TX Text Control

Class Library Programmer's Guide

Version 7.0

TX Text Control 7.0

Contents

Introduction

This programmer's guide contains the information necessary to use the Text Control Class Library. The Text Control Class Library is a set of C++ classes that encapsulate the functionality necessary to use Text Control in applications written with the Microsoft Foundation Class Library. Using Text Control, you can create all kinds of text-based applications with highly sophisticated formatting and display capabilities which are normally the exlusive domain of large word processing packages.

System Requirements

Using the Text Control Class Library requires the following minimum configuration:

- Windows 95/98, Windows NT 4.0 or Windows 2000.
- Microsoft Visual C++ 6.0.
- The Microsoft Foundation Class Library 6.0.

How this Manual is Organized

- Part 1 of this manual, *"Class Library User's Guide"*, is a tutorial that can be used to learn how to use the Text Control Class Library. It covers the following topics:
  - A step-by-step guide creating a simple word processor
  - Adding additional Text Control features like headers and footers or hypertext links.
  - How to create your own modified version of the Text Control Class Library.
- Part 2, *"Class Library Reference"*, contains more detailed information of all the classes' member functions and how these functions work together. It also covers the following topics:
  - Information about how the Text Control Classes are integrated in the Microsoft Foundation Classes.

- Several articles describing how the Text Control Class Library realizes the more advanced Text Control features.

The Files You Work With

After Text Control has successfully been installed you can find all required files in the following sub-directories under the main installation directory:

- \BIN contains all DLL files of the Text Control Class library and the Text Control kernel. The Class Library DLL is contained in the following versions:
    - TXCLASSES.DLL
      Retail version using the ANSI character format
    - TXCLASSESD.DLL
      Debug version using the ANSI character format
    - TXCLASSESU.DLL
      Retail version using the Unicode character format
    - TXCLASSESDU.DLL
      Debug version using the Unicode character format
- \HELP contains the Text Control online help files.
- \TXCLASSES\INC contains the Class Library's include files. More information about how to integrate these files can be found in the next chapter.
- \TXCLASSES\LIB contains the import library files of the Class Library. More information about how to link your appliction with these files can be found in the next chapter.
- \TXCLASSES\SRC contains the source files of the Class Library. For more information on how to modify and compile the Class Library see "*Building Your Own Class Library*".

Distributing your Applications

The following table shows all the files necessary for Text Control to operate properly. You must ensure that these files exist on your client's machine and they are the correct version. If your client's machine has older versions of these files, you should update them.

1	TXCLASSES.DLL
2	TX32.DLL TXTLS32.DLL WNDTLS32.DLL TXOBJ32.DLL IC32.DLL IC32.INI TX_BMP32.FLT TX_TIF32.FLT TX_WMF32.FLT TX_RTF32.DLL TX_HTM32.DLL TX_WORD.DLL
3	MFC42.DLL (6.00.8447.0)
4	TX_GIF32.FLT

The first file (group 1) is the DLL file containing the Text Control Class Library. This file should be installed in the same directory as your application's executable file. If your application is based on the Unicode character format, you must distribute the Unicode version (TXCLASSESU.DLL).

The files listed in the second group are the Text Control kernel DLL files. They must be installed in the same directory as the TXCLASSES.DLL. You must always install all of them.

You should also verify that the Microsoft Foundation Class Library (group 3) is installed on your client's computer. This file must be installed in the Windows system directory. Please refer to Microsoft's redistribution policy if you need to redistribute them. If your application is based on the Unicode character format you must distribute the Unicode version (MFC42U.DLL).

The last file (group 4) is a filter to use the GIF image format with Text Control. Unisys Corporation holds patent rights to the LZW technology used in this filter. If a customer wants to use the GIF file format, he is required to obtain a license from Unisys and send a copy of the license agreement to The Imaging Source Europe GmbH. We will then send him the GIF filter free of charge.

Class Library User's Guide

In this tutorial, you will learn how to use Microsoft Visual C++ to build a Text Control based word processor with the Text Control Class Library working together with the Microsoft Foundation Class Library. It is assumed that you have some knowledge of C++ and of programming with the Microsoft Foundation Class Library.

The tutorial is divided into several parts each of which covers a number of Text Control features:

- Part 1, "*Creating a Simple Word Processor*", begins with a 10 step tour how to create a simple word processor with Visual C++'s Application Wizard.
- Part 2, "*Extend Your Application's Menus*", shows how to add Text Control's predefined menus, how to create an own menu command and how to access the Text Control.
- Part 3, "*Adding A Button Bar and a Status Bar*", shows how to add a Button Bar and a Status Bar to the application's frame window.
- Part 4, "*Working with File Formats*", shows how to enable your application to load and save all the file formats that Text Control supports.

Each of the tutorial's parts adds more features to the starter application created in part 1. The resulting program is the *TXWords* demo program distributed with Text Control. The source code of each part can be found in the *Samples\VisualC\TXWords1 ... n* sub-directories.

Creating a Simple Word Processor

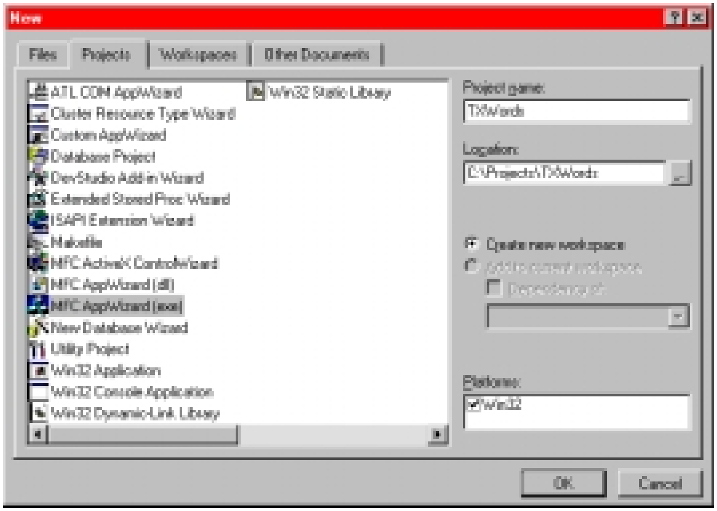
This chapter shows you how to create a simple word processor from scratch with just a few lines of code. It will be able to load and save files, use the clipboard and will have dialog boxes for character and paragraph formatting, a ruler and a full keyboard and mouse interface.The following step-by-step instructions cover the following topics:

- Creating the starter application.
- Performing Visual C++ project settings.
- Adding the Text Control Class Library.
- Using the MFC document/view architecture.

Step 1: Use the Visual C++ AppWizard to Create a Project

Start Application Wizard:

- From the Visual C++ *File* menu select *New*.
- Make sure you're on the *Projects* tab.
- Select *MFC AppWizard (exe)*.
- In the *Location* box enter the desired project base directory (e.g. *C:\Projects*).
- Enter the name of your Project in the *Project Name* box (This tutorial assumes *TXWords* as the project name)



Project name: TXWords

Location: C:\Projects\TXWords

● Create new workspace

○ Add to current workspace

☐ Dependency of:

Platforms: ☑ Win32

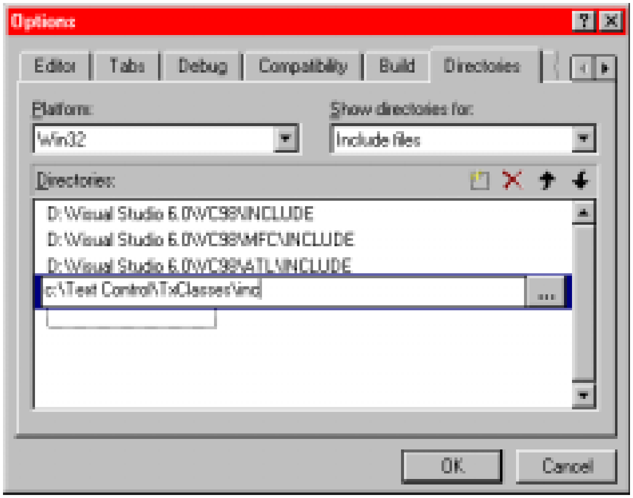
- Click on *OK*.

Proceed in the following dialogs as follows:

1. On page 1 don't change the default settings. Click on *Next*.
2. On page 2 don't change the default settings. Click on *Next*.
3. On page 3 deselect support for *ActiveX Controls*.Click on *Next*.
4. On page 4 deselect *Initial status bar*, because Text Control has its own status bar. Click on *Next.*
5. On page 5 don't change the default settings. Click on *Next*.
6. On page 6 don't change the default settings. Click on *Finish.*

   Now a dialog box appears, summarizing all the settings made in the previous steps. Click on *OK* to start the code generation process.

Step 2: Add Text Control's Include Files to Your Project



Platform: Win32

Show directories for: Include files

Directories:

D:\Visual Studio 6.0\VC98\INCLUDE
D:\Visual Studio 6.0\VC98\MFC\INCLUDE
D:\Visual Studio 6.0\VC98\ATL\INCLUDE
c:\Text Control\TxClasses\inc

In Visual C++, select *Tools - Options* from the menu, select the *Directories* tab, and add the *\TXClasses\Inc* subdirectory to the list of include paths. (i.e. if your Text Control installation directory is *C:\TextControl*, add *C:\TextControl\TXClasses\Inc* to the list of include paths). Close this dialog by clicking on *OK.*

Step 3: Add Text Control's Import Libraries to Your Project

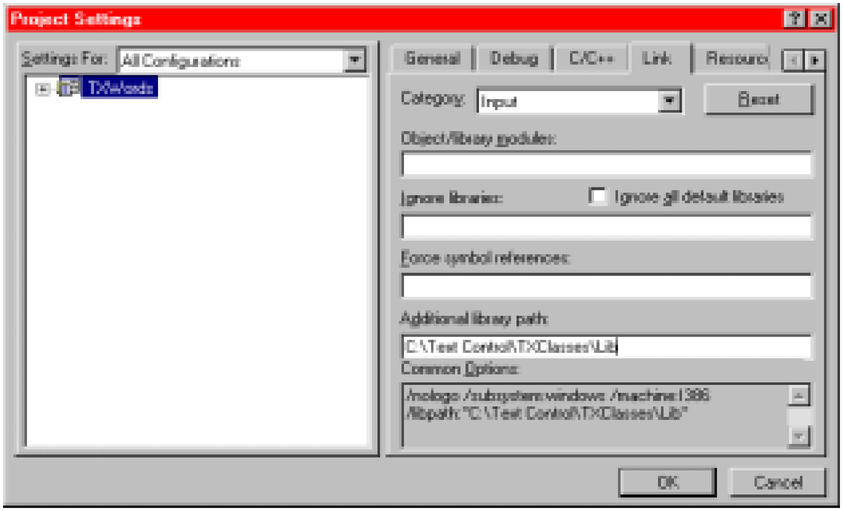
- From the *Project* menu select *Settings*.
- Select the *Link* tab.

- Under *Category* select *Input.*
- In the *Object/Library modules* text field enter the following depending on the configuration you selected under *Settings For:*

For this configuration	**Add this to *Object/Library modules***
Win32 Debug	*TXClassesD.lib*
Win32 Release	*TXClasses.lib*

(Use *TXClassesU.lib* and *TXClassesDU.lib* instead, when you develop an application based on the Unicode character format.)

- Select *Settings for: All configurations*.
- In the *Additional library path* text field, enter the *\TXClasses\Lib* subdirectory, i.e. enter *C:\TextControl\TXClasses\Lib* if your Text Control installation directory is *C:\TextControl*.



Settings For: All Configurations

TXWords

Category: Input

Object/library modules:

Ignore libraries:

☐ Ignore all default libraries

Force symbol references:

Additional library path: C:\Text Control\TXClasses\Lib

Common Options:

```
/nologo /subsystem:windows /machine:I386
/libpath:"C:\Text Control\TXClasses\Lib"
```

Step 4: Enable Runtime Type Information (RTTI)

- While still in the *Project Settings* dialog, select *Settings for: All configurations.*
- On the *C++* tab select the *C++ Language category*.

- Select the *Enable Run-Time Type Information (RTTI)* check box.
- Close the *Project Settings* dialog by clicking on *OK*.

Note: If you forget this last step, you will get an error while compiling your *TXWords* project. RTTI is absolutely necessary for the TXClasses DLL to work properly.

Step 5: Copy Text Control's DLL Files

Before running your program make sure the Text Control DLL files are in the output directory of your project. The Text Control DLL files can be found in the *\Bin* subdirectory of the Text Control installation directory. For more information see "*Introduction - The Files You Work With*".

If you build an application based on the ANSI character format:

- Copy TXCLASSES.DLL to *C:\Projects\TXWords\Release.*
- Copy TXCLASSESD.DLL to *C:\Projects\TXWords\Debug.*

If you build an application based on the Unicode character format:

- Copy TXCLASSESU.DLL to *C:\Projects\TXWords\Release.*
- Copy TXCLASSESUD.DLL to *C:\Projects\TXWords\Debug.*

Copy all other Text Control DLL files to both directories. A complete list can be found in "*Introduction - Distributing your Applications*".

Step 6: Derive Your View Class from CTXView

In *TXWordsView.h:*

- Add the following before the declaration of the class *CTXWordsView*:

```
#include "TXView.h"
```

- Derive your *CTXWordsView* class from **CTXView**:

```
class CTXWordsView : public CTXView
```

In *TXWordsView.cpp:*

- Replace every occurrence of **CView** with **CTXView**.

Step 7: Derive Your Document Class from CTXDoc

In *TXWordsDoc.h:*

- Add the following before the declaration of the class *CTXWordsDoc*:

```
#include "TXDoc.h"
```

- Derive your *CTXWordsDoc* class from **CTXDoc**:

```
class CTXWordsDoc : public CTXDoc
```

In *TXWordsDoc.cpp:*

- Replace every occurrence of **CDocument** with **CTXDoc**.

Step 8: Add Code to Load and Save Documents

In *TXWordsDoc.cpp* add the following line to *CTXWordsDoc::Serialize()* (the added line is marked with ⌦):

```
void CTXWordsDoc::Serialize(CArchive& ar)
{
⌦   CTXDoc::Serialize(ar);

    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}
```

Step 9: Add Code to Print Documents

In *TXWordsView.cpp* change **CTXWordsView::OnPreparePrinting**. The function's code should look like the following:

```
BOOL CTXWordsView::OnPreparePrinting(CPrintInfo* pInfo)
{
    return CTXView::OnPreparePrinting(pInfo);
}
```

Step 10: Compile and Run Your Application

- Verify that you have completed all steps exactly as they are documented here. (The sub-directory *Samples\VisualC\TXWords1* contains the code created in this chapter.)
- Hit F7 (or select *Build TXWords*.exe from the *Build* menu) to start the compilation process.

After compilation, you can run the application with Visual C++'s *Build* - *Execute TxWords.exe* command. When *TXWords* runs, an MDI application window appears with a menu bar containing *File*, *Edit*, *View*, *Window* and *Help* menus and a default toolbar. The application window contains one open document window with a ruler at its top. You can type in text, copy and paste it via the clipboard and save and load the text using the *File* - *Open* and the *File* - *Save* menus. You can also print the document or view the printing output with the print preview command.

Extending Your Application's Menus

In addition to the generic file and edit commands you have seen in the previous chapter, Text Control's view class contains predefined command handlers that can change font and paragraph attributes and insert tables, images and OLE objects.

In this part you will add predefined resources to access Text Control's predefined command handlers. You will also create your own command handler to learn how to extend the predefined menus.

Add Text Control's Predefined Resources

Text Control's predefined resources are located in the *\TXClasses\Res* subdirectory. (i.e. if your Text Control installation directory is *C:\TextControl*, the resource directory is *C:\TextControl\TXClasses\Res*). The subdirectories contain the resources for different languages. The currently available languages are English U.S. (*\enu*) and German (*\deu*).

To add the resources perform the following steps:

- In the Workspace window select the *ResourceView* tab.
- With *File* - *Open* open the *TXClasses* resource file (i.e. *\TXClasses\Res\enu\TXClasses.rc*).
- Double-click on the menu resource of *TXClasses.rc* and select the menu with the identifier TX_IDR_TXVIEW. Press the CTRL key and drag and drop this menu in your project's Workspace window.
- Perform the same operation with the TX_IDD_TABLEINSERT dialog box and the tool bar (TX_IDR_TXVIEW).
- Double-click the string table in *TXClasses.rc*, choose *Edit* - *Select All* and then *Edit* - *Copy*. Then double-click your application's string table in the Workspace window and choose *Edit* - *Paste*.

  Your application's resources now should contain an additional menu (TX_IDR_TXVIEW), an additional dialog box (TX_IDD_TABLEINSERT), an additional toolbar and additional strings in your application's string table.

  Note: All the resource identifiers of Text Control are prefixed with *TX_*.
- Close *TXClasses.rc*. Leaving *TXClasses.rc* open results in a conflict with *Resource.h*.

Copy the Help Menu

The menu previously created with the Application Wizard (IDR_TXWORDTYPE) is no longer required, as the TX_IDR_TXVIEW menu is your new menu. Before deleting it, you should copy your application's help menu:

- Double-click the old menu and select the help menu. Choose *Edit* - *Copy*.
- Double-click the new menu and choose *Edit* - *Paste*. The help menu now appears at rightmost submenu of the TX_IDR_TXVIEW menu.

Load the Copied Toolbar

The **CMainFrame** class which implements the application's main frame window, by default loads the toolbar created through the Application Wizard. To make the copied toolbar available perform the following:

In **CMainFrame::OnCreate** change

```
m_wndToolBar.LoadToolBar(IDR_MAINFRAME)
```

to

```
m_wndToolBar.LoadToolBar(TX_IDR_TXVIEW)
```

Add an Additional Menu Command

The following shows you how to extend the previously inserted predefined menu with an additional menu command. You should be familiar with Application Studio and Class Wizard to add a menu entry and a corresponding command handler. The following steps add a new *View* - *Whitespace* menu command:

- In the Workspace Window double-click the TX_IDR_TXVIEW menu.
- Double-click the new entry field at the bottom of the *View* sub-menu. The *Menu Item Properties* dialog box appears. Enter ID_VIEW_WHITESPACE as ID, *Whitespace* as caption and *View Whitespace* as prompt.
- Choose *View* - *Class Wizard* and associate the *View* menu with the *CTxwordsView* class.
- For the new ID_VIEW_WHITESPACE command select the Command message and click *Add Function*. Accept the default function name *OnViewWhitespace*.
- Click *Edit Code*. Class Wizard creates the handler function and opens the *TXWordsView.cpp* file.

The remaining steps are to fill the empty handler function with code that accesses the Text Control. To access the Text Control use the member function **CTXView::GetTextControl** and to view the whitespace characters use the member function **CTXTextControl::SetMode**. The following steps add the necessary code:

- Fill the command handler with the following line of code:

```
void CTxwordsView::OnViewWhitespace()
{
   GetTextControl()->SetMode(TF_SHOWWHITESPACE);
}
```

- To be able to use the **CTXTextControl** class add the following include statement at the top of *TXWordsView.cpp*:

```
#include "TXTextControl.h"
```

Compile and Run Your Application

- Verify that you have completed all steps exactly as they are documented here. (The sub-directory *Samples\VisualC\TXWords2* contains the code created in this chapter.)
- Hit F7 (or select *Build TXWords*.exe from the *Build* menu) to start the compilation process.

After starting the application you should now be able to format your inserted text with font and paragraph attributes and to insert tables, images and OLE objects. The *View* menu contains several commands for changing the zooming factor and the page view, the *Edit* menu has entries for searching and replacing text. You can also view whitespace characters with you manually inserted menu command.

Adding a Button Bar and a Status Bar

In this part you will add code to the starter application that is necessary to integrate Text Control's toolbars. With Text Control's Button Bar you can set text formatting attributes, like fonts and their size and styles. Text Control's Status Bar shows the state of several keyboard keys and information text like menu command descriptions.

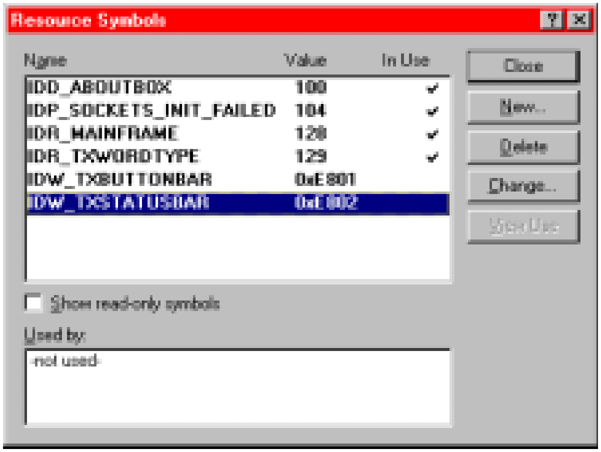
Add Member Variables to CMainFrame

- At the top of *MainFrm.h* add the following two *#include* statements:
```
#include "TXBBBar.h"
#include "TXSBBar.h"
```
- In the **CMainFrame** class declaration below add two protected members:
```
protected:
   CTXButtonBar m_wndBB;
   CTXStatusBar m_wndSB;
```

Add New Resources

Create two new resource IDs for the two bars:

- Select *View->Resource Symbols* from the menu, and click on *New*. In the dialog box that appears, enter IDW_TXBUTTONBAR as the name and 0xE801 as the value.
- Repeat those steps for a new resource symbol with the name IDW_TXSTATUSBAR and the value 0xE802. (You can choose any values for the resource symbols, as long as they are in the range 0xE800...0xE8FF.)



