

# Developing Mixed Visual Basic/COBOL Applications\*

Wayne Rippin

If you are developing applications for Microsoft Windows, sooner or later you'll encounter one of the varieties of Microsoft Visual Basic (VB). Besides this development tool, there is also Visual Basic for Applications (VBA), the scripting language used by a wide range of desktop applications sold by Microsoft and other vendors. Even if you are primarily a COBOL programmer, because VB has such widespread support, it is often beneficial to combine the strengths of the two languages and develop a mixed VB/COBOL application.

Building this kind of application usually involves using VB to develop the application front end and COBOL to develop the business-processing back end. One way of calling COBOL from VB is to develop the COBOL code as a standard COBOL subprogram and then compile and link it to form a dynamic link library (DLL). Both VB and VBA provide mechanisms to call a DLL, and virtually all COBOL compilers available for the Windows platform support DLL creation. If you know that the VB and COBOL code will always run on the same machine and that you are not likely to want to migrate the application to the Web or other environments, then creating a standