error, data buffers not available.
<b>2716</b>	Unexpected error, data buffers not available.
<b>2718</b>	Unexpected error, data buffers not available.
<b>2719</b>	Unexpected error, table cell list not available.
<b>271B</b>	Unexpected error, table cell list not available.
<b>271D</b>	Unexpected error, table cell list not available.
<b>2C00</b>	Unexpected error, global object list not available.

<b>2C0D</b>	Unexpected error, global object list not available.
<b>3101</b>	Unexpected error, table cell list not available.
<b>3303</b>	Cannot read from a specified text file.

## **DBS\_E\_UNEXPECTED**

<b>Code</b>	<b>Description</b>
<b>0201</b>	Unexpected error, invalid text handle.
<b>0303</b>	Unexpected error, field identifier mismatch.
<b>0305</b>	Unexpected error, field data list not available.
<b>0307</b>	Unexpected error, field identifier not found.
<b>0309</b>	Unexpected error, field data list not available.
<b>030B</b>	Unexpected error, field identifier not found.
<b>0409</b>	Unexpected error, invalid object count.
<b>0504</b>	Unable to get data from the font manager.
<b>0505</b>	Unable to create an inserted font.
<b>0700</b>	Unable to get data from the font manager during the TX_ENLARGEFONT message.
<b>0702</b>	Unable to get data from the font manager during the TX_SETFONTATTR message.
<b>0704</b>	Unable to get data from the font manager.
<b>0706</b>	Unable to initialize the font manager.
<b>0707</b>	Unable to get data from the font manager during the TX_GETFONTATTR message.
<b>0708</b>	Unable to lock local fontlist.
<b>0709</b>	Global fontlist not available.
<b>0800</b>	Unable to create a new Image-Control window.
<b>0A02</b>	Unable to create a new TX window, <b>CreateWindow</b> failed.
<b>0A04</b>	Invalid formatting device.
<b>0A05</b>	Unexpected error, invalid count of fixed positioned objects.
<b>0B02</b>	Unexpected error, invalid local handle.
<b>0B03</b>	Unexpected error, unable to get font data.
<b>0B04</b>	Unexpected error, invalid local handle.

---

<b>0B05</b>	Unexpected error, unable to get paragraph data.
<b>0B07</b>	Unexpected error, invalid text description.
<b>0B0B</b>	Unexpected error, table list not available.
<b>0B0C</b>	Unexpected error, table attribute list not available.
<b>0B11</b>	Unexpected error, invalid handle.
<b>0D00</b>	Unexpected error, invalid pointer.
<b>0D06</b>	Unexpected error, unknown object.
<b>0D07</b>	Unexpected error, invalid global handle.
<b>1005</b>	General internal error, data lost.
<b>1202</b>	Unexpected error, paragraph data lost.
<b>1205</b>	Unexpected error, paragraph data lost.
<b>1900</b>	Unable to copy text between linked windows.
<b>1F00</b>	Unable to get text that should be copied to the following window in a chain of linked windows.
<b>1F01</b>	Unable to test whether text needs to be copied to the previous window in a chain of linked windows.
<b>1F02</b>	Unexpected error while text was being copied to a linked window.
<b>1F03</b>	Text cannot be copied to the previous window in a chain of linked windows.
<b>1F05</b>	Unexpected error, unable to perform the text operation.
<b>1F06</b>	Unexpected error, unable to perform the text operation.
<b>1F07</b>	Unexpected error, unable to perform the text operation.
<b>1F08</b>	Unexpected error, unable to perform the text operation.
<b>2008</b>	Unexpected error, invalid local handle.
<b>200A</b>	Unexpected error, invalid local handle.
<b>2302</b>	Unexpected error, paragraph data not available.
<b>2701</b>	Unexpected error, invalid text format.
<b>2702</b>	Unexpected data error.
<b>2703</b>	Unexpected error, invalid text format.
<b>2704</b>	Unexpected error, data buffers not available.
<b>2709</b>	Unexpected error, data buffers not available.
<b>270E</b>	Unexpected error, data buffers not available.
<b>270F</b>	Unexpected error, data buffers not available.
<b>2900</b>	Error during undo operation: Invalid position value.

<b>2901</b>	Error during undo operation: Invalid undo buffer.
<b>2902</b>	Error during undo operation: Invalid undo buffer.
<b>2903</b>	Error during undo operation: Format could not be restored.
<b>2B01</b>	Invalid object identifier. The object cannot be deleted.
<b>2C0A</b>	Object cannot be created. Identifier mismatch.
<b>2E00</b>	Unexpected error, global object list not available.
<b>3102</b>	Unexpected error, table list or table attribute list not available.

### ***Error Code Table (OLE module)***

The following error codes belong to module number 5 which is the module handling the insertion of OLE objects. These codes are only possible in the 32 bit version.

<b>Group Code</b>	<b>Code</b>	<b>Description</b>
<b>DBS_E_INVALIDARG</b>	<b>0200</b>	Cannot find the specified programmatic identifier in the registration database.
<b>DBS_E_UNEXPECTED</b>	<b>0201</b>	Cannot insert the specified OLE object.
<b>DBS_E_CLIPBOARD</b>	<b>0202</b>	Cannot paste an OLE object from the clipboard.
<b>DBS_E_INVALIDARG</b>	<b>0203</b>	OLE object cannot be inserted, the specified filename may be invalid.

# Appendix C

## Development of a Text Filter

This appendix describes how to develop installable filters that allow word-processed files in various formats to be placed into a Text Control and, on the reverse side, text created with a Text Control to be exported in an external format. The interface is given through the messages TX\_DATAIN, TX\_DATAOUT, TX\_IMPORTTEXTFILE, TX\_IMPORTTEXTBUFFER and TX\_EXPORTTEXT.

The developer of a filter is responsible for implementing the C-language routines that Text Control calls when one of the messages, listed above, are used by an application. The filter must be built as a Windows library which Text Control can load dynamically with the **LoadLibrary** function.

The filter library must export the following functions:

TX_Import	@1
TX_Export	@2
WEP	@3 (16 bit only)
TX_Ex_Import	@5 (optional)
TX_Ex_Export	@6 (optional)
GetFilterInfo	@7 (16 bit: recommended, 32 bit: required)

The **TX\_Import** and **TX\_Ex\_Import** functions provide the data translation necessary to describe a foreign format in terms of the Text Control's internal data structures. Appendix A contains a description of these internal structures. The functions are described later in this appendix. **TX\_Import** generates the generic text and format structures Text Control uses internally. **TX\_Ex\_Import** generates these same structures and additionally data like document settings that can be exchange with an application via the TX\_DATAIN message. When **TX\_Ex\_Import** cannot be found Text Control calls **TX\_Import**.

The **TX\_Export** and **TX\_Ex\_Export** functions provide the mechanism by which data is transformed from the internal Text Control text format into a foreign format. In addition to the generic text formats **TX\_Ex\_Export** receives document settings

the filter can save in the document. When **TX\_Ex\_Export** cannot be found Text Control calls **TX\_Export**.

The **WEP** function is required for every Windows 16 bit library. For a description of the **WEP** function and how to build a Windows library see the Windows Programmer's Reference.

The following shows the definitions of the functions which must be provided by the filter.

---

## GetFilterInfo

### Syntax

**void GetFilterInfo**(*lpFilterInfo*)

This function returns information about the capabilities of the text filter. Text Control calls this function before calling any other function in the filter.

<u>Parameter</u>	<u>Type/Discription</u>
<i>lpFilterInfo</i>	<b>LPTXFILTERINFO</b> Points to a TXFILTERINFO structure which is defined as follows:

```
typedef struct tagTXFILTERINFO {
    DWORD      dwTXFilterInfoSize;
    DWORD      dwMinFileFormat;
    DWORD      dwMaxFileFormat;
    BOOL       bIsUnicodeFilter;
    BYTE       reserved[8];
    char       reserved1[16];
    char       reserved2[16];
} TXFILTERINFO;
```

The TXFILTERINFO structure has the following fields:

<u>Field</u>	<u>Description</u>
<i>dwTXFilterInfoSize</i>	Specifies the size, in bytes, of this data structure. Text Control initializes this member before calling the function. The filter must fill this member before the function returns.

<i>dwMinFileFormat</i>	The filter must fill this member with the version number of the oldest Text Control text format it can process.
<i>dwMaxFileFormat</i>	The filter must fill this value with the version number of the newest Text Control text format it can process.
<i>bIsUnicodeFilter</i>	The filter must set this member to TRUE when it can process Unicode strings, otherwise it must set this value to FALSE. The 32 bit version of Text Control only uses filters that can process Unicode strings.
<i>reserved[8]</i>	Reserved field. Must be initialized with zero.
<i>reserved1[16]</i>	Reserved field. Must be initialized with zero.
<i>reserved2[16]</i>	Reserved field. Must be initialized with zero.

**Comments** This function must be declared with the WINAPI declarator. It must be exported in the EXPORT section of the filter's module definition file with the ordinal number @7.

---

## TX\_Import

**Syntax** **WORD WINAPI TX\_Import**(*hInData*, *hFile*, *lphOutData*)

This function transforms data given in an external format into the Text Control's internal format. The data is either given through a buffer handle or through a file handle.

**Parameter**

*hInData*

**Type/Discription**

**HGLOBAL** Specifies a global data handle that identifies a buffer containing the data in an external format. *hInData* contains zero if the data is given through the file handle.

<i>hFile</i>	<b>HFILE</b> Identifies a file. The data to be transformed is read from that file at the current file position.
<i>lphOutData</i>	<b>HGLOBAL FAR*</b> Points to a variable of type HGLOBAL. The filter must allocate a global buffer for the translated data and has to copy the handle that identifies the buffer to the variable <i>lphOutData</i> points to.

