Sharing Information with the Clipboard

CHAPTER

17

IN THIS CHAPTER

Once upon a time, humankind struggled just to survive. People lived in dark caves, hunted for food with spears and rocks, and communicated with grunt-like sounds and hand motions. They worshipped fire because it gave them light under which they worked on their very slow computers. Computers back then could run only one application at a time due to hardware and software limitations. The only way to share information was to save it on disk and to pass the disk along for others to copy to their machines.

Nowadays, at least the equipment and software have improved. With operating systems such as Windows 95/98 and Windows NT/2000, multiple applications can be run simultaneously, which makes life much easier and more productive for the computer user. One of the advantages gained from Windows is that information can be shared between applications on the same machine. Two of the earlier technologies for sharing information are the Win32 Clipboard and Dynamic Data Exchange (DDE). You can make it possible for your users to copy information from one application to another with little effort using either of these.

This chapter shows you how to use Delphi's encapsulation of the Win32 Clipboard. Previous editions of this book covered DDE as well. However, with powerful interprocess communication technologies such as COM, we can't, in all good conscience, refer you back to a dead technology. Later, in Chapter 23, "COM and ActiveX," we'll discuss COM in greater depth. For simple implementations of sharing information between applications, the Clipboard is still a very solid solution.

In the Beginning, There Was the Clipboard

If you're an experienced Windows programmer, you might already be familiar with the Win32 Clipboard—at least in functionality. If you're new to Windows programming but have been using Windows, you've probably been using the Clipboard all along but never really understood how it's implemented.

Almost any application that has an Edit menu makes use of the Clipboard. So what exactly is the Clipboard? It's simply an area of memory and a set of Win32 API functions that enable applications to store and retrieve information to and from that area in memory. You can copy a portion of your source code from the Delphi editor, for example, and paste that same code into the Windows Notepad or any other editor.

Why does Win32 require a special set of functions and messages in order to use the Clipboard? Copying data to the Clipboard is more than just allocating an area of memory and placing data in that area. Other applications have to know how to retrieve that data and whether the data is in a format that the application supports. Win32 takes care of the memory management and enables you to copy, paste, and query about the information on the Clipboard.

Clipboard Formats

Win32 supports 25 predefined formats that applications can copy to or paste from the Clipboard. The most common formats are as follows:

`CF_BITMAP`	Specifies bitmap data.
`CF_DIB`	Specifies bitmap data along with the bitmap's palette information.
`CF_PALETTE`	Specifies a color palette.
`CF_TEXT`	Specifies a character array where each line ends with a carriage return/linefeed. This is the most commonly used format.

You can refer to the Win32 API online help under "`SetClipboardData`" if you're curious about less-common formats. Additionally, Win32 enables you to define your own private Clipboard formats, as illustrated later in this chapter.

Before Delphi, you had to call various Clipboard functions directly and were responsible for ensuring that your application didn't do anything ill-advised with the Clipboard's contents. With Delphi, you just use the global variable `Clipboard`. `Clipboard` is a Delphi class that encapsulates the Win32 Clipboard.

Using the Clipboard with Text

We already showed you how to use the Clipboard with text in Chapter 16, "MDI Applications." Specifically, this had to do with the text editor in the MDI application. We created menu items for cutting, copying, pasting, deleting, and selecting text.

In the MDI application, the editor, a `TMemo` component, covers the client area of the form. The `TMemo` component has its own functions that interact with the global `Clipboard` object. These functions are `CutToClipBoard()`, `CopyToClipBoard()`, and `PasteFromClipBoard()`. The methods `ClearSelection()` and `SelectAll()` aren't necessarily `Clipboard` interface routines, but they enable you to select the text you want to copy to the Clipboard. Listing 17.1 shows the event handlers for the Edit menu items.

LISTING 17.1 Clipboard Operations on Text

```
procedure TMdiEditForm.mmiCutClick(Sender: TObject);
begin
  inherited;
  memMainMemo.CutToClipBoard;
end;
```

continues

LISTING 17.1 Continued

```
procedure TMdiEditForm.mmiCopyClick(Sender: TObject);
begin
  inherited;
  memMainMemo.CopyToClipBoard;
end;

procedure TMdiEditForm.mmiPasteClick(Sender: TObject);
begin
  inherited;
  memMainMemo.PasteFromClipBoard;
end;
```

