TX Text Control

Getting Started

TX Text Control 7.0

Contents

Getting Started

Now that you have TX Text Control, the first thing you will want to do is to install it onto your system. To perform the installation, simply insert the Text Control CD into your CD ROM drive and run the *Setup.exe* program. If you have downloaded the Text Control installation file from the Text Control web site, extract the .ZIP file and run the contained *Setup.exe* program.

By choosing all the default options while *Setup.exe* is running you will install both, the Text Control ActiveX Control and the Text Control C++ Class Library. Choosing *Custom* in the *Setup Type* dialog box you can select either the ActiveX Control or the Class Library or both.

Once the installation is complete, the quickest way to get an overview of what you can do with TX Text Control is to look at the sample programs. The most comprehensive sample included with Text Control, is the *TX Text Control Words* demo program, that can either be started directly from the last dialog of the setup program or through an item in the Text Control program folder.

TX Text Control Words illustrates many of the features in TX Text Control, but it only begins to demonstrate the full creative potential available with TX. Because of TX Text Control's scalable design, you do not have to stick to standard word processing style user interfaces. You can design the user interface that is right for your application and you can pick and choose from the appropriate controls included with Text Control.

The remainder of this book is devoted to creating a simple word processor with TX Text Control. For ActiveX Control users it contains examples for the Microsoft Visual Basic, Borland Delphi, and Microsoft Visual C++ development enviroments.

For Class Library users it contains a short tutorial on how to integrate TX Text Control in the Microsoft Visual C++ development enviroment.

Starting with the ActiveX

Introduction

Welcome to TX Text Control, the comprehensive text integration tool in a single ActiveX control. Using Text Control, you can create all kinds of text-based applications with the ease of programming that is characteristic of Microsoft Visual Basic, with highly sophisticated formatting and display capabilities which are normally the exlusive domain of large word processing packages.

System Requirements

The Text Control ActiveX control requires the following minimum configuration:

- Windows 95/98, Windows NT 4.0 or Windows 2000.
- Microsoft Visual Basic, Borland Delphi, Microsoft Visual C++ or one of many other development platforms which support ActiveX controls.

Distributing your Applications

The table below shows all the files necessary for the Text Control ActiveX 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	TX4OLE.OCX
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	MFC40.DLL
	MSVCRT40.DLL
4	TX_GIF32.FLT

The first file (group 1) is the Text Control ActiveX server containing the ActiveX controls. These controls must be registered in the registration database on your client's machine.

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

You should also verify that the Microsoft foundation class library files (group 3) are installed on your client's computer. These files must be installed in the Windows system directory. Please refer to Microsoft's redistribution policy if you need to redistribute them.

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.

A Simple Word Processor in Visual Basic

This chapter shows you how to create a small 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, a status bar and full keyboard and mouse

interface. The source code for this example is contained in the *Simple* sample source directory.

Creating the Project

Assuming that you have already run the Text Control installation program and started Visual Basic, the next step is to create a project for the text processor. To do this begin by selecting the *New Project* command from the file menu. Then use the *Tools / Custom Controls...* command to include the file 'tx4ole.ocx' into the new project. You will see four additional icons appear at the bottom of the toolbox, representing the Text Control and its Status Bar, Button Bar and Ruler:

 The Text Control Icon

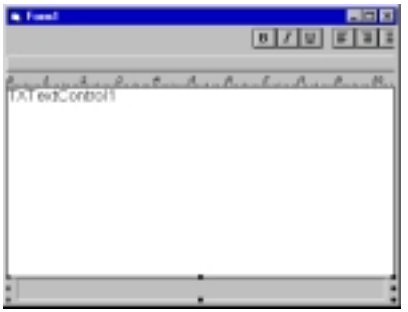
 The Status Bar Icon

 The Button Bar Icon

 The Ruler Icon

Creating the Controls

The next step is to put these four controls onto a form and connect them. Click on the Text Control icon and draw it on the form. In the same way, create a Ruler and a Button Bar on top of the Text Control, and a Status Bar below it. Your form should now look like the diagram on the right:

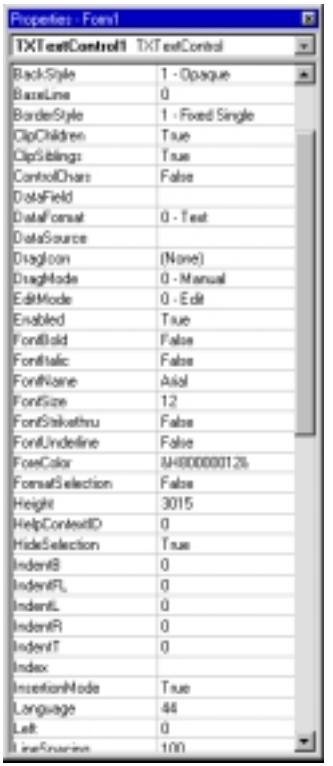


Connecting the Controls

Add the following code to the form's **Load** event procedure:

```
Private Sub Form_Load()
    TXTextControl1.ButtonBarHandle = TXButtonBar1.hWnd
    TXTextControl1.RulerHandle = TXRuler1.hWnd
    TXTextControl1.StatusBarHandle = TXStatusBar1.hWnd
End Sub
```

Running the Program



The text processor is not yet finished, but we can make a first attempt at running it to see what it can do. Click the 'Start' button. You can type in some text, select it with the mouse, copy it to the clipboard (use the <CTRL>+<C> and <CTRL>+<V> keys as long as there is no menu), select a different font, set tabs and do lots of other things. All of these features have been built into the Text Control and can be used with almost no programming effort.

You will have noticed, however, that some features are still missing. For instance, if you resize the main window, the controls keep their old sizes. There is no menu, and there are no scrollbars either. We will fix this in the coming chapters.

Adding Scrollbars

To add scrollbars, click on the Text Control window to have its property list displayed. Click on the **Scrollbars** property and select *3 - Both*. Select the **PageWidth** property and enter 12000, which is about the width of a letter in twips, the currently selected measurement. Set **PageHeight** to 15000 for now.

Resizing the Controls

Two steps are involved in making the controls resize properly when the main window is resized.

- Set the **Align** property to *1 - Align Top* for the Button Bar, the Ruler and the Text Control. Set it to *2 - Align Bottom* for the Status Bar. This will adjust everything except the height of the Text Control.

- Open the code window for the form which contains the Text Control. In the combo boxes on top of the code window, select 'Form' in the 'Object:' box and 'Resize' in the 'Proc:' box. The code window should show an empty procedure for the **Resize** event:

```
Private Sub Form_Resize ()
End Sub
```

Extend it as follows:

```
Private Sub Form_Resize ()
    TXTextControl1.Height = ScaleHeight - TXRuler1.Height _
    - TXStatusBar1.Height - TXButtonBar1.Height
End Sub
```

This line of code will cause the Text Control's height to be adjusted every time the size of the form is altered. (The ' _' character is used to extend one logical line of code to two or more physical lines).

Adding a Menu

In this section, you will add a menu to the text processor to enable you to call the Text Control's built-in dialog boxes.

Use the Visual Basic Menu Editor to create a *Format* menu with the items *Character...* and *Paragraph...*.

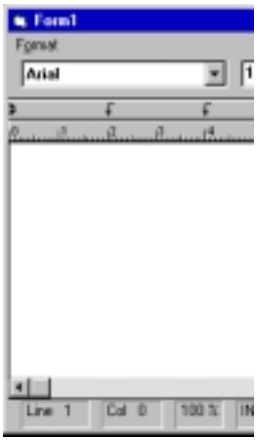
Name the items 'mnuFomat_Character' and 'mnuFormat_Paragraph'. (Please refer to the Visual Basic documentation if you need help with creating menus).

Add the following code to the Click procedures of the menu items:

```
Private Sub mnuFormat_Character_Click()
   TXTextControl1.FontDialog
```

```
End Sub

Private Sub mnuFormat_Paragraph_Click()
   TXTextControl1.ParagraphDialog
End Sub
```



Start the program again. You should be able to use the menu items to call the Font and Paragraph dialog boxes.

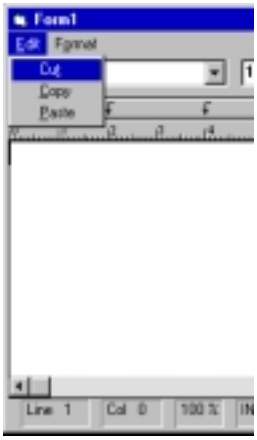
Now for the *Edit* menu. Again use the Menu Design Window and create an *Edit* menu containing items for *Cut*, *Copy*, and *Paste*. The code for these menu items is:

```
Private Sub mnuEdit_Cut_Click()
   TXTextControl1.Clip 1
End Sub

Private Sub mnuEdit_Copy_Click()
   TXTextControl1.Clip 2

End Sub

Private Sub mnuEdit_Paste_Click()
   TXTextControl1.Clip 3
End Sub
```



Having added these menu items, you can exchange formatted text with other word processors via the clipboard.