**Return Value** The return value specifies an error code value. The following error codes are possible:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
FE_NO_ERROR	No error.
FE_NOT_MY_FILE	Wrong file type.
FE_BAD_TOKEN	Logical error in input file.
FE_NO_MEM	Unable to allocate enough global memory.
FE_READ_DATA	File read error.
FE_WRITE_DATA	File write error.
FE_OPEN_FILE	Unable to open file.
FE_FILE_TOO_BIG	Input file is too big.
FE_UNSUPP	Currently not supported.
FE_INTERNAL	Unspecified internal error.

**Comments** This function must be declared with the WINAPI declarator. It must be exported in the EXPORT section of the filter's module definition file with the ordinal number @1.

The buffer allocated for the translated data is freed by Text Control.

The Text Control's internal format is described in appendix A. The header file "tx.h" should be included to have definitions for all the necessary structures and error codes.

---

## TX\_Ex\_Import

### Syntax

**WORD WINAPI TX\_Ex\_Import**(*hInData*, *hFile*, *lpInBuffer*, *lphOutData*, *lpFilterIO*, *lpdwBytesRead*)

This function is an extended version of **TX\_Import**. It transforms data given in an external format into the Text Control's internal format with the data is either given through a buffer handle, a buffer pointer or a file handle. Additionally data can be exchanged defined through the TXFILTERIO structure.

#### Parameter

#### Type/Discription

*hInData*

**HGLOBAL** Specifies a global data handle that identifies a buffer containing the data in an external format. *hInData* contains zero if the data is given through *hFile* or *lpInBuffer*.

*hFile*

**HFILE** Identifies a file. The data to be transformed is read from that file at the current file position. *hFile* contains HFILE\_ERROR when the data is given through *hInData* or *lpInBuffer*.

*lpInBuffer*

**const BYTE \_huge\*** Points to a memory buffer containing the data in an external format. *lpInBuffer* is zero when the data is given through *hInData* or *hFile*.

*lphOutData*

**HGLOBAL FAR\*** Points to a variable of type HGLOBAL. The filter must allocate a global buffer for the translated data and has to copy the handle that identifies the buffer to the variable *lphOutData* points to.

<i>lpFilterIO</i>	<b>TXFILTERIO FAR*</b> points to a TXFILTERIO structure to exchange additional data. The structure members contain default values on function entry the filter can use to resolve relative values. On function exit the filter fills the structure members with information not available with the Text Control's text format. See appendix A for a detailed description of this structure.
<i>lpdwBytesRead</i>	<b>DWORD FAR*</b> The filter must copy the number of read bytes to the variable this parameter points to.

**Return Value** The return value specifies an error code value. For a list of possible error codes see the description of the **TX\_Import** function.

**Comments** This function must be declared with the WINAPI declarator. It must be exported in the EXPORT section of the filter's module definition file with the ordinal number @5.

The buffer allocated for the translated data is freed by Text Control.

---

## TX\_Export

**Syntax** **WORD WINAPI TX\_Export**(*hInData*, *hFile*, *lphOutData*)

This function transforms data given in the Text Control's internal format into an external format.

<u>Parameter</u>	<u>Type/Discription</u>
<i>hInData</i>	<b>HGLOBAL</b> Specifies a global data handle that identifies a buffer containing the data in the Text Control's internal format.
<i>hFile</i>	<b>HFILE</b> Identifies a file. The transformed data is written to that file at the current file position. If <i>hFile</i> contains <b>HFILE_ERROR</b> , the function has to allocate a global buffer for the transformed data.

*lphOutData*                    **HGLOBAL FAR\***    Points to a variable of type HGLOBAL. If *hFile* contains HFILE\_ERROR, the filter must allocate a global buffer for the transformed data and has to copy the handle that identifies the buffer to the variable pointed to by *lphOutData*.

**Return Value**                The return value specifies an error code value. For a list of possible error codes see the description of the **TX\_Import** function.

**Comments**                    This function must be declared with the WINAPI declarator. It must be exported in the EXPORT section of the filter's module definition file with the ordinal number @2.

The Text Control's internal format is described in appendix A. The header file "tx.h" should be included to have definitions for all the necessary structures and error codes.

---

## TX\_Ex\_Export

**Syntax**                        **WORD WINAPI TX\_Export**(*hInData*, *hFile*, *lphOutData*, *lpFilterIO*, *lpdwBytesWritten*)

This function is an extended version of **TX\_Export**. It transforms data given in the Text Control's internal format into an external format.

<u>Parameter</u>	<u>Type/Discription</u>
<i>hInData</i>	<b>HGLOBAL</b> Specifies a global data handle that identifies a buffer containing the data in the Text Control's internal format.
<i>hFile</i>	<b>HFILE</b> Identifies a file. The transformed data is written to that file at the current file position. If <i>hFile</i> contains HFILE_ERROR, the function has to allocate a global buffer for the transformed data.

<i>lphOutData</i>	<b>HGLOBAL FAR*</b> Points to a variable of type HGLOBAL. If <i>hFile</i> contains HFILE_ERROR, the filter must allocate a global buffer for the transformed data and has to copy the handle that identifies the buffer to the variable pointed to by <i>lphOutData</i> .
<i>lpFilterIO</i>	<b>TXFILTERIO FAR*</b> points to a TXFILTERIO structure to exchange additional data. The filter can save this additional data in the document. See appendix A for a detailed description of this structure.
<i>lpdwBytesWritten</i>	<b>DWORD FAR*</b> The filter must copy the number of written bytes to the variable this parameter points to.

***Return Value*** The return value specifies an error code value. For a list of possible error codes see the description of the **TX\_Import** function.

***Comments*** This function must be declared with the WINAPI declarator. It must be exported in the EXPORT section of the filter's module definition file with the ordinal number @6.

The Text Control's internal format is described in appendix A. The header file "tx.h" should be included to have definitions for all the necessary structures and error codes.

# Appendix D

## Status Bar Control

A status bar has been designed as a separate module that handles a Status-Bar-Control window. This window can be created as a child window anywhere within the client area of its parent window. It shows the current state of a Text Control window or an information string.

The parent window must pass on the following Text Control notification messages to the Status-Bar-Control window in order to show the current state:

TN_FORCEUPDATE	Sent when the status bar should update its contents.
TN_KEYSTATECHANGED	Sent when the character insertion mode, or the state of either the NUMLOCK key or CAPSLOCK key is changed.
TN_KILLFOCUS	Sent when a Text Control loses the input focus.
TN_POSCHANGED	Sent when the current input position changes.
TN_SETFOCUS	Sent when a Text Control receives the input focus.
TN_ZOOMED	Sent when a Text Control is zoomed.

The Status-Bar-Control uses the window handle sent with the notification to ask the TX window about its current state.

A Status-Bar-Control has two display modes. The first mode shows three areas that display page number, line number and character position. The second mode shows an information string. The Status-Bar-Control switches to the first mode if it receives a TX notification message. It switches to the second mode if it receives a WM\_SETTEXT message, containing the information text. The application can

send a WM\_SETTEXT message whenever it wants the Status-Bar-Control to show an information string.

The messages WM\_GETFONT and WM\_SETFONT can be used to alter the font which the Status-Bar-Control uses to show its contents. All WM\_xxx messages are described in the Windows SDK manuals.

The following describes functions and messages an application can use to create a Status-Bar-Control window and to alter its appearance:

## Functions

---

### CreateStatusBar

```
#include "ttools.h"
```

**Syntax**      **HWND CreateStatusBar**(*hWndParent*, *wChildID*, *lpRect*, *lpLogFont*, *lpszStrings*, *dwStyle*)

This function creates a Status-Bar-Control child window.

<u>Parameter</u>	<u>Type/Description</u>
<i>hWndParent</i>	<b>HWND</b> Identifies the parent window of the Status-Bar-Control window.
<i>wChildID</i>	<b>WORD</b> Is the child window identifier.
<i>lpRect</i>	<b>LPRECT</b> Points to a <b>RECT</b> data structure which contains the position and size of the Status-Bar-Control window in client area coordinates of its parent window.
<i>lpLogFont</i>	<b>LPLOGFONT</b> Points to a <b>LOGFONT</b> data structure that defines a logical font. The Status-Bar-Control window will use this font to display its contents. If <i>lpLogFont</i> is NULL, a default font is used.
<i>lpszStrings</i>	<b>LPCSTR</b> Points to a buffer containing the strings, that the Status-Bar-Control will use to display page number, line number and character position. The string must have

three parts, each terminated with a zero character. Each part must contain a format description in the form specified for the **wprintf()** function as described in the Windows SDK. The complete string must end with two terminating zeros. See the following comments section for more information.

If *lpzStrings* is NULL, the Status-Bar-Control displays only the numbers.

*dwStyle*

**DWORD** Specifies the style of the Status-Bar-Control window. This parameter can be one of the following values:

<b><u>Style</u></b>	<b><u>Meaning</u></b>
STS_LEFTALIGN	Positions the text output areas on the left side of the client area.
STS_RIGHTALIGN	Positions the text output areas on the right side of the client area.
STS_NOBORDER	Suppresses the Status-Bar-Control window's border.
STS_NOPAGE	Suppresses the page number.
STS_NOLINE	Suppresses the line number.
STS_NOCOLUMN	Suppresses the column number.
STS_NOZOOM	Suppresses the zooming factor.
STS_NOKEYSTATES	Suppresses the insertion mode and the CAPSLOCK and NUMLOCK keystates.

**Return Value** The return value identifies the Status-Bar-Control window. It is zero if an error has occurred.

**Comments**

To use this function the header file TXTOOLS.H must be included into the application's source file and the object files have to be linked with TXTOOLS.LIB.

