Review:
TOLEAutomationClient Component

by Micha Somers

When you need to use the features of an application, such as MS Word or Excel, in your own application, OLE automation can be very useful: it allows your program to control other applications and facilitates the exchange of complex objects between your program and other applications.

Unfortunately, the 16-bit version of Delphi does not directly support OLE automation. Delphi 2.0 on the other hand does support OLE automation, but only for 32-bit applications. But for those who do not immediately switch to Windows 95 there is still a way to use OLE automation. The `TOLEAutomationClient` component will give you the necessary support to integrate these OLE objects into your own applications, giving them lots of extra power. It's a commercial component, written and sold by a developer in New Zealand.

First of all, I will describe what OLE automation is and how it may be used, and after that I will show a simple example in which the `TOLEAutomationClient` component is used.

What Is OLE Automation?

OLE automation can be used for data exchange between applications as well as for controlling other applications. With OLE automation it is possible to exchange complex information such as objects. It is even possible to set and retrieve the properties of such an object and call its methods, just as you would do with properties and methods of your own objects made in Delphi.

In order to use OLE automation there has to be an OLE automation server. The OLE automation server application exposes some objects to OLE automation client applications. Objects of an OLE automation server application are not automatically available to the OLE automation client application. Since Delphi is a compiled language, before you can call a method or refer to a property for an object, you need to let Delphi know that the object belongs to a class that contains that method or property. This is where the `TOLEAutomationClient` component gives support. When you use the this component, an object called `TOleObject` will become available. You will need this object to make so-called *mirror classes*.

Actually, you will have to build your own copy of the structure of the OLE objects you are going to use, including the relations between them. The way to do that in Delphi is to create a new unit and to declare a new class (a mirror class) with the required methods and properties. This new class is derived from the class `TOleObject`.

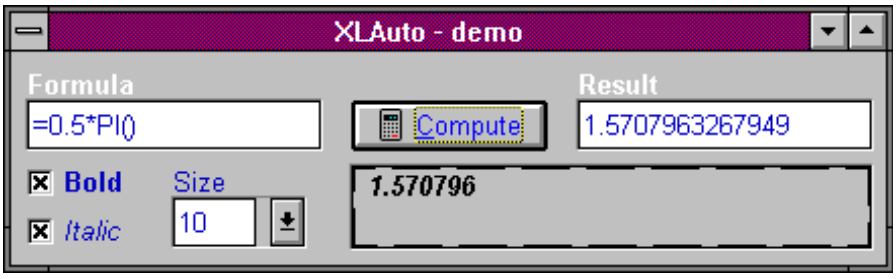
Which objects are exposed depends on the OLE automation server application. Word, for example, offers just one object called `Word.Basic` whereas Excel offers several objects. A piece of Delphi code using a Word object looks like the example in Listing 1. The code in Listing 2 (over the page) shows the definition of the mirror class.

Although many operations can be performed on this Word object, Word is not the most interesting example because it exposes just one object. Excel 5.0, on the other hand, offers a whole hierarchy of objects. An example of a command for changing the font size of a cell in Excel is given by:

```
ExcelObj.ActiveWorkSheet.Cells(
  1, 1).Font.Size := 11;
```

In this example you can see that the object `ExcelObj` consists of an object called `ActiveWorkSheet`. This `ActiveWorkSheet` has a function method called `Cells` giving a `Range` object as result. This object consists of a `Font` object. Finally, this `Font` object has a property called `Size`.

In order to make it possible to access the objects within an object, you will have to declare



➤ *Figure 1: Excel OLE automation client application*

```
var
  WordOb: TWordObject;
begin
  WordOb := TWordObj.CreateObject('word.basic');
  WordOb.FileNew('normal.dot');
  WordOb.FileClose;
  WordOb.Release;
end;
```

➤ *Listing 1*

so-called nested mirror classes. The declaration of properties, methods and nested mirror classes might look like the example in Listing 3.

OLE Automation Example

To illustrate the power of this TOLEAutomationClient component I have constructed a simple application which opens an existing Excel worksheet, gives Excel the command to compute the result of a formula and place the result in a cell. The worksheet will be saved when the computation has been finished. The application also gives the user the possibility to change the font style of the cell.

In order to see the changes to the data, a TOleContainer component is placed on the form and is linked to the corresponding data (an existing Excel file). Figure 1 shows the result of computing 0.5*PI().

Notice that the result in the OleContainer is less accurate than the result in the edit box. This is because the TOLEAutomationClient gives the exact internal result whereas the OleContainer presents the result the way it is shown by the OLE server.

The complete source code with the definition of the mirror classes for Excel is included on the disk with this issue, plus an executable version of the example program. It shows you how powerful this component is when the mirror classes have been defined. Of course, the *source code* can only be used if you have the TOLEAutomationClient component. The executable works if you have Excel installed on your computer. By double clicking on the OLE container at run-time you can link the OLE container to the file XLAUTOVB.XLS – see the README.TXT file for more information. This way you will see the changes to the data immediately in your OLE container.

Conclusions

Although it requires more effort to use OLE automation with this component than with the OLE automation facilities that Delphi 2.0 offers (in Delphi 2.0 you do not have to declare mirror classes), it still remains an easy way to make your applications more powerful. Besides, Delphi 2.0 only supports OLE automation for 32-bit applications, but TOLEAutomationClient can be used for 16-bit applications. So, for those of us who will continue developing 16-bit applications, this TOLEAutomationClient component might be a very useful tool.

The documentation explains exactly and in a simple and understandable way how to declare and implement the mirror classes. Further, the documentation contains some examples (Word, Excel) which will help you a lot.

One disadvantage of using OLE automation in your programs is that it might require a lot of memory, depending on the OLE server application it uses. You have to take care that all (nested) objects are released correctly after you have used them.

Another disadvantage is that you cannot check at compile time if your mirror class corresponds with the OLE object. Only at run time you can find out if your mirror class definition corresponds or not. Actually, this problem is not specific for this component because Delphi 2.0 has the same problem.

```
TWordObject = class(TOleObject)
public
  procedure FileNew(Template: String; NewTemplate: Integer);
  procedure FileClose(CloseMode: Integer);
end;
procedure TWordObject.FileNew(Template: String; NewTemplate: Integer);
const
  pszName: PChar =
    '12345678901234567890123456789012345678901234567890123456789012345678901234567890';
begin {create a file based on a specific template}
  SetOleMethodArg('Integer', NewTemplate);
  StrPCopy(pszName, Template);
  SetOleMethodArg('PChar', pszName);
  CallOleProc('FileNew');
end;
procedure TWordObject.FileClose(CloseMode: Integer);
begin {close the current file, with or without saving}
  SetOleMethodArg('Integer', CloseMode);
  CallOleProc('FileClose');
end;
```

➤ *Listing 2*

```
type
  TExcelFont=class(TOleObject)
   private
    function GetSize: Integer;
    procedure SetSize(iSize: Integer);
   public
    property Size: Integer read GetSize write SetSize;
  end;
  TExcelRange=class(TOleObject)
   private
    FFont: TExcelFont;
    function GetFont: TExcelFont;
   public
    property Font: TExcelFont read GetFont write FFont;
  end;
  TExcelWorkSheet=class(TOleObject)
   public
    function Cells(rowIndex, columnIndex: Integer): TExcelRange;
  end;
  TExcelObj=class(TOleObject)
   private
    FActiveWorkSheet: TExcelWorkSheet;
    function GetActiveWorkSheet: TExcelWorkSheet;
   public
    property ActiveWorkSheet: TExcelWorkSheet read GetActiveWorkSheet
      write FActiveWorkSheet;
  end;
```

➤ *Listing 3*

OLE automation is a powerful means of exchanging complex objects between different applications. The `TOLEAutomationClient` component has been developed to ease the use of OLE automation within your 16-bit Delphi programs by doing all the hard work for you. Instead of programming low-level instructions, you can spend your precious time on integrating powerful applications with your own programs!

The component is sold direct via email (raike@iconz.co.nz or CompuServe 100236,1656) and for US$39.95 it can be yours. A number of software tool retailers also stock the product, so it should be easy enough to obtain. There are no runtime fees or royalties. Because it is a native Delphi component, there is no need to distribute additional files.

Micha Somers is a Knowledge Engineer from the Netherlands developing Delphi, Pascal and C++ applications. You can contact him by email at micha@bolesian.nl

Contact details for the developer of TOLEAutomationClient:

Email: sraike@iconz.co.nz
CompuServe: 100236,1656
Fax: +64-9-832-0088
Mail: Software Developer (Raike), 66 Simpson Road, Swanson, Auckland 8, New Zealand