Finally, we shall add one last menu. Create a *File* menu including the items *Load...* and *Save As...*. Place a common dialog box icon on the form and enter the following code, which will call the common dialog box to get a file name from the user, and will then load respectively save the selected file:

```
Private Sub mnuFile_Load_Click()
   On Error Resume Next

   ' Create an "Open File" dialog box
   CommonDialog1.Filter = "TX Demo (*.tx)|*.tx"
```

```
   CommonDialog1.DialogTitle = "Open"
   CommonDialog1.Flags = cdlOFNFileMustExist Or _
      cdlOFNHideReadOnly
   CommonDialog1.CancelError = True
   CommonDialog1.ShowOpen
   If Err Then Exit Sub

   ' Pass the filename to the text control
   TXTextControl1.Load CommonDialog1.filename, 0
End Sub

Private Sub mnuFile_SaveAs_Click()
   On Error Resume Next

   ' Create a "Save File" dialog box
   CommonDialog1.Filter = "TX Demo (*.tx)|*.tx"
   CommonDialog1.DialogTitle = "Save As"
   CommonDialog1.Flags = cdlOFNOverwritePrompt Or _
      cdlOFNHideReadOnly
   CommonDialog1.CancelError = True
   CommonDialog1.ShowSave
   If Err Then Exit Sub

   ' Open the selected file
   TXTextControl1.Save CommonDialog1.filename, 0
End Sub
```

A Simple Word Processor in Delphi

This chapter shows you how to create a small 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, a status bar and full keyboard and mouse interface.

The source code for this example is contained in the *Simple* sample source directory.

Creating the Project

Assuming that you have already run the Text Control installation program and started Delphi, the next step is to create a project for the text processor. To do this begin by selecting the *New Application* command from the file menu. If you have already imported Text Control into Delphi, its icons are shown when the ActiveX tab is selected. Otherwise, click on *Controls / Import ActiveX...* and choose TX Text Control from the given list. Click *Install* and then *OK* until all dialog boxes have been closed. Now you will see the following four additional icons when the ActiveX tab is selected:

 The Text Control Icon

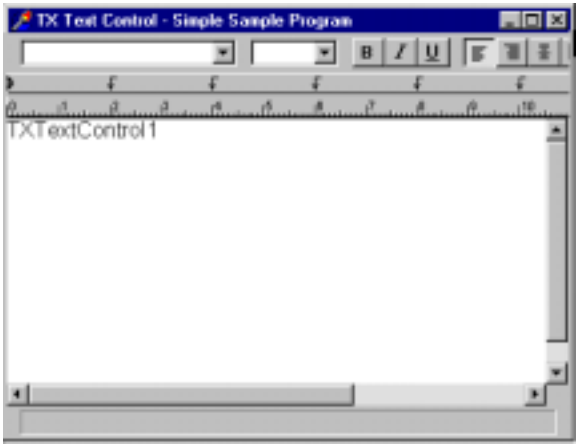
 The Status Bar Icon

 The Button Bar Icon

 The Ruler Icon

Creating the Controls

The next step is to put these four controls on a form and connect them. Run Delphi and create a new project. Select the ´OCX´ page in the component palette to have the 4 Text Control icons displayed. Click on the Text Control icon and draw it on the form. In the same way, create a Ruler and a Button Bar on top of the Text Control, and a Status Bar below it. Your form should now look like the diagram on the right:



Connecting the Controls

Add the following code to the form's FormShow Event procedure:

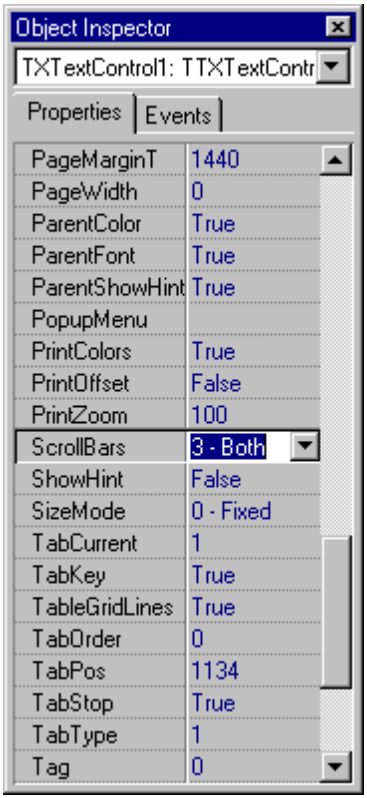
```
procedure TForm1.FormShow(Sender : TObject);
begin
   TXTextControl1.ButtonBarHandle := TXButtonBar1.hWnd;
   TXTextControl1.RulerHandle := TXRuler1.hWnd;
   TXTextControl1.StatusBarHandle := TXStatusBar1.hWnd;
end;
```

Running the Program

The text processor is not yet finished, but we can make a first attempt at running it and seeing what it can do. Click the Start button. You can type in some text, select it with the mouse, copy it to the clipboard (use the <CTRL>+<C> and <CTRL>+<V> keys as long as there is no menu), select a different font, set tabs and do lots of other things. All of

these features have been built into the Text Control and can be used with almost no programming effort.

You will have noticed, however, that some features are still missing. For instance, if you resize the main window, the controls keep their old sizes. There is no menu, and there are no scrollbars either. We will fix this in the coming chapters.



Adding Scrollbars

To add Scroll Bars, click on the Text Control window to have its property list displayed. Click on the Scrollbars property and enter *3 - Both*. Select the PageWidth property and enter 12000, which is about the width of a letter in twips, the currently selected measurement. Set PageHeight to 15000 for now.

Resizing the Controls

Two steps are involved in making the controls resize properly when the main window is resized:

- Set the **Align** property to *alTop* for the Button Bar, the Ruler and the Text Control. Set it to *alBottom* for the Status Bar. This will adjust everything except the height of the Text Control.
- Change to the events listing in the property window and double-click the **OnResize** event. The code window should show an empty procedure for the **Resize** event:

```
procedure TForm1.FormResize(Sender: TObject);
begin

end;
```

Extend it as follows:

```
procedure TForm1.FormResize(Sender: TObject);
begin
   TXTextControl1.Height := ClientHeight - TXRuler1.Height
      - TXStatusBar1.Height - TXButtonBar1.Height;
   TXTextControl1.Width := ClientWidth;
   TXRuler1.Width := ClientWidth;
end;
```

This line of code will cause the Text Control's height and width to be adjusted every time the size of the form is altered.

Adding a Menu

In this chapter, you will add a menu to the text processor to enable you to call the Text Control's built-in dialog boxes.

Use the Delphi Menu Component to create a *Format* menu with the items *Character...* and *Paragraph...*. (Please refer to the Delphi documentation if you need help with creating menus).

Add the following code to the Click procedures of the menu items:

```
procedure TForm1.Character1Click(Sender: TObject);begin
   TXTextControl1.FontDialog
end;
```

```
procedure TForm1.Paragraph1Click(Sender: TObject);begin
   TXTextControl1.ParagraphDialog;
end;
```

Start the program again. You should be able to use the menu items to call the font and paragraph dialog boxes.

Again use the *Menu Design Window* and create an *Edit* menu containing items for *Cut*, *Copy*, and *Paste*. The code for these menu items is:

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
   TXTextControl1.Clip (1);
end;

procedure TForm1.Copy1Click(Sender: TObject);
begin
   TXTextControl1.Clip (2);
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
   TXTextControl1.Clip (3);
end;
```

After adding these menu items, you can exchange formatted text with other word processors via the clipboard.

The last menu for now shall be a simple file menu. Create a *File* menu including the items *Load...* and *Save As...*. Place a common dialog box icon on the form and enter the following code, which will call the common dialog box to get a file name from the user, and will then load respectively save the selected file:

```
procedure TForm1.Load1Click(Sender:
TObject);
const
   TXT_FIlE = 1;
   TXM_FILE = 3;
begin
   OpenDialog1.Title := 'Open file';
   OpenDialog1.Filename := '';
   OpenDialog1.Filter
      := 'Text Control Demo (*.txm)|*.txm
            |Plain text(*.txt)|*.txt';
   OpenDialog1.FilterIndex := 1;
   If OpenDialog1.Execute then begin;