The string which *lpszStrings* points to must have the following form:

```
[ptext]%format\0[ltext]%format\0[ctext]%format\0\0
```

The parts in brackets are optional, all other parts are required. The various parts have the following meanings:

<b><u>Field</u></b>	<b><u>Meaning</u></b>
<i>ptext</i>	Text for the Status-Bar-Control area that displays the page number.
<i>ltext</i>	Text for the Status-Bar-Control area that displays the line number.
<i>ctext</i>	Text for the Status-Bar-Control area that displays the character position.
<i>%format</i>	A Format string in the same form as used for the <b>wsprintf()</b> function. The Status-Bar-Control uses that format to display the number.
\0	Terminating zero characters.

---

## Messages

---

---

### STB\_GETLANGUAGE

This message returns the language that the Status Bar Control uses to display its status strings. It works in the same way as the TX\_GETLANGUAGE message.

---

---

### STB\_SETLANGUAGE

This message sets the language that the Status Bar Control uses to display its status strings. It works in the same way as the TX\_SETLANGUAGE message.

---

---

### STB\_SETSTRINGS

This message changes the strings that the Status-Bar-Control uses to display page number, line number and character position.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Specifies whether the Status-Bar-Control is to be redrawn. A value of TRUE redraws the Status-Bar-Control.
<i>lParam</i>	Points to a buffer containing new the strings that the Status-Bar-Control will use to display page number, line number and character position. For an explanation of possible formats, see the description of the <b>CreateStatusBar</b> function.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**STB\_SETSTYLE**

This message changes the style of a Status-Bar-Control.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies whether the Status-Bar-Control is to be redrawn. A value of TRUE redraws the Status-Bar-Control.
<i>lParam</i>	Specifies the new Status-Bar-Control style. For an explanation of possible styles, see the description of the <b>CreateStatusBar</b> function.

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

# Appendix E

## Button Bar Control

A button bar has been designed as a separate window that is called from here on a Button-Bar-Control window. This window can be created as a child window anywhere within the client area of its parent window. It shows the current paragraph and font settings of the Text Control that has the input focus.

An application must pass on the following notification messages of a Text Control to the Button-Bar-Control window to update its contents:

TN_CHARFORMATCHANGED	Sent when the characters covered by the current selection are newly formatted.
TN_FORCEUPDATE	Sent when the button bar should update its contents.
TN_KILLFOCUS	Sent when a Text Control loses the input focus.
TN_PGCHANGED	Sent when the current input position changes to another paragraph.
TN_PGFORMATCHANGED	Sent when the paragraphs covered by the current selection are newly formatted.
TN_POSCHANGED	Sent when the current input position changes.
TN_SETFOCUS	Sent when a Text Control gets the input focus.

The button bar saves the Text Control window handle passed on with the TN\_SETFOCUS message and sends font and paragraph messages to that window when the user clicks an element of the button bar. To disable a button bar, for example when no Text Control exists, the application can send the WM\_ENABLE message with the *wParam* parameter set to FALSE. The next TN\_SETFOCUS notification re-enables the button bar automatically.

During creation the application can use the **BUTTONINFO** data structure to alter the buttons' default appearing order or to add additional custom buttons.

The following describes the function to create a Button-Bar-Control window:

## Functions

---

### CreateButtonBarControl

```
#include "txttools.h"
```

**Syntax**      **HWND CreateButtonBarControl**(*hWndParent*, *wChildID*, *lpRect*, *dwStyle*, *wButtons*, *lpButtonArray*)

This function creates a Button-Bar-Control child window.

<u>Parameter</u>	<u>Type/Description</u>
<i>hWndParent</i>	<b>HWND</b> Identifies the parent window of the Button-Bar-Control window.
<i>wChildID</i>	<b>UINT</b> Specifies the child window identifier.
<i>lpRect</i>	<b>LPRECT</b> Points to a <b>RECT</b> data structure which contains the position and the width of the Button-Bar-Control window in client area coordinates of its parent window.
<i>dwStyle</i>	<b>DWORD</b> Specifies the style of the Button-Bar-Control window. See the description of the <b>BBM_SETSTYLE</b> message for a list of possible values.
<i>wButtons</i>	<b>WORD</b> Specifies the number of buttons that the Button-Bar-Control will display. This parameter can be zero if the <i>lpButtonArray</i> parameter is not used.
<i>lpButtonArray</i>	<b>BUTTONINFO FAR*</b> Points to an array of <b>BUTTON-INFO</b> structures that contain information about styles and command IDs for all buttons. The number of structures this array contains is specified through the <i>wButtons</i> parameter. The buttons appear visually on the screen in the same order they appear in this array. For more information about the <b>BUTTONINFO</b> structure, see the following

comments section. If *lpButtonArray* is zero the button bar uses default buttons for font and paragraph settings in a default order.

**Return Value** The return value identifies the Button-Bar-Control window. It is zero if an error has occurred.

**Comments** To use this function the header file TXTOOLS.H must be included into the application's source file and the object files have to be linked with TXTOOLS.LIB.

The window's height is adapted to the buttons which the button bar contains. To obtain the height of the window the application should use the **GetWindowRect** function.

The **BUTTONINFO** data structure has the following form:

```
typedef struct tagBUTTONINFO {
    UINT  nIDResource;
    UINT  nIDCommand;
    UINT  nStyle;
    int   nPixOffset;
} BUTTONINFO;
```

The **BUTTONINFO** structure fields have the following meaning:

<b><u>Field</u></b>	<b><u>Description:</u></b>
<i>nIDResource</i>	Specifies either a resource identifier of a bitmap or a predefined button identifier. If the value is greater than <b>BBID_MAXBUTTON</b> it is interpreted as a bitmap resource identifier. The button bar loads that bitmap from the instance of the module which belongs to the specified parent window and creates a custom button with that bitmap. The bitmap must be of the standard size, i.e. 16 pixels wide and 15 pixels high. Otherwise <i>nIDResource</i> can be one of the following predefined button identifiers:

<b><u>Value</u></b>	<b><u>Meaning:</u></b>
<b>BBID_FONTS</b>	Specifies a Combo-Box to select a font family.

	BBID_POINTSIZE	Specifies a Combo-Box to select a pointsize.
	BBID_BOLD	Specifies a button for bold font style.
	BBID_ITALIC	Specifies a button for italic font style.
	BBID_UNDERLINE	Specifies a button for underlined fonts.
	BBID_PGLEFT	Specifies a button for left paragraph alignment.
	BBID_PGRIGHT	Specifies a button for right paragraph alignment.
	BBID_PGCENTERED	Specifies a button for centered paragraphs.
	BBID_PGJUSTIFIED	Specifies a button for justified paragraphs.
	BBID_LEFTTAB	Specifies a button for left aligned tabulators.
	BBID_RIGHTTAB	Specifies a button for right aligned tabulators.
	BBID_CENTEREDTAB	Specifies a button for centered tabulators.
	BBID_DECIMALTAB	Specifies a button for decimal tabulators.
<i>nIDCommand</i>		Specifies a command identifier for a user defined button to associate it with a menu identifier. The button bar sends a WM_COMMAND message to its parent window if a user clicks the button. This field must be zero for predefined buttons. The WM_COMMAND message has the same form as messages sent from a menu.
<i>nStyle</i>		Specifies a button style for a user defined button. It can be one of the following values:

	<u>Value</u>	<u>Meaning:</u>
	BBBS_BUTTON	Specifies a standard push-button.
	BBBS_CHECKBOX	Specifies a check-box button.
	This value is ignored for predefined buttons.	
<i>nPixOffset</i>	Specifies in pixels, an offset for the button's horizontal position. This offset is the distance between the left border and the right border of the previous button. This value can be used to group buttons of similar features.	

## Messages

---

### BBM\_GETLANGUAGE

This message returns the language that the Button Bar Control uses to display resources. It works in the same way as the TX\_GETLANGUAGE message.

---

### BBM\_SETLANGUAGE

This message sets the language that the Button Bar Control uses to display resources. It works in the same way as the TX\_SETLANGUAGE message.

---

### BBM\_GETSTYLE

This message returns a Button Bar Control's styles.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value specifies the styles. See the description of the `BBM_SETSTYLE` message for a list of possible values.

---

## BBM\_SETSTYLE

This message sets new styles for a Button Bar Control.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Specifies the new styles. It can be a combination of the following values:

<b><u>Value</u></b>	<b><u>Meaning</u></b>
BBS_3D	Paints the Button Bar Control with three-dimensional effects.
BBS_FLAT	Paints the Button Bar Control without visual effects.
BBS_3DBUTTONS	Paints the Button Bar Control's buttons with three-dimensional effects.
BBS_FLATBUTTONS	Paints the Button Bar Control's buttons without visual effects.
BBS_DBLCLKS	The Button Bar Control opens the Text Control's built-in font and paragraph dialog boxes if the user doubleclicks in an area that does not represent a button.
BBS_NODBLCLKS	The Button Bar Control does not open the Text Control's built-in dialog box.
BBS_BORDER	Paints the Button Bar Control with a border.

---

BBS_NOBORDER	Suppresses the Button Bar Control window's border.
--------------	--

***Return Value***     The return value specifies the previous styles.

***Comments***         The Button Bar Control is refreshed when new styles affect the visual appearance.

The style values are grouped. The following values must not be used together:

BBS\_3D and BBS\_FLAT

BBS\_3DBUTTONS and BBS\_FLATBUTTONS

BBS\_DBLCLKS and BBS\_NODBLCLKS

BBS\_BORDER and BBS\_NOBORDER

The default values are BBS\_FLAT, BBS\_3DBUTTONS, BBS\_DBLCLKS and BBS\_BORDER.