As illustrated in Listing 17.1, you need only call the `TMemo` methods to perform the `Clipboard` functions. You also can place text on the Clipboard manually by using the `Clipboard.AsText` property. Back in the 16-bit environment, the `AsText` property was limited to 255 characters and you had to use the `SetTextBuf()` and `GetTextBuf()` methods to copy larger strings to the Clipboard. This is no longer the case in 32-bit Delphi because the `AsText` property's string type now means long strings. You'll notice that `SetTextBuf()` and `GetTextBuf()` are still supported as well.

```
Clipboard.AsText := 'Delphi Rules';
```

NOTE

The `Clipboard` function's `GetTextBuf()` and `SetTextBuf()` methods use Pascal `PChar` types as buffers to pass and retrieve data from the Clipboard. When using such methods, you can typecast long strings as `PChar` types so that you don't have to do any converting of `String` types to `PChar` types.

Using the Clipboard with Images

The Clipboard can also copy and paste images. You saw how this can be done in the same MDI sample program. The event handlers that performed the Clipboard operations are shown in Listing 17.2.

LISTING 17.2 Clipboard Operations on a Bitmap

```
procedure TMdiBMPForm.mmiCopyClick(Sender: TObject);
begin
  inherited;
  ClipBoard.Assign(imgMain.Picture);
end;
```

```
procedure TMdiBMPForm.mmiPasteClick(Sender: TObject);
{ This method copies the contents from the clipboard into imgMain }
begin
  inherited;
  // Copy clipboard content to imgMain
  imgMain.Picture.Assign(ClipBoard);
  ClientWidth := imgMain.Picture.Width;
  { Adjust clientwidth to adjust the scollbars }
  VertScrollBar.Range := imgMain.Picture.Height;
  HorzScrollBar.Range := imgMain.Picture.Width;
end;
```


TIP

In order to access the `Clipboard` global variable, you must include `ClipBrd` in the `uses` clause of the unit that will be using `Clipboard`.

In Listing 17.2, the `mmiCopyClick()` event handler uses the `Clipboard.Assign()` method to copy the image to the Clipboard. Using this approach, you can paste the image into another Win32 application that supports the `CF_BITMAP` format, such as Windows Paint (`PBrush.EXE`).

`mmiPasteClick()` uses the `Image.Assign()` method to copy the image from the Clipboard and readjusts the scrollbars accordingly.

NOTE

`CF_PICTURE` is not a standard Win32 Clipboard format. Instead, it's a private format used by Delphi applications to determine whether the Clipboard data is in a `TPicture`-compatible format, such as bitmaps and metafiles. If you were to register your own graphic format, `TPicture` will support that format as well. Look up `TPicture` in Delphi's online help for further information on `TPicture`-compatible formats.

Creating Your Own Clipboard Format

Imagine working with an address entry program. Suppose that you're entering a record that differs only slightly from the record previously entered. It would be convenient if you could copy the contents from the previous record and paste them to the current record, instead of having to enter each field again. You might want to use the same information in other applications as well, perhaps as the address in a letter. The next example shows you how to create an object that knows about the Win32 Clipboard and can save its special formatted data to the

Clipboard. You also learn how to store your information as CF_TEXT format so that you can retrieve the same data in other applications that support the CF_TEXT format.

Creating a Clipboard-Aware Object

You might be thinking that one way to define custom Clipboard formats would be to create a descendant TClipboard class that knows about the newly defined format. This special TClipboard class could contain the specialized methods for dealing with the custom format. Although such a class would suffice in an isolated case, it would become tedious to maintain as you continue to need additional formats or as you need to redefine your data. If 70 different vendors came up with their own TClipboard descendant classes for their custom Clipboard formats, you'd have a major problem trying to use just two of the formats. The TClipboard descendants would conflict with each other.

A better approach would be to define an object around your data and then make the object aware of the TClipboard object, rather than the reverse. This singleton pattern to the Clipboard is the approach that Borland uses with its Delphi components. A TMemo component knows how to place its data on the Clipboard, just as a TImage component knows how to place its data on the Clipboard. All components use the same TClipboard object, so there's no conflict. This is the approach we'll show you in this section to define a custom Clipboard format, which is basically a record with a person's name, age, and birth date information. The unit for defining the data, along with the Clipboard methods to copy and paste the data to and from the Clipboard, is shown in Listing 17.3.

LISTING 17.3 A Unit That Defines Custom Clipboard Data

```
unit cbdata;
interface
uses
  SysUtils, Windows, clipbrd;

const

  DDGData = 'CF_DDG'; // constant for registering the clipboard format.
type

  // Record data to be stored to the clipboard
  TDataRec = packed record
    LName: string[10];
    FName: string[10];
    MI: string[2];
    Age: Integer;
    BirthDate: TDateTime;
  end;
```

```
  { Define an object around the TDataRec that contains the methods
    for copying and pasting the data to and from the clipboard }
  TData = class
  public
    Rec: TDataRec;
    procedure CopyToClipBoard;
    procedure GetFromClipBoard;
  end;

var
  CF_DDGDATA: word; // Receives the return value of RegisterClipboardFormat().

implementation

procedure TData.CopyToClipBoard;
{ This function copies the contents of the TDataRec field, Rec, to the
  clipboard as both binary data, as text. Both formats will be
  available from the clipboard }
const
  CRLF = #13#10;
var
  Data: THandle;
  DataPtr: Pointer;
  TempStr: String[50];
begin
  // Allocate SizeOf(TDataRec) bytes from the heap
  Data := GlobalAlloc(GMEM_MOVEABLE, SizeOf(TDataRec));
  try
    // Obtain a pointer to the first byte of the allocated memory
    DataPtr := GlobalLock(Data);
    try
      // Move the data in Rec to the memory block
      Move(Rec, DataPtr^, SizeOf(TDataRec));
      { Clipboard.Open must be called if multiple clipboard formats are
        being copied to the clipboard at once. Otherwise, if only one
        format is being copied the call isn't necessary }
      ClipBoard.Open;
      // First copy the data as its custom format
      ClipBoard.SetAsHandle(CF_DDGDATA, Data);
      // Now copy the data as text format
      with Rec do
        TempStr := FName+CRLF+LName+CRLF+MI+CRLF+IntToStr(Age)+CRLF+
                DateTimeToStr(BirthDate);
      ClipBoard.AsText := TempStr;
      { If a call to Clipboard.Open is made you must match it
        with a call to Clipboard.Close }
```

continues

LISTING 17.3 Continued

```
      Clipboard.Close
    finally
      // Unlock the globally allocated memory
      GlobalUnlock(Data);
    end;
  except
    { A call to GlobalFree is required only if an exception occurs.
      Otherwise, the clipboard takes over managing any allocated
      memory to it.}
    GlobalFree(Data);
    raise;
  end;
end;

procedure TData.GetFromClipBoard;
{ This method pastes memory saved in the clipboard if it is of the
  format CF_DDGDATA. This data is stored in the TDataRec field of
  this object. }
var
  Data: THandle;
  DataPtr: Pointer;
  Size: Integer;
begin
  // Obtain a handle to the clipboard
  Data := ClipBoard.GetAsHandle(CF_DDGDATA);
  if Data = 0 then Exit;
  // Obtain a pointer to the memory block referred to by Data
  DataPtr := GlobalLock(Data);
  try
    // Obtain the size of the data to retrieve
    if SizeOf(TDataRec) > GlobalSize(Data) then
      Size := GlobalSize(Data)
    else
      Size := SizeOf(TDataRec);
    // Copy the data to the TDataRec field
    Move(DataPtr^, Rec, Size)
  finally
    // Free the pointer to the memory block.
    GlobalUnlock(Data);
  end;
end;

initialization
  // Register the custom clipboard format
  CF_DDGDATA := RegisterClipBoardFormat(DDGData);
end.
```

This unit performs several tasks. First, it registers the new format with the Win32 Clipboard by calling the `RegisterClipboardFormat()` function. This function returns a value that identifies this new format. Any application that registers this same format, as specified by the string parameter, will obtain the same value when calling this function. The new format is also available on the `ClipBoard`'s list of formats, which can be accessed by the `Clipboard.Formats` property.

The unit also defines the record containing the data to be placed onto the Clipboard and the object that encapsulates this record. The record, `TDataRec`, has string fields to hold a person's name, an integer field to hold the person's age, and a `TDataTime` field to hold the person's birth date.

The object encapsulating `TDataRec`, `TData`, defines the methods `CopyToClipboard()` and `GetFromClipboard()`.

`TData.CopyToClipboard()` places the contents of the field `TData.Rec` onto the Clipboard as two formats: `CF_DDGDATA` and `CF_TEXT`. `CF_TEXT`, which, as you know, is an already-defined Clipboard format. The text version of `TData.Rec`'s contents are placed on the Clipboard by concatenating its fields as strings separated by carriage return/line feed characters. The non-string fields are converted to strings before formulating the final string that gets saved to the Clipboard. `ClipBoard.SetAsHandle()` first places a given handle onto the Clipboard in the format specified by its parameter. In this case, the parameter is the newly defined Clipboard format `CF_DDGDATA`.

Before calling `Clipboard.SetAsHandle()`, however, the method prepares a valid `THandle` that it must pass to `SetAsHandle()`. This handle represents the block of memory that contains the data being sent to the Clipboard. See the sidebar titled "Working with `THandle`s." The following line tells the Win32 system to allocate `Sizeof(TDataRec)` bytes of memory that may be moved, if necessary, and to return a handle to that memory to the variable `Data:`

```
Data := GlobalAlloc(GMEM_MOVEABLE, SizeOf(TDataRec));
```

A pointer to the memory is obtained with the following statement:

```
DataPtr := GlobalLock(Data);
```

The data is then moved to the memory block with the `Move()` function. In the remaining lines of code, the `ClipBoard.Open()` method opens the Clipboard to prevent other applications from using it while it's being given data:

```
ClipBoard.Open;
try
  ClipBoard.SetAsHandle(CF_DDGDATA, Data);
  with Rec do
    TempStr := FName+CRLF+LName+CRLF+MI+CRLF+IntToStr(Age)+CRLF+
      DateTimeToStr(BirthDate);
  ClipBoard.AsText := TempStr;
```

```
finally
  Clipboard.Close
End;
```

Typically, it's not necessary to call `Open()` unless you're sending multiple formats to the Clipboard, as you're doing here. This is because each assignment to the Clipboard using one of its methods (such as `ClipBoard.SetTextBuf()`) or properties (such as `ClipBoard.AsText`) causes the Clipboard to erase its previous contents because they, too, call `Open()` and `Close()` internally. By calling `ClipBoard.Open()` first, you prevent this from happening and therefore can assign multiple formats simultaneously. Had you not called the `Open()` method, only the `CF_TEXT` format would be available on the Clipboard after executing this method. The lines after the call to `Open()` simply assign the data to the Clipboard and then call the `ClipBoard.Close()` method accordingly.

At this point, the Win32 system is responsible for managing memory allocated for the Clipboard with the `GlobalAlloc()` function. A call to `GlobalFree()` would be necessary only if an exception occurred during the copy process. Don't call `GlobalFree()` otherwise because Win32 has taken over that memory management for the Clipboard.

With both `CF_DDGDATA` and `CF_TEXT` formats available on the Clipboard, you can paste the data back into either this sample program or other applications, as we'll illustrate momentarily.

`TData.GetFromClipboard()` does just the opposite—it retrieves data from the Clipboard in the `CF_DDGDATA` format and places that data in the `TData.Rec` field. The commentary in the listing explains how this method operates. The sample application that we'll show next illustrates how to use this unit. Notice that this Clipboard object can be easily modified to store any type of record you might define.

NOTE

Do not free the handle returned from `GetAsHandle()`; it doesn't belong to your application—it belongs to the Clipboard. Therefore, the data that the handle references should be copied.

Working with THandles

A `THandle` is nothing more than a 32-bit variable that represents an index of a table where the Win32 system maintains information about a memory block. There are many types of `THandle`s, and Delphi encapsulates most of them with `TIcons`, `TBitmaps`, `TCanvas`, and so on.

Certain Win32 functions, like the various Clipboard functions, use the heap to manipulate Clipboard data. To get access to heap memory, you make use of the memory allocation function shown in the following list:

`GlobalAlloc()`	Allocates a number of bytes specified from the heap and returns a `THandle` to that memory object
`GlobalFree()`	Frees the memory allocated with `GlobalAlloc()`
`GlobalLock()`	Returns a pointer to a global memory object received from `GlobalAlloc()`
`GlobalUnlock()`	Unlocks memory previously locked with `GlobalLock()`

Using the Custom Clipboard Format

The main form for the project that illustrates the use of the custom Clipboard format is shown in Figure 17.1.

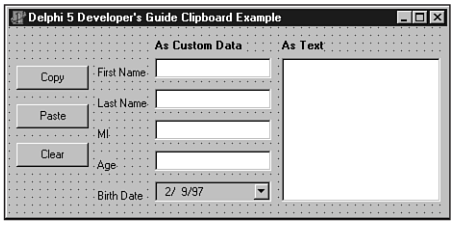

FIGURE 17.1
The main form for the custom Clipboard format example.

As shown, this form contains the controls required to fill the `TDataRec` field of the `TData` object. Listing 17.4 shows the source code for this form. The project resides on the CD as `Ddgcbp.dpr`.