   // Pass the filename to the text control
```

```
   If UpperCase(copy(OpenDialog1.Filename,
      length(OpenDialog1.filename)-2, 3)) = 'TXM' then begin
         TXTextControl1.Load(OpenDialog1.Filename,
                                  0, TXM_FILE, 0);
   end
   else
      TXTextControl1.Load(OpenDialog1.Filename,
                           0, TXT_FILE, 0);
   end;
end;

procedure TForm1.Saveas1Click(Sender: TObject);
const
   TXM_FILE = 3;
begin
   SaveDialog1.Title := 'Save as ...';
   SaveDialog1.Filename := '';
   SaveDialog1.Filter := 'Text Control Demo (*.txm)|*.txm';
   SaveDialog1.FilterIndex := 1;
   SaveDialog1.DefaultExt := 'txm';
   if SaveDialog1.Execute then begin;
      // Pass the filename to the text control
      TXTextControl1.Save(SaveDialog1.Filename,
                                  0, TXM_FILE, 0);
   end;
end;
```

Using the ActiveX Control in Visual C++

This chapter discusses how to use the TX Text Control ActiveX in Visual C++ 4.x, 5.x and 6.x. We're assuming that you already have working knowledge of Visual C++, or at least are familiar with the Visual C++ documentation and online help.

Creating Applications in Visual C++

Creating a Dialog, CFormView, or CView Based Application

1. Start Visual C++.
2. From the File menu, choose New. The New dialog box appears
3. **VC 4.x:** In the New box, select Project Workspace and click OK.
   VC 5.x/6.x: In the New box, select Projects Tab.
4. The New Project Workspace dialog appears.
5. Browse to the desired directory path.
6. In the Name text box, type a name for your project. This will create a sub-directory of that name in the current path.
7. From the Type list, select MFC AppWizard(exe) to create a project based on the MFC library.
8. **VC 4.x**: Click the Create button.
   VC 5.x/6.x: Click the OK button.

The MFC AppWizard - Step 1 Dialog appears.

If you wish to create a Dialog based application, click the Dialog radio button, click NEXT and procede to the section, Dialog Based Applications. If you wish to create a CFormView based application, click the "Single Document" or "Multiple Documents" radio button, click NEXT and procede to the section, CFormView Based Application. If you wish to create a CView based application, click the "Single Document" or "Multiple Documents" radio button, click NEXT and procede to the section, CView Based Applications.

Dialog Based Applications

1. In the Step 2 dialog, click on the OLE Controls (**VC 5.x/6.x**: ActiveX Controls) check box to add built-in support for OCX

products.

2. Click on NEXT button.
   The Step 3 dialog will appear.
3. In the Step 3 dialog, you can accept the default options by clicking the NEXT button.
4. In Step 4, you can accept the default options by clicking the FINISH button. VC++ will build your project.
   The New Project Information dialog will appear.
5. Click OK

CFormView Based Applications

1. In the Step 2 dialog you can accept the default options by clicking the NEXT button.
2. In the Step 3 dialog, click on the OLE Controls (**VC 5.x/6.x**: ActiveX Controls) check box to add built-in support for OCX products.
3. Click on Next button.
4. In the Step 4 and 5 dialogs you can accept the default options by clicking the NEXT button.
5. In the Step 6 dialog, select the class view name from the class list at the top of the dialog.
   CView will appear in the Base Class listbox.
6. In the Base Class listbox, change CView to CFormView.
7. Then click on the FINISH button to have VC++ build your project.

CView Based Applications

1. In the Step 2 dialog you can accept the default options by clicking the NEXT button.
2. In the Step 3 dialog, click on the OLE Controls (**VC 5.x**: ActiveX Controls) check box to add built-in support for OCX products.
3. Click on Next button.
4. In the Step 4 and 5 dialogs you can accept the default options by clicking the NEXT button.

5. In the Step 6 dialog, click on the FINISH button to have VC++ build your project.

Adding the Text Control Component to your Project

To insert a Text Control component into your project:

1. **VC 4.x**: From the Insert menu, choose Components.
   The Component Gallery dialog box appears.
   Select the OLE Controls tab.
   If the Text Control Text Control icon is not visible in the Gallery, click Customize to add the control.
   Select the control from the Component list on the right and click OK. This returns you to the Component Gallery.
   VC 5.x/6.x: From the Project / Add to Project menu choose Components and Controls.
   Open the Registered ActiveX Controls folder.
2. Select the Text Control icon in the Gallery and click Insert.
   The Confirm Classes dialog will display.
3. Click OK to confirm and exit the dialog.
4. Repeat steps 2 and 3 for the Status Bar, Ruler, and Button Bar controls.
5. Click Close to exit the Component Gallery.

The Text Control and its tools should now appear in the Control palette.

When VC++ adds components to your project, it creates CPP and H source files defining the class, properties, and methods for the control. It is a good idea to take a look at these files to understand what they contain. Methods and properties are not accessed the same in C++ as they are in many other languages like Visual Basic. When these files are generated, VC++ creates both a Get and Set function for most methods and properties. Text Control, for example, has a Text property. VC++ will create both a GetText and SetText member functions.

Adding the Component to your Dialog or CFormView:

1. In the Resource Editor, bring up the dialog that you want to place Text Control into.
2. Click on the Text Control component in the Editor's Control palette.

3. Draw the component on the dialog box.
4. Now this can be placed and sized as desired using the handles around the control.
5. Click on the right mouse button to bring up a floating menu. The design-time properties for the control can be viewed and modified through this menu.

Assigning Member Variables

Once you have added the text control to the dialog, it will be necessary to assign a member variable to each control to gain access to the methods and properties at runtime.

1. From the View menu, choose ClassWizard.
2. Select the Member Variables tab.
3. Select the control in the Control ID window for which you wish to add a variable and click the Add Variable button.
   The Add Member Variable dialog will display.
4. Type in the member variable name e.g. something like m_txctrl. Accept the default variable category and type, by clicking OK.
5. The MFC ClassWizard dialog will display the variable you added in the Control ID window.
6. Repeat steps 3 and 4 for each of the Text Control controls, specifying a new name for each.
7. Once you have added all the variables, click on OK in the MFC ClassWizard dialog to return to your project.

Adding the Text Control Component to your CView:

1. In the file list, bring up the header file for the view (<projname>view.h).
2. At the top of the file, include each of the Text Control control header files:

```
#include "tx4ole.h"
#include "txbbar.h"
#include "txruler.h"
```

```
#include "txsbar.h"
```

3. In the Attributes section, as a public member, add the following to create member variables for each of the controls in your view:

```
CTX4OLE  m_txctrl;
CTXBBAR  m_txbbar;
CTXRULER m_txruler;
CTXSBAR  m_txsbar;
```

4. Now through the file list, bring up the C++ source file for the view (<projname>view.cpp).
5. Start the ClassWizard. Make sure the view class is selected as the Class Name.
6. Select the View object in the Object Id listbox.
7. Select the "Create" message in the Messages listbox.
   The Create handler will initially come up with the following code:

```
return CWnd::Create(lpszClassName, lpszWindowName, dwStyle, rect,
pParentWnd, nID, pContext);
```

Change this to the following:

```
if (CWnd::Create(lpszClassName, lpszWindowName, dwStyle, rect,
                 pParentWnd, nID, pContext) == 0)
   return 0;

WCHAR szLic[] = L“AB-12345TS-1234567890“;
BSTR bstrKey = SysAllocString(szLic);
BOOL bSuccess = m_txctrl.Create(NULL, dwStyle, rect, this, 1000,
                                NULL, NULL, bstrKey);
SysFreeString(bstrKey);
if (!bSuccess)
   return 0;
if (m_txbbar.Create("TextControl ButtonBar", dwStyle, rect,
                 this, 1001) == 0)
   return 0;
if (m_txruler.Create("TextControl Ruler", dwStyle, rect,
                 this, 1002) == 0)
   return 0;
if (m_txsbar.Create("TextControl StatusBar", dwStyle, rect,
```

```
                    this, 1003) == 0)
   return 0;
return TRUE;
```

8. Start the ClassWizard. Select view class as the Class Name.
9. Select the View object in the Object Id listbox.
10. Select the "WM_SIZE" message in the Messages listbox.
11. Click on the Add Function button to create the OnSize handler function for this message.
12. Add the following code to the handler:

```
if (m_txctrl.m_hWnd && m_txbbar.m_hWnd && m_txruler.m_hWnd &&
m_txsbar.m_hWnd) {
   m_txctrl.MoveWindow(0, 60, cx, cy-(25+60));
   m_txbbar.MoveWindow(0, 0, cx, 30);
   m_txruler.MoveWindow(0, 30, cx, 30);
   m_txsbar.MoveWindow(0, cy-25, cx, 25);
}
```

Licensing the Control

The code added in the previous section, uses a license string to create a Text Control. Text Control is shipped as a CD version and as a trial version that can be downloaded and unlocked. The license string for the CD version users is the Text Control serial number. The license string for the trial version users is the customer key followed by the serial number when the trial version is unlocked. When you use the locked trial version to test Text Control's features, use only your customer key as license string. In the code example above the customer key is "AB-12345" and the serial number is "TS-1234567890".

Connecting the Text Control Controls

Connecting the Controls:

1. In the Create handler, add the following code:

```
m_txctrl.SetButtonBarHandle(m_txbbar.GetHWnd());
m_txctrl.SetRulerHandle(m_txruler.GetHWnd());
m_txctrl.SetStatusBarHandle(m_txsbar.GetHWnd());
```

Handling Events in your Dialog or CFormView:

Assigning Message Handlers:

1. Start ClassWizard
2. In the Class Name listbox, select the Dialog or CFormView class that was created.
3. In the Messages listbox, select the desired message to handle and click on Add Function button to add a handler for this. For our example, select the "Click" event and click on the Add Function button to add the handler for this.
4. Click on the Edit Code button to edit the new function.
5. Add the following code in the function:

```
MessageBox ("Click Event","You clicked on the document");
```

6. Run the program and when the document is clicked on, the message "You click on the document".

Setting Properties in Visual C++

You can easily set specific properties for each of the controls you include in your project.

To set properties for a control:

1. Double-click on the control in your project that you wish to set properties for. The Control Properties dialog will display.
2. Select the appropriate tab for the property settings you wish to modify.
   Properties are grouped together in categories, such as paragraphs, fonts, and pages.
3. Modify the property settings as needed. For more information on each property, see 'Text Control Properties, Events, and Methods.'
4. Once you have set the properties for the active control, close the Control Properties dialog to return to your project.
5. Repeat steps 1 through 4 for each control.

What Comes Next

Now that you've created your first TX Text Control project, you can see how quick and easy it is to add word processing functionality into your application. Naturally, TX has many more features than these simple examples demonstrate such as OLE Object support, image embedding, table support, headers and footers, macro fields, hypertext links, undo/redo, printing, and zooming. To find out more information about these features and how to use them, please use the following resources:

Online Manuals

The TX Text Control ActiveX comes with an extensive online manual, the *ActiveX Programmer's Guide*, that provides complete information on programming and using TX Text Control. The *ActiveX Programmer's Guide* describes the ActiveX interface, including a complete property, method and event reference. It also discusses the various sample programs.

Sample Programs

In addition to the reference manual, the Text Control ActiveX comes with a wealth of sample programs. These are all stored under the *Samples* directory, that is a subdirectory of your selected Text Control installation directory. Each sample project is located within its own subdirectory.

Readme Files

Before you get too involved with Text Control, you should take a quick look at the *Read me* and *Notes for ActiveX Users* documents. It contains the latest and greatest news concerning Text Control, including new information since the manuals were printed. You can start them from the Text Control program folder.

Technical Support

Free technical support for Text Control can be obtained at http://www.textcontrol.com/support or contact The Imaging Source Europe GmbH by fax at +49-421-33591-80.

Starting with the Class Library

Introduction

Welcome to TX Text Control, the comprehensive text integration tool with an extensive C++ 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.

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.

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, save and print files, use the clipboard and will have a full keyboard and mouse interface.The following step-by-step instructions cover the following topics:

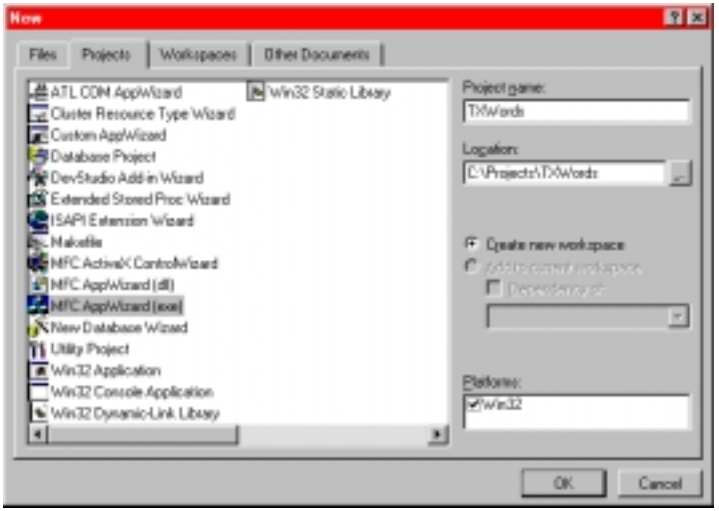
- Creating the starter application.
- Performing Visual C++ project settings.

- 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)



- Click on *OK*.

Proceed in the following dialogs as follows:

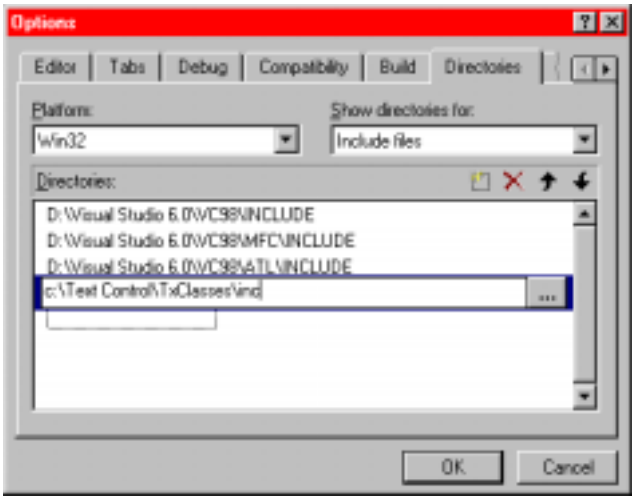
1. On page 1 don't change the default settings. Click on *Next*.
2. On page 2 don't change the default settings. Click on *Next*.

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

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

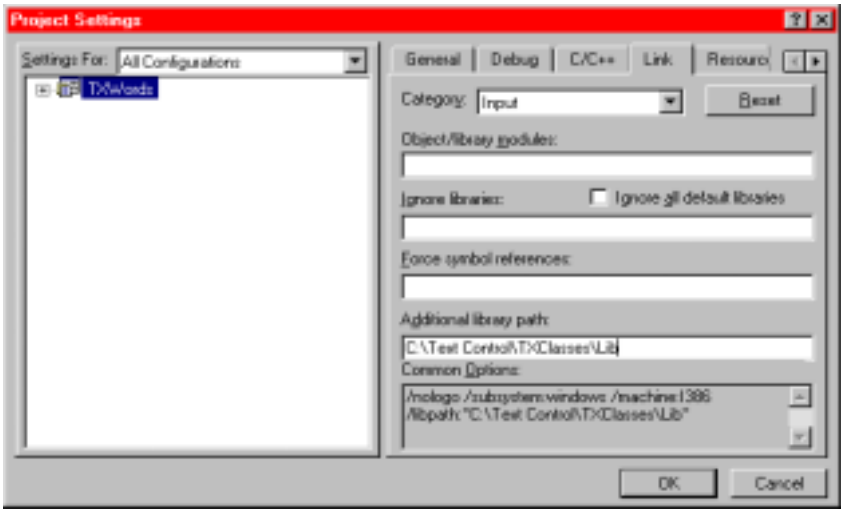
- From the *Project* menu select *Settings*.
- Select the *Link* tab.
- Under *Category* select *Input.*

- 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*.



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.

What Comes Next

Now that you've created your first TX Text Control project, you can see how quick and easy it is to add word processing functionality into your application. Naturally, TX has many more features than this short step-by-step guide demonstrate such as OLE Object support, image embedding, table support, headers and footers, macro fields, hypertext links, undo/redo, printing, and zooming. To find out more information about these features and how to use them, please use the following resources:

Online Manuals

The TX Text Control Class Library comes with an extensive online manual, the *Class Library Programmer's Guide*, that provides complete information on programming and using TX Text Control. The *Class Library Programmer's Guide* contains a tutorial that continues creating the word processor you have just started. Additionally it is a reference of all the classes' member functions and how these functions work

together. It also contains several articles describing how the Text Control Class Library realizes the more advanced Text Control features.

Readme Files

Before you get too involved with Text Control, you should take a quick look at the *Read me* file. It contains the latest and greatest news concerning Text Control, including new information since the manuals were printed. You can start it from the Text Control program folder.

Technical Support

Free technical support for Text Control can be obtained at http://www.textcontrol.com/support or contact The Imaging Source Europe GmbH by fax at +49-421-33591-80.