# Appendix F

## Ruler Control

A ruler has been designed as a separate window that is called from here on a Ruler-Control window. This window can be created as a child window anywhere within the client area of its parent window. It shows a ruler with the current tabulator and indent settings for the Text Control window which has the input focus.

An application must pass on the following Text Control notification messages to the Ruler-Control window to update its contents:

TN_FORCEUPDATE	Sent when the ruler should update its contents.
TN_HMOVED	Sent after a Text Control has been moved horizontally.
TN_HSCROLL	Sent when the Text Control's client area is scrolled horizontally.
TN_PAGEFORMATCHANGED	Sent when current page format has been changed.
TN_PGCHANGED	Sent when the current input position changes to another paragraph.
TN_PGFORMATCHANGED	Sent when the paragraphs covered by the current selection are newly formatted.
TN_KILLFOCUS	Sent when the Text Control loses the input focus.
TN_SETFOCUS	Sent when a Text Control receives the input focus.
TN_ZOOMED	Sent when the Text Control is zoomed.

The Ruler-Control saves the Text Control window handle passed on with the TN\_SETFOCUS message and sends paragraph messages to that window when the user alters the tabulator or indent settings. To disable a Ruler-Control, for example when no Text Control exists, the application can send the WM\_ENABLE message with the *wParam* parameter set to FALSE. The next TN\_SETFOCUS notification

re-enables the Ruler-Control automatically. A disabled Ruler-Control shows only the ruler without any tabulator or indent marks.

The functions and messages which an application can use to create a Ruler-Control window or to change its appearance are described as follows:

## Functions

---

### CreateRulerControl

```
#include "txtools.h"
```

**Syntax**      **HWND CreateRulerControl**(*hWndParent*, *wChildID*, *lpRect*, *hWndButtonBar*, *dwStyle*)

This function creates a Ruler-Control child window.

<u>Parameter</u>	<u>Type/Description</u>
<i>hWndParent</i>	<b>HWND</b> Identifies the parent window of the Ruler-Control window.
<i>wChildID</i>	<b>UINT</b> Specifies the child window identifier.
<i>lpRect</i>	<b>LPRECT</b> Points to the <b>RECT</b> data structure which contains the position and width of the Ruler-Control window in the client area coordinates of its parent window.
<i>hWndButtonBar</i>	<b>HWND</b> Identifies a Button-Bar-Control window. The Ruler-Control uses the tabulator style setting of this button bar to select the tabulator style of a newly created tabulator. If <i>hWndButtonBar</i> is zero all newly created tabulators are left aligned.
<i>dwStyle</i>	<b>DWORD</b> Specifies the style of the Ruler-Control window. This parameter can be a combination of the following values:

<u>Style</u>	<u>Meaning</u>
RS_TABULATORS	The Ruler-Control shows tabulator settings.
RS_LEFTINDENT	The Ruler-Control shows a left indent mark.
RS_FIRSTINDENT	The Ruler-Control shows a mark for the additional indent of the first line.
RS_RIGHTINDENT	The Ruler-Control shows a right indent mark.
RS_INDENTS	The Ruler-Control shows marks for all indents.
RS_POSITION	The Ruler-Control shows the current position during moving a tabulator or an indent mark.
RS_NOBORDER	Suppresses the Ruler-Control window's border.

If *dwStyle* is zero the Ruler-Control contains all elements.

***Return Value*** The return value identifies the Ruler-Control window. It is zero if an error has occurred.

***Comments*** To use this function the header file TXTOOLS.H must be included into the application's source file and the object files have to be linked with TXTOOLS.LIB.  
The window's height is adapted to the necessary height for the ruler and the tabulator marks. To obtain the height of the window the application should use the **GetWindowRect** function.

---

## Messages

---

### RM\_SETBUTTONBAR

---

This message connects a new Button-Bar-Control with a Ruler-Control.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Identifies a Button-Bar-Control window. The Ruler-Control uses the tabulator style setting of the button bar to select the tabulator style of a newly created tabulator. If <i>wParam</i> is zero all newly created tabulators are left aligned.
<i>lParam</i>	Is not used.

**Return Value** The return value is zero if the value specified by *wParam* is not a valid window handle. Otherwise it is nonzero.

---

### RM\_SETUNITS

This message alters the scaling, the number of units and the number of scale lines.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Contains the scaling of the ruler as a percentage of a centimeter.
<i>lParam</i>	Contains the number of units in the low-order word and the number of scale lines in the high-order word. The number of units is the digit the ruler window will display at the first main scale line. The number of scale lines is the number of steps the ruler will display between two main scale lines.

**Return Value** The return value is zero if an error has occurred. Otherwise it is nonzero.

**Comments** The default setting is a scale with one centimeter steps subdivided into ten smaller steps.

**Example** To scale the ruler in inches *wParam* must contain 254 to indicate that one inch is 254 percent of one centimeter. The low-order word of *lParam* must contain 1 to display digits in the range of 0, 1, 2, 3, ...The high-order word of *lParam* can contain 4 or 8 to subdivide one inch into 4 or 8 parts.

# Appendix G

## Debugging Functions

Debugging functions make it easier for developers to integrate the Text Control library into an application. The only currently supported function tests a buffer that contains the text and the formatting information in the Text Control's internal format as described in appendix A. This is important to insert data with the TX\_PASTEDATA message or to develop a filter described in appendix C. For time optimization this function should only be called during development.

---

### DebugTXFormat

**Syntax**      **LONG** DebugTXFormat(*hFormat*, *lpdwOffset*)

This function checks a text format description according to appendix A. This function should be used for debugging purposes only.

**Parameter**

**Type/Discription**

*hFormat*

**HGLOBAL**    Identifies a global memory object that contains the format description.

*lpdwOffset*

**LPWORD**    Points to a DWORD variable. If an error has occurred the function copies an offset value to that variable specifying the location of the detected error. Otherwise if no error has occurred the value specifies the amount of tested data. If *lpdwOffset* is zero the function ignores it.

**Return Value**

The return value specifies an internal error code in the same form as returned by the **GetErrorCode** function. It is zero if no error has been detected.

## ***Error Code Table (Expansion for Debugging functions)***

<b>Code</b>	<b>Description</b>
<b>2400</b>	Discarded global handle.
<b>2401</b>	Invalid global handle, <b>GlobalLock</b> failed.
<b>2402</b>	Invalid buffer size. The buffer size is less than that necessary for an empty TextControl.
<b>2403</b>	Invalid buffer size. The buffer size is less than the specified text size.
<b>2404</b>	Text contains invalid control characters.
<b>2405</b>	Invalid text size. The real text size is not the same as the specified text size.
<b>2406</b>	The specified size of format data is less than the minimum size of format data.
<b>2407</b>	Invalid buffer size. The buffer size is less than the sum of the specified text and format data sizes.
<b>2408</b>	Invalid format data size. The specified format data size is not equal to the sum of the specified structures.
<b>2409</b>	Error in the list of FONTBLOCK structures: Invalid <i>wBlockStart</i> value.
<b>240A</b>	Error in the list of FONTBLOCK structures: Invalid <i>wBlockLength</i> value.
<b>240B</b>	Error in the list of FONTBLOCK structures: Invalid <i>wFntNum</i> value.
<b>240C</b>	Error in the list of FONTBLOCK structures: The <i>wBlockStart</i> value of the following structure is not equal to <i>wBlockStart + wBlockLength</i> .
<b>240D</b>	Error in the list of FONTBLOCK structures: Two following blocks of length zero.
<b>240E</b>	Error in the list of FONTBLOCK structures: Invalid <i>wBlockStart</i> value. Value is less than the <i>wBlockStart</i> value of the previous structure.

- 
- 240F** Error in the list of FONTBLOCK structures: The *wBlockLength* value of the last structure does not end at the end of the text.
- 2410** Error in the list of PGBLOCK structures: The count of PGBLOCK structures is not equal to the count of paragraphs in the text.
- 2411** Error in the list of PGBLOCK structures: Invalid *wPgStart* value.
- 2412** Error in the list of PGBLOCK structures: Invalid *wPgStop* value.
- 2413** Error in the list of PGBLOCK structures: Invalid *wPgNum* value.
- 2414** Error in the list of PGBLOCK structures: Invalid *wPgStop* value. The value does not specify the position of a paragraph end character in the text.
- 2415** Error in the list of PGBLOCK structures: Invalid *wPgStart* value. The value does not specify the position of the character that follows a paragraph end character.
- 2416** Error in the list of PGBLOCK structures: The *wPgStop* value of the last character does not specify the position of the terminating zero character.
- 2417** Error in the list of FONTBLOCK structures: Two following blocks have identical data.
- 2418** General error: Unable to allocate enough global memory to continue debugging.
- 2419** Error in list of CELLPOSITION structures: Invalid *wCellStart* value.
- 241A** Error in list of CELLPOSITION structures: Invalid *wCellStop* value.
- 241B** Error in list of CELLPOSITION structures: Invalid *wAttrRefNum* value.
- 241C** Error in list of CELLPOSITION structures: Invalid *wCellStop* value. The value does not specify the position of a paragraph end character.
- 241D** Error in list of CELLPOSITION structures: Invalid *wCellStart* value. The value does not follow the previous *wCellStop* value. *wCellStart* must be *wCellStop+1*.
- 241E** Error in list of CELLPOSITION structures: Invalid *lxPos* value. The value must be *lxPos+lxExt* of its left neighbour cell.

- 241F** Error in the list of FIELDADATA structures: The number of FIELDADATA structures is different from that of the *wFieldDataEntries* member.
- 2420** Error in the list of FIELDADATA structures: Invalid value in the *wField* member. The number specified here could not be found in the list of FONTBLOCKS.
- 2421** Error in the list of FIELDADATA structures: Only one of the members *dwData* and *dwDataSize* may contain a non-zero value.

# Appendix H

## Object Window Messages

---

### WMX\_COPYDATA

This message is used to copy the complete data of an object to the buffer pointed to by *lParam*. A Text Control sends this message to an object window during the processing of a TX\_COPYDATA message, a TX\_SAVE message or for undo operations.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to the buffer that is to receive the data.

**Return Value** The return value must be a pointer to the next free position behind the copied data. It must be set to zero if an error has occurred or if there is no data to copy.

**Comments** This message is not sent if the previously sent WMX\_GETDATASIZE message returns zero, meaning that there is no data to copy.

---

### WMX\_GETDATASIZE

This message is used to obtain the size of the buffer used for the WMX\_COPYDATA message. A Text Control sends this message to an object window during the processing of a TX\_COPYDATA message, a TX\_SAVE message or for undo operations.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value must contain the data size or zero if there is no data to copy.

**Comments** This message is sent more than once.

---

## WMX\_GETWINDOW

This message is used to obtain an object's window handle. A Text Control sends this message to its parent window to give the parent the opportunity of creating a window with the specified child identifier and returning it to the Text Control. A Text Control sends this message during an undo process or during a loading operation with the TX\_PASTEDATA or the TX\_LOAD message.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies a child window identifier. It is the identifier of the window connected to a Text Control via the TX_OBJ_EMBED message.
<i>lParam</i>	Specifies the object's identifier.

**Return Value** The return value should be the window handle of the object. The Text Control automatically becomes the parent window.

---

## WMX\_GETZOOM

This message is used to obtain the object's current zooming factor.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value specifies the object's zooming factor.

## WMX\_PASTEDATA

This message is used to reload the complete data of an object from a buffer pointed to by *lParam*. A Text Control sends this message to an object window during an undo process or during a loading operation using the TX\_PASTEDATA or the TX\_LOAD message.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a buffer that contains the data.

**Return Value** The return value must point to the buffer position behind the pasted data. It must be set to zero if an error has occurred.

---

## WMX\_PRINT

This message submits the contents of an object to the device identified by the *wParam* parameter. A Text Control sends this message to an object window during the processing of a TX\_PRINT or a TX\_PRINTPAGE message.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Contains a printer device context.
<i>lParam</i>	Points to the RECT data structure that contains the bounding rectangle of the part of the object to be printed. This rectangle is given in device pixels with an origin at the upper left corner of the object window's client area.

**Return Value** The return value must be set to zero if an error has occurred or if there is nothing to print. Otherwise it must be set to nonzero.

**WMX\_ZOOM**

This message informs the object about a new zooming factor. A Text Control sends this message to an object window during the processing of a TX\_ZOOM message.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains the new zooming factor as a percentage. This value is between 10 and 250.
<i>lParam</i>	Is not used.

***Return Value*** The return value must be zero if the window cannot be zoomed. Otherwise it must be nonzero.

***Comments*** The object must adjust its window size and position but to avoid refreshing errors it need not repaint itself. This is because it receives a WM\_PAINT message at a later date.

# IC Image Control

Reference Manual

# *IC Image Control 2.2*

Information in this document is subject to change without notice and does not represent a commitment on the part of DBS GmbH. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

©Copyright: DBS GmbH, 1991-98. All rights reserved.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

---

# *IC Image Control 2.20 Release Notes*

## **New features:**

The 32 bit version of Image Control has been extended to support Unicode, the character set for all languages. All messages and functions with string parameters have been implemented twice, one version for ANSI and one version for Unicode. See the new chapter 1.8 "*ANSI and Unicode*" for more information and a complete list of the extended messages and functions and how to use them. Unicode support is available on Windows NT and Windows 95/98.

## **Update Notes:**

The Image Control's file format has been extended to support Unicode. The following extensions have been made:

- The new format version number is 220.
- An additional array of extended data has been appended. See appendix B "*The IC Image Control File Format*" for more information about this array.

# ***IC Image Control 1.60 Release Notes***

## **New features:**

Images can be displayed scaled. The new messages IC\_GETSCALING and IC\_SETSCALING support this feature.

The new mode flags IF\_CHANGEABLE and IF\_READONLY which can be set via the IC\_SETMODE message are implemented to suppress or to activate the part of the built-in keyboard interface that alter or delete images.

## **Update Notes:**

The message IC\_GETIMAGESIZE has been expanded to support scaling factors.

The Image Control's internal file format has been changed to support the scaling messages. See appendix B *"The IC Image Control File Format"* for more information.

# *IC Image Control 1.50 Release Notes*

## **New features:**

The new function ICGetVersion returns the Image Control's current version number.

The new message IC\_GETZOOM returns the Image Control's zooming factor.

An Image Control window sends the new notification message ICN\_CHANGED when its image is changed via the new implemented keyboard interface.

An Image Control window sends the new notification message ICN\_CLEARED when its image is deleted via the new implemented keyboard interface.

## **Update Notes:**

The messages IC\_COPYDATA and IC\_SAVE optionally copies or saves the actual image data instead of the image's filename.

If an Image Control window has the current input focus it reacts to the following keyboard keys:

### **Key:**

DEL

CTRL+INS or CTRL+C

SHIFT+DEL or CTRL+X

SHIFT+INS or CTRL+V

### **Reaction:**

Deletes the actual image.

Copies the image to the clipboard.

Copies the image to the clipboard and deletes it.

Inserts an image from the clipboard.

# *IC Image Control 1.20/1.30 Release Notes*

## **New features:**

The new function **GetFilterStrings** returns information about currently available image import filters. These filters are installed in the private initialization file IC.INI. See the description of the **GetFilterStrings** function and the new chapter 1.6 "*Filter Selection*" for more information.

The new message IC\_GETIMAGESIZE can be used to get the size of the currently registered image.

The messages WM\_COPY and WM\_PASTE have been implemented to paste images from, or to copy images to, the clipboard.

## **Update Notes:**

The message IC\_SETIMAGE has been expanded to support filter selection.

The messages IC\_SETIMAGE and IC\_GETIMAGE have been expanded to handle images with their data instead of their filenames.

The Image Control's internal file format has been changed to support multiple filters. The new version number is 120. See appendix B "*The IC Image Control File Format*" for more information.

# *IC Image Control 1.12 Release Notes*

## **New features:**

This release includes a new graphics import filter for Windows bitmap files stored in DIB format (device-independent bitmap). The Image Control uses this filter for files with the extension \*.BMP.

The newly included filter BMP.DLL supports color palettes. To display colored images the application has to process the Windows messages WM\_PALETTECHANGED and WM\_QUERYNEWPALETTE. The following code shows how to handle these messages:

```
static HPALETTE  hPalAppl;           // application's logical palette
HPALETTE        hOldPal;           // old palette
HDC              hDC;
UINT             iChange = 0;       // number of changed palette entries

case WM_PALETTECHANGED:
    if ((HWND)wParam == hWnd) {     // ignore message if this appl. changed
        return 0;                  // the palette
    }                               // else fall through

case WM_QUERYNEWPALETTE:           // select application's logical palette

    if (hPalAppl) {
        hDC = GetDC(hWnd);
        hOldPal = SelectPalette(hDC, hPalAppl, FALSE);
        iChange = RealizePalette(hDC);
        if (hOldPal) {
            SelectPalette(hDC, hOldPal, TRUE);
            RealizePalette(hDC);
        }
        ReleaseDC(hWnd, hDC);
        if (iChange) {
            RepaintAll(hWnd);
        }
    }
    return iChange;
```

# *IC Image Control 1.10 Release Notes*

## **New features:**

The included graphics import filter for images stored in TIFF format can read the following additional format styles:

- 1-Dimensional Modified Huffman compression (CCITT Group 3).
- LZW compression.
- PackBits compression.

# 1. Introduction

## 1.1 What is IC

The IC Image Control is an efficient programming tool from DBS. It contains functions for image handling and display. It is especially useful in combination with the editor module TX Text Control because it allows you to insert images into formatted text.

## 1.2 The Library Files (16 bit version)

IC.DLL	The dynamic link library of the Image Control module.
IC.H	The Image Control's include file for your application.
IC.LIB	The import library file to be linked with the calling module.
IC.INI	A private initialization file containing all available image import filters.
TX_TIFF.FLT	The import filter for handling TIFF files.
TX_BMP.FLT	The import filter for handling bitmap files.
TX_WMF.FLT	The import filter for handling Windows Metafiles.
TX_GIF.FLT	The import filter for handling GIF files.

Additional filters for various other bitmap and vector image formats are available. Please contact DBS GmbH for further information.

## 1.3 The Library Files (32 bit version)

IC32.DLL	The dynamic link library of the Image Control module.
IC.H	The Image Control's include file for your application.
IC32.LIB	The import library file to be linked with the calling module.

IC32.INI	A private initialization file containing all available image import filters.
TX_TIF32.FLT	The import filter for handling TIFF files.
TX_BMP32.FLT	The import filter for handling bitmap files.
TX_WMF32.FLT	The import filter for handling Windows Metafiles.
TX_GIF32.FLT	The import filter for handling GIF files.

Additional filters for various other bitmap and vector image formats are available. Please contact DBS GmbH for further information.

## 1.4 *The Source Files*

If your version includes the source code of the module, the source files are to be found in subdirectories: \IC for the Image Control, \IC\TIFF for the TIFF filter, \IC\WMF for the WMF filter, \IC\GIF for the GIF filter and \IC\BMP for the bitmap filter. The include, lib and dll files are copied to the \INC, \LIB and \DLL directories during the make operation.

## 1.5 *How to use the Image Control*

The first step is to include the header file IC.H in your application. The file IC.LIB/IC32.LIB must be linked to the application, the executable library files listed above must be copied to the application start up directory or to a directory that is included in the PATH environment variable.