LISTING 17.4 Source Code for the Custom Clipboard Format Example

```
unit MainFrm;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, clipbrd, Mask, ComCtrls;
type
```

continues

LISTING 17.4 Continued

```
  TMainForm = class(TForm)
    edtFirstName: TEdit;
    edtLastName: TEdit;
    edtMI: TEdit;
    btnCopy: TButton;
    btnPaste: TButton;
    meAge: TMaskEdit;
    btnClear: TButton;
    lblFirstName: TLabel;
    lblLastName: TLabel;
    lblMI: TLabel;
    lblAge: TLabel;
    lblBirthDate: TLabel;
    memAsText: TMemo;
    lblCustom: TLabel;
    lblText: TLabel;
    dtpBirthDate: TDateTimePicker;
    procedure btnCopyClick(Sender: TObject);
    procedure btnPasteClick(Sender: TObject);
    procedure btnClearClick(Sender: TObject);
  end;

var
  MainForm: TMainForm;

implementation
uses cbdata;

{$R *.DFM}

procedure TMainForm.btnCopyClick(Sender: TObject);
// This method copies the data in the form's controls onto the clipboard
var
  DataObj: TData;
begin
  DataObj := TData.Create;
  try
    with DataObj.Rec do
    begin
      FName     := edtFirstName.Text;
      LName     := edtLastName.Text;
      MI        := edtMI.Text;
      Age       := StrToInt(meAge.Text);
      BirthDate := dtpBirthDate.Date;
      DataObj.CopyToClipBoard;
    end;
```

```
  finally
    DataObj.Free;
  end;
end;

procedure TMainForm.btnPasteClick(Sender: TObject);
{ This method pastes CF_DDGDATA formatted data from the clipboard to
  the form's controls. The text version of this data is copied to the
  form's TMemo component. }
var
  DataObj: TData;
begin
  btnClearClick(nil);
  DataObj := TData.Create;
  try
    // Check if the CF_DDGDATA format is available
    if ClipBoard.HasFormat(CF_DDGDATA) then
      // Copy the CF_DDGDATA formatted data to the form's controls
      with DataObj.Rec do
      begin
        DataObj.GetFromClipBoard;
        edtFirstName.Text := FName;
        edtLastName.Text  := LName;
        edtMI.Text        := MI;
        meAge.Text        := IntToStr(Age);
        dtpBirthDate.Date := BirthDate;
      end;
  finally
    DataObj.Free;
  end;
  // Now copy the text version of the data to form's TMemo component.
  if ClipBoard.HasFormat(CF_TEXT) then
    memAsText.PasteFromClipBoard;
end;

procedure TMainForm.btnClearClick(Sender: TObject);
var
  i: integer;
begin
  // Clear the contents of all controls on the form
  for i := 0 to ComponentCount - 1 do
    if Components[i] is TCustomEdit then
      TCustomEdit(Components[i]).Text := '';
end;

end.
```

When the user clicks the Copy button, it copies the data contained in the `TEdit`, `TDateTimePicker`, and `TMaskEdit` controls to the `TDataRec` field of a `TData` object. It then invokes the `TData.CopyToClipboard()` method, which places the data onto the Clipboard.

When the Paste button is clicked, the opposite happens. First, if the data in the Clipboard is of the type `CF_DDGDATA`, it's copied from the Clipboard and placed into the edit controls on the form. The text representation of the data is also copied and placed into the main form's `TMemo` component. The result of a paste operation is shown in Figure 17.2. You can also paste the text representation of the data into another Windows application, such as Notepad.

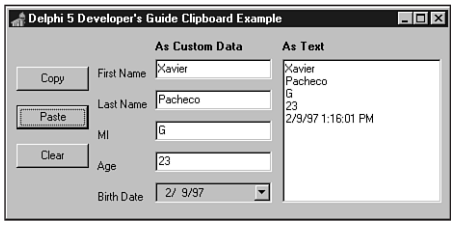

FIGURE 17.2
Pasted data on the main form.

The Clear button empties the contents of all controls on the main form.

Summary

Sharing data with other applications is an extremely useful technique. By enabling your applications to share data with other applications, you make it more usable and your users more productive. This chapter shows you how to use the Clipboard's built-in functions to work with Delphi controls. It also demonstrates how to create your own custom Clipboard formats. Another even more powerful method of interprocess communication is COM, which we'll cover in depth in later chapters.