

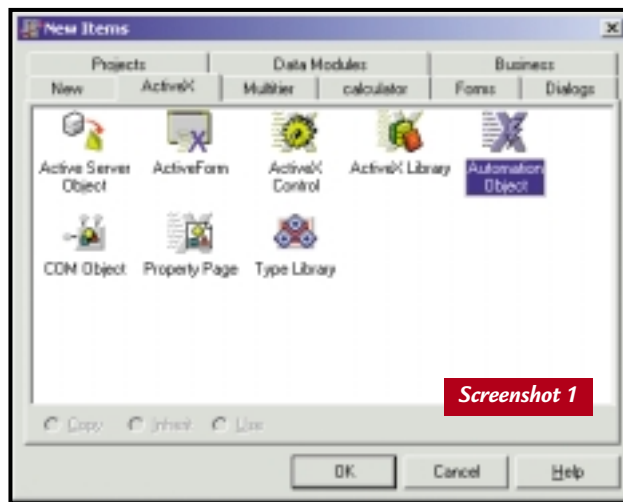


# Set it on auto

Tim Anderson takes you through the vital steps to build your own **automation server** in Delphi.

**T**he key debate in software development today is about the future of Windows in the web era. Trying to judge what proportion of software development will be for Windows, Java, Linux or whatever, is a guessing game that anyone can play. What is obvious, though, is that most business applications will need to support a variety of clients, probably via a web browser.

For those working primarily on Windows today, there is a way to future-proof an application, which is to create a COM automation interface to its features. The application remains a Windows application, but once its functionality is available through COM, it becomes feasible to support a diverse range of clients. Through Active Server Pages (ASPs) you can create browser-independent web applications. You can also create macros in Visual Basic for applications that access your application. This is the hallmark of a next-generation application, one that makes few assumptions about how it will be used. A useful advanced book on Delphi COM development is *Delphi COM Programming*, by Eric Harmon (Macmillan Technical Publishing, ISBN 1-57870-221-6). This US publication is priced at £27 and is available from [www.amazon.co.uk](http://www.amazon.co.uk).



**Choose Automation Object to add automation server capabilities to an existing Delphi 5.0 project**

What follows here is an explanation of how to create an automation server using Delphi. Some of this is easier in Visual Basic, but VB hides so much of the workings of COM that you don't get such a good feel for how it works.

The starting point for an automation in Delphi (4.0 or higher) is to start a new application and save it. This application is going to perform calculations, so call it Calculator. Then choose File, New, ActiveX, Automation object. This presents a dialog box as in screenshot 1. The Delphi 4.0 version is a little more

sparse. In both versions 4.0 and 5.0 you have to choose a class name and an instancing model, while Delphi 5.0 also offers you a threading model.

The Class Name (or CoClass Name in Delphi 5.0) is the name applications will use to create objects of this

class. In this case you could call it Calc. The complete class name will be Calculator.Calc, with the first part taken from the project name.

The instancing option plays a significant role. Internal is for objects that are private to the application, while Single Instance means that each time an object of this type is created, a new instance of the server is started.

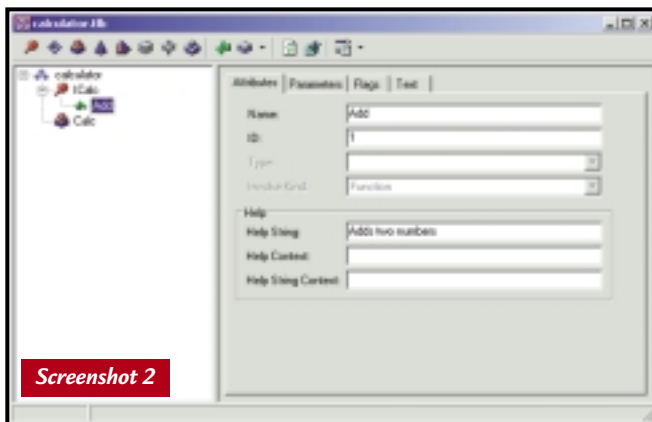
This is easy for the developer, but is also inefficient. Normally you will want to choose Multiple Instance, where a single server instance supports the creation of multiple instances of the object.

Delphi 5.0 includes another couple of options. Choosing Threading Model Defaults to Apartment, will mean objects only communicate with the thread that created them. In this case, you will still need to synchronise access to global data. Single threading avoids this but is restrictive, as it limits access to the server to a single thread. Free threading means the server must be completely thread-safe, and the Both option combines Apartment and Free. Apartment is a good choice for less experienced COM developers.

The Event support code option in Delphi 5.0 is not covered here, as this object will not use events.

## Editing the type library

Next, the type library editor opens (screenshot 2). This is where the interface to the COM server is defined. To keep things simple, this object has just one method, Add, which sums two numbers. Right-click ICalc in the tree view on the left, and choose Add, Method. Name this method Add. Select the method, and in the Help string field on the Attributes tab describe it as 'Adds two numbers'.



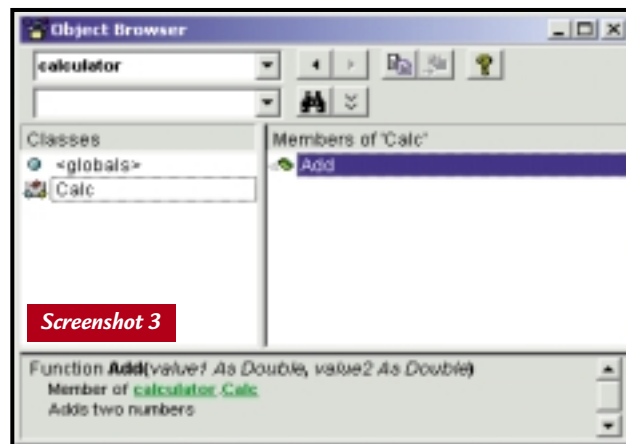
**Delphi's type library editor is where you define the interface for the new Automation object. You can also define constants and types that become available to clients**

Then choose the Parameters tab. Here Delphi 4.0 and 5.0 differ. In both cases you need to add two parameters of type Double. In Delphi 4.0, you can also give the method a return value of type Double. In Delphi 5.0, the return value must be left as HRESULT. To provide the return value, add a third parameter, called Result (for example), of type VARIANT\*. Click the Modifier button and set the parameter flags to Out andRetVal. This must be the last parameter in this list. Elsewhere in the code, you will find that the Delphi 5.0 method looks like a function returning a Double, despite appearances in the type library.

You have now defined the interface for the COM object. The next step is to implement its functionality. Delphi automatically creates a unit for this purpose. It has a Delphi class, TCalc, which is declared like this:

```
TCalc = class(TAutoObject, ✓  
ICalc)
```

(Key: ✓ code string continues)



***The new server appears in Visual Basic's object browser, complete with a help text describing the member function***

This means that the object inherits from TAutoObject, which supports necessary COM methods like QueryInterface, and implements ICalc, which is the interface you have defined in the type library.

In the body of the TCalc.Add function

Browser, and select the calculator library (screenshot 3). You'll note that the Calc object and the Add method are listed, along with with the help text explaining what it does. Complete the application by adding a button and a

type this code:

```
Result := ✓  
value1 + ✓  
value2;
```

Now run the application. The first time it runs, Delphi registers the automation object.



Next, try out the server, for example in Visual Basic or VBA. Open up Project References, find calculator library in the list, and check it. Now choose View, Object



# hands on

## visual programming

label to a form, with the following code:

```
Dim mycalc As New   
calculator.Cacl  
Label1.Caption =   
mycalc.Add(43.786,76.213)
```




Note that it is hard to get the parameters wrong, as VB checks the types. When you run the application, there is a flash as the Delphi application runs and exits, and the result appears.

The flash is not very appealing, so here are three things you could do about it. First, you could have created an in-process server, also known as a COM DLL, without any user interface. Second, you can run the server before running VB, in which case the object is created from the running instance and performance is better. Note that you can also detect whether the application was started standalone or via automation. Add `comserv` to the `uses` clause of a unit, and then inspect the `startmode` property of the global `comserver` object. This can be `smAutomation`, `smStandalone`, `smRegServer` or `smUnRegServer`, and lets you determine how the application presents itself in these different contexts.

If you do not have VB or VBA, you can also test the application in Delphi. Start a new application, and add to the `uses` clause of the main form the file

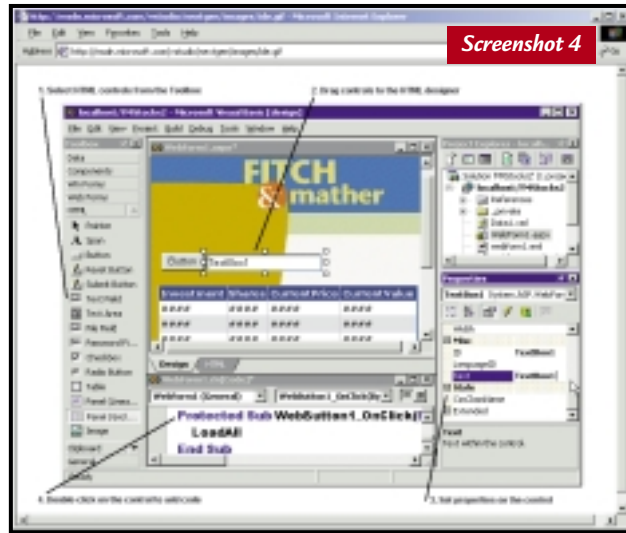
## It is as if Microsoft has just noticed what VB developers have been requesting for years

`calculator_tlb`, which was generated by the calculator application. Now you can write code like this:

```
procedure   
TForm1.Button1Click(Sender:   
TObject);  
var  
mycalc: ICalc;  
begin  
mycalc := CoCalc.create;  
Label1.Caption :=   
mycalc.Add(45.2,34.5);  
end;  
end;
```

This code is essentially the same as that used in Visual Basic. In Delphi 5.0 you can also use `Project`, `Import Type Library`, which will create a component that makes it even easier to use automation servers.

This simple example disguises the complexity of COM, and it is only fair to



**Visual Basic 7.0 web forms let you develop ASP applications almost as if they were ordinary Windows apps**

observe that, in its detail, COM development is challenging to master. On top of that, you need to face issues such as synchronising multiple threads of execution, which traditional VB or Delphi development generally avoids. It is worth the effort, since exposing an application's features for automation genuinely does take it to a new level, and for commercial developers gives an immediate advantage

over rivals that do not offer it. It is also a vital step towards web-enabling Windows applications.

### Visual Basic grows up

The next version of Visual Basic is likely to address the main weaknesses of the current version, according to Microsoft's recent announcements. It is as if Microsoft has just noticed what developers have been requesting for years, even back in VB 3.0 days. Microsoft's website announces these new features as 'language innovations', which is stretching a point since it is only catching up with other languages. Here is a quick summary:

● **Inheritance** Currently Visual Basic supports interface inheritance, which means you can declare that a new class implements an existing interface.

However, the work of implementing that interface or delegating its methods to an object of an existing class is down to the developer. A new `inherits` keyword now supports implementation inheritance, which is what most people mean by inheritance anyway. With this keyword, a class can acquire all the functionality of the class from which it inherits. The

derived class can override existing methods or add new ones of its own.

● **Visual Inheritance** In Visual Basic 7.0, you will be able to inherit visually, for example to have a form inherit from an existing form. The link will be retained, so that changes to the parent form will appear in the child form as well.

● **Overloading** Overloaded methods are methods with the same name as an existing one, but which take different parameters. This lets you write more intuitive code.

● **Constructors** Since Visual Basic 4.0, constructors have been severely hobbled by the inability to receive parameters. This means there is no safe way to instantiate an object, except with some convoluted code. For example, VB developers sometimes use a factory function that does take parameters. The factory function creates a new object, sets key properties according to the parameters passed, and returns the object instance. In the next version of VB this will not be needed, as constructors themselves will take parameters. It may seem small, but this is a huge step forward for those wanting to write robust and professional code.

● **Web forms** These were covered in a previous column, but it is important to note that VB 7.0 is intended to make ASP development radically easier (see screenshot 4). A web form designer will

be built in, and behave in a similar way to a traditional VB form, for the developer. Technically, the key improvement is that the web form will be pure HTML without any script. A special tag will link the HTML page to the VB class that drives it. To work, this requires changes in ASP itself.

● **Free Threading** A Thread object that works with function pointers will also be added, to let you write genuinely multi-threaded applications. Function pointers let you pass the address of a function to the thread's constructor, so it can run that function in the background while the application continues to execute.

● **Exception handling** Visual Basic is at last getting structured exception handling, based on Try, Catch, Finally blocks. Try starts the block; the Catch block executes if there are errors; the Finally section always runs, so you can have clean-up code that cannot be

bypassed. Visual C++ and Delphi programmers already use similar constructs, so again this is VB catching up, not being especially clever, but it is welcome even so.

● **Option Strict** This is another big one. It prevents many of the implicit type coercions that are meant to make VB easier to use, but in fact make it more error-prone.

All these improvements are fantastic, and may help to stem the flow of VB developers to other languages, which typically happens once they discover that VB's most frustrating features have a high price in terms of productivity. There are a couple of additional points to note. One is that some of these features are intimately linked to changes in COM itself, which is no surprise since VB is already linked to COM. For example, it appears that COM may be getting inheritance. It is also important to await

the small print. Not everything may be as it first appears, and it is problems of detail that so often spoil great new features. It is also worth noting that VB's selling point will always be ease of use, and it is virtually inconceivable that the elegance of Java or the flexibility of C++ could be achieved without changing the language beyond recognition.

The other disappointment concerns timing. Microsoft is vague on the subject, but is hinting that a beta release in the second half of 2000 may lead to a final version in the first half of 2001. That is a long time to wait since the release of VB 6.0, by which time the catch-up features in VB may look less exciting.

## CONTACTS

Tim Anderson welcomes your comments on the Visual Programming column. Contact him via the PCW editorial office or email: [visual@pcw.co.uk](mailto:visual@pcw.co.uk)