Now you can create a window by calling **CreateImageControl**. Windows of this type are referred to as Image Control in the following text. Save the returned window handle because it is needed for communication with the Image Control. The communication is completely handled with functions like **PostMessage**, **SendMessage** or the window manager functions. All functions of the Image Control except window creation and error handling are accessible through messages. The Image Control is established as a child of the controlling window.

To connect an image with a newly created window send the IC\_SETIMAGE message and use the returned values to contract or expand the window size before showing the window. The image file name sent with the IC\_SETIMAGE message can be obtained with the Windows common dialog box created by the Windows function **GetOpenFileName**. The **GetFilterStrings** function can be used to initialize this dialog box with available image filters. The **GetFilterStrings** function retrieves all filters installed in the private initialization file IC.INI. In the case of an existing image the dialog box can also be initialized with the IC\_GETIMAGE message.

## 1.6 How to use Images with a Text Control

If you are using the Image Control as part of the Text-Control, including the header file IC.H and linking your application with IC.LIB/IC32.LIB is not necessary. Instead of creating a window with **CreateImageControl**, use the message TX\_CREATEIMAGE. To initialize a file selecting dialog box use the messages TX\_GETIMAGE and TX\_GETIMAGEFILTERS.

## 1.7 Filter Selection

The Image Control uses the image import filter specified through the *wParam* parameter of the IC\_SETIMAGE or TX\_CREATEIMAGE message to read the image data. If this parameter contains zero it uses the filter belonging to the specified file extension. All possible file extensions are listed in the private initialization file IC.INI/IC32.INI. The entries in this file have the following format:

```
<graphics format>=<filter>,<extension>
```

These fields have the following meaning:

<b><u>Field</u></b>	<b><u>Meaning</u></b>
<graphics format>	Name of the graphics format supported by this filter. (for example, Tagged Image Format).
<filter>	Name of the filter executable minus the file extension if the filter has a .FLT extension (for example, if the filter is TIFF.FLT, the entry is TIFF, if the filter is TIFF.DLL, the entry must be TIFF.DLL).

<extension> File extension of the files which this filter supports. The string is limited to three characters.

If the Image Control cannot find the specified file extension, it tries to select a filter automatically. If the file contains an unknown format the Image Control returns an error code.

## 1.8 ANSI and Unicode

Like the Text Control module the Image Control can also be used either from an ANSI or a Unicode application. The Image Control include file IC.H defines an ANSI and a Unicode version for each message and function which has string or character parameters. The chapter 1.14 of the Text Control manual gives more information about how this is implemented.

The following is a complete list of all messages and functions that have two implementations. ANSI strings can contain characters from double-byte character sets.

### **Message:**

IC\_GETIMAGE

IC\_SETIMAGE

### **changes:**

*lParam* changed from **LPSTR** to **LPTSTR**

*lParam* changed from **LPSTR** to **LPTSTR**

### **Function:**

GetFilterStrings

### **changes:**

The returned data buffer contains Unicode or ANSI strings.

---

## 2. Function Directory

---

### CreateImageControl

**Syntax**            **HWND CreateImageControl**(*hWndParent*, *wChildID*, *lpRect*)

This function creates an Image Control child window.

<u>Parameter</u>	<u>Type/Discription</u>
<i>hWndParent</i>	<b>HWND</b> Identifies the parent window of the Image Control being created.
<i>wChildID</i>	<b>WORD</b> Represents the child window identifier.
<i>lpRect</i>	<b>LPRECT</b> Points to a <b>RECT</b> data structure containing the position and size of the Image Control window in client area coordinates of the parent window.

**Return Value**    The return value identifies the new Image Control window. It is zero if an error has occurred.

---

### GetFilterStrings

**Syntax**            **HGLOBAL GetFilterStrings**(*void*)

This function returns a buffer which contains pairs of null-terminated strings specifying image filters. The format of the buffer has the same form as described

for the *lpstrFilter* member of an **OPENFILENAME** structure and can be used to initialize the **GetOpenFileName** dialog box. See the description of the **OPENFILENAME** structure in Windows SDK for more information.

**Return Value** The return value is a global memory handle identifying the buffer that holds the strings. Each string ends with a terminating zero, the buffer itself ends with two terminating zeros. The return value is zero if an error has occurred. If an application has finished using the buffer it must free it with the **GlobalFree** function.

---

## GetICErrorCode

**Syntax** **LONG GetICErrorCode(void)**

This function returns an internal error code and can be called if the parent window has received an **IC\_ERRCODE** notification message.

**Return Value** The return value contains an error number in the low-order word and a module number in the high-order word. The module number is 3 or 4 for the programming tool described in this manual but it can also be the number of other modules used by the Image Control for special purposes.

The error numbers associated with the Image Control module are described in the error code table in appendix A. For a description of error codes associated with other module numbers, refer to the reference manuals of those modules.

---

## ICGetVersion

**Syntax** **LONG ICGetVersion(void)**

This function returns the current version number. The version number can be different from the format version number explained in appendix B.

**Return Value** The return value contains the Image Control's version number in its low-order word. For example, for the release IC Image Control 2.2 the version number is 220.

## 3. Message Overview

The Image Control processes the following Windows messages:

<b><u>Message</u></b>	<b><u>Description</u></b>
WM_CLEAR	Deletes the actual image.
WM_COPY	Copies image data to the clipboard.
WM_CREATE	Initializes the internal data structures.
WM_CUT	Copies image data to the clipboard and deletes the image.
WM_DESTROY	Deletes the internal data structures.
WM_ERASEBKGD	Paints the background.
WM_KILLFOCUS	Hides the selection frame.
WM_LBUTTONDOWN	Sends an ICN_CLICKED notification code to the parent window.
WM_LBUTTONDBLCLK	Sends an ICN_DOUBLECLICKED notification code to the parent window.
WM_MOUSEACTIVATE	Prevents the parent window from receiving this message.
WM_PAINT	Paints the contents of the Image Control.
WM_PASTE	Exchange an existing image with an image contained in the clipboard.
WM_RBUTTONDOWN	Sends this message to the parent window.
WM_SETFOCUS	Shows the selection frame.

The Image Control uses the following private messages:

<b><u>Message</u></b>	<b><u>Description</u></b>
IC_COPYDATA	Copies the Image Control data to a buffer.
IC_GETDATASIZE	Returns the size of the buffer (in bytes) that is needed for the IC_COPYDATA message.

IC_GETIMAGE	Returns the image data or the path name of the image associated with the currently selected image control window.
IC_GETIMAGESIZE	Returns the image size.
IC_GETMODE	Returns information about the mode flags currently set.
IC_GETSCALING	Returns the current scaling factors.
IC_GETZOOM	Returns the current zooming factor.
IC_LOAD	Fills an Image Control with image data from a file.
IC_PASTEDATA	Sets the data of an Image Control previously saved with the IC_COPYDATA message.
IC_PRINT	Prints a specific portion of the image.
IC_SAVE	Stores the Image Control data in a file.
IC_SETIMAGE	Registers a new image for the Image Control.
IC_SETMODE	Sets various mode flags.
IC_SETSCALING	Sets new scaling factors.
IC_ZOOM	Zooms the Image Control window.

The Image Control sends the following notification messages through a WM\_COMMAND message to its parent window to inform the application about special conditions:

<b><u>Message</u></b>	<b><u>Description</u></b>
ICN_CHANGED	Indicates that the Image Control window's image has been changed.
ICN_CLEARED	Indicates that the Image Control window's image has been deleted.
ICN_CLICKED	Indicates that the Image Control has been clicked.
ICN_DOUBLECLICKED	Indicates that the user has double-clicked the Image Control.
ICN_ERRCODE	Informs the parent window that an error has occurred.

---

## 4. Message Directory

---

### IC\_COPYDATA

This message copies the image data and all the format information of an Image Control window to a buffer pointed to by *lParam*.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	If <i>wParam</i> contains IF_SAVEASDATA this message copies the data itself. Otherwise it copies the image's filename.
<i>lParam</i>	Points to a buffer that is to receive the data.

**Return Value** The return value is a pointer to the next free position behind the copied data. It is zero if an error has occurred.

**Comments** This message is implemented to save the Image Control data in memory. For more information on how to retrieve the data from memory see the description of the IC\_PASTEDATA message. To save the data to a file, use the IC\_SAVE message. To get the minimum buffer size in bytes use the IC\_GETDATASIZE message with the same value of the *wParam* parameter.

The data is formatted according to appendix B.

---

### IC\_GETDATASIZE

This message returns the minimum size, in bytes, of a buffer that can be used to save the contents of the Image Control in memory.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	If <i>wParam</i> contains IF_SAVEASDATA this message returns the size of the image data. Otherwise it returns the size of the image's filename.

*lParam* Is not used.

**Return Value** The return value contains the data size. It is zero if an error has occurred.

**Comments** This message is necessary to calculate the size of the buffer for the IC\_COPYDATA message.

## IC\_GETIMAGE

This message retrieves the image data or a full DOS path name of the file containing the image data for the currently registered image.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the size of the buffer which <i>lParam</i> points to, including the null-terminating character. This parameter is not used if <i>lParam</i> contains zero.
<i>lParam</i>	If this parameter is zero the return value is a global data handle. Otherwise this parameter must point to a buffer that is to receive the path name.

**Return Value** If *lParam* is zero the return value is a global data handle which identifies a buffer containing the image data. Otherwise the return value is the length, in bytes, of the copied path name. The return value is zero if an error has occurred, if no image is currently registered or if no path name exists.

**Comments** If the return value is a global data handle the caller must free it with the **GlobalFree** function.

**Unicode** The size specified through *wParam* is in characters, when UNICODE is defined, otherwise it is in bytes.

## IC\_GETIMAGESIZE

This message returns the size, in pixels, of the currently registered image.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	If this parameter is nonzero the return value contains scaled sizes according to the currently set scaling factors. Otherwise the return value contains the original size values.
<i>lParam</i>	Is not used.

**Return Value** The return value is a long value that contains the size, in pixels, of the current image. The width is in the low-order word and the height is in the high-order word. If the low-order word is zero an error has occurred or the specified Image Control window has no registered image.

---

## IC\_GETMODE

This message returns information about the different modes of the Image Control like the current background mode or the current display mode.

<u>Parameter</u>	<u>Description</u>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

**Return Value** The return value is a combination of the following values:

<u>Value</u>	<u>Meaning</u>
IF_CHANGEABLE	The image is changeable via the keyboard interface.
IF_DRAWGRAYRECT	If a gray shaded rectangle is drawn instead of the image.
IF_DRAWIMAGE	If the image is drawn on the screen.
IF_FILTERDIALOG	If the preference dialog boxes of the image import filters are to be opened.
IF_FRAMED	If a border line is drawn.

IF_NOFILTERDIALOG	If the preference dialog boxes of the image import filters are to be ignored.
IF_NOTFRAMED	If the border line is hidden.
IF_OPAQUE	If the background mode is opaque.
IF_READONLY	The image is not changeable via the keyboard interface.
IF_TRANSPARENT	If the background mode is transparent.

---

## IC\_GETSCALING

This message returns the current scaling factors in percent.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

***Return Value*** The return value specifies the current scaling factors in percent. The low-order word contains the scaling factor in x-direction, the high-order word contains the scaling factor in y-direction.

---

## IC\_GETZOOM

This message returns the current zooming factor in percent.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Is not used.

***Return Value*** The return value specifies the current zooming factor in percent.

## IC\_LOAD

This message fills the internal data buffers of the Image Control with data stored in a file. This data must have been stored with the IC\_SAVE message.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains a DOS file handle. The file pointer must be positioned at the head of the data.
<i>lParam</i>	Points to a POINT data structure. The Image Control copies the size of the image, in pixels, to this buffer. If <i>lParam</i> is zero, the image size is not copied.

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

***Comments*** The file pointer is only moved if the return value is nonzero.

The image size retrieved by *lParam* includes the current zooming factor.

---

## IC\_PASTEDATA

This message sets the data of an Image Control previously obtained with the IC\_COPYDATA message. The data contains all image and format information needed by the Image Control. The data must be formatted according to appendix B.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Is not used.
<i>lParam</i>	Points to a buffer that contains the data.

***Return Value*** The return value points to the buffer position behind the pasted data. It is zero if an error has occurred.

**IC\_PRINT**

This message submits the contents of the Image Control to the device identified by the *wParam* parameter.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains a printer device context.
<i>lParam</i>	Points to a RECT data structure that contains the bounding rectangle of the part of the image to be printed. This rectangle is given in device pixels with an origin at the upper left corner of the image. The image must be copied to the device given by <i>wParam</i> with an origin at (0, 0).

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

***Comments*** The Image Control is always printed at 100 percent, regardless of the current zooming factor.

---

**IC\_SAVE**

This message saves the data of an Image Control in a file.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains a DOS file handle.
<i>lParam</i>	If <i>lParam</i> contains IF_SAVEASDATA the actual image data is stored. Otherwise a reference to the actual image file is stored.

***Return Value*** The return value is zero if an error has occurred. Otherwise it is nonzero.

***Comments*** The Image Control does not save the size of its window.

## IC\_SETIMAGE

This message registers a new image for the Image Control window.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	The high-order bit indicates the meaning of the <i>lParam</i> parameter. The lower 15 bits specify an image filter as an index into the buffer returned by the <b>GetFilterStrings</b> function. The first pair of strings has an index value of 1. If the buffer returned by <b>GetFilterStrings</b> is used to initialize the <i>lpstrFilter</i> member of an <b>OPENFILENAME</b> structure, another member of that structure, <i>nFilterIndex</i> , can be used to initialize this parameter. See the Windows SDK for more information about the <b>OPENFILENAME</b> structure. If <i>wParam</i> is set to 0, the Image Control automatically tries to select a filter.
<i>lParam</i>	If the high-order bit of the <i>wParam</i> parameter is set, <i>lParam</i> must be a global data handle which identifies a buffer containing the image data. Otherwise <i>lParam</i> points to a null-terminated string which is the full DOS path name for the file containing the new image.

***Return Value*** The return value is a long value that contains the size, in pixels, of the new image. The width is in the low-order word and the height is in the high-order word. If the low-order word is zero an error has occurred and the high-order word contains one of the error code values described in the following list:

<u>Value</u>	<u>Meaning</u>
0	General error, use <b>GetICErrorCode</b> for more information.
1	The given file does not exist or cannot be opened.
2	The given file is of an unknown type.
3	The given import file contains an unsupported compression scheme.
4	The given import file contains an unsupported version.
5	The given import file contains an unsupported style.
6	The specified filter cannot be found.
7	The specified filter uses an unknown interface.

*Comments*      If no error has occurred the image size returned includes the current zooming factor.

---

## IC\_SETMODE

This message sets several different modes of the Image Control. Changing one mode does not alter the other mode settings.

<u>Parameter</u>	<u>Description</u>										
<i>wParam</i>	Is the new mode value. It can be one or more of the following values:										
	<table> <thead> <tr> <th><u>Value</u></th> <th><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td>IF_CHANGEABLE</td> <td>Activates the part of the keyboard interface that alters or deletes images.</td> </tr> <tr> <td>IF_DRAWGRAYRECT</td> <td>If a gray shaded rectangle is to be drawn for fast updating.</td> </tr> <tr> <td>IF_DRAWIMAGE</td> <td>If the image is to be drawn on the screen.</td> </tr> <tr> <td>IF_FILTERDIALOG</td> <td>If an image import filter has an internal dialog box to set preferences this mode opens this dialog.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Meaning</u>	IF_CHANGEABLE	Activates the part of the keyboard interface that alters or deletes images.	IF_DRAWGRAYRECT	If a gray shaded rectangle is to be drawn for fast updating.	IF_DRAWIMAGE	If the image is to be drawn on the screen.	IF_FILTERDIALOG	If an image import filter has an internal dialog box to set preferences this mode opens this dialog.
<u>Value</u>	<u>Meaning</u>										
IF_CHANGEABLE	Activates the part of the keyboard interface that alters or deletes images.										
IF_DRAWGRAYRECT	If a gray shaded rectangle is to be drawn for fast updating.										
IF_DRAWIMAGE	If the image is to be drawn on the screen.										
IF_FILTERDIALOG	If an image import filter has an internal dialog box to set preferences this mode opens this dialog.										

IF_FRAMED	If a border line is to be drawn.
IF_NOFILTERDIALOG	This mode ignores filter dialog boxes to set preferences.
IF_NOTFRAMED	If the border line is to be hidden.
IF_OPAQUE	If the background mode is to be set to opaque.
IF_READONLY	Blocks the part of the keyboard interface that alters or deletes images.
IF_TRANSPARENT	If the background mode is to be set to transparent.

The bitwise OR operator can be used to specify more than one value.

*lParam* Is not used.

### **Comments**

The Image Control is completely updated if a new background mode or a new display mode is set. The gray shaded rectangle is only drawn on the screen, it is not printed.

The mode flags are grouped. The following groups list mode flags that should not be used together:

IF\_OPAQUE and IF\_TRANSPARENT  
 IF\_NOTFRAMED and IF\_FRAMED  
 IF\_DRAWIMAGE and IF\_DRAWGRAYRECT  
 IF\_NOFILTERDIALOG and IF\_FILTERDIALOG  
 IF\_CHANGEABLE and IF\_READONLY

The first value of each group is the default value.

---

## **IC\_SETSCALING**

This message sets scaling factors which are used to scale the current image.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
-------------------------	---------------------------

<i>wParam</i>	Contains one of the following values:
---------------	---------------------------------------

---

	<b><u>Value</u></b>	<b><u>Meaning</u></b>
	IF_SCALED	The scaling factors specified through the <i>lParam</i> parameter are used to scale the image.
	IF_SCALEDTOWINDOW	The current image is scaled so that it fits into the Image Control's actual window size.
	IF_UNSCALED	The actual scaling factors are reset to unscald.
<i>lParam</i>	Specifies new scaling factors. The low-order word contains the scaling factor in x-direction, the high-order word contains the scaling factor in y-direction.	
<b><i>Return Value</i></b>	The return value is zero if the scaling factors could be set. Otherwise it is nonzero.	

---

## IC\_ZOOM

This message sets a new zooming factor for the Image Control. This factor is stated in percent. A value of 100 means the original size.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Contains the new zooming factor in percent. The value must be between 10 and 250.
<i>lParam</i>	Is not used.

***Return Value*** The return value is zero if the window cannot be zoomed. Otherwise it is nonzero.

***Comments*** The Image Control window size is adjusted but the Image Control is not updated. The **InvalidateRect** function must be used to update the appropriate portion of the parent window's client area.

---

## 5. Notification Messages

---

### ICN\_CHANGED

This code specifies that the actual contents of an Image Control window have been changed with the SHIFT+INS or CTRL+V keyboard key. The parent window receives this code through a WM\_COMMAND message from an Image Control.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the Image Control ID.
<i>lParam</i>	Contains a handle that identifies the Image Control in its low-order word and the ICN_CHANGED notification code in its high-order word.

---

### ICN\_CLEARED

This code specifies that the actual contents of an Image Control window have been deleted with the DEL, SHIFT+DEL or CTRL+X keyboard key. The parent window receives this code through a WM\_COMMAND message from an Image Control.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the Image Control ID.
<i>lParam</i>	Contains a handle that identifies the Image Control in its low-order word and the ICN_CLEARED notification code in its high-order word.

## ICN\_CLICKED

This code specifies that the user has clicked an Image Control. The parent window receives this code through a WM\_COMMAND message from an Image Control.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the Image Control ID.
<i>lParam</i>	Contains a handle that identifies the Image Control in its low-order word and the ICN_CLICKED notification code in its high-order word.

---

## ICN\_DOUBLECLICKED

This code specifies that the user has double-clicked an Image Control. The parent window receives this code through a WM\_COMMAND message from an Image Control.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the Image Control ID.
<i>lParam</i>	Contains a handle that identifies the Image Control in its low-order word and the ICN_DOUBLECLICKED notification code in its high-order word.

---

## ICN\_ERRCODE

This code informs the parent window that an error has occurred. The parent window receives this code through a WM\_COMMAND message from an Image Control. Further information can be obtained by calling the **GetICErrorCode** function.

<b><u>Parameter</u></b>	<b><u>Description</u></b>
<i>wParam</i>	Specifies the Image Control ID.
<i>lParam</i>	Contains a handle that identifies the Image Control in its low-order word and the ICN_ERRCODE notification code in its high-order word.

# Appendix A

This appendix describes the error codes returned by the **GetICErrorCode** function. The error codes described here belong to the modules with the numbers 3 (Image Control module) or 4 (graphics import filter).

The following error codes are sorted according to its group codes:

## *Error Code Table (Image Control module)*

### **DBS\_E\_OUTOFMEMORY**

<b>0101</b>	Unable to allocate enough memory to register a new image.
<b>0301</b>	Unable to allocate enough memory to register the image to be loaded.
<b>0401</b>	Unable to allocate enough memory.
<b>0703</b>	Unable to allocate enough memory.
<b>0902</b>	Unable to allocate enough memory.
<b>0904</b>	Unable to allocate enough memory.
<b>0B00</b>	Unable to allocate enough memory.

### **DBS\_E\_UNEXPECTED**

<b>0B02</b>	IC_PRINT message: Unexpected error.
-------------	-------------------------------------

### **DBS\_E\_FILEIO**

<b>0302</b>	File error, unable to read data from file.
<b>0403</b>	File error, unable to write data to file.
<b>0700</b>	File error: cannot create temporary file to import clipboard data.
<b>0701</b>	File error: cannot write to temporary file.
<b>0702</b>	File error: cannot read temporary file data.

**DBS\_E\_CLIPBOARD**

- 0600**        **OpenClipboard** failed.
- 0601**        Unable to get data from the clipboard.
- 0602**        Clipboard import error: invalid data handle.
- 0603**        Clipboard export error: cannot copy data to the clipboard.

**DBS\_E\_DLLNOTLOADED**

- 0504**        The specified graphics import filter cannot be loaded.

**DBS\_E\_DLLINCOMPATIBLE**

- 0505**        The specified graphics import filter uses an unknown interface.

**DBS\_E\_INVALIDARG**

- 0900**        IC\_PASTEDATA message: invalid argument.
- 0903**        IC\_COPYDATA message: invalid argument.
- 0B01**        Invalid argument.

**DBS\_E\_INVALIDFORMAT**

- 0901**        IC\_PASTEDATA message: unsupported data format.

## *Error Code Table (graphics import filter)*

**DBS\_E\_OUTOFMEMORY**

- 010B**        Unable to allocate enough memory during import operation.

**DBS\_E\_NOMEMORYACCESS**

- 010F**        Locking memory failed.

**DBS\_E\_UNGROUPED**

- 0101**        The specified file does not exist or cannot be opened.
- 0102**        The specified file contains a bitmap or a picture that is too big.
- 0103**        The file contains a bitmap that is completely white.
- 0107**        The specified file is of an unknown type.
- 0109**        Current file data is bad.
- 010A**        The import operation has been aborted.
- 010D**        The generated metafile is too big.
- 0129**        The import file contains an unsupported compression scheme.
- 012A**        The import file contains an unsupported file version.
- 012B**        The import file contains an unsupported style.

# Appendix B

## The IC Image Control File Format

The following describes the IC Image Control file format. The Image Control supports all prior versions of the format, but, to maximize working speed, the current format should be used.

The format has the following form:

```

WORD  wFormatID;
WORD  wVersion;
DWORD dwFormatSize;
short nXScale;
short nYScale;
WORD  wWinWidth;
WORD  wWinHeight;
short nFileNameLength;
short nFilterNameLength;
char  szExtension[4];
char  szFileName[];
char  szFilterName[];
union {
    BYTE    ImageData[];
    ICDITEM ExtData[];
}

```

These fields have the following meanings:

<b><u>Field</u></b>	<b><u>Description</u></b>
<i>wFormatID</i>	Specifies an identifier which supports older version formats. This value must be zero for version 120 and higher.
<i>wVersion</i>	Specifies the version number of the format. This number is currently 220.
<i>dwFormatSize</i>	Specifies the total size, in bytes, of the format structure. This value is returned by the IC_GETDATASIZE message.

---

<i>nXScale, nYScale</i>	Specifies the scaling factors in horizontal and vertical direction in percent if the scaling mode of the Image Control is set to IF_SCALED. Otherwise these values should be set to zero.
<i>wWinWidth</i>	Specifies the window width in TWIPS (1/1440 Inch) if the scaling mode of the Image Control is set to IF_SCALEDTOWINDOW. Otherwise this value should be set to zero.
<i>wWinHeight</i>	Specifies the window height in TWIPS (1/1440 Inch) if the scaling mode of the Image Control is set to IF_SCALEDTOWINDOW. Otherwise this value should be set to zero.
<i>nFileNameLength</i>	Specifies the size, in bytes, of the <i>szFileName[ ]</i> array without a terminating zero. If this value is zero the <i>szFileName[ ]</i> array does not exist and the image is specified through the <i>ImageData[ ]</i> array. Otherwise, when this value is non-zero, additional data is specified through the <i>ExtData[ ]</i> array.
<i>nFilterNameLength</i>	Specifies the size, in bytes, of the <i>szFilterName[ ]</i> array without a terminating zero.
<i>szExtension[4]</i>	Contains the file extension belonging to the filter stored in the <i>szFilterName[ ]</i> array.
<i>szFileName[ ]</i>	Contains the full path and file name of the file containing the image data. If no file name is specified this array must be given a terminating zero.
<i>szFilterName[ ]</i>	Contains the name of the filter that should be used to read the image data. If this string contains no path the filter must be in the same directory as the Image Control module. If the string contains no extension the .FLT extension is assumed. If no filter name is specified this array must be given a terminating zero.
<i>ImageData[ ]</i>	Specifies an array of bytes containing the image data if the <i>szFileName[ ]</i> array does not specify a filename. If the image is specified by a filename the <i>ExtData[ ]</i> array is used instead of this array.

*ExtData[ ]* Is an array of ICDITEM structures containing additional Image Control data. The size of the array must be determined through the *dwSize* members of the ICDITEM structures. This array is an alternative to the *ImageData[ ]* array. See the following comments section for a description of the ICDITEM structure.

### Comments

The *szFilterName[ ]* array and/or the *szExtension[ ]* array can contain empty strings. In this event the Image Control uses the following algorithm to select a filter:

- If no filter name is specified, *szExtension[ ]* is used to select a filter from the private initialization file IC.INI/IC32.INI.
- If no extension is specified the extension contained in *szFileName[ ]* is used to select a filter from IC.INI/IC32.INI.
- If *szFileName[ ]* contains an unknown extension, the Image Control automatically tries to select a filter.

The ICDITEM structure is the header structure of an entry in the *ExtData[ ]* array. The data itself follows the header structure and its size is determined through the *dwSize* member. ICDITEM has the following form:

```
typedef struct tagICDITEM {
    DWORD dwID;
    DWORD dwSize;
    DWORD dwReserved;
} ICDITEM;
```

<b><u>Field</u></b>	<b><u>Description</u></b>										
<i>dwID</i>	Identifies the data of this item. The following values are possible:										
	<table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left;"><b><u>Value</u></b></th> <th style="text-align: left;"><b><u>Meaning</u></b></th> </tr> </thead> <tbody> <tr> <td style="vertical-align: top;">ICI_END</td> <td style="vertical-align: top;">Identifies the end of the data array.</td> </tr> <tr> <td style="vertical-align: top;">ICI_FILEW</td> <td style="vertical-align: top;">The data is a filename formatted as a zero-terminated Unicode string.</td> </tr> <tr> <td style="vertical-align: top;">ICI_FILTERW</td> <td style="vertical-align: top;">The data is a filtername formatted as a zero-terminated Unicode string.</td> </tr> <tr> <td style="vertical-align: top;">ICI_EXTW</td> <td style="vertical-align: top;">The data is a file extension formatted as a zero-terminated Unicode string.</td> </tr> </tbody> </table>	<b><u>Value</u></b>	<b><u>Meaning</u></b>	ICI_END	Identifies the end of the data array.	ICI_FILEW	The data is a filename formatted as a zero-terminated Unicode string.	ICI_FILTERW	The data is a filtername formatted as a zero-terminated Unicode string.	ICI_EXTW	The data is a file extension formatted as a zero-terminated Unicode string.
<b><u>Value</u></b>	<b><u>Meaning</u></b>										
ICI_END	Identifies the end of the data array.										
ICI_FILEW	The data is a filename formatted as a zero-terminated Unicode string.										
ICI_FILTERW	The data is a filtername formatted as a zero-terminated Unicode string.										
ICI_EXTW	The data is a file extension formatted as a zero-terminated Unicode string.										

<i>dwSize</i>	Determines the amount, in bytes, of the data that follows this structure.
<i>dwReserved</i>	Reserved for future use. Must be set to zero.