Name	Value	In Use
IDD_ABOUTBOX	100	✓
IDP_SOCKETS_INIT_FAILED	104	✓
IDR_MAINFRAME	128	✓
IDR_TXWORDTYPE	129	✓
IDW_TXBUTTONBAR	0xE801	
IDW_TXSTATUSBAR	0xE802	

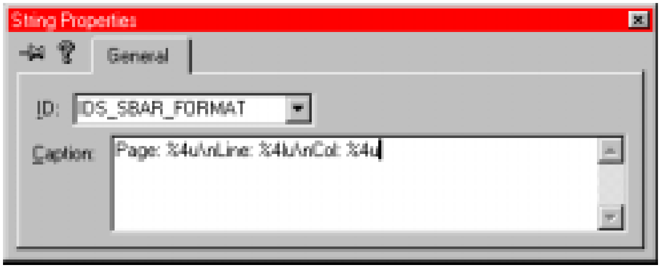
☐ Show read-only symbols

Used by:
-not used-

- Close the *Resource Symbols* dialog by clicking *Close*.

Add a new string to the application's string table:

- In the Workspace window select the *ResourceView* tab.
- Expand the *TXWords Resources* folder and the *String Table* folder.



ID: IDS_SBAR_FORMAT

Caption: Page: %4u\nLine: %4lu\nCol: %4u

- Double click on the *String Table* entry (not the folder), and create a new string resource by double-clicking on the last (empty) entry in the list of existing strings.
- Name the new string resource IDS_SBAR_FORMAT and give it the value:

  `Page: %4u\nLine: %4lu\nCol: %4u`

Create the Button Bar and the Status Bar Window

In **CMainFrame::OnCreate( )** add statements to create a button bar and a status bar. The resulting code should look like this (added lines are marked with ⌦):

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)
      return -1;

   if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT,
              WS_CHILD|WS_VISIBLE|CBRS_TOP|CBRS_GRIPPER
              |CBRS_TOOLTIPS|CBRS_SIZE_DYNAMIC)
            || !m_wndToolBar.LoadToolBar(TX_IDR_TXVIEW))
   {
      TRACE0("Failed to create toolbar\n");
      return -1;       // fail to create
   }

   // TODO: Delete these three lines if you don't
   // want the toolbar to be dockable
   m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
   EnableDocking(CBRS_ALIGN_ANY);
   DockControlBar(&m_wndToolBar);
⌦       if (!m_wndBB.Create(this,
⌦               IDW_TXBUTTONBAR,WS_CHILD|WS_VISIBLE|CBRS_TOP,
⌦               BBS_FLAT | BBS_FLATBUTTONS))
⌦       {
⌦          TRACE0("Failed to create button bar\n");
⌦          return -1;      // fail to create
⌦       }
⌦
⌦       m_wndBB.SetDefaultStrings();
⌦       m_wndBB.SetBarStyle(m_wndBB.GetBarStyle()
⌦          | CBRS_TOOLTIPS | CBRS_FLYBY);
⌦
⌦       // get format string for page, line and column
⌦       CString strSBFormat;
⌦       strSBFormat.LoadString(IDS_SBAR_FORMAT);
⌦
⌦       if (!m_wndSB.Create(this,
⌦            IDW_TXSTATUSBAR, WS_CHILD|WS_VISIBLE
⌦            |CBRS_SIZE_FIXED|CBRS_ALIGN_BOTTOM,
⌦            STS_LEFTALIGN|STS_NOBORDER, strSBFormat, NULL))
⌦       {
⌦          TRACE0("Failed to create status bar\n");
⌦          return -1;      // fail to create
⌦       }
        return 0;
     }
```

Make Your CMainFrame a CTXToolContainer

In *MainFrm.h* implement the following:

- At the top add the following *#include* statement:

  `#include "TXToolContainer.h"`

- In addition to **CMDIFrameWnd**, derive **CMainFrame** from **CTXToolContaine**r:

```
class CMainFrame : public CMDIFrameWnd , public
CTXToolContainer
```

- Add the declarations of the **CTXToolContainer** virtual methods to the class *CMainFrame*:

```
public:
   CTXButtonBar*  GetButtonBar();
   CTXStatusBar*  GetStatusBar();
```

- At the bottom of *MainFrm.h* add the inline implementation of the **CTXToolContainer** virtual methods:

```
inline
CTXButtonBar* CMainFrame::GetButtonBar()
{
   return &m_wndBB;
}

inline
CTXStatusBar* CMainFrame::GetStatusBar()
{
   return &m_wndSB;
}
```

Enable the Display of Menu Command Descriptions

Add the following declaration to *MainFrm.h (put it* in one of the *public* sections):

```
virtual CWnd* GetMessageBar();
```

Add the following code to *MainFrm.cpp*:

```
CWnd* CMainFrame::GetMessageBar()
{
   return (m_wndSB.GetSafeHwnd() ? &m_wndSB : NULL);
}
```

Compile and Run Your Application

- Verify that you have completed all steps exactly as they are documented here. (The sub-directory *Samples\VisualC\TXWords3* contains the code created in this chapter.)

- Hit F7 (or select *Build TXWords.exe* from the *Build* menu) to start the compilation process.

Working with File Formats

Each application needs its own format to save its own application-specific data. In addition the most applications support other formats to load or exchange data with other applications. In this part of the tutorial you will add code that enables the application to load and save its data in its own format and in all the text formats that Text Control supports.

Define the Application's Document Format

In part 1 of this tutorial, Step 8, you added a line of code in the **CTXWordsDoc::Serialize** function. This added line loads or saves the text and its formatting attributes of your documents. This function can be further extended to load and save additional data that is specific to your application. The Application Wizard has created *TODO* comments to show where to add storing and loading code.

Currently when you use the *File Open* and *File Save* dialog boxes your documents have no type description and no file extension. To give your documents a type description and a file extension perform the following steps:

- In the Workspace window select the *ResourceView* tab. Open the string table and double-click the IDR_TXWORDTYPE entry.

This is the document type string consisting of seven substrings, seperated through *\n* characters. See **CDocTemplate::GetDocString** in the MFC documentation for more information about the meanings of these substrings.

- Extend this string to the following:

```
\nTXWord\nTXWord\nTX Words Format (*.txw)\n.txw\n
TXWords.Document\nTXWord Document
```

Now your document's type description is *TX Words Format (*.txw)* and your document's file extension is *.txw*. This description string now appears in the *File Open* and *File Save* dialog boxes.

Load and Save Additional Text Formats

Text Control currently supports six text formats:

- Its own native format (*.TX).
- Plain text (*.TXT).
- Plain Unicode text (*.TXT).
- Rich Text Format (*.RTF).
- Hyper Text Markup Language (*.HTM, *.HTML).
- Microsoft Word Format (.DOC).

To load and save documents using one of these formats, perform the following step:

- In **CTXWordsApp::InitInstance( )** add the following line of code before the document's template is created:

```
CTXDoc::EnableFileFormats(this);
```

To support for example only Rich Text Format, perform the following steps:

- At the top of *TXWords.cpp* add the following include statement:

```
#include "TXFileFormats.h"
```

- In **CTXWordsApp::InitInstance( )** add the following line of code:

```
CTXDoc::EnableFileFormats(this, FORMAT_RTF);
```

Compile and Run Your Application

- Verify that you have completed all steps exactly as they are documented here. (The sub-directory *Samples\VisualC\TXWords4* contains the code created in this chapter.)
- Hit F7 (or select *Build TXWords.exe* from the *Build* menu) to start the compilation process.

Reference

Using the Text Control Class Library

Headers and Footers

Using Headers and Footers

Headers and footers can only be used when a page size has been set with **CTXTextControl::SetPageSize**. Headers and footers are only visible on the screen if one of the page view modes (TF_PAGEVIEW or TF_EXTPAGEVIEW) has been selected. View modes can also be set with **CTXTextControl::SetPageSize**.

Headers and/or footers must be enabled with **CTXTextControl::HFEnable**. This function 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 **CTXTextControl:: HFActivate** 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. Override **CTXNotifyHandler::OnTnHFActivated** and **CTXNotifyHandler::OnTnHFDeActivated** to handle these notifications.

CTXTextControl::HFEnable 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 **CTXTextControl::HFGetPosition** and **CTXTextControl::HFSetPosition** 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. **CTXTextControl::HFGetEnabled** 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, use **CTXTextControl::HFDisable**.

Programming Headers and Footers

Headers and footers are seperate 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 **CTXTextControl::TableInsert**, 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 **CTXTextControl::HFSelect** to use a certain message for a certain text part. For example the following code returns the length of a header's text:

```
LONG lTextSize;
HFSelect(TF_HF_HEADER);
lTextSize = GetTextLength();
HFSelect(TF_HF_AUTO);
```

The first line selects the header, independent of the current input focus, the second line gets the length 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 **HFSelect** calls.

Almost all member functions of the **CTXTextControl** class can be used in this way with some exceptions. The following is a complete list of these exceptions:

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

- all functions that handle scrolling
- operations with headers and footers
- printing operations
- operations with chains of linked Text Controls

2. The following member functions of **CTXTextControl** always affect all text parts (main text, headers and footers), independent of the currently selected part. These functions are:

- **Get/SetBackgroundColor**
- **Get/SetLanguage**
- **Get/SetCaretExt**
- **Get/SetMode**
- **GetSupportedFonts**
- **GetSupportedSizes**
- **GetDevice**
- **SetDevicePrinter/Screen/Standard**
- **Get/SetZoom**

3. The following member functions of **CTXTextControl** can only be used with headers and footers after selection with **HFSelect**:

- **LoadFile**
- **LoadFromMemory**
- **ResetContents**
- **SaveFile**
- **SaveToMemory**

Tables

Using Tables

Tables can be inserted into a Text Control either with **CTXTextControl::TableInsert** or as part of a document formatted with the RTF, HTML or Microsoft Word 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 **CTXTextControl::TableDeleteLines**. 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 a built-in dialog box can be opened with the **CTXTextControl::TableAttrDialog**. 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, use **CTXTextControl::TableIsPossible**. A second way to change a table's attributes is to use **CTXTextControl::TableGetAttr** and **CTXTextControl::TableSetAttr**. These functions need a table identifier and a row and column number as parameters.

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 to which the input position belongs.Then the cells' positions and extensions can be changed with a built-in mouse interface.

Programming with Table Identifiers

Like OLE objects, images and marked text fields each table has an unique identifier which is set by Text Control. This identifier is returned from **CTXTextControl::TableInsert**. A programmer can select an own identifier for each table with the *nTableID* parameter of **CTXTextControl::TableInsert**. Selecting an own identifier is not necessary but recommended when a table's text or attributes are to be changed from the programmer 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 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. Override **CTXNotifyHandler::OnTnTableCreated** to define a new user-defined table identifier for this new table.

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. These identifiers can also be changed with **CTXNotifyHandler::OnTnTableCreated**.

When tables are imported with **CTXTextControl::LoadFile** or **CTXTextControl::LoadFromMemory**, TN_TABLE_CREATED notifications are sent only when the *bReplaceSel* parameter is set to TRUE 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. Override **CTXNotifyHandler::OnTnTableDeleted** to perform actions in this case.

Several member functions of the **CTXTextControl** class accept table identifiers. These identifiers can be either Text Control defined or user-defined. If more than one table with a certain identifier exists, these functions perform the operation with the originally inserted table. In chains of linked windows these functions can be called for any Text Control in the chain regardless of which Text Control currently contains the table.

Marked Text Fields

Using Marked Text Fields

A set of member functions of the **CTXTextControl** class 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, 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 **CTXTextControl::FieldInsert** to define a marked text field. The whole communication works with the unique numbers returned by this function. The current text can be changed or retrieved with **CTXTextControl::FieldChangeText** and **CTXTextControl::FieldGetText**, **CTXTextControl::FieldGetPosition** retrieves the current text position of a field. Special attributes can be selected with **CTXTextControl::Field HasAttr** and **CTXTextControl::FieldSetAttr**. 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. All of these notification messages can be handled by overriding the appropriate **CTXNotifyHandler** member function **OnTnFieldxxx**.

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 can be defined with **CTXTextControl::FieldInsert** or can be altered with **CTXTextControl::FieldSetAttr**.

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 **CTXTextControl::SetCaretExt**. 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 **CTXTextControl::FieldSetData**. 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 **CTXTextControl::FieldSetData**.

Other parts of the program can use **CTXTextControl::FieldGetData** 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, **CTXTextControl::FieldGetData** can be used with the field identifier, which has been sent with the TN_FIELD_CLICKED notification message. **CTXTextControl::FieldGetData** 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 **CTXTextControl::FieldSetData** 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

Special types of marked text fields are fields that display the current page number and that provide support for hypertext links. These fields can be inserted with **CTXTextControl::InsertPageNumber, CTXTextControl::InsertLink**, and **CTXTextControl::InsertTarget**. **CTXTextControl:: FieldGetType** returns a type identifier for these fields. The following types are possible:

Type	Description
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.

All of these fields have the same general properties as standard marked text fields with the following exceptions: Fields of the type FT_LINKTARGET 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 fields which are inserted with **CTXTextControl::InsertLink** (types FT_EXTERNALLINK and FT_INTERNALLINK), Text Control manages 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. With **CTXTextControl::ChangeLink** the target of a link can be altered, **CTXTextControl::GetLinkLocation** retrieves the information to where the link points.

Targets in documents can be inserted with **CTXTextControl::InsertTarget**. These fields have the type

FT_LINKTARGET and are identified through names. **CTXTextControl::ChangeTarget** alters this name and **CTXTextControl::GetTargetName** asks for the name of a certain target.

When the user clicks on a field of the type FT_EXTERNALLINK or FT_INTERNALLINK a TN_FIELD_LINKCLICKED notification is sent. **CTXNotifyHandler::OnTnFieldLinkClicked** informs the application about the type of hypertext link (external or internal) and about the information to where the link points.

CTXTextControl::FieldGoto can be used to scroll to an internal link position and **CTXTextControl::FieldGetNext** can be used to enumerate all fields of a certain type.

Inserting OLE Objects

Insertion

OLE objects can be inserted into a Text Control document with **CTXTextControl::InsertOleObject**, **CTXTextControl:: InsertOleProgID**, **CTXTextControl::InsertOleFile** or **CTXTextControl::InsertOleLinkFile**.

CTXTextControl::InsertOleObject opens the OLE built-in *Insert Object* dialog box where the user can select one of the system-registered OLE servers. Depending on the specified insertion mode, 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 **CTXTextControl::ObjGetAttr**. 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 control bars.

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 overlayed 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.

Loading and Saving

OLE objects are integrated into the Text Control's text format like any other objects. Therefore all functions that support loading and saving 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 specified with **CTXTextControl::PrintPage**.

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.

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 **CTXTextControl::SetLanguage** this setting can be altered independent of the system language. The documentation of **CTXTextControl::SetLanguage** lists all currently available built-in languages. To alter the language of the Button Bar and Status Bar use **CTXButtonBar::SetLanguage** and **CTXStatusBar::SetLanguage**.

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

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

Furthermore Microsoft Visual C++ project files are contained and 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 **CTXTextControl::FontDialog**, **CTXTextControl::ParagraphDialog** and **CTXTextControl::TableAttrDialog**.
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 TXRES\BMP subdirectory.

To avoid conflicts with other programs that also use their 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 use the file name of the resource library including this path with **CTXTextControl::SetLanguage**.

At runtime Text Control determines resources in the following way:

1. When **CTXTextControl::SetLanguage** 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 **CTXTextControl::SetLanguage** has been called with an identifier of a built-in language Text Control displays resources in this language independent of the system language.

3. When **CTXTextControl::SetLanguage** has been called with a file name of a resource library Text Control tries to load the resources from this library. Previously set 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 functions of these classes.

Text Control Classes

CTXButtonBar

#include <TXBBBar.h>

The **CTXButtonBar** class provides the functionality of Text Control's Button Bar. This is a seperate child window that can be created in the client area of any parent window. It provides buttons and combo boxes for setting font and paragraph attributes. A button bar can be specified as a parameter of a Ruler Bar's **Create** function. This Ruler Bar then uses the tabulator type settings of the specified Button Bar.

To show or change the font and paragraph attributes of a certain Text Control with a Button Bar, the Button Bar must be connected with this Text Control. To perform this, use **TXTextControl::ConnectToolBar**. The Button Bar then displays the settings of the connected Text Control after the Text Control has obtained the input focus. Several Text Controls can be connected with a single Button Bar. To disconnect a Button Bar use **CTXTextControl::DisconnectToolBar**.

To create a Button Bar, first call the constructor **CTXButtonBar** to construct the **CTXButtonBar** object, then call the **Create** member function to create the window and attach it to the **CTXButtonBar** object.

Member Functions

CTXButtonBar::Create

Description: This member function creates a Button Bar child window. Button Bar child windows must be created in two steps. First call the constructor which creates the **CTXButtonBar** object. Then call **Create**, which creates the Button Bar child window and attaches it to **CTXButtonBar**.

Syntax: **BOOL Create(CWnd*** *pParentWnd,* **UINT** *nID,*
DWORD *dwStyle* **= WS_CHILD | WS_VISIBLE | CBRS_TOP,**
DWORD *dwButtonBarStyle* = **BBS_FLAT | BBS_FLATBUTTONS);**

Parameter	Description
pParentWnd	Specifies the Button Bar's parent window. It must not be **NULL**.
nID	Specifies the Button Bar's identifier. This identifier must be in the range IDW_CONTROLBAR_FIRST+1 to IDW_CONTROLBAR_LAST. See Microsoft Technical Note 31 for more information.
dwStyle	Specifies the Button Bar's window styles.
dwButtonBarStyle	Specifies additional Button Bar styles. See the following Values section for possible settings.

Return Value: The function returns **TRUE** if the Button Bar window could be created, otherwise it returns **FALSE**.

Values: The following list contains possible values for the *dwButtonBarStyle* parameter:

Style	Meaning
BBS_3D	Paints the Button Bar with three-dimensional effects.
BBS_FLAT	Paints the Button Bar without visual effects.
BBS_3DBUTTONS	Paints the Button Bar's buttons with three-dimensional effects.
BBS_FLATBUTTONS	Paints the Button Bar's buttons without visual effects.

See also: **CTXButtonBar::CTXButtonBar**

CTXButtonBar::CTXButtonBar

Description: Constructs a **CTXButtonBar** object.

See also: **CTXButtonBar::Create**

CTXButtonBar::SetLanguage

Description:	This member function sets the language which the Button Bar uses to display its buttons. The language is specified either through an identifier or through the filename of a resource library.

Syntax:	**BOOL SetLanguage(UINT** *nLang***);**
	BOOL SetLanguage(const CString& *strLang***);**

Parameter	Description
nLang	Specifies a language identifier. For possible values see **CTXTextControl::SetLanguage**.
strLang	Specifies the filename including its full path of a resource library. See the chapter *Using the Text Control Class Library - Resources* for more information about creating a resource library.

Return Value: The return value is **FALSE** if an error has occurred or if the specified language has already been set, otherwise it is **TRUE**.

CTXDoc

#include <TXDoc.h>

The **CTXDoc** class provides the basic functionality for user-defined document classes in applications using Text Control and the **TXView** class. The **CTXDoc** class provides all the standard operations available through MFC's document/view architecture. Additionally it supports loading and saving the document's text in other text formats like RTF, HTML or Microsoft Word. To implement a Text Control document in a typical application, perform the following:

- Derive a class from **CTXDoc**.
- Define member variables for your application-specific data.
- Overwrite the **CObject::Serialize** function in your document class to write and read the application-specific data.
- In your document's **Serialize** function, call the base class's implementation **CTXDoc::Serialize** to write and read the text contained in the Text Control of the associated **CTXView** class.

Member Functions

CTXDoc::EnableFileFormats

Description: This member function enables all the text formats that Text Control supports. These formats can then be accessed through the *File Open* and *File Save* dialog boxes.

Syntax: **BOOL EnableFileFormats(CWinApp*** *pApp*, **DWORD** *dwFormatMask* = **FORMAT_ALL);**

Parameter	**Description**
pApp	Points to the single **CWinApp** object for the application.
dwFormatMask	Specifies the text formats that the application wants to support. See the following Values section for possible values.

Return Value: The function returns **TRUE** if the specified text formats could be enabled. Otherwise it returns **FALSE**.

Values: The following lists possible values for the *dwFormatMask* parameter:

Value	Meaning
FORMAT_TX	Text Control's native format (*.TX)
FORMAT_ANSI	Plain text (*.TXT)
FORMAT_UNICODE	Plain Unicode text (*.TXT)
FORMAT_RTF	Rich Text Format (*.RTF)
FORMAT_DOC	Microsoft Word Format (*.DOC)
FORMAT_HTM	Hyper Text Markup Language (*.HTM)
FORMAT_ALL	All supported formats.

CTXNotifyHandler

#include <TXNotifyHandler.h>

The **CTXNotifyHandler** class implements a member function for each notification message that a Text Control windows sends to its parent window. Derive your handler from this class and overwrite the functions that belong to the notification you want to handle. A pointer to the derived handler can be specified as a parameter of **CTXTextControl::Create**.

All of the following member functions have the same first parameter which is a pointer to the **CTXTextControl** object that causes the notification. All member functions also have the same boolean return value. If a function returns **FALSE,** which is a handler's default implementation, further processing continues. Otherwise, if a function returns **TRUE** further processing stops.

CTXNotifyHandler Class Members

General Notifications	
OnTnChanged	Called after the contents of a Text Control has been changed.
OnTnDoubleClicked	Called after a Text Control has been double-clicked.
OnTnErrCode	Called after an error has occurred.
OnTnHExpand	Called after a Text Control has automatically expanded its window width.
OnTnHMoved	Called after a Text Control's window has been moved horizontally relative to its parent window.
OnTnHScroll	Called after the horizontal scroll position has been changed.
OnTnKeyStateChanged	Called after the character insertion mode or after the state of the NUMLOCK or CAPSLOCK key has been changed

OnTnKillFocus	Called after the Text Control has lost the input focus.
OnTnModeChanged	Called after one of the modes has been changed.
OnTnParaChanged	Called after the character input position has been moved to another paragraph.
OnTnPosChanged	Called after the character input position has been moved to another position.
OnTnSetFocus	Called after a Text Control has obtained the input focus.
OnTnVExpand	Called after a Text Control has automatically expanded its window height.
OnTnVScroll	Called after the vertical scroll position has been changed.
OnTnZoomed	Called after a Text Control has been zoomed.

Formatting Changes

OnTnCharFormatChanged	Called after character attributes of the currently selected text have been changed.
OnTnPageFormatChanged	Called after page attributes have been changed.
OnTnParaFormatChanged	Called after paragraph attributes of the currently selected text have been changed.

Image, Window and OLE Object Notifications

OnTnObjClicked	Called after an object has been clicked.
OnTnObjCreated	Called after a new object has been pasted from the clipboard.
OnTnObjDblClicked	Called after an object has been double-clicked.
OnTnObjDeleted	Called after an object has been deleted.
OnTnObjMoved	Called after an object has been moved.

OnTnObjSized	Called after an object has been sized.

Marked Text Field Notifications

OnTnFieldChanged	Called after the text of a marked text field has been changed.
OnTnFieldClicked	Called after a user has clicked on a marked text field.
OnTnFieldCreated	Called after a new marked text field has been pasted from the clipboard.
OnTnFieldDblClicked	Called after a user has double-clicked on a marked text field.
OnTnFieldDeleted	Called after a marked text field has been deleted.
OnTnFieldEntered	Called after the current input position has been moved to a position that belongs to a marked text field.
OnTnFieldLeft	Called after the current input position has been moved to a position that does not belong to the marked text field at the previous input position.
OnTnFieldSetCursor	Called when the cursor is being moved over a marked text field.

Hypertext Link Notifications

OnTnFieldLinkClicked	Called after the user has clicked on a marked text field that represents the source of a hypertext link.

Table Notifications

OnTnTableCreated	Called after after a new table has been pasted from the clipboard.
OnTnTableDeleted	Called after a table has been deleted.

Header and Footer Notifications

OnTnHFActivated	Called after a header or footer has been activated.

OnTnHFDeActivated	Called after a header or footer has been deactivated.

Member Functions

CTXNotifyHandler::OnTnChanged

Description: The specified Text Control calls this member function after its contents have been changed.

Syntax: **BOOL OnTnChanged(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnCharFormatChanged

Description: The specified Text Control calls this member function after character attributes of its currently selected text have been changed.

Syntax: **BOOL OnTnCharFormatChanged(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnDoubleClicked

Description: The specified Text Control calls this member function after it has been double-clicked.

Syntax: **BOOL OnTnDoubleClicked(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnErrCode

Description: The specified Text Control calls this member function after an error has occurred.

Syntax: **BOOL OnTnErrCode(CTXTextControl*** *pTX***, WORD** *wModule,* **WORD** *wErrCode,* **WORD** *wGroupCode***);**

Parameter	Description
wModule	Specifies a module number. It can be one of the following values: **Value: Meaning:**

01	The error has occurred in TX32.DLL or one of the text filters.
03	The error has occurred in IC32.DLL.
04	The error has occurred in an image filter module.
05	The error has occurred in TXOBJ32.DLL.

wErrCode Specifies an error code. See *Error Codes* for possible values.

wGroupCode Specifies a group code. Possible group codes are listed in the following values section.

Values: The following is a list of possible error group codes:

Code (Value)	**Description**
DBS_E_UNGROUPED (00)	A special error condition has occurred. See the description of the error code for more information.
DBS_E_OUTOFMEMORY (01)	Not enough storage is available to complete the operation.
DBS_E_NOMEMORYACCESS (02)	Invalid access to a memory location.
DBS_E_UNEXPECTED (03)	Unexpected failure.
DBS_E_FILEIO (04)	A file read/write operation cannot be performed.
DBS_E_CLIPBOARD (06)	A clipboard operation cannot be performed. The clipboard cannot be opened or emptied or clipboard data cannot be accessed.
DBS_E_DLLNOTLOADED (07)	An operation cannot be performed because a helper DLL or filter needed for the operation cannot be found or loaded.

DBS_E_DLLINCOMPATIBLE (08)	An operation cannot be performed because a helper DLL or Filter needed for the operation is too old.
DBS_E_DLLOBSOLETE (09)	A helper DLL or Filter DLL needed for the operation is obsolete but can be used for the operation.
DBS_E_INVALIDARG (0A)	One or more arguments are invalid.
DBS_E_NOTIMPL (0B)	The feature is not implemented.
DBS_E_INVALIDFORMAT (0C)	An operation cannot be performed because data has an invalid format.

CTXNotifyHandler::OnTnFieldChanged

Description: The specified Text Control calls this member function after the text of a marked text field has been changed.

Syntax: **BOOL OnTnFieldChanged(CTXTextControl*** *pTX*, **UINT** *nFieldID***);**

Parameter	Description
nFieldID	Is the identifier of the field the text of which has been changed.

CTXNotifyHandler::OnTnFieldClicked

Description: The specified Text Control calls this member function after a user has clicked on a marked text field.

Syntax: **BOOL OnTnFieldClicked(CTXTextControl*** *pTX*, **UINT** *nFieldID***);**

Parameter	Description
nFieldID	Is the identifier of the field that has been clicked.

CTXNotifyHandler::OnTnFieldCreated

Description: The specified Text Control calls this member function after a new marked text field has been pasted from the clipboard.

Syntax: **BOOL OnTnFieldCreated(CTXTextControl*** *pTX,* **UINT** *nFieldID***);**

Parameter	Description
nFieldID	Is the identifier of the field that has been created.

CTXNotifyHandler::OnTnFieldDblClicked

Description: The specified Text Control calls this member function after a user has double-clicked on a marked text field.

Syntax: **BOOL OnTnFieldDblClicked(CTXTextControl*** *pTX,* **UINT** *nFieldID***);**

Parameter	Description
nFieldID	Is the identifier of the field that has been double-clicked.

CTXNotifyHandler::OnTnFieldDeleted

Description: The specified Text Control calls this member function after a marked text field has been deleted. It does not call the function when the field is deleted because the window is completely destroyed or because the complete text contents are exchanged.

Syntax: **BOOL OnTnFieldDeleted(CTXTextControl*** *pTX,* **UINT** *nFieldID***);**

Parameter	Description
nFieldID	Is the identifier of the field that has been deleted.

CTXNotifyHandler::OnTnFieldEntered

Description: The specified Text Control calls this member function after its current input position has been moved to a position that belongs to a marked text field. The function is only called if the current input position has been moved using the keyboard. If the current input position has been moved with the mouse **CTXNotifyHandler::OnTnFieldClicked** is called.

Syntax: **BOOL OnTnFieldEntered(CTXTextControl*** *pTX,* **UINT** *nFieldID***);**

Parameter	Description
nFieldID	Is the identifier of the field that has been entered.

CTXNotifyHandler::OnTnFieldLeft

Description: The specified Text Control calls this member function after its current input position has been moved to a position that does not longer belong to the marked text field at the previous input position.

Syntax: **BOOL OnTnFieldLeft(CTXTextControl*** *pTX,* **UINT** *nFieldID***);**

Parameter	Description
nFieldID	Is the identifier of the field that has been left.

CTXNotifyHandler::OnTnFieldLinkClicked

Description: The specified Text Control calls this member function after the user has clicked on a marked text field that represents the source of a hypertext link.

Syntax: **BOOL OnTnFieldLinkClicked(CTXTextControl*** *pTX,* **UINT** *nFieldID,* **UINT** *nFieldType,* **const CString&** *strLink***);**

Parameter	Description
nFieldID	Is the identifier of the field that has been clicked.
nFieldType	Specifies the type of the clicked marked text field. It can be one of the following values:

Type	Description

	FT_EXTERNALLINK	The field is the source of a hypertext link to a location outside of the document.
	FT_INTERNALLINK	The field is the source of a hypertext link to a location in the same document.
strLink	Specifies the location to where the link points.	

CTXNotifyHandler::OnTnFieldSetCursor

Description: The specified Text Control calls this member function while the cursor is moved over a marked text field.

Syntax: **BOOL OnTnFieldSetCursor(CTXTextControl*** *pTX,* **UINT** *nFieldID,* **BOOL&** *bCursorSet***);**

Parameter	Description
nFieldID	Is the identifier of the field over which the cursor is moved.
bCursorSet	If this parameter retrieves **FALSE**, which is the default value, Text Control sets the cursor to the vertical arrow cursor. Otherwise if this parameter retrieves **TRUE** the Text Control does not set a cursor.

CTXNotifyHandler::OnTnHExpand

Description: The specified Text Control calls this member function after it has automatically expanded its window width.

Syntax: **BOOL OnTnHExpand(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnHFActivated

Description: The specified Text Control calls this member function after a header or footer has been activated.

Syntax: **BOOL OnTnHFActivated(CTXTextControl*** *pTX,* **UINT** *nHeaderFooter***);**

Parameter	Description
nHeaderFooter	Specifies which kind of header or footer has been activated. It can be one of the following values:

Value	Description
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.

CTXNotifyHandler::OnTnHFDeActivated

Description: The specified Text Control calls this member function after a header or footer has been deactivated.

Syntax: **BOOL OnTnHFDeActivated(CTXTextControl*** *pTX,* **UINT** *nHeaderFooter***);**

Parameter	Description
nHeaderFooter	Specifies which kind of header or footer has been deactivated. It can be one of the following values:

Value	Description
TF_HF_HEADER	A header has been deactivated.
TF_HF_1STHEADER	The special header for the first page has been deactivated.
TF_HF_FOOTER	A footer has been deactivated.
TF_HF_1STFOOTER	The special footer for the first page has been deactivated.

CTXNotifyHandler::OnTnHMoved

Description: The specified Text Control calls this member function after its window has been moved horizontally relative to its parent window.

Syntax: **BOOL OnTnHMoved(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnHScroll

Description: The specified Text Control calls this member function after its horizontal scroll position has been changed.

Syntax: **BOOL OnTnHScroll(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnKeyStateChanged

Description: The specified Text Control calls this member function after the character insertion mode or after the state of the NUMLOCK or CAPSLOCK key has been changed.

Syntax: **BOOL OnTnKeyStateChanged(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnKillFocus

Description: The specified Text Control calls this member function after it has lost the input focus.

Syntax: **BOOL OnTnKillFocus(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnModeChanged

Description: The specified Text Control calls this member function after one of its modes has been changed. See **CTXTextControl::SetMode** for a list of possible modes.

Syntax: **BOOL OnTnModeChanged(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnObjClicked

Description: The specified Text Control calls this member function after one of its inserted images, windows or OLE objects has been clicked.

Syntax: **BOOL OnTnObjClicked(CTXTextControl*** *pTX*, **UINT** *nObjID***);**

Parameter	Description
nObjID	Is the identifier of the object that has been clicked.

CTXNotifyHandler::OnTnObjCreated

Description: The specified Text Control calls this member function after a new image, window or OLE object has been pasted from the clipboard.

Syntax: **BOOL OnTnObjCreated(CTXTextControl*** *pTX*, **UINT** *nObjID***);**

Parameter	Description
nObjID	Is the identifier of the object that has been created.

CTXNotifyHandler::OnTnObjDblClicked

Description: The specified Text Control calls this member function after one of its inserted images, windows or OLE objects has been double-clicked.

Syntax: **BOOL OnTnObjDblClicked(CTXTextControl*** *pTX*, **UINT** *nObjID***);**

Parameter	Description
nObjID	Is the identifier of the object that has been double-clicked.

CTXNotifyHandler::OnTnObjDeleted

Description: The specified Text Control calls this member function after one of its inserted images, windows or OLE objects has been deleted.

Syntax: **BOOL OnTnObjDeleted(CTXTextControl*** *pTX*, **UINT** *nObjID***);**

Parameter	Description
nObjID	Is the identifier of the object that has been deleted.

CTXNotifyHandler::OnTnObjMoved

Description: The specified Text Control calls this member function after one of its inserted images, windows or OLE objects has been moved relative to its client area.

Syntax: **BOOL OnTnObjMoved(CTXTextControl*** *pTX,* **UINT** *nObjID***);**

Parameter	Description
nObjID	Is the identifier of the object that has been moved.

CTXNotifyHandler::OnTnObjSized

Description: The specified Text Control calls this member function after one of its inserted images, windows or OLE objects has been sized.

Syntax: **BOOL OnTnObjSized(CTXTextControl*** *pTX,* **UINT** *nObjID***);**

Parameter	Description
nObjID	Is the identifier of the object that has been sized.

CTXNotifyHandler::OnTnPageFormatChanged

Description: The specified Text Control calls this member function after page attributes have been changed.

Syntax: **BOOL OnTnPageFormatChanged(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnParaChanged

Description: The specified Text Control calls this member function after its character input position has been moved to another paragraph.

Syntax: **BOOL OnTnParaChanged(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnParaFormatChanged

Description: The specified Text Control calls this member function after the paragraph attributes of its currently selected text have been changed.

Syntax: **BOOL OnTnParaFormatChanged(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnPosChanged

Description: The specified Text Control calls this member function after its character input position has been moved.

Syntax: **BOOL OnTnPosChanged(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnSetFocus

Description: The specified Text Control calls this member function after it has obtained the input focus.

Syntax: **BOOL OnTnSetFocus(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnTableCreated

Description: The specified Text Control calls this member function after a new table has been pasted from the clipboard.

Syntax: **BOOL OnTnTableCreated(CTXTextControl*** *pTX,* **UINT&** *nTableID***);**

Parameter	Description
nTableID	Is the identifier of the table that has been created. It can be changed to a user-defined value. This value must be between 10 and 0x7FFF.

CTXNotifyHandler::OnTnTableDeleted

Description: The specified Text Control calls this member function after a table has been deleted.

Syntax: **BOOL OnTnTableDeleted(CTXTextControl*** *pTX*, **UINT** *nTableID***);**

Parameter	Description
nTableID	Is the identifier of the table that has been deleted.

CTXNotifyHandler::OnTnVExpand

Description: The specified Text Control calls this member function after it has automatically expanded its window height.

Syntax: **BOOL OnTnVExpand(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnVScroll

Description: The specified Text Control calls this member function after its vertical scroll position has been changed.

Syntax: **BOOL OnTnVScroll(CTXTextControl*** *pTX***);**

CTXNotifyHandler::OnTnZoomed

Description: The specified Text Control calls this member function after it has been zoomed.

Syntax: **BOOL OnTnZoomed(CTXTextControl*** *pTX***);**

CTXRulerBar

#include <TXRLBar.h>

The **CTXRulerBar** class provides the functionality of the Ruler Bar. This is a seperate child window that can be created in the client area of any parent window. It provides a ruler with handles to change paragraph indents or table borders. Tabulators can be set or deleted. To define types for tabulators, the Ruler Bar can be connected with a Button Bar that contains buttons for tabulator types.

To show or change the paragraph attributes of a certain Text Control with a Ruler Bar, the Ruler Bar must be connected with this Text Control. To perform this, use **CTXTextControl::ConnectToolBar**. The Ruler Bar then displays the settings of the connected Text Control after the Text Control has obtained the input focus. Several Text Controls can be connected with a single Ruler Bar. To disconnect a Ruler Bar use **CTXTextControl::DisconnectToolBar**.

To create a Ruler Bar, first call the constructor **CTXRulerBar** to construct the **CTXRulerBar** object, then call the **Create** member function to create the window and attach it to the **CTXRulerBar** object.

The **CTXView** class automatically creates a Text Control with a connected Ruler Bar.

Member Functions

CTXRulerBar::Create

Description: This member function creates a Ruler Bar child window. Ruler Bar child windows must be created in two steps. First call the constructor which creates the **CTXRulerBar** object. Then call **Create**, which creates the Ruler Bar child window and attaches it to **CTXRulerBar**.

Syntax:

BOOL Create(CWnd* *pParentWnd,* **UINT** *nID,*
DWORD *dwStyle* **= WS_CHILD | WS_VISIBLE | CBRS_TOP,**
CTXButtonBar* *pButtonBar* = **NULL**,
DWORD *dwRulerBarStyle* = **RS_ALLPARTS);**

Parameter	Description
pParentWnd	Specifies the Ruler Bar's parent window. It must not be **NULL**.
nID	Specifies the Ruler Bar's identifier. This identifier must be in the range IDW_CONTROLBAR_FIRST+1 to IDW_CONTROLBAR_LAST. See Microsoft Technical Note 31 for more information.
dwStyle	Specifies the Ruler Bar's window styles.
pButtonBar	Specifies a Button Bar. The Ruler Bar uses the tabulator style stetting of this Button Bar to select the tabulator style of a newly created tabulator. If *pButtonBar* is zero all newly created tabulators are left-aligned.
dwRulerBarStyle	Specifies additional Ruler Bar styles. See the following Values section for possible settings.

Return Value: The function returns **TRUE** if the Ruler Bar window could be created, otherwise it returns **FALSE**.

Values: The following list contains possible values for the *dwRulerBarStyle* parameter:

Style	Meaning
RS_ALLPARTS	The Ruler Bar displays all its elements.
RS_FIRSTINDENT	The Ruler Bar displays a mark for the additional indent of the first line.
RS_INDENTS	The Ruler Bar displays marks for all indents.
RS_LEFTINDENT	The Ruler Bar displays a left indent mark.
RS_POSITION	The Ruler Bar displays the current position when moving a tabulator or an indent mark.
RS_RIGHTINDENT	The Ruler Bar displays a right indent mark.
RS_TABLECOL	The Ruler Bar displays table columns if the current input position is in a table.
RS_TABULATORS	The Ruler Bar displays tabulator settings.

See also: **CTXRulerBar::CTXRulerBar**

CTXRulerBar::CTXRulerBar

Description: Constructs a **CTXRulerBar** object.

See also: **CTXRulerBar::Create**

CTXStatusBar

#include <TXSBBar.h>

The **CTXStatusBar** class provides the functionality of Text Control's Status Bar. This is a seperate child window that can be created in the client area of any parent window. It provides a row of text output panes that show the status of the NUM LOCK and CAPS LOCK keys, the character insertion mode, the current zooming factor and the page, line and column number of the current text input position. Furthermore, it can display message lines for example menu help-message lines that briefly explain a selected menu command.

To display the status of a certain Text Control, a Status Bar must be connected with this Text Control. To perform this use **CTXTextControl::ConnectToolBar**. The Status Bar then displays the status of the connected Text Control after this Text Control has obtained the input focus. Several Text Controls can be connected with a single Status Bar. To disconnect a Status Bar use **CTXTextControl::DisconnectToolBar**.

To create a Status Bar, first call the constructor **CTXStatusBar** to construct the **CTXStatusBar** object, then call the **Create** member function to create the window and attach it to the **CTXStatusBar** object.

Member Functions

CTXStatusBar::Create

Description: This member function creates a Status Bar child window. Status Bar child windows must be created in two steps. First call the constructor which creates the **CTXStatusBar** object. Then call **Create**, which creates the Status Bar child window and attaches it to **CTXStatusBar**.

Syntax: **BOOL Create(CWnd*** *pParentWnd*, **UINT** *nID*,
DWORD *dwStyle* **= WS_CHILD | WS_VISIBLE |**
CBRS_SIZE_FIXED | CBRS_ALIGN_BOTTOM,
DWORD *dwStatusBarStyle* = **STS_LEFTALIGN |**

STS_NOBORDER,
const CString& *strFormat* = **"",**
CFont* *pFont* **= NULL);**

Parameter	**Description**
pParentWnd	Specifies the Status Bar's parent window. It must not be **NULL**.
nID	Specifies the Status Bar's identifier. This identifier must be in the range IDW_CONTROLBAR_FIRST+1 to IDW_CONTROLBAR_LAST. See Microsoft Technical Note 31 for more information.
dwStyle	Specifies the Status Bar's window styles.
dwStatusBarStyle	Specifies additional Status Bar styles. See the following Values section for possible settings.
strFormat	Specifies a format string. The Status Bar gets the information from this string how to display the page, line and column number of the current text input position. See the following Remarks section for more information. If an empty string is specified the Status Bar displays only the numbers.
pFont	Specifies a font object. The Status Bar uses this font to display its information.

Return Value: The function returns **TRUE** if the Status Bar window could be created, otherwise it returns **FALSE**.

Remarks: The string specified through *strFormat* must have the following form:
[*ptext*]%*format*\n[*ltext*]%*format*\n[*ctext*]%*format*
The parts in brackets are optional, all other parts are required. The various parts have the following meanings:

Part	**Meaning**
ptext	Text for the Status Bar pane that displays the page number of the current text input position.

ltext	Text for the Status Bar pane that displays the line number of the current text input position.
ctext	Text for the Status Bar pane that displays the column number of the current text input position.
%*format*	A format string in the same form as used for the **wsprintf** function contained in the Windows SDK. The Status Bar uses this format to display the number.

Values: The following list contains possible values for the *dwStatusBarStyle* parameter:

Style	**Meaning**
STS_LEFTALIGN	Positions the text output panes on the left side of the client area.
STS_RIGHTALIGN	Positions the text output panes at the right side of the client area.
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 character insertion mode and the CAPSLOCK and NUMLOCK keystates.

See also: **CTXStatusBar::CTXStatusBar**

CTXStatusBar::CTXStatusBar

Description: Constructs a **CTXStatusBar** object.

See also: **CTXStatusBar::Create**

CTXStatusBar::SetLanguage

Description: This member function sets the language which the Status Bar uses to display text in some of its text panes. The language is specified either through an identifier or through the filename of a resource library.

Syntax: **BOOL SetLanguage(UINT** *nLang***);**
BOOL SetLanguage(const CString& *strLang***);**

Parameter	Description
nLang	Specifies a language identifier. For possible values see **CTXTextControl::SetLanguage**.
strLang	Specifies the filename including its full path of a resource library. See the chapter *Using the Text Control Class Library - Resources* for more information about creating a resource library.

Return Value: The return value is **FALSE** if an error has occurred or if the specified language has already been set, otherwise it is **TRUE**.

CTXTextControl

#include <TXTextControl.h>

The **CTXTextControl** class provides the functionality of a Text Control. A Text Control can be created either from a dialog template or directly in your code. In both cases, first call the constructor **CTXTextControl** to construct the **CTXTextControl** object, then call the **Create** member function to create the window and attach it to the **CTXTextControl** object.

Construction can be a one-step process in a class derived from **CTXTextControl**. Write a constructor for the derived class and call **Create** from within the constructor. **CTXTextControl** inherits significant functionality from **CWnd**.

If you want to handle Windows notification messages sent by a Text Control you can use either a Text Control notification handler or MFC's **ON_NOTIFY** message map entry. The Text Control notification handler is implemented through the **CTXNotifyHandler** class. This class has a member function for each possible notification. Derive your handler from this class and override the functions that belong to the notification you want to handle. A pointer to the derived handler can be specified as parameter of **CTXTextControl::Create**.

CTXTextControl Class Members

Construction and Initialization

CTXTextControl	Constructs a **CTXTextControl** object.
Create	Creates and initializes the child window associated with the **CTXTextControl** object.

Selection Operations

Clear	Deletes the current selection (if any).
GetSel	Retrieves the position of the current selection.
GetSelText	Returns selected text.
GetText	Retrieves the text in a generic text format.
GetTextLength	Returns the number of characters.

ReplaceSel	Replaces the current selection with new text.
SetSel	Sets a new text selection.

Loading and Saving

LoadFile	Loads formatted or unformatted text from a file.
LoadFromMemory	Loads formatted or unformatted text from a memory buffer.
ResetContents	Deletes all contents in a Text Control.
SaveFile	Saves formatted or unformatted text into a file.
SaveToMemory	Saves formatted or unformatted text in a memory buffer.

Clipboard Operations

CanCopy	Determines if a part of a Text Control document can be copied to the clipboard.
CanPaste	Determines if the contents of the clipboard can be pasted.
Copy	Copies the current selection to the clipboard.
Cut	Cuts the current selection to the clipboard.
Paste	Inserts data from the clipboard.

Undo Operations

CanRedo	Determines if an undone editing operation can be restored.
CanUndo	Determines if an editing operation can be undone.
EmptyUndoBuffer	Resets the undo buffer.
Redo	Restores the last undone edit operation.
Undo	Undoes the last edit operation.

Printing Operations

PrintControl	Prints the contents of a Text Control that is used without built-in scroll interface.

PrintPage	Prints a single page.

Search and Replace Functions

FindText	Searches for a text string.
ReplaceText	Finds and replaces text within the Text Control's contents.

Character Formatting Operations

EnlargeFont	Enlarges the pointsizes of all fonts in the current selection.
FontDialog	Opens Text Control's built-in font dialog box.
GetBaseLine	Returns the baseline alignment.
GetFont	Retrieves font name and size.
GetFontAttr	Returns font attributes.
GetTextColor	Retrieves the text color and the text background color.
ReduceFont	Reduces the pointsizes of all fonts in the current selection.
SetBaseLine	Sets a new baseline alignment.
SetFont	Sets a new font with a new size.
SetFontAttr	Sets font attributes.
SetTextColor	Sets a new text color.
SetTextBkColor	Sets a new text background color.

Paragraph Formatting Operations

GetLineSpacing	Retrieves the line spacing.
GetParaAlignment	Returns the paragraph alignment.
GetParaFormatFlags	Informs about advanced paragraph formatting attributes.
GetParaFrame	Retrieves paragraph frame attributes.
GetParaIndents	Retrieves paragraph indents.
GetTabs	Retrieves tab positions and types.

ParagraphDialog	Opens Text Control's built-in paragraph attributes dialog box.
SetLineSpacing	Sets a new line spacing.
SetParaAlignment	Sets a new paragraph alignment.
SetParaFormatFlags	Sets new advanced paragraph formatting options.
SetParaFrame	Sets new attributes for paragraph frames.
SetParaIndents	Sets new paragraph indents.
SetTabs	Sets new tab positions and types.

Page and Document Operations

GetDevice	Returns the current formatting device.
GetPageCount	Returns the number of pages in the document.
GetPageMargins	Retrieves the page margins.
GetPageSize	Retrieves the page size and the current view settings.
GetSupportedFonts	Retrieves all fonts the current device supports.
GetSupportedSizes	Retrieves all sizes of a certain font.
GetTXScrollPos	Returns the scroll position.
InsertPageNumber	Inserts a marked text field displaying the current page number.
SetDevicePrinter	Sets a printer as formatting device.
SetDeviceScreen	Sets the screen as formatting device.
SetDeviceStandard	Sets the standard printer as formatting device.
SetPageMargins	Sets new page margins.
SetPageSize	Sets a new page size and/or a new view setting.
SetTXScrollPos	Sets a new scroll position.

General Operations

GetBackgroundColor	Retrieves the current background color.
GetCaretExt	Returns the current caret extension.

GetLanguage	Returns the current language setting.
GetLineAndCol	Retrieves page, line and column number of the current input position.
GetMode	Informs about all current mode settings.
GetTextSize	Retrieves width and height of the text.
GetZoom	Returns the zooming factor.
InputPosFromPoint	Returns the character position at a given geometric position.
SetBackgroundColor	Sets a new background color.
SetCaretExt	Sets the width of the caret.
SetLanguage	Sets a new language.
SetLineAndCol	Sets a new text input position from a page, line and column number.
SetMode	Sets one or more of Text Control's modes.
SetZoom	Sets a new zooming factor.

Line Operations

GetBaseLinePos	Returns a line's baseline position.
GetLineCount	Returns the number of text lines.
GetLineRect	Retrieves a line's rectangular area.
LineFromChar	Returns the line at a given character position.
LineFromPoint	Returns the line at a given geometric position.
LineIndex	Returns a line's character index.

Inserted Images, Windows and OLE Objects

GetImageFilters	Informs about supported image filters.
InsertImage	Inserts an image.
InsertOleFile	Inserts an embedded OLE object from a file.
InsertOleLinkFile	Inserts a linked OLE object from a file.
InsertOleObject	Opens a dialog box and inserts an OLE object.
InsertOleProgID	Inserts an OLE object via its programmatic identifier.

InsertWindow	Inserts a window through its window handle.
ObjDelete	Deletes an inserted object.
ObjGetAttr	Retrieves an inserted object's attributes.
ObjGetIDispatch	Retrieves an inserted object's dispatch interface pointer.
ObjGetNext	Enumerates inserted objects.
ObjOleCancel	Deactivates an inserted OLE object.
ObjSetDistances	Sets distances between an inserted object and the text.
ObjSetMovable	Sets the movable state of an inserted object.
ObjSetScaling	Sets the scaling factors of an inserted object.
ObjSetSizeable	Sets the sizeable state of an inserted object.

Marked Text Field Functions

FieldChangeText	Alters the text of a marked text field.
FieldDelete	Deletes a marked text field.
FieldFromCaretPos	Returns the field at the current input position.
FieldGetData	Retrieves related data.
FieldGetNext	Enumerates marked text fields.
FieldGetPosition	Retrieves the starting and the ending character position of a marked text field.
FieldGetText	Retrieves the text of a marked text field.
FieldGetType	Retrieves the type of a marked text field.
FieldGoto	Scrolls to a marked text field.
FieldHasAttr	Informs about a field's attributes.
FieldInsert	Inserts a marked text field.
FieldSetAttr	Sets a field's attributes.
FieldSetData	Relates data to a marked text field.

Hypertext Link Support

ChangeLink	Changes the target to where a hypertext link points.

ChangeTarget	Changes the name of a hypertext target.
GetLinkLocation	Retrieves the location to where a hypertext link points.
GetTargetName	Retrieves the name of a hypertext target.
InsertLink	Inserts a hypertext link.
InsertTarget	Inserts a hypertext target.

Table Functions

TableAttrDialog	Opens Text Control's built-in dialog box for setting table attributes.
TableDeleteLines	Deletes table lines.
TableFromCaretPos	Returns the table with the current input position.
TableGetAttr	Retrieves table attributes.
TableGetCellPosition	Retrieves the starting and ending character position of a table cell.
TableGetCellText	Retrieves a table cell's text.
TableGetNext	Enumerates tables.
TableGetRowsAndCols	Retrieves the number of rows and columns.
TableInsert	Inserts a table.
TableIsPossible	Retrieves information whether a table can be inserted or changed.
TableSetAttr	Sets table attributes.
TableSetCellText	Sets a table cell's text.

Operations with Headers and Footers

HFActivate	Activates a header or a footer.
HFDisable	Deletes a header or a footer or disables settings.
HFEnable	Inserts a header or a footer or enables settings.
HFGetEnabled	Informs about which header or footer is enabled.

HFGetPosition	Returns a header's or a footer's position on the page.
HFSelect	Enables the programmer to manipulate the contents of a header or footer.
HFSetPosition	Sets a header's or a footer's position on the page.

Operations with Chains of Linked Text Controls

GetLinkWnd	Returns a certain window in a chain.
GetLinkWndCount	Returns the number of windows in a chain.
GetLinkWndFromOffset	Returns the window belonging to a certain character offset.
GetLinkWndNumber	Returns the number of a certain window in the chain.
GetLinkWndOffset	Returns the character offset of a certain window's first character.
SetLinkWnd	Sets a new successor Text Control.

Tool Bar Support

ConnectToolBar	Connects a Tool Bar with this Text Control.
DisconnectToolBar	Disconnects a Tool Bar from this Text Control.

Member Functions

CTXTextControl::CanCopy

Description:	This member function informs whether a part of a Text Control's document has been selected and can be copied to the clipboard.
Syntax:	**BOOL CanCopy( );**
Return Value:	The return value is **TRUE** if something can be copied to the clipboard. Otherwise it is **FALSE**.

CTXTextControl::CanPaste

Description: This member function informs whether the clipboard contains a format that can be pasted into a Text Control's document.

Syntax: **BOOL CanPaste( );**

Return Value: The return value is **TRUE** if something can be pasted. Otherwise it is **FALSE**.

CTXTextControl::CanRedo

Description: This member function informs whether an previously undone edit operation can be restored.

Syntax: **BOOL CanRedo(DWORD&** *dwRes* **= dwNULL);**

Parameter	Description
dwRes	Informs what kind of operation can be restored. It can be one of the following values:

Value:	**Meaning:**
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.

Return Value: The return value is **TRUE** if an undone operation can be restored. Otherwise it is **FALSE**.

CTXTextControl::CanUndo

Description: This member function informs whether an edit operation can be undone.

Syntax: **BOOL CanUndo(DWORD&** *dwRes* **= dwNULL);**

Parameter	Description
dwRes	Informs what kind of operation can be undone. It can be one of the following values:

Value:	**Meaning:**
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.

Return Value: The return value is **TRUE** if an edit operation can be undone. Otherwise it is **FALSE**.

CTXTextControl::ChangeLink

Description: This member function changes the target to where a hypertext link points.

Syntax: **BOOL ChangeLink(UINT** *nFieldID,* **const CString&** *strLinkTarget,* **BOOL** *bExternal* = **TRUE);**

Parameter	Description
nFieldID	Is the identifier of the marked text field that defines the hypertext link in the text.
strLinkTarget	Specifies the location to where the hypertext link points. This can be an address or a file name if the link point to an external location. If the link points to a location inside the same document it must be the name of a target field.
bExternal	Must be set to **TRUE** if *strLinkTarget* defines a location outside of the document, otherwise this parameter must be set to **FALSE**.

Return Value: The return value is **TRUE** if the function was successful, otherwise it is **FALSE**.

See also: **CTXTextControl::ChangeTarget, CTXTextControl::InsertLink, CTXTextControl::InsertTarget**

CTXTextControl::ChangeTarget

Description: This member function changes the name of a hypertext target.

Syntax: **BOOL ChangeTarget(UINT** *nFieldID*, **const CString&** *strTargetName***);**

Parameter	**Description**
nFieldID	Is the identifier of the marked text field that defines the hypertext target in the text.
strTargetName	Specifies the target's new name.

Return Value: The return value is **TRUE** if the target's name could be changed, otherwise it is **FALSE**.

CTXTextControl::Clear

Description: This member function deletes the text of the current selection, if any.

Syntax: **void Clear( );**

CTXTextControl::ConnectToolBar

Description: This member function connects one of the Text Control's tool bars with this Text Control. The connected Button Bar, Ruler Bar or Status Bar shows this Text Control's current state only if it has the input focus.

Syntax: **BOOL ConnectToolBar(CTXButtonBar*** *pButtonBar***);**
BOOL ConnectToolBar(CTXRulerBar* *pRulerBar***);**
BOOL ConnectToolBar(CTXStatusBar* *pStatusBar***);**

Return Value: The return value is **TRUE** if the tool bar could be connected, otherwise it is **FALSE**.

See also: **CTXTextControl::DisconnectToolBar**

CTXTextControl::Copy

Description: This member function copies the text of the current selection (if any) to the clipboard.

Syntax: **void Copy( );**

CTXTextControl::Create

Description: This member function creates a Text Control child window. Text Control child windows must be created in two steps. First call the constructor which creates the **CTXTextControl** object. Then call **Create**, which creates the Text Control child window and attaches it to **CTXTextControl**.

Syntax: **BOOL Create(CWnd*** *pParentWnd,* **UINT** *nID,* **const CRect&** *rcSize,* **CTXNotifyHandler*** *pHandler* = **NULL, LPLOGFONT** *lpLogFont* = **NULL);**

Parameter	Description
pParentWnd	Specifies the Text Control's parent window. It must not be **NULL**.
nID	Specifies the Text Control's identifier.
rcSize	Specifies the Text Control's size and position in client area coordinates of its parent window.
pHandler	Points to a notification handler object. This parameter can be zero if you do not want to handle notifications or if you want to use MFC's message map entries to handle notifications. See "*Notifications*" for more information.
lpLogFont	Points to a **LOGFONT** data structure which defines the logical font the Text Control will use.

Return Value: The function returns **TRUE** if the Text Control window could be created, otherwise it returns **FALSE**.

See also: **CTXTextControl::CTXTextControl**

CTXTextControl::CTXTextControl

Description: Constructs a **CTXTextControl** object.

See also: **CTXTextControl::Create**

CTXTextControl::Cut

Description: This member function deletes the text of the current selection (if any) and copies the deleted text to the clipboard.

Syntax: **void Cut( );**

CTXTextControl::DisconnectToolBar

Description: This member function disconnects a previously connected tool bar from this Text Control.

Syntax: **BOOL DisconnectToolBar(CTXButtonBar*** *pButtonBar***);**
BOOL DisconnectToolBar(CTXRulerBar* *pRulerBar***);**
BOOL DisconnectToolBar(CTXStatusBar* *pStatusBar***);**

Return Value: The return value is **TRUE** if the tool bar could be disconnected, otherwise it is **FALSE**.

See also: **CTXTextControl::ConnectToolBar**

CTXTextControl::EmptyUndoBuffer

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

Syntax: **void EmptyUndoBuffer( );**

CTXTextControl::EnlargeFont

Description:	This member function enlarges the pointsizes of all fonts in the current selection.
Syntax:	**BOOL EnlargeFont(CSize&** *szMin* = **szNULL);**

Parameter	Description
szMin	Text Control fills this variable with its new minimum window size (in pixels). It is only useful if the Text Control is used without the built-in scroll-interface.

Return Value:	The return value is **TRUE**, if the font sizes could be enlarged. Otherwise it returns **FALSE**.

CTXTextControl::FieldChangeText

Description:	This member function alters the text of a marked text field.
Syntax:	**BOOL FieldChangeText(UINT** *nFieldID*, **const CString&** *strNewText***);**

Parameter	Description
nFieldID	Is the identifier of the marked text field.
strNewText	Specifies the marked text field's new text.

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

CTXTextControl::FieldDelete

Description:	This member function deletes a marked text field. The field is deleted independent of its attributes.
Syntax:	**BOOL FieldDelete(UINT** *nFieldID*, **BOOL** *bDeleteText* = **FALSE);**

Parameter	Description
nFieldID	Is the identifier of the marked text field.

bDeleteText	If this parameter is **TRUE**, the field is deleted including its text. Otherwise the field's text is not deleted.

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

Remarks: If a marked text field is deleted with this function, the Text Control does not send a TN_FIELD_DELETED notification message.

CTXTextControl::FieldFromCaretPos

Description: This member function returns the field identifier of the field containing the current input position.

Syntax: **UINT FieldFromCaretPos( );**

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.

CTXTextControl::FieldGetData

Description: This member function retrieves the data, previously related to a marked text field with **FieldSetData**.

Syntax:

BOOL FieldGetData(UINT *nFieldID*, **CString&** *strData***);**
BOOL FieldGetData(UINT *nFieldID*, **DWORD&** *dwData***);**
BOOL FieldGetData(UINT *nFieldID*, **CByteArray&** *arBuf***);**

Parameter	**Description**
nFieldID	Is the identifier of the marked text field.
strData	Retrieves string data from the marked text field.
dwData	Retrieves a 4-byte value from the marked text field.
arBuf	Retrieves byte-data from the marked text field.

Return Value: The return value is **FALSE** if an error has occurred or if the specified field does not exist. Otherwise it is **TRUE**.

CTXTextControl::FieldGetNext

Description: This member function returns the identifier of a marked text field that follows the specified field in the Text Control's current text.

Syntax: **UINT FieldGetNext(WORD** *wFieldType*, **UINT** *nFieldID* = **0);**

Parameter	Description
wFieldType	Specifies the group of fields. It can be a combination of any of the values described in the following Values section. If *wFieldType* is zero, all fields are handled.
nFieldID	Specifies a field's identifier. If this parameter is zero, the first field's identifier is 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.

Values: The *wFieldType* parameter can be a combination of the following values:

Value	Meaning
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.
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.

CTXTextControl::FieldGetPosition

Description: This member function retrieves the start and end character positions of a marked text field.

Syntax: **BOOL FieldGetPosition(UINT** *nFieldID*, **DWORD&** *dwPosStart*, **DWORD&** *dwPosEnd***);**

Parameter	Description
nFieldID	Is the identifier of the marked text field.
dwPosStart	Receives the field's start position.
dwPosEnd	Receives the field's start position.

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

Remarks: 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.

CTXTextControl::FieldGetText

Description: This member function retrieves the text of a marked text field.

Syntax: **BOOL FieldGetText(UINT** *nFieldID***, CString&** *strBuffer***);**

Parameter	Description
nFieldID	Is the identifier of the marked text field.
strBuffer	Is a buffer receiving the field's text.

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

CTXTextControl::FieldGetType

Description: This member function retrieves the type of a marked text field.

Syntax: **BYTE FieldGetType(UINT *nFieldID*);**

Parameter	Description
nFieldID	Is the identifier of the marked text field.

Return Value: The return value is the type of the specified marked text field. It can be one of the following values:

Type	Description
FT_EXTERNALLINK	The field is the source of a hypertext link to a location outside of the document.
FT_INTERNALLINK	The field is the source of a hypertext link to a location in the same document.
FT_LINKTARGET	The field is a position in a document which is the target of a hypertext link.
FT_PAGENUMBER	The field displays the current page number.
FT_STANDARD	The field is a standard marked text field without a special type.

CTXTextControl::FieldGoto

Description: This member function 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 Text Control's client area.

Syntax: **BOOL FieldGoto(UINT *nFieldID*);**
BOOL FieldGoto(const CString& *strTargetname*);

Parameter	Description
nFieldID	Specifies the identifier of the marked text field to which should be scrolled.
strTargetname	Specifies the name of the marked text field to which should be scrolled if the field is a hypertext target

Return Value: The return value is **FALSE** if the specified field does not exist. Otherwise it is **TRUE**.

CTXTextControl::FieldHasAttr

Description: This member function returns **TRUE** if a marked text field has the specified attributes.

Syntax: **BOOL FieldHasAttr(UINT** *nFieldID,* **DWORD** *dwAttr***);**

Parameter	Description
nFieldID	Is the identifier of the marked text field.
dwAttr	Specifies one or more field attributes. See **TXTextControl::FieldInsert** for more information which attributes are possible.

Return Value: The return value is **TRUE** if the field has the specified attributes. Otherwise it is **FALSE**.

See Also: **TXTextControl::FieldSetAttr**

CTXTextControl::FieldInsert

Description: This member function inserts a new marked text field at the current input position or defines selected text as a marked text field.

Syntax: **UINT FieldInsert(const CString&** *strFieldText* = "", **DWORD** *dwAttr* = **0, UINT** *nReserved* = **0);**

Parameter	Description
strFieldText	Specifies the field's text. If text is selected Text Control defines this text as the field's field.

dwAttr	Specifies one or more attributes described in the following Values section.
nReserved	A reserved parameter for future use.

Return Value: The return value is the identifier for the newly created field. It is zero if an error has occurred.

Values: The *dwAttr* parameter can be a combination of the following values:

Value	Meaning
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 a marked text field can be changed.
TF_UNCHANGEABLE	Set if the text of a marked text field cannot be changed.
TF_EXTEDITMODE	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_NORMALEDITMODE	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 **CTXTextControl::SetCaretExt**.

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.

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_EXTEDITMODE
TF_SHOWCURFIELDNORMAL and TF_SHOWCURFIELDGRAY
TF_USETEXTCARET and TF_USEFIELDCARET
TF_DISABLEDBLCLICKS and TF_ENABLEDBLCLICKS

The default attributes for a newly created field 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 that case TN_FIELD_DELETED notifications are not sent.

CTXTextControl::FieldSetAttr

Description: This member function sets attributes for the specified marked text field. Changing one attribute does not alter other attributes.

Syntax: **BOOL FieldSetAttr(UINT** *nFieldID,* **DWORD** *dwAttr***);**

Parameter	Description
nFieldID	Is the identifier of the marked text field.
dwAttr	Specifies one or more field attributes. See **TXTextControl::FieldInsert** for more information which attributes are possible.

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

CTXTextControl::FieldSetData

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

Syntax: **BOOL FieldSetData(UINT** *nFieldID*, **const CString&** *strData***);**
BOOL FieldSetData(UINT *nFieldID*, **DWORD** *dwData***);**
BOOL FieldSetData(UINT *nFieldID*, **LPBYTE** *pBuf*, **DWORD** *dwDataSize***);**

Parameter	Description
nFieldID	Is the identifier of the marked text field.
strData	Stores string data.
dwData	Stores a 4-byte value.
pBuf	Points to a buffer containing general byte-data.
dwDataSize	Specifies the number of bytes stored in the buffer pBuf points to.

Return Value: The return value is **FALSE** if the specified field does not exist or when the data could not be stored. Otherwise it is **TRUE**.

CTXTextControl::FindText

Description: This member function opens the system-defined modeless search dialog box (first prototype) or searches for a specified text string (second prototype). This makes it possible for the user to find text within a Text Control's contents.

Syntax: **void FindText( );**
UINT FindText(const CString& *strFindWhat*, **DWORD** *dwFlags* = **TXFR_MATCHCASE, LONG** *lStart* = **0);**

Parameter	**Description**
strFindWhat	Specifies the string to search for.
dwFlags	Specifies a combination of the following flags:

Value	**Description**
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.
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.

Parameter	**Description**
lStart	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.

Return Value: 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.

CTXTextControl::FontDialog

Description: This member function opens a modal dialog box which contains all available fonts and pointsizes for the currently selected printer. Font attributes and values for subscript and superscript can also be set.

Syntax: **BOOL FontDialog(CSize&** *szMin* = **szNULL, BOOL&** *bChanged* = **bNULL);**

Parameter	**Description**
szMin	Text Control fills this variable with its new minimum window size (in pixels). It is only useful if the Text Control is used without the built-in scroll-interface.
bChanged	Retrieves **TRUE** if the dialog box has been left with *Ok*. Otherwise it retrieves **FALSE**.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetBackgroundColor

Description: This member function retrieves a RGB value for the background color of the Text Control.

Syntax: **BOOL GetBackgroundColor(COLORREF&** *colBack***, BOOL&** *bIsSysColor* = **bNULL);**

Parameter	**Description**
colBack	Retrieves the background color.
bIsSysColor	Retrieves **TRUE** if *colBack* is the system color for the window background. Otherwise it retrieves **FALSE**.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetBaseLine

Description: This member function returns the baseline alignment value of the currently selected text.

Syntax: **WORD GetBaseLine(WORD&** *wBaseAlign* = **wNULL);**

Parameter	Description
wBaseAlign	Retrieves the baseline alignment value in *twips*. Nothing is retrieved if the return value is FA_NOCOMMONS.

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

Value	Meaning
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 align is superscript.
FA_SUBSCRIPT	The common baseline align is subscript.

CTXTextControl::GetBaseLinePos

Description: This member function returns the baseline position of the specified line. The dimensions are given in *twips* 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 **GetTXScrollPos**.

Syntax: **DWORD GetBaseLinePos(LONG** *lIndex***);**

Parameter	Description
lIndex	Specifies the index of the line which baseline position should be returned. The index of the first line is zero.

Return Value: The return value specifies the requested baseline position in *twips*.

CTXTextControl::GetCaretExt

Description: This member function returns the current extension of the caret in pixel.

Syntax: **CSize GetCaretExt( );**

Return Value: Specifies the caret extension in pixels.

CTXTextControl::GetDevice

Description: This member function returns the device for which the text is currently formatted (screen, standard device or printer).

Syntax: **DWORD GetDevice(CString&** *strDevName,* **WORD** *wMaxChar* = **255);**

Parameter	Description
strDevName	Retrieves the device name if the return value is TF_PRINTER.
wMaxChar	Specifies the device name's maximum length.

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

Value	Meaning
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.

Remarks: The name of the printer is copied in the same format as that used in the WIN.INI file, for example:
`PostScript Printer,PSCRIPT,LPT1:`

CTXTextControl::GetFont

Description: This member function retrieves the common fontname and size of all currently selected fonts.

Syntax: **UINT GetFont(CString&** *strFont***, BOOL** *bPoints* = **TRUE);**

Parameter	Description
strFont	Retrieves the typeface string. The string is set to an empty string if no common typeface exists.
bPoints	When set to TRUE the returned pointsize is in *points*, otherwise it is returned in *twips*.

Return Value: The return value is the common pointsize. It is zero if no common pointsize exists.

CTXTextControl::GetFontAttr

Description: This member function returns a bitmask of the font attributes for all fonts in the current selection.

Syntax: **DWORD GetFontAttr( );**

Return Value: The return value is zero if an error has occurred. Otherwise it is one or more of the following values, indicating the common attributes:

Value	Meaning
FA_NOCOMMONS	No common font attributes.
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 doubled underlined.
FA_UL_WORDSONLY	Words are underlined, word gaps are omitted.
FA_UL_REDZIGZAG	Each font is underlined with a red zigzag line.

CTXTextControl::GetImageFilters

Description: This member function retrieves pairs of null-terminated strings specifying image filters. The first string in each pair is a display string that describes the filter (for example, „Windows Bitmap“), and the second string specifies the filter pattern (for example, „*.BMP"). This is the same format as described in the Windows SDK for the *lpstrFilter* member of an **OPENFILENAME** structure and therefore the strings can be used to initialize the **GetOpenFileName** dialog box.

Syntax: **BOOL GetImageFilters(CString&** *strFilters***);**

Parameter	**Description**
strFilters	Retrieves the pairs of strings. The last pair ends with two terminating zero characters.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetLanguage

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

Syntax: **UINT GetLanguage();**

Return Value: The return value is the language identifier. See **CTXTextControl::SetLanguage** for possible values.

CTXTextControl::GetLineAndCol

Description: This member function retrieves page, line and column number of the current input position. All values are one-based. Text Control's status bar uses this function to display the page, line and column number.

Syntax: **BOOL GetLineAndCol(DWORD&** *dwLine,* **DWORD***& dwCol,* **UINT&** *nPage***);**

Parameter	**Description**
dwLine	Retrieves the line number.
dwCol	Retrieves the column number.
nPage	Retrieves the page number.

Return Value: The return value is **FALSE** if an error has occurred, otherwise it is **TRUE**.

CTXTextControl::GetLineCount

Description: This member function returns the number of text lines in the Text Control.

Syntax: **long GetLineCount( );**

Return Value: The return value is the number of text lines.

CTXTextControl::GetLineRect

Description: This member function 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 *twips* with an origin at the upper left corner of the Text Control's complete text. The relationship between the upper left corner of the complete text and the upper left corner of the Text Control's client area can be obtained with **CTXTextControl::GetTXScrollPos**.

Syntax: **void GetLineRect(LONG** *lLineIndex*, **CRect&** *rcLine***);**

Parameter	Description
lLineIndex	Specifies the index of the line whose rectangle is to be retrieved. The index of the first line is zero.
rcLine	Retrieves the line's rectangle.

CTXTextControl::GetLineSpacing

Description: This member function retrieves the line spacing of all selected paragraphs.

Syntax: **void GetLineSpacing(WORD&** *wLineSpace***, WORD&** *wPercent =* **wNULL);**

Parameter	Description
wLineSpace	Retrieves the line spacing in *twips*. It is set to zero if there is no common value.
wPercent	Retrieves the line spacing as a percentage of the font size. It is set to zero if there is no common value.

CTXTextControl::GetLinkLocation

Description: This member function retrieves the location to where a hypertext link points.

Syntax: **BOOL GetLinkLocation(UINT** *nFieldID*, **CString&** *strText***);**

Parameter	Description
nFieldID	Is the identifier of a marked text field.
strText	Retrieves the location to where the link points.

Return Value: The return value is **FALSE** if an error has occurred or if the specified marked text field does not represent the source of a hypertext link. Otherwise it returns **TRUE**.

CTXTextControl::GetLinkWnd

Description: This member function searches for the handle of a window that is part of a chain of Text Control windows.

Syntax: **HWND GetLinkWnd(DWORD** *dwLnkWnd***);**

Parameter	Description
dwLnkWnd	Specifies the relationship between the window for which this function is called and the returned window. Possible values are listed in the following Values section.

Return Value: The return value is the handle of the requested window. It is zero if the window could not be found.

Values: Possible values for *dwLnkWnd* are:

Value	Meaning
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.

CTXTextControl::GetLinkWndCount

Description: This member function returns the total number of windows that belong to a chain of linked windows.

Syntax: **UINT GetLinkWndCount( );**

Return Value: Is the number of windows in the chain.

CTXTextControl::GetLinkWndFromOffset

Description: This member function returns the window of a chain of linked Text Controls that contains the specified one-based character offset.

Syntax: **HWND GetLinkWndFromOffset(LONG** *lOffset***);**

Parameter	Description
lOffset	Specifies a one-based character offset.

Return Value: The return value is the window containing the character offset, or zero if the window could not be found.

CTXTextControl::GetLinkWndNumber

Description:	This member function returns the chain position of a window within a chain of linked Text Controls. The first window is assigned position one.
Syntax:	**UINT GetLinkWndNumber( );**
Return Value:	Specifies the position number.

CTXTextControl::GetLinkWndOffset

Description:	This member function returns the one-based character offset of this window's first character relative to the complete text in a chain of linked Text Controls.
Syntax:	**LONG GetLinkWndOffset();**
Return Value:	The return value is the offset of this window's first character in the chain.

CTXTextControl::GetMode

Description: This member function returns all the Text Control's current mode settings.

Syntax: **DWORD GetMode(DWORD&** *dwModeEx* = **dwNULL, CSize&** *szMax* = **szNULL);**

Parameter	**Description**
dwModeEx	Retrieves extended mode settings. Extended mode settings are described in the following Values section.
szMax	This parameter is only useful when the Text Control operates in autoexpand mode. It is filled with the current maximum window size to which the window can expand (in pixels).

Return Value: The return value specifies Text Control's current mode settings. It can be a combination of the following values:

Value	Meaning
TF_AUTOEXPAND	The Text Control's window will be automatically expanded when text insertion or format changes result in text that does not fit into the Text Control anymore.
TF_FIXED	The Text Control's window size is fixed and is not automatically expanded.
TF_FRAMED	The Text Control window is drawn with a frame of 1 pixel width.
TF_NOTFRAMED	The Text Control window is drawn without a frame.
TF_SHOWSELNA	A text selection remains visible when the control looses the input focus.
TF_HIDESELNA	A text selection is hidden when the control looses the input focus.
TF_SHOWWHITESPACE	Control characters are made visible.
TF_HIDEWHITESPACE	Control characters are hidden.
TF_OVERWRITE	Newly inserted characters overwrite existing characters.
TF_INSERT	Newly inserted characters are inserted.
TF_REPLACESEL	The text of a current selection is deleted before new text is inserted.
TF_KEEPSEL	The text of a current selection is not deleted before new text is inserted.
TF_OPAQUE	The control's background is opaque.
TF_TRANSPARENT	The control's background is transparent.

Values: The *dwModeEx* parameter retrieves the following extended mode settings:

Value	Meaning
TF_DISPLAY	Text Control only displays text.
TF_READONLY	Text control displays text and the user can select and copy it.

TF_EDIT	Text Control displays text and the user can select and edit it.
TF_NOWAITCURSOR	Text Control does not change the cursor to an hourglass during long time operations.
TF_WAITCURSOR	Text Control changes the cursor to an hourglass during long time operations.
TF_TOPINDENTFIRSTPG	Text Control allows a top indent for the first paragraph in the text.
TF_NOTOPINDENTFIRSTPG	Text Control suppresses a top indent of the first paragraph.
TF_ERRORBOXES	Text Control displays error message boxes.
TF_NOERRORBOXES	Text Control suppresses all error message boxes.
TF_SHOWGRIDLINES	Text Control shows grid lines in tables.
TF_HIDEGRIDLINES	Text Control hides grid lines in tables.

CTXTextControl::GetPageCount

Description: This member function returns the current number of pages.

Syntax: **UINT GetPageCount(LONG** *lHeight***);**

Parameter	**Description**
lHeight	Specifies the height of the page in *twips*. This parameter is ignored if Text Control operates in page view mode.

Return Value: The return value is the number of pages.

CTXTextControl::GetPageMargins

Description: This member function retrieves the current page margins.

Syntax: **BOOL GetPageMargins(CRect&** *rcMargins***);**

Parameter	Description
rcMargins	Retrieves the page margins in *twips*.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetPageSize

Description: This member function retrieves the document's page size and view settings.

Syntax: **BOOL GetPageSize(CSize&** *szText,* **UINT&** *nViewMode,* **UINT&** *nScrollInterface***);**

Parameter	Description
szText	Retrieves the document's page size without page margins. A value of zero indicates that a page size has not been set. In this case the text is formatted in the borders of the Text Control's client area.
nViewMode	Retrieves the current view mode. See **CTXTextControl::SetPageSize** for possible values.
nScrollInterface	Retrieves the current scroll interface settings. See **CTXTextControl::SetPageSize** for possible values.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetParaAlignment

Description: This member function returns the paragraph alignment value of the currently selected paragraphs.

Syntax: **DWORD GetParaAlignment( );**

Return Value: The return value is zero if an error has occurred. Otherwise it is one of the following values:

Value	Meaning
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.

CTXTextControl::GetParaFormatFlags

Description: This member function returns advanced formatting attributes of all selected paragraphs.

Syntax: **DWORD GetParaFormatFlags( );**

Return Value: The return value is a combination of the formatting attributes. Possible values are listed in the Values section of **CTXTextControl::SetParaFormatFlags**.

CTXTextControl::GetParaFrame

Description: This member function retrieves the appearance, style, line width and text distance for the frames of all selected paragraphs.

Syntax: **DWORD GetParaFrame(WORD&** *wWidth,* **WORD&** *wDistance***);**

Parameter	Description
wWidth	Retrieves the paragraph frame's line width in *twips*. Zero indicates that the selected paragraphs have different frame widths.
wDistance	Retrieves the distance between frame and text in *twips*. A value of -1 indicates that the selected paragraphs have different distances.

Return Value: The return value is a combination of frame appearance and style flags. The following values are possible:

Value	Meaning
BF_LEFTLINE	The frame has a left line.
BF_RIGHTLINE	The frame has a right line.
BF_TOPLINE	The frame has a top line.
BF_BOTTOMLINE	The frame has a bottom line.
BF_BOX	The frame is a complete box.
BF_TABLINES	The frame includes vertical lines at each tabulator position.
BF_TABLE	The frame is a complete box including vertical lines at each tabulator position.
BF_SINGLE	The lines are single lines.
BF_DOUBLE	The lines are doubled lines.
BF_BOXCONNECT	The frame is connected with the frames of the neighbouring paragraphs.

CTXTextControl::GetParaIndents

Description: This member function retrieves the paragraph indents of all selected paragraphs.

Syntax: **BOOL GetParaIndents(CRect&** *rcIndents***, int&** *iFirstIndent*, **CSize&** *szMaxNew* = **szNULL);**

Parameter	Description
rcIndents	Retrieves the paragraphs' indent values. If a value contains TR_IGNORED, no common value of this indent exists for the selected paragraphs.
iFirstIndent	Retrieves an additional left indent of the first line. This value can be negative indicating that the left indent of the first line is smaller than the left indent of the following lines. TR_IGNORED is retrieved if no common value exists for all paragraphs.
szMaxNew	Retrieves maximum values for a combination of new indents that can be set with **SetParaIndents**. The x-value is the maximum value for the sum of left indent, right indent and 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.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetSel

Description: This member function retrieves the starting and ending positions of the current text selection.

Syntax: **BOOL GetSel(long&** *lStart*, **long&** *lEnd***);**

Parameter	Description
lStart	Specifies the zero-based text input position where the user has started the current text selection.
lEnd	Specifies the zero-based input position where the user has ended the current text selection.

Return Value: The return value is **FALSE** if an error has occurred, otherwise it is **TRUE**.

CTXTextControl::GetSelText

Description: This member function returns currently selected text.

Syntax: **CString GetSelText( );**

Return Value: The return value is a string variable containing the selected text.

CTXTextControl::GetSupportedFonts

Description: This member function retrieves all the font names which are supported by the current output device.

Syntax: **BOOL GetSupportedFonts(CStringArray&** *arFonts***);**

Parameter	Description
arFonts	Retrieves the font names.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetSupportedSizes

Description: This member function retrieves all point sizes which are supported for a certain font by the current output device.

Syntax: **BOOL GetSupportedSizes(const CString&** *strFontName*, **CStringArray&** *arSizes***);**

Parameter	Description
strFontName	Specifies the name of the font, the sizes of which are to be retrieved.
arSizes	Retrieves the font sizes.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetTabs

Description: This member function retrieves common tab positions and types for all selected paragraphs.

Syntax: **BOOL GetTabs(LPTABSCT** *pTabs*, **BOOL** *bTwips* = **TRUE);**

Parameter	Description
pTabs	Points to an array of type TABSCT and size NTABS. See *Data Structures* for a description of the TABSCT structure.
bTwips	When this parameter is set to **TRUE** the retrieved position values are in *twips*, otherwise they are in pixels.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::GetTargetName

Description: This member function retrieves the name of a hypertext target.

Syntax: **BOOL GetTargetName(UINT** *nFieldID*, **CString&** *strText***);**

Parameter	Description
nFieldID	Is the identifier of a marked text field.
strText	Retrieves the name of a hypertext target.

Return Value: The return value is **FALSE** if an error has occurred or if the specified marked text field does not represent the target of a hypertext link. Otherwise it returns **TRUE**.

CTXTextControl::GetText

Description: This member function retrieves the Text Control's text. The text is in Text Control's generic text format and can be used to work with functions that use character indices like **CTXTextControl::SetSel** or **CTXTextControl::LineFromChar**. To get the text in a Windows compatible generic text format, for example to exchange it with a Windows Edit Control, use **CTXTextControl::SaveToMemory** with the format identifier set to TF_FORMAT_ANSI or TF_FORMAT_UNICODE.

Syntax: **BOOL GetText(CString&** *strText*, **DWORD** *dwCount* = **0);**

Parameter	Description
strText	Retrieves the text.
dwCount	Specifies the maximum number of characters to be copied, including the terminating zero character. If *dwCount* is zero the complete text is retrieved.

Return Value: The return value is **TRUE** if text is retrieved. Otherwise it is **FALSE**.

CTXTextControl::GetTextColor

Description: This member function retrieves RGB values for the text color and the text background color of the currently selected text.

Syntax: **DWORD GetTextColor(COLORREF&** *colFG*, **COLORREF&** *colBG***);**

Parameter	Description
colFG	Retrieves the text color.
colBG	Retrieves the text background color.

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

Value	Meaning
CV_UNDEFINED	The current selection contains more than one text color.
CV_TEXTDEFAULT	The text color is the system color for the window text.
CV_TEXTUSER	The text color is a user-defined value.

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

Value	**Meaning**
CV_UNDEFINED	The current selection contains more than one color for the text background.
CV_BKDEFAULT	The text background color is the system color for the window background.
CV_BKCONTROL	The text background color is the Text Control's background color, set with **CTXTextControl::SetBackgroundColor**.
CV_BKUSER	The text background color is a user-defined value.

CTXTextControl::GetTextLength

Description: This member function returns the length of the text in characters.

Syntax: **DWORD GetTextLength( );**

CTXTextControl::GetTextSize

Description: This member function retrieves the dimensions of the text in *twips*.

Syntax: **BOOL GetTextSize(CSize&** *szText***);**

Parameter	**Description**
szText	Retrieves the width and the height of the text the Text Control currently contains.

Return Value: The return value is **TRUE** if the function is successful. Otherwise it is **FALSE**.

CTXTextControl::GetTXScrollPos

Description: This member function returns the current scroll position.

Syntax: **DWORD GetTXScrollPos(WORD** *wDir***);**

Parameter	Description
wDir	Specifies the direction. It can be one of the following values:

Value	Meaning
TF_HSCROLL	Returns the horizontal scroll position.
TF_VSCROLL	Returns the vertical scroll position.

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

CTXTextControl::GetZoom

Description: This member function returns the current zoom factor in percent.

Syntax: **UINT GetZoom( );**

CTXTextControl::HFActivate

Description: This member function 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.

Syntax: **BOOL HFActivate(LONG** ***lWhat*****);**

Parameter	Description
lWhat	When this parameter is zero the currently activated header or footer is deactivated. Otherwise it specifies the header or footer to activate and can be one of the following values:

Value	Description
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 **TRUE** if the header or footer could be activated. Otherwise it is **FALSE**.

CTXTextControl::HFDisable

Description: This member function disables certain parts of the header and footer functionality.

Syntax: **BOOL HFDisable(LONG** *lWhat***);**

Parameter	**Description**
lWhat	When this parameter is zero, all currently enabled header and footer functionality is disabled and all allocated memory is freed. Other possible values are described in Values.

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

Values: *lWhat* can be a combination of the following values:

Value	**Description**
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_NOMOUSEDBLCLK	Enables activation through mouse double-clicks.

TF_HF_SOLIDFRAME	Enables activation through mouse double-clicks.
TF_HF_UNFRAMED	Resets the border to framed.

CTXTextControl::HFEnable

Description: This member function enables the usage of headers and footers. Headers and footers can only be used when a user-defined page size has been set with **CTXTextControl::SetPageSize**.

This message can only be used to add a certain header or footer or a certain style setting. To disable a certain functionality use **CTXTextControl::HFDisable**. For example when activation with mouse clicks is enabled, calling this function with TF_HF_SOLIDFRAME displays an activated header or footer with a solid frame. Activation with mouse clicks remains active.

Syntax: **BOOL HFEnable(LONG l***What***);**

Parameter	Description
lWhat	Specifies what to enable. See the following Values section for possible values.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Values: *lWhat* can be a combination of the following values:

Value	Description
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_NOMOUSEDBLCLK	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.

CTXTextControl::HFGetEnabled

Description: This member function returns which headers and/or footers are enabled for the current document.

Syntax: **DWORD HFGetEnabled( );**

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

Value	Description
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.

CTXTextControl::HFGetPosition

Description: This member function 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.

Syntax: **DWORD HFGetPosition(LONG** *lWhat***);**

Parameter	Description
lWhat	Specifies the header or footer the position of which is requested. It can be one of the following values:

Value	Meaning
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.

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

CTXTextControl::HFSelect

Description: This member function defines, whether a certain Text Control function handles a header, a footer or the main text. The Text Control's button bar, ruler and status bar need the default automatic mode for correct working. Therefore when a text part selection is not longer needed it should be reset to the default automatic mode.

Syntax: **BOOL HFSelect(LONG** *lWhat***);**

Parameter	Description
lWhat	Specifies the text part to select. It can be one of the following values:

Value	Meaning
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	Selects the special footer for the first page.
TF_HF_AUTO	Selects the automatic mode. A function call handles the text part with the current input position. This is the default setting.
TF_HF_MAINTEXT	Selects the main text.

Return Value: The return value is **TRUE** if the selection was successful. Otherwise, it is **FALSE**.

CTXTextControl::HFSetPosition

Description: This member function 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.

Syntax: **BOOL HFSetPosition(LONG** *lWhat*, **LONG** *lPos***);**

Parameter	**Description**
lWhat	Specifies the header or footer the position of which is to be set. It can be one of the following values:

Value	**Meaning**
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.

lPos	Specifies the new position.

Return Value: The return value is **TRUE** if the position could be set, otherwise it is **FALSE**.

CTXTextControl::InputPosFromPoint

Description: This member function 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.

Syntax: **long InputPosFromPoint(const CPoint&** *ptPos*, **BOOL** *bTwips* = **TRUE);**

Parameter	**Description**
ptPos	Specifies the geometric position.
bTwips	When this parameter is set to **TRUE** the position values are in *twips*, otherwise they are in *pixels*.

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.

CTXTextControl::InsertImage

Description: This member function inserts an image in a Text Control's document.

Syntax: **UINT InsertImage(**
const CString& *strFileName*,
WORD *wImageFlags* = **0,**
UINT *nFilterIndex* = **0,**
LONG *lTextPos* = **-1,**
WORD *wInsertMode* = **EOM_INSERTASCHAR,**
BOOL *bMoveable* = **TRUE,**
BOOL *bSizeable* = **TRUE,**
const CPoint& *ptPos* = **CPoint(0, 0),**
const CSize& *szScale* = **CSize(100, 100),**

const CRect& *rcDistances* = **CRect(0, 0, 0, 0),**
WORD& *wError* = **wNULL);**

Parameter	Description
strFileName	Specifies the image's filename.
wImageFlags	Specifies mode settings for the image. The following values are possible: **Value** — **Meaning** ICF_GRAYED — The image is displayed in fast display mode. ICF_SAVEASDATA — 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.
nFilterIndex	Specifies an image filter as an index of the string pairs retrieved through **CTXTextControl::GetImageFilters**. The first pair of strings has the index value 1. If the string pairs are used to initialize the *lpstrFilter* member of an **OPENFILENAME** structure, another member of that structure, *nFilterIndex*, can be used to initialize this parameter. See the Windows SDK for more information about the **OPENFILENAME** structure. If *nFilterIndex* is set to 0, the Text Control automatically tries to select a filter.
lTextPos	Specifies the text position where to insert the image. If *lTextPos* is -1 the image is inserted at the current input position.
wInsertMode	Specifies how the image is handled when the text is formatted. See the following Values section for possible values.

bMoveable	The image can be moved by depressing the ALT key and then dragging it with the mouse when this parameter is **TRUE**.
bSizeable	The image can be sized with the mouse (by depressing the ALT key) when this parameter is **TRUE**.
ptPos	Specifies the position where to insert the image. The position values must be in *twips* with an origin at the upper left corner of the complete text. The relationship between the upper left corner of the complete text and the upper left corner of the Text Control's client area can be obtained with **CTXTextControl::GetTXScrollPos**.
szScale	Specifies scaling factors.
rcDistances	Specifies distances between the image and the text.
wError	Retrieves an error code. This parameter is set only when the function returns zero. It can be one of the following values:

Value	**Meaning**
0	General error.
1	The file does not exist or cannot be opened.
2	The file is of an unknown type.
3	The file contains an unsupported compression scheme.
4	The file contains an unsupported version.
5	The file contains an unsupported style.
6	The filter cannot be found.
7	The filter uses an unknown interface.

Return Value: The return value is the image's object identifier when the function was successful. Otherwise it is zero.

Values: The following insertion modes are possible for the *wInsertMode* parameter:

Value	Meaning
EOM_INSERTASCHAR	The image is handled like a single character in the text. In this case the *ptPos* and the *rcDistances* parameters are ignored.
EOM_DISPLACELINE	The text flow stops at the top border of the image and continues at the bottom border. Empty areas on the left or right side of the object are not filled. In this case the *lTextPos* parameter is ignored.
EOM_DISPLACEWORD	Same as EOM_DISPLACELINE but empty areas on the left or right side of the image are filled with text so that a line's text is interrupted by the object. In this case the *lTextPos* parameter is ignored.

CTXTextControl::InsertLink

Description: This member function inserts a hypertext link in the document.

Syntax: **UINT InsertLink(const CString&** *strLinkText*, **const CString&** *strLinkTarget*, **BOOL** *bExternal* = **TRUE);**

Parameter	Description
strLinkText	Specifies the link's textual representation.
strLinkTarget	Specifies the location to where the hypertext link points. This can be an address or a file name if the link point to an external location. If the link points to a location inside the same document it must be the name of a target field.
bExternal	Must be set to **TRUE** if *strLinkTarget* defines a location outside of the document, otherwise this parameter must be set to **FALSE**.

Return Value: The return value is the identifier of the newly created marked text field field which defines the hypertext link. See

CTXTextControl::FieldInsert for more information about this identifier.

See Also: **CTXTextControl::ChangeLink, CTXTextControl::InsertTarget, CTXTextControl::FieldGoto**

CTXTextControl::InsertOleFile

Description: This member function inserts a newly created embedded OLE object from a file in a Text Control's document.

Syntax:

UINT InsertOleFile(
const CString& *strFileName,*
LONG *lTextPos* = **-1,**
WORD *wInsertMode* = **EOM_INSERTASCHAR,**
BOOL *bMoveable* = **TRUE,**
BOOL *bSizeable* = **TRUE,**
const CPoint& *ptPos* = **CPoint(0, 0),**
const CSize& *szScale* = **CSize(100, 100),**
const CRect& *rcDistances* = **CRect(0, 0, 0, 0));**

Parameter	Description
strFileName	Specifies the filename.

For a description of all other parameters see **CTXTextControl::InsertOleObject**.

Return Value: The return value is the object's identifier when the function was successful. Otherwise it is zero.

CTXTextControl::InsertOleLinkFile

Description: This member function inserts a newly created linked OLE object from a file in a Text Control's document.

Syntax:

UINT InsertOleLinkFile(
const CString& *strFileName***,**
LONG *lTextPos* = **-1,**
WORD *wInsertMode* = **EOM_INSERTASCHAR,**

BOOL *bMoveable* = **TRUE,**
BOOL *bSizeable* = **TRUE,**
const CPoint& *ptPos* = **CPoint(0, 0),**
const CSize& *szScale* = **CSize(100, 100),**
const CRect& *rcDistances* = **CRect(0, 0, 0, 0));**

Parameter	Description
strFileName	Specifies the filename.

For a description of all other parameters see **CTXTextControl::InsertOleObject**.

Return Value: The return value is the object's identifier when the function was successful. Otherwise it is zero.

CTXTextControl::InsertOleObject

Description: This member function opens the system-defined *OLE Insert* dialog box and inserts the chosen OLE object in a Text Control's document.

Syntax:

UINT InsertOleObject(
LONG *lTextPos* = **-1,**
WORD *wInsertMode* = **EOM_INSERTASCHAR,**
BOOL *bMoveable* = **TRUE,**
BOOL *bSizeable* = **TRUE,**
const CPoint& *ptPos* = **CPoint(0, 0),**
const CSize& *szScale* = **CSize(100, 100),**
const CRect& *rcDistances* = **CRect(0, 0, 0, 0));**

Parameter	Description
lTextPos	Specifies the text position where to insert the object. If *lTextPos* is -1 the object is inserted at the current input position.
wInsertMode	Specifies how the object is handled when the text is formatted. See the Values section of **CTXTextControl::InsertImage** for possible values.
bMoveable	The object can be moved with the mouse when this parameter is **TRUE**.

bSizeable	The object can be sized with the mouse when this parameter is **TRUE**.
ptPos	Specifies the position where to insert the object. The position values must be in *twips* with an origin at the upper left corner of the complete text. The relationship between the upper left corner of the complete text and the upper left corner of the Text Control's client area can be obtained with **CTXTextControl::GetTXScrollPos**.
szScale	Specifies scaling factors.
rcDistances	Specifies distances between the object and the text.

Return Value: The return value is the object's identifier when the function was successful. Otherwise it is zero.

CTXTextControl::InsertOleProgID

Description: This member function inserts an OLE object given through its programmatic identifier. The programmatic identifier is stored under the *ProgID* key in the registration database. For example the programmatic identifier of the Text Control ActiveX is TX.TextControl.110.

Syntax:
UINT InsertOleProgID(
const CString& *strProgID,*
LONG *lTextPos* = **-1,**
WORD *wInsertMode* = **EOM_INSERTASCHAR,**
BOOL *bMoveable* = **TRUE,**
BOOL *bSizeable* = **TRUE,**
const CPoint& *ptPos* = **CPoint(0, 0),**
const CSize& *szScale* = **CSize(100, 100),**
const CRect& *rcDistances* = **CRect(0, 0, 0, 0));**

Parameter	Description
strProgID	Specifies the OLE object's programmatic identifier.

For a description of all other parameters see **CTXTextControl::InsertOleObject**.

Return Value: The return value is the object's identifier when the function was successful. Otherwise it is zero.

CTXTextControl::InsertPageNumber

Description: This member function function inserts a marked text field that displays the current page number.

Syntax: **UINT InsertTarget(DWORD** *dwReserved***);**

Parameter	Description
dwReserved	A reserved parameter for future use. It must be set to zero.

Return Value: The return value is the identifier of the newly created marked text field.

CTXTextControl::InsertTarget

Description: This member function function inserts a hypertext target in the document.

Syntax: **UINT InsertTarget(const CString&** *strTargetName***);**

Parameter	Description
strTargetName	Specifies the target's name.

Return Value: The return value is the identifier of the newly created marked text field field which defines the hypertext target. See **CTXTextControl::FieldInsert** for more information about this identifier.

See Also: **CTXTextControl::ChangeTarget, CTXTextControl::InsertLink, CTXTextControl::FieldGoto**

CTXTextControl::InsertWindow

Description: This member function inserts an externally created window like a Windows button in a Text Control's document. The child window identifier of this window must not be larger than 0x7FFF.

Syntax: **UINT InsertWindow(**
HWND *hWnd*,
LONG *lTextPos* = **-1,**
WORD *wInsertMode* = **EOM_INSERTASCHAR,**
BOOL *bMoveable* = **TRUE,**
BOOL *bSizeable* = **TRUE,**
const CPoint& *ptPos* = **CPoint(0, 0),**
const CSize& *szScale* = **CSize(100, 100),**
const CRect& *rcDistances* = **CRect(0, 0, 0, 0));**

Parameter	Description
hWnd	Specifies a valid window handle.

For a description of all other parameters see **CTXTextControl::InsertOleObject**.

Return Value: The return value is the object's identifier when the function was successful. Otherwise it is zero.

CTXTextControl::LineFromChar

Description: This member function returns the line number of the line which contains the character with the specified character position.

Syntax: **long LineFromChar(long** *lChar***);**

Parameter	Description
lChar	Specifies a zero-based character index.

Return Value: The return value is a line index, started with 0 for the first line. The return value is -1 if an error has occurred.

CTXTextControl::LineFromPoint

Description: This member function 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 Text Control's client area.

Syntax: **long LineFromPoint(const CPoint&** *ptPos***);**

Parameter	**Description**
ptPos	Specifies a geometric position in pixels.

Return Value: The return value is a line index, started with 0 for the first line. The return value is -1 if an error has occurred.

CTXTextControl::LineIndex

Description: This member function returns the character index of a given line. The character index is the number of characters from the beginning of the Text Control to the specified line.

Syntax: **long LineIndex(long** *lLine***);**

Parameter	**Description**
lLine	Specifies a zero-based line index.

Return Value: The return value is the character index of the specified line.

CTXTextControl::LoadFile

Description: This member function loads formatted or unformatted text from a file.

Syntax:

```
BOOL LoadFile(
        CFile& fFile,
        WORD wFormat = TF_FORMAT_TX,
        BOOL bReplaceSel = FALSE,
        DWORD& dwBytesRead = dwNULL);
BOOL LoadFile(
        const CString& strFilename,
        WORD wFormat = TF_FORMAT_TX,
        BOOL bReplaceSel = FALSE,
        DWORD& dwBytesRead = dwNULL);
```

Parameter	**Description**
fFile	Specifies a file from which the text is loaded.
strFilename	Specifies the name of a file from which the text is loaded.

wFormat	Specifies the text format. Possible values are listed in the following Values section.
bReplaceSel	The loaded text replaces the current selection or inserts the text at the current input position when this parameter is **TRUE**. Otherwise the loaded text replaces the complete contents of the Text Control.
dwBytesRead	Retrieves the number of read bytes.

Return Value: The return value is FALSE if an error has occurred. Otherwise it is TRUE.

Values: The following is a list of all text formats that Text Control supports. 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.

Value	**Meaning**
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. Text Control supports the formats of Word 6 (WordPad), Word 95, Word 97 and Word 2000.

CTXTextControl::LoadFromMemory

Description: This member function loads formatted or unformatted text from a buffer.

Syntax:
BOOL LoadFromMemory(
LPBYTE *lpBuf,*
WORD *wFormat* = **TF_FORMAT_TX,**
BOOL *bReplaceSel* = **FALSE,**
DWORD& *dwBytesRead* = **dwNULL);**

Parameter	Description
lpBuf	Points to a buffer containing the text to load. For text-based formats the buffer must be zero-terminated.
wFormat	Specifies the text format. Possible values are listed in the Values section for **CTXTextControl::LoadFile**.
bReplaceSel	The loaded text replaces the current selection or inserts the text at the current input position when this parameter is **TRUE**. Otherwise the loaded text replaces the complete contents of the Text Control.
dwBytesRead	Retrieves the number of read bytes.

Return Value: The return value is FALSE if an error has occurred. Otherwise it is TRUE.

CTXTextControl::ObjDelete

Description: This member function deletes an inserted image, OLE object or window.

Syntax: **BOOL ObjDelete(UINT** *nObjID* = **0);**

Parameter	Description
nObjID	Specifies the object's identifier. If this parameter is zero the currently selected object is deleted.

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

See Also: **CTXTextControl::InsertImage, CTXTextControl::InsertOle***xxx***, CTXTextControl::InsertWindow**.

CTXTextControl::ObjGetAttr

Description: This member function retrieves information about the attributes of an inserted object like insertion mode, position or scaling factors.

Syntax: **BOOL ObjGetAttr(**
UINT *nObjID ,*
LONG& *lTextPos***,**
WORD& *wInsertMode***,**
BOOL& *bMoveable***,**
BOOL& *bSizeable***,**
CPoint& *ptPos***,**
CSize& *szScale***,**
CRect& *rcDistances***);**

Parameter	Description
nObjID	Specifies the object's identifier. If this parameter is zero information about the currently selected object is retrieved.
lTextPos	Retrieves the object's character position in the text. This parameter is only useful when the object's insertion mode is EOM_INSERTASCHAR.
wInsertMode	Retrieves the object's insertion mode. See the Values section of **CTXTextControl::InsertImage** for possible values.

bMoveable	Retrieves TRUE if the object can be moved with the mouse.
bSizeable	Retrieves TRUE if the object can be sized with the mouse.
ptPos	Retrieves the object's geometric position. This parameter is only useful if the object's insertion mode is EOM_DISPLACELINE or EOM_DISPLACEWORD.
szScale	Retrieves the object's scaling factors.
rcDistances	Retrieves the distances between the object and the text. This parameter is only filled when the object's insertion mode is EOM_DISPLACELINE or EOM_DISPLACEWORD.

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

See Also: **CTXTextControl::InsertImage, CTXTextControl::InsertOle***xxx***, CTXTextControl::InsertWindow**.

CTXTextControl::ObjGetIDispatch

Description: This member function returns a pointer to an inserted object's dispatch interface. It can be used to call properties and methods for an object.

Syntax: **BOOL ObjGetIDispatch(UINT** *nObjID*, **LPDISPATCH*** *pDisp***);**

Parameter	Description
nObjID	Specifies the object's identifier.
pDisp	Retrieves the dispatch interface pointer. Text Control calls the **AddRef** method for the object before returning, so the calling application must call the **Release** method when it is done with the object.

Return Value: The return value is **FALSE** if the object has no dispatch interface. Otherwise it is **TRUE**.

CTXTextControl::ObjGetNext

Description: This member function returns the identifier of an inserted object that follows the specified object in the Text Control's internal list of objects. This function can be used to enumerate inserted objects.

Syntax: **UINT ObjGetNext(UINT** *nObjID* = **0, DWORD** *dwFlags* = **0);**

Parameter	**Description**
nObjID	Specifies the object's identifier.
dwFlags	Specifies certain types of objects. See the following Values section for possible values. When this parameter is zero all objects are enumerated.

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

Values: The following lists possible values for the *dwFlags* parameter:

Value	**Meaning**
OGN_ASCHARONLY	Returns only identifiers of objects that act as single characters (insertion mode: EOM_INSERTASCHAR).
OGN_FIXEDONLY	Returns only identifiers of objects which have been inserted with the EOM_DISPLACELINE or EOM_DISPLACEWORD insertion mode.
OGN_IMAGESONLY	Returns only identifiers of objects which have been inserted with **CTXTextControl::InsertImage**.
OGN_EXTERNALSONLY	Returns only identifiers of objects which have been inserted with **CTXTextControl::InsertWindow**.
OGN_OLEOBJECTSONLY	Returns only identifiers of OLE objects.

See Also: **CTXTextControl::InsertImage, CTXTextControl::InsertOle***xxx***, CTXTextControl::InsertWindow**.

CTXTextControl::ObjOleCancel

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

Syntax: **void ObjOleCancel( );**

CTXTextControl::ObjSetDistances

Description: This member function sets new distances between the text and an inserted object. This function can only be used for objects inserted with the insertion mode EOM_DISPLACELINE or EOM_DISPLACEWORD.

Syntax: **BOOL ObjSetDistances(UINT** *nObjID* = **0**, **const CRect&** *rcDistances* = **CRect(0, 0, 0, 0));**

Parameter	Description
nObjID	Specifies the object's identifier. If this parameter is zero the currently selected object is used.
rcDistances	Specifies new distances between the object and the text.

Return Value: The return value is **TRUE** if the new distances could be set. Otherwise it is **FALSE**.

See Also: **CTXTextControl::InsertImage, CTXTextControl::InsertOle***xxx***, CTXTextControl::InsertWindow**.

CTXTextControl::ObjSetMovable

Description: This member function changes the movable state of an inserted object.

Syntax: **BOOL ObjSetMovable(UINT** *nObjID* = **0**, **BOOL** *bMoveable* = **TRUE);**

Parameter	**Description**
nObjID	Specifies the object's identifier. If this parameter is zero the currently selected object is used.
bMovable	When this parameter is TRUE the object can be moved with the mouse. Otherwise it cannot be moved.

Return Value: The return value is **TRUE** if the new state could be set. Otherwise it is **FALSE**.

See Also: **CTXTextControl::InsertImage, CTXTextControl::InsertOle***xxx***, CTXTextControl::InsertWindow**.

CTXTextControl::ObjSetScaling

Description: This member function sets new scaling factors for an inserted object.

Syntax: **BOOL ObjSetScaling(UINT** *nObjID* = **0**, **const CSize&** *szScale* = **CSize(100, 100));**

Parameter	**Description**
nObjID	Specifies the object's identifier. If this parameter is zero the currently selected object is used.
szScale	Specifies new scaling factors.

Return Value: The return value is **TRUE** if the new scaling factors could be set. Otherwise it is **FALSE**.

See Also: **CTXTextControl::InsertImage, CTXTextControl::InsertOle***xxx***, CTXTextControl::InsertWindow**.

CTXTextControl::ObjSetSizeable

Description: This member function changes the sizeable state of an inserted object.

Syntax: **BOOL ObjSetSizeable(UINT** *nObjID* = **0**, **BOOL** *bSizeable* = **TRUE);**

Parameter	Description
nObjID	Specifies the object's identifier. If this parameter is zero the currently selected object is used.
bSizeable	When this parameter is TRUE the object can be sized with the mouse. Otherwise it cannot be sized.

Return Value: The return value is **TRUE** if the new state could be set. Otherwise it is **FALSE**.

See Also: **CTXTextControl::InsertImage, CTXTextControl::InsertOle***xxx***, CTXTextControl::InsertWindow**.

CTXTextControl::ParagraphDialog

Description: This member function 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 paragraph.

Syntax: **BOOL ParagraphDialog(BOOL&** *bChanged* = **bNULL);**

Parameter	Description
bChanged	Retrieves **TRUE** if the dialog box has been left with *Ok*. Otherwise it retrieves **FALSE**.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::Paste

Description: This member function inserts data from the clipboard at the current input position. Data is inserted only if the Text Control has the input focus and if the clipboard contains data in a recognized format.

Syntax: **void Paste();**

CTXTextControl::PrintControl

Description: This member function prints the contents of a Text Control that is used without built-in scroll interface. The contents are printed on the printer's paper with the same offset, the Text Control window has relative to the client area of its parent window. Use this function to print several controls that cover different small text areas on a single page.

Syntax: **BOOL PrintControl(HDC** *hDC***);**

Parameter	Description
hDC	Specifies a printer's device context.

Return Value: The return value is **FALSE** if an errror has occurred. Otherwise it is **TRUE**.

CTXTextControl::PrintPage

Description: This member function prints a single page. It can only be used when the TextControl operates in the page view mode.

Syntax:
BOOL PrintPage(
HDC *hDC,*
UINT *nPage,*
const CPoint& *ptOffset* = **CPoint(0, 0),**
DWORD *dwOptions* = **0,**
WORD *wScale* = **100);**

Parameter	Description
hDC	Specifies a printer device context.
nPage	Specifies the number of the page to print. The first page has the number one.
ptOffset	Specifies an additional printing offset. Text Control adds this offset to the currently set page margins. The values can be negative to print to a position less than the page margins.

dwOptions	Specifies print options. It must contain TF_PRINTCOLORS if text colors are to be printed. If *dwOptions* contains zero, text is printed in black.
wScale	Specifies a scaling factor in percent. This value can range from 10 to 400.

Return Value: The return value is **FALSE** if an errror has occurred. Otherwise it is **TRUE**.

CTXTextControl::Redo

Description: This member function restores the last undone edit operation.

Syntax: **BOOL Redo( );**

Return Value: The return value is **FALSE** if the redo operation fails. Otherwise it is **TRUE**.

CTXTextControl::ReduceFont

Description: This member function reduces the pointsizes of all fonts in the current selection.

Syntax: **BOOL ReduceFont(CSize&** *szMin* = **szNULL);**

Parameter	Description
szMin	Text Control fills this variable with its new minimum window size (in pixels). It is only useful if the Text Control is used without the built-in scroll-interface.

Return Value: The return value is **TRUE**, if the font sizes could be reduced. Otherwise it returns **FALSE**.

CTXTextControl::ReplaceSel

Description: This member function replaces the currenet selection with the specified text.

Syntax: **BOOL ReplaceSel(const CString&** *strText***);**

Parameter	Description
strText	Specifies the replacement text.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::ReplaceText

Description: This member function 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.

Syntax: **void ReplaceText( );**

CTXTextControl::ResetContents

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

Syntax: **BOOL ResetContents( );**

Return Value: The return value is **TRUE** if everything could be deleted. Otherwise it is **FALSE**.

CTXTextControl::SaveFile

Description: This member function saves formatted or unformatted text into a file.

Syntax:

```
BOOL SaveFile(
        CFile& fFile,
        WORD wFormat,
        BOOL bCurSel = FALSE,
        DWORD& dwBytesWritten = dwNULL);
BOOL SaveFile(
        const CString& strFilename,
        WORD wFormat,
        BOOL bCurSel = FALSE,
        DWORD& dwBytesWritten = dwNULL);
```

Parameter	Description
fFile	Specifies a file into which the text will be written.
strFilename	Specifies the name of a file into which the text will be written.
wFormat	Specifies the text format. Possible values are listed in the Values section for **CTXTextControl::LoadFile**.
bReplaceSel	When this parameter is **TRUE** the currently selected text is saved. Otherwise the Text Control's complete text is saved.
dwBytesWritten	Retrieves the number of written bytes.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SaveToMemory

Description: This member function saves formatted or unformatted text in a memory buffer.

Syntax: **BOOL SaveToMemory(CByteArray&** *arBuf,* **WORD** *wFormat,* **BOOL** *bCurSel* = **FALSE);**

Parameter	Description
arBuf	Retrieves the saved text.
wFormat	Specifies the text format. Possible values are listed in the Values section for **CTXTextControl::LoadFile**.
bCurSel	When this parameter is **TRUE** the currently selected text is saved. Otherwise the Text Control's complete text is saved.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SetBackgroundColor

Description: This member function 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.

Syntax: **BOOL SetBackgroundColor(BOOL** *bSysColor*, **COLORREF** *newColor***);**

Parameter	**Description**
bSysColor	Indicates if the background color should be set to the system color for the window background. If this value is **TRUE**, *newColor* is ignored.
newColor	Specifies a RGB value that identifies the new background color.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SetBaseLine

Description: This member function sets a new baseline alignment value for the currently selected text.

Syntax: **BOOL SetBaseLine(WORD** *wFlag* = **FA_STANDARD**, **LONG** *lBaseAlign* = **0);**

Parameter	**Description**
wFlag	Specifies the type of alignment:

Value	**Meaning**
FA_STANDARD	The new alignment is set to zero.
FA_SUPERSCRIPT	The new alignment is superscript.
FA_SUBSCRIPT	The new alignment is subscript.

lBaseAlign	Specifies the new baseline alignment value in *twips*. It is limited to 48 pt = 960 twips.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SetCaretExt

Description: This member function sets the width of the caret. The caret's height depends on the current font.

Syntax: **BOOL SetCaretExt(UINT** *nWidth*, **BOOL** *bTextCaret* = **TRUE);**

Parameter	**Description**
nWidth	Specifies the caret's new width in pixels. 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.
bTextCaret	When this parameter is **TRUE** the new width is set for the caret in standard text sections. When this parameter is **FALSE** the new width is set for the caret in marked text fields.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE.**

CTXTextControl::SetDevicePrinter

Description: This member function sets a new device to which the text of the Text Control is formatted.

Syntax: **BOOL SetDevicePrinter(const CString&** *strPrinter*, **BOOL&** *bChanged* = **bNULL, CSize&** *szMin* = **szNULL);**

Parameter	**Description**
strPrinter	Specifies the name of the new device. This name must be in the same format as that used in the WIN.INI file,

	for example: PostScript Printer,PSCRIPT,LPT1:
bChanged	Retrieves **TRUE** if the device has been changed and settings like fontnames have been adapted. Otherwise it retrieves **FALSE** if the specified device is the same as the current device.
szMin	Text Control fills this variable with its new minimum window size (in pixels) if the new device could not be set because the Text Control's client area was too small to display the text with adapted fonts. It is only useful if the Text Control is used without the built-in scroll-interface.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE.**

CTXTextControl::SetDeviceScreen

Description: This member function sets the screen as the formatting device.

Syntax: **BOOL SetDeviceScreen(BOOL&** *bChanged* = **bNULL, CSize&** *szMin* = **szNULL);**

Parameter	**Description**
bChanged	Retrieves **TRUE** if the device has been changed and settings like fontnames have been adapted. Otherwise it retrieves **FALSE** if the specified device is the same as the current device.
szMin	Text Control fills this variable with its new minimum window size (in pixels) if the new device could not be set because the Text Control's client area was too small to display the text with adapted fonts. It is only useful if the Text Control is used without the built-in scroll-interface.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE.**

CTXTextControl::SetDeviceStandard

Description: This member function sets the system-defined standard device, specified in the [windows] section of the WIN.INI file.

Syntax: **BOOL SetDeviceStandard(BOOL& bChanged = bNULL, CSize&** *szMin* = **szNULL);**

Parameter	Description
bChanged	Retrieves **TRUE** if the device has been changed and settings like fontnames have been adapted. Otherwise it retrieves **FALSE** if the specified device is the same as the current device.
szMin	Text Control fills this variable with its new minimum window size (in pixels) if the new device could not be set because the Text Control's client area was too small to display the text with adapted fonts. It is only useful if the Text Control is used without the built-in scroll-interface.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE.**

CTXTextControl::SetFont

Description: This member function sets a new font with a new size for all selected fonts.

Syntax: **BOOL SetFont(const CString&** *strFont,* **WORD** *wFontSize* = **0, BOOL** *bPoints* = **TRUE, CSize&** *szMin* = **szNULL);**

Parameter	Description
strFont	Specifies the name of the new font.
wFontSize	Specifies a new font size. If this parameter is set to null, only the name is set and all sizes remain the same.
bPoints	If set to **TRUE** *wFontSize* specifies *points*. Otherwise it specifies *twips*.

szMin	Text Control fills this variable with its new minimum window size (in pixels). It is only useful if the Text Control is used without its built-in scroll-interface.

Return Value: The return value is **FALSE** if an error has occurred, otherwise it is **TRUE**.

CTXTextControl::SetFontAttr

Description: This member function sets or resets font attributes for all fonts of the selected text.

Syntax: **BOOL SetFontAttr(DWORD** *dwFlags,* **CSize&** *szMin* = **szNULL);**

Parameter	Description
dwFlags	Can contain one or more of the values listed in the following Values section.
szMin	Text Control fills this variable with its new minimum window size (in pixels). It is only useful if the Text Control is used without its built-in scroll-interface.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Values: The following are the possible font attributes:

Value	Meaning
FA_STANDARD	Resets all attributes of all fonts.
FA_BOLD	Sets each font to bold.
FA_ITALIC	Sets each font to italic.
FA_UNDERLINE	Sets each font to underline.
FA_STRIKEOUT	Sets each font to strike out.
FA_NOBOLD	Resets the bold attribute of each font.
FA_NOITALIC	Resets the italic attribute of each font.
FA_NOUNDERLINE	Resets each underlined font.
FA_NOSTRIKEOUT	Resets each struck out font.

FA_UL_DOUBLE	Sets each font to doubled underline.
FA_UL_REDZIGZAG	Adds a red zigzag line to each font. This underline attribute does not reset other underline attributes.
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 the doubled underline attribute of each font.
FA_UL_NOREDZIGZAG	Resets the red zigzag line attribute of each font.
FA_UL_NOWORDSONLY	Resets each font that has the words only attribute. This value can only be used in combination with FA_NOUNDERLINE or FA_UL_NODOUBLE.
FA_TOGGLE	Toggles the specified attributes instead of adding or resetting them. This flag can be set with any combination. Toggling an attribute results in deleting this attribute if *wFlags* contains the same value as all fonts of the current selection.

CTXTextControl::SetLanguage

Description: This member function sets the language which Text Control uses to display informations strings, warnings or dialog boxes. The language is specified either through an identifier or through the filename of a resource library.

Syntax: **BOOL SetLanguage(UINT** *nLang***);**
BOOL SetLanguage(const CString& *strLang***);**

Parameter	Description
nLang	Specifies a language identifier. The following identifiers are possible:

Language	Identifier
English	01
French	33
Spanish	34
Italian	39
German (Switzerland)	41
German (Austria)	43
German	49
Japanese	81

Parameter	Description
strLang	Specifies the filename including its full path of a resource library. See the chapter *Using the Text Control Class Library - Resources* for more information about creating a resource library.

Return Value: The return value is **FALSE** if an error has occurred or if the specified language has already been set, otherwise it is **TRUE**.

CTXTextControl::SetLineAndCol

Description: This member function sets a new text input position from a page, line and column number. All values start with number 1.

Syntax: **BOOL SetLineAndCol(UINT** *nLine,* **UINT** *nCol,* **UINT** *nPage* = **0);**

Parameter	Description
nLine	Specifies the line number.
nCol	Specifies the column number.
nPage	Specifies the page number. When Text Control works in a view mode that does not display pages, this parameter is ignored and should be set to zero.

Return Value: The return value is **TRUE** if the specified input position could be set. Otherwise it is **FALSE**.

CTXTextControl::SetLineSpacing

Description: This member function sets a new linespacing for all currently selected paragraphs.

Syntax: **BOOL SetLineSpacing(WORD** *wLineSpace*, **BOOL** *bTwips* = **TRUE);**

Parameter	Description
wLineSpace	Specifies a new linespacing value.
bTwips	If this parameter is set to **TRUE**, *wLineSpace* must be a value in *twips*. If set to **FALSE**, *wLineSpace* must be a value in percent of the font size. Before setting the linespacing in *twips*, **CTXTextControl::SetParaFormatFlags** can be used to specify whether the linespacing is used as a minimum, or as an absolute value.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Remarks: To realize double linespacing, *wLineSpace* must contain 200 and *bTwips* must be **FALSE**.

Minimum and maximum values are:
10% to 400% or 57 to 5669 *twips* (1 to 100 mm).

CTXTextControl::SetLinkWnd

Description: This member function informs this Text Control 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 following 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. Chains of linked windows can only be built with Text Controls that work without their built-in scroll.interfaces.

Syntax: **BOOL SetLinkWnd(HWND** *hWnd***);**

Parameter	Description
hWnd	Specifies the window handle of the successor window. Create a new Text Control window and use **CTXTextControl.m_hWnd** for this parameter. If this parameter is zero, Text Control disconnects its successor window.

Return Value: The return value is **FALSE** if the windows could not be linked. Otherwise it is **TRUE**.

CTXTextControl::SetMode

Description: This member function sets Text Control's different working modes. Changing one mode does not alter the other mode settings.

Syntax: **BOOL SetMode(DWORD** *dwNewMode***,**
DWORD *dwMaxAutoSize* = **0, DWORD** *dwNewModeEx* = **0);**

Parameter	Description
dwNewMode	Specifies one or more mode settings. See the following Values section for possible values.
dwMaxAutoSize	This parameter is for the TF_AUTOEXPAND mode only. It specifies maximum values for the Text Control's window size (in pixels). 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.
dwNewModeEx	Specifies one or more extended mode settings. See also the following Values section for possible values.

Return Value: The return value is FALSE if one of the new modes could not be set. Otherwise it is TRUE.

Values: The following modes can be set with the *dwNewMode* parameter:

Value	Meaning
TF_AUTOEXPAND	The Text Control's window will be automatically expanded when text insertion or format changes result in text that does not fit into the Text Control anymore.
TF_FIXED	The Text Control's window size is fixed and is not automatically expanded.
TF_FRAMED	The Text Control window is drawn with a frame of 1 pixel width.
TF_NOTFRAMED	The Text Control window is drawn without a frame.
TF_SHOWSELNA	A text selection remains visible when the control looses the input focus.
TF_HIDESELNA	A text selection is hidden when the control looses the input focus.
TF_SHOWWHITESPACE	Control characters are made visible.
TF_HIDEWHITESPACE	Control characters are hidden.
TF_OVERWRITE	Newly inserted characters overwrite existing characters.
TF_INSERT	Newly inserted characters are inserted.
TF_REPLACESEL	The text of a current selection is deleted before new text is inserted.
TF_KEEPSEL	The text of a current selection is not deleted before new text is inserted.
TF_OPAQUE	The control's background is opaque.
TF_TRANSPARENT	The control's background is transparent.

The following modes can be set with the *dwNewModeEx* parameter:

Value	Meaning
TF_DISPLAY	Text Control only displays text.
TF_READONLY	Text Control displays text and the user can select and copy it.
TF_EDIT	Text Control displays text and the user can select and edit it.
TF_NOWAITCURSOR	Text Control does not change the cursor to an hourglass during long time operations.
TF_WAITCURSOR	Text Control changes the cursor to an hourglass during long time operations.
TF_TOPINDENTFIRSTPG	Text Control allows a top indent for the first paragraph in the text.
TF_NOTOPINDENTFIRSTPG	Text Control suppresses a top indent of the first paragraph.
TF_ERRORBOXES	Text Control displays error message boxes.
TF_NOERRORBOXES	Text Control suppresses all error message boxes.
TF_SHOWGRIDLINES	Text Control shows grid lines in tables.
TF_HIDEGRIDLINES	Text Control hides grid lines in tables.

CTXTextControl::SetPageMargins

Description: This member function sets new page margins.

Syntax: **BOOL SetPageMargins(const CRect&** *rectMargin*, **BOOL** *bReformat* = **FALSE);**

Parameter	Description
rectMargin	Specifies the new margins.

bReformat	If this parameter is **TRUE** the Text Control reformats the complete text. Otherwise the text is not reformatted. If this function is combined with **CTXTextControl::SetPageSize**, it should be called first with *bFormat* set to **FALSE** to avoid doubled reformatting.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Remarks: Page margins are only shown on the screen if the Text Control operates in one of the page view modes. See **CTXTextControl::SetPageSize** for more information.

CTXTextControl::SetPageSize

Description: This member function sets the document's page size and view settings.

Syntax: **BOOL SetPageSize(const CSize&** *szText,* **UINT** *nViewMode,* **UINT** *nScrollInterface***);**

Parameter	**Description**
szText	Specifies the document's page size without page margins. A value of zero means that the text is formatted in the borders of the Text Control's client area.
nViewMode	Specifies a view mode. See the following Values section for possible values. This parameter has only effect when the sizes set through *szText* are non-zero.
nScrollInterface	Specifies scroll interface settings. See the following Values section for possible values. The settings of this parameter has only effect when the sizes set through *szText* are non-zero.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Values: The following is a list of Text Control's document view modes:

Value	Meaning
TF_NORMALVIEW	Text Control displays the text without pages and margins.
TF_PAGEVIEW	Text Control displays pages with margins, borders and a gray background.
TF_EXTPAGEVIEW	Text Control displays three-dimensional pages which are centered in the windows visible area.

The following is a list of Text Control's scroll interface settings:

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	Text Control updates its client area whilst moving the scrollbar's scroll box (thumb).
TF_THUMBPOSITION	Text Control updates its client area when the scrollbar's scroll box (thumb) has reached a new position.

CTXTextControl::SetParaAlignment

Description: This member function sets a new paragraph alignment value for all selected paragraphs.

Syntax: **BOOL SetParaAlignment(WORD** *wAlignment***);**

Parameter	Description
wAlignment	Specifies one of the following values:

Value	Meaning
TF_LEFT	Set left-aligned paragraphs.

TF_RIGHT	Set right-aligned paragraphs.
TF_CENTER	Set centered paragraphs.
TF_BLOCK	Set to block formatted paragraphs.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SetParaFormatFlags

Description: This member function sets advanced paragraph formatting attributes.

Syntax: **BOOL SetParaFormatFlags(DWORD** *dwFlags***);**

Parameter	Description
dwFlags	Specifies the new formatting. Possible values are listed in the following Values section.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Values: The following are the advanced attributes:

Value	Meaning
TF_ATLEASTLINESPACING	If a specified line spacing is too small to show all the line's contents, Text Control enlarges the line spacing between lines accordingly, so that nothing is cropped.
TF_EXACTLINESPACING	A specified line spacing is used as exact value, 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.

CTXTextControl::SetParaFrame

Description: This member function sets appearance flags, frame width and frame distance values for all paragraphs of the current selection.

Syntax: **BOOL SetParaFrame(WORD** *wFlags*, **WORD** *wWidth* = **0, WORD** *wDistance* = **-1);**

Parameter	Description
wFlags	Specifies the appearance and the style of the paragraph frame. It can be a combination of the values listed in the following Values section.
wWidth	Specifies the paragraph frame's line width in *twips*. If this parameter is set to zero it is ignored.
wDistance	Specifies the distance between the frame and the text in twips. If this parameter is set to -1 it is ignored.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Values: For a paragraph frame's appearance and styles the following values are possible:

Value	Meaning
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.

CTXTextControl::SetParaIndents

Description: This member function sets new indent values for all currently selected paragraphs.

Syntax: **BOOL SetIndents(const CRect&** *rcIndents,* **int** *iFirstIndent,* **BOOL&** *bChanged* = **bNULL);**

Parameter	**Description**
rcIndents	Specifies the paragraphs' new indents.
iFirstIndent	Specifies an additional left indent for the first line. This value can be negative indicating that the left indent of the first line is smaler than the left indent of the following lines.
bChanged	Retrieves TRUE if the new values could not be accepted because they are too large for the currently set page size.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SetSel

Description: This member function sets a new text selection.

Syntax: **BOOL SetSel(long** *lStart***, long** *lEnd***);**

Parameter	Description
lStart	Specifies the zero-based text input position where the new selection starts.
lEnd	Specifies the zero-based text input position where the new selection ends.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Remarks: If the start position is zero and the end position is -1 the entire text is selected.

CTXTextControl::SetTabs

Description: This member function sets new tab positions and types.

Syntax: **BOOL SetTabs(LPTABSCT** *pTabs*, **BOOL** *bTwips* = **TRUE);**

Parameter	Description
pTabs	Points to an array of type TABSCT and size NTABS. See *Data Structures* for a description of the TABSCT structure.
bTwips	When this parameter is set to **TRUE** all tab position values must be in *twips*, otherwise they must be in pixels.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SetTextBkColor

Description: This member function sets a new text background color for the currently selected text.

Syntax: **BOOL SetTextBkColor(DWORD** *dwDefColor*, **COLORREF** *newColor***);**

Parameter	**Description**
dwDefColor	Specifies one of the following values:

Value	**Meaning**
CV_BKDEFAULT	The new text background color is the system color for the window background.
CV_BKCONTROL	The new text background color is Text Control's background color.
CV_BKUSER	The new text color is specified through *newColor*.

Parameter	**Description**
newColor	Specifies a RGB value that identifies the new text background color.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

Remarks: If the Text Control's background mode is TF_TRANSPARENT and the *dwDefColor* parameter contains CV_BKDEFAULT or CV_BKCONTROL the text background color is not drawn.

CTXTextControl::SetTextColor

Description: This member function sets a new text color for the currently selected text.

Syntax: **BOOL SetTextColor(BOOL** *bSysColor*, **COLORREF** *newColor***);**

Parameter	**Description**
bSysColor	Indicates if the text color should be set to the system color for window text. If this value is **TRUE**, *newColor* is ignored.
newColor	Specifies a RGB value that identifies the new text color.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SetTXScrollPos

Description: This member function sets a new scroll position.

Syntax: **BOOL SetTXScrollPos(WORD** *wDir*, **DWORD** *dwPos***);**

Parameter	Description
wDir	Specifies the direction. It can be one of the following values: **Value** — **Meaning** TF_HSCROLL — Sets the horizontal scroll position. TF_VSCROLL — Sets the vertical scroll position.
dwPos	Specifies the new scroll position in *twips*. The text associated with this position is displayed at the top of the Text Control's client area.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::SetZoom

Description: This member function sets a new zooming factor for the Text Control. This factor is given as a percentage. A value of 100 means the original size.

Syntax: **BOOL SetZoom(UINT** *nNewZoom*, **BOOL** *bUpdate* = **FALSE);**

Parameter	Description
wNewZoom	Specifies the new zooming factor in percent. It must be between 10 and 400.
bUpdate	Updates the appropiate portion of the parent window's client area, if set to **TRUE**.

Return Value: The return value is **FALSE** if the window could not be zoomed or if the specified zooming factor has already been set. Otherwise it is **TRUE**.

CTXTextControl::TableAttrDialog

Description: This member function opens a built-in dialog box for setting table attributes such as frames and distances between frame and text.

Syntax: **BOOL TableAttrDialog(BOOL&** *bChanged* = **bNULL);**

Parameter	Description
bChanged	Retrieves **TRUE** if the dialog box has been left with *Ok*. Otherwise it retrieves **FALSE**.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**. The return value also is FALSE if the selection contains no table, or more than one table, or if the selected table is mixed with other text. When this function is called as a reaction to a menu, use **CTXTextControl::TableIsPossible** to get the information whether or not table attributes can be set.

CTXTextControl::TableDeleteLines

Description: This member function deletes the currently selected table lines or the table line at the current input position.

Syntax: **BOOL TableDeleteLines( );**

Return Value: The return value is **FALSE** 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 **TRUE**.

CTXTextControl::TableFromCaretPos

Description: This member function retrieves the identifier and the number of row and column of the table with 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.

Syntax: **UINT TableFromCaretPos(WORD&** *wRow* = **wNULL, WORD&** *wCol* = **wNULL);**

Parameter	Description
wRow	Retrieves a table row number.
wCol	Retrieves a table column number.

Return Value: The return value is the identifier of the table with the current input position.

CTXTextControl::TableGetAttr

Description: This member function retrieves information about the attributes of one or more table cells.

Syntax:

BOOL TableGetAttr(
UINT *nTableID,*
WORD *wRow,*
WORD *wCol,*
CRect& *rcFrameWidth,*
CRect& *rcDistances,*
COLORREF& *crBkColor* **= dwNULL**,
long& *lxPos* = **lNULL**,
long& *lxExt* = **lNULL);**

Parameter	Description
nTableID	Specifies a table's identifier.
wRow	Specifies a row in this table. If this parameter is zero the attributes of all rows are retrieved.
wCol	Specifies a column in this table. If this parameter is zero the attributes of all columns are retrieved.
rcFrameWidth	Retrieves the width of the cell's frame lines in *twips*.
rcDistances	Retrieves the distances between the cell's frame and the cell's text in *twips*.
crBkColor	Retrieves the cell's background color as an RGB value.
lxPos	Retrieves the cell's horizontal position in *twips*.
lxExt	Retrieves the cell's width in *twips*.

Return Value: The return value is **FALSE** if an error has occurred or if the table or the specified cell in this table does not exist. Otherwise it returns **TRUE**.

Remarks: If the specified cells are formatted differently, the appropriate parameter for a certain attribute retrieves -1.

CTXTextControl::TableGetCellPosition

Description: This member function retrieves the indexes (one-based) of a table cell's first and last character.

Syntax: **BOOL TableGetCellPosition(UINT** *nTableID,* **WORD** *wRow,* **WORD** *wCol,* **DWORD&** *dwStart,* **DWORD&** *dwEnd***);**

Parameter	Description
nTableID	Specifies a table's identifier.
wRow	Specifies a row in this table.
wCol	Specifies a column in this table.
dwStart	Retrieves the index of the table cell's first character.
dwEnd	Retrieves the index of the table cell's last character.

Return Value: The return value is **FALSE** if an error has occurred or if the specified table identifier does not exist, otherwise it is **TRUE**.

Remarks: If tables are used in chains of linked Text Controls 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 character position of the table in this window use **CTXTextControl::GetLinkWndFromOffset** and **CTXTextControl::GetLinkWndOffset**.

CTXTextControl::TableGetCellText

Description: This member function retrieves the text of a table cell.

Syntax: **BOOL TableGetCellText(UINT** *nTableID,* **WORD** *wRow,* **WORD** *wCol,* **CString&** *strText***);**

Parameter	Description
nTableID	Specifies a table's identifier.
wRow	Specifies a row in this table.
wCol	Specifies a column in this table.
strText	Retrieves the text of the specified cell.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::TableGetNext

Description: This member function returns an enumeration number of a table that follows the specified table in the Text Control's current text. It can be used to enumerate all tables. In a list of linked Text Controls the tables in all windows are enumerated.

Syntax: **UINT TableGetNext(UINT** *nEnum* = **0, UINT&** *nTableID* = **uiNULL);**

Parameter	Description
nEnum	Specifies an enumeration number. The function returns the enumeration number of the table that follows the table with this number. If this parameter is zero the first table's enumeration number is returned.
nTableID	Retrieves the table's identifier. This is the same value set with **CTXTextControl::TableInsert**.

Return Value: The return value is the enumeration number of the next table. It can be used for the next **TableGetNext** function call. The return value is zero when the last table has been reached or when the specified enumeration number was invalid.

CTXTextControl::TableGetRowsAndCols

Description: This member function returns the number of rows and columns for the specified table.

Syntax: **UINT TableGetRowsAndCols(WORD&** *wRows,* **WORD&** *wCols,* **UINT** *nTableID* = **0);**

Parameter	Description
wRows	Retrieves the number of rows.
wCols	Retrieves the number of columns.
nTableID	Specifies a table's identifier. If this parameter is zero the function retrieves the number of rows and columns for the table at the current text input position.

Return Value: The return value is the table's identifier. This is the same value as specified through *nTableID* or the identifier of the table with the current input position. The return value is zero if an error has occurred or if the current input position is not inside a table and *nTableID* has been set to zero.

CTXTextControl::TableInsert

Description: This member function inserts a new table into the text.

Syntax: **UINT TableInsert(WORD** *wRows,* **WORD** *wCols,* **UINT** *nTableID* = **0);**

Parameter	Description
wRows	Specifies the number of rows in the new table.
wCols	Specifies the number of columns in the new table.
nTableID	Specifies the table identifier for the new table. It must be a value between 10 and 0x7FFF. If this parameter is 0, Text Control chooses its own identifier for the new table.

Return Value: The return value is the table's identifier. It is either the specified identifier or an identifier choosed through Text Control. The return value is zero if the new table could not be created.

CTXTextControl::TableIsPossible

Description: This member function returns **TRUE** if the specified action is possible.

Syntax: **BOOL TableIsPossible(UINT** *nAction =* **TF_TABLE_CANINSERT);**

Parameter	Description
nAction	Specifies the action to perform. Possible values are listed in the Values section.

Return Value: The return value is **TRUE** if the specified action can be performed. Otherwise it is **FALSE**.

Values: The following actions can be requested:

Value	Meaning
TF_TABLE_CANINSERT	**TRUE** is returned if a table can be inserted at the current input position. **FALSE** is returned if a section of text has been selected or the current input position is inside a table.
TF_TABLE_CANDELETELINES	**TRUE** is returned if selected table lines can be deleted. **FALSE** is returned if no table line is selected or if the current input position is outside a table.
TF_TABLE_CANCHANGEATTR	**TRUE** is returned if the attributes of selected table lines can be altered. **FALSE** is returned if the selection is not completely within a single table.

CTXTextControl::TableSetAttr

Description: This member function sets new attributes for one or more table cells.

Syntax: **BOOL TableSetAttr(**
UINT *nTableID,*

```
WORD wRow,
WORD wCol,
CRect& rcFrameWidth = CRect(-1, -1, -1, -1),
CRect& rcDistances = CRect(-1, -1, -1, -1),
COLORREF& crBkColor = -1,
long lxPos = -1,
long lxExt = -1);
```

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td><i>nTableID</i></td><td>Specifies a table's identifier.</td></tr>
<tr><td><i>wRow</i></td><td>Specifies a row in this table. If this parameter is zero the attributes of all columns are changed.</td></tr>
<tr><td><i>wCol</i></td><td>Specifies a column in this table. If this parameter is zero the attributes of all columns are changed.</td></tr>
<tr><td><i>rcFrameWidth</i></td><td>Sets the width of the cell's frame lines in <i>twips</i>. Each value set to -1 is ignored.</td></tr>
<tr><td><i>rcDistances</i></td><td>Sets the distances between the cell's frame and the cell's text in <i>twips</i>. Each value set to -1 is ignored.</td></tr>
<tr><td><i>crBkColor</i></td><td>Sets the cell's background color. The following values are possible:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td><b>RGB(</b><i>r</i>, <i>g</i>, <i>b</i><b>)</b></td><td>Specifies an RGB color value.</td></tr>
<tr><td>CV_SYS_COLOR</td><td>The color is set to the system color for the window background.</td></tr>
<tr><td>CV_CTL_COLOR</td><td>The color is set to the currently defined control background.</td></tr>
<tr><td>-1</td><td>The parameter is ignored.</td></tr>
</table>
</td></tr>
<tr><td><i>lxPos</i></td><td>Sets the cell's horizontal position in <i>twips</i>. If set to -1 this parameter is ignored.</td></tr>
<tr><td><i>lxExt</i></td><td>Sets the cell's width in <i>twips</i>. If set to -1 this parameter is ignored.</td></tr>
</table>

Return Value: The return value is **FALSE** if an error has occurred or if the table or the specified cell in this table does not exist. Otherwise the return value is **TRUE**.

CTXTextControl::TableSetCellText

Description: This member function alters the text of a table cell.

Syntax: **BOOL TableSetCellText(UINT** *nTableID*, **WORD** *wRow*, **WORD** *wCol*, **const CString&** *strText***);**

Parameter	Description
nTableID	Specifies a table's identifier.
wRow	Specifies a row in this table.
wCol	Specifies a column in this table.
strText	Specifies the new text for the given cell.

Return Value: The return value is **FALSE** if an error has occurred. Otherwise it is **TRUE**.

CTXTextControl::Undo

Description: This member function undoes the last edit operation.

Syntax: **BOOL Undo( );**

Return Value: The return value is **FALSE** if the undo operation fails. Otherwise it is **TRUE**.

Remarks: Use **CTXTextControl::CanUndo** to determine whether an operation can be undone. If this function is called although there is no operation that can be undone the Text Control beeps.

CTXToolContainer

#include <TXToolContainer.h>

The **CTXToolContainer** class is a base class that can be used with classes that have embedded objects of the type **CTXButtonBar**, **CTXRulerBar** and/or **CTXStatusBar**. To use this class, derive the class which contains embedded tool bars from **CTXToolContainer** and override the member functions associated with the contained tool bars.

For example if a **CFrameWnd** derived class, called **CMainFrame,** has an embedded object of the type **CTXButtonBar**, derive **CMainFrame** from **CTXToolContainer** and override **CTXToolContainer::GetButtonBar**.

The **CTXView** class looks for a tool container and uses the tool container's member functions to connect the tool bars with its embedded Text Control.

CTXToolContainer Class Members

Overridables	
GetButtonBar	Retrieves a **CTXButtonBar** object.
GetRulerBar	Retrieves a **CTXRulerBar** object.
GetStatusBar	Retrieves a **CTXStatusBar** object.

Member Functions

CTXToolContainer::GetButtonBar

Description:	This member function retrieves a **CTXButtonBar** object associated with the class used as tool container. If the tool container has a Button Bar override this function and return a pointer to this **CTXButtonBar** object. The default implementation returns zero to indicate that there is no Button Bar.
Syntax:	**CTXButtonBar* GetButtonBar( );**
Return Value:	The return value is a pointer to a **CTXButtonBar** object or zero if there is no Button Bar.

CTXToolContainer::GetRulerBar

Description:	This member function retrieves a **CTXRulerBar** object associated with the class used as tool container. If the tool container has a Ruler Bar override this function and return a pointer to this **CTXRulerBar** object. The default implementation returns zero to indicate that there is no Ruler Bar.
Syntax:	**CTXRulerBar* GetRulerBar( );**
Return Value:	The return value points to a **CTXRulerBar** object or zero if there is no Ruler Bar.

CTXToolContainer::GetStatusBar

Description:	This member function retrieves a **CTXStatusBar** object associated with the class used as tool container. If the tool container has a Status Bar override this function and return a pointer to this **CTXStatusBar** object. The default implementation returns zero to indicate that there is no Status Bar.
Syntax:	**CTXStatusBar* GetStatusBar( );**
Return Value:	The return value points to a **CTXStatusBar** object or zero if there is no Status Bar.

CTXView

#include <TXView.h>

The **CTXView** class, with **CTXDoc**, provides the functionality of a Text Control within the context of MFC's document view architecture. Each instance of this class contains an embedded Text Control object. **CTXView::GetTextControl** provides access to the embedded Text Control. In addition to the functionality provided through the embedded Text Control the **CTXView** class has command handler functions for predefined menu resources.

To be able to handle notification messages sent by the embedded Text Control, the **CTXView** class is implemented as a Text Control notify handler. Therefore the view contains all the overridable member functions described for the **CTXNotifyHandler** class. Override all the functions associated with the notification messages you want to handle.

CTXView Class Members

Attributes

GetTextControl	Retrieves the Text Control associated with the view.
GetRulerBar	Retrieves the Ruler Bar associated with the view.
GetButtonBar	Retrieves the Button Bar connected with the view.
GetStatusBar	Retrieves the Status Bar connected with the view.

Overridables

CreateTextControl	Creates the embedded Text Control.
GetDefaultMode	Retrieves the default mode settings for the embedded Text Control.
GetDefaultModeEx	Retrieves the default extended mode settings for the embedded Text Control.

Member Functions

CTXView::CreateTextControl

Description: This member function is called by the view to create its associated Text Control. Override this function if you want to alter the default creation mechanism. The view calls this function with itself as parent window and as notify handler.

Syntax: **CTXTextControl* CreateTextControl(CWnd*** *pParentWnd,* **UINT** *nID,* **const CRect&** *rcSize,* **CTXNotifyHandler*** *pNotifyHandler***);**

Parameter	Description
pParentWnd	Specifies the Text Control's parent window.
nID	Specifies the Text Control's identifier.
rcSize	Specifies the Text Control's size and position in client area coordinates of its parent window.
pNotifyHandler	Points to a notification handler object.

Return Value: The return value is a pointer to the created **CTXTextControl** object. This pointer is retrieved through following **CTXView::GetTextControl** calls.

CTXView::GetButtonBar

Description: This member function retrieves the **CTXButtonBar** object connected with this **CTXView** object.

Syntax: **CTXButtonBar* GetButtonBar( );**

Return Value: The return value is a **CTXButtonBar** object. It is zero if there is no connected Button Bar.

CTXView::GetDefaultMode

Description: This member function is called by the view to get default mode settings for its embedded Text Control. Mode settings are documented for the *dwNewMode* parameter of the **CTXTextControl::SetMode** function. The default implementation of this function retrieves TF_OPAQUE, TF_FIXED, TF_SHOWSELNA, TF_NOTFRAMED, TF_INSERT, TF_REPLACESEL and TF_HIDEWHITESPACE. Override this function to use other mode settings as default.

Syntax: **DWORD GetDefaultMode( );**

Return Value: The return value is a combination of the default mode settings.

CTXView::GetDefaultModeEx

Description: This member function is called by the view to get default extended mode settings for its embedded Text Control. Extended mode settings are documented for the *dwNewModeEx* parameter of the **CTXTextControl::SetMode** function. The default implementation of this function retrieves TF_EDIT, TF_WAITCURSOR, TF_NOTOPINDENTFIRSTPG, TF_ERRORBOXES and TF_SHOWGRIDLINES. Override this function to use other extended mode settings as default.

Syntax: **DWORD GetDefaultModeEx( );**

Return Value: The return value is a combination of the default extended mode settings.

CTXView::GetRulerBar

Description: This member function retrieves the **CTXRulerBar** object associated with this **CTXView** object.

Syntax: **CTXRulerBar* GetRulerBar( );**

Return Value: The return value is a **CTXRulerBar** object. It is zero if there is no associated Ruler Bar.

CTXView::GetStatusBar

Description:	This member function retrieves the **CTXStatusBar** object connected with this **CTXView** object.
Syntax:	**CTXStatusBar* GetStatusBar( );**
Return Value:	The return value is a **CTXStatusBar** object. It is zero if there is no connected Status Bar.

CTXView::GetTextControl

Description:	This member function retrieves the **CTXTextControl** object associated with this **CTXView** object.
Syntax:	**CTXTextControl* GetTextControl( );**
Return Value:	The return value is the **CTXTextControl** object for this view. It is zero if there is no associated Text Control.

Data Structures

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:

Field	Description	
nTabFlag	Specifies the type of the tabstop. It can be any one of the following values:	
	Value	**Meaning**
	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 system-defined decimal sign is located at the tab position.
wTabPos	Specifies the x-coordinate of the tab position.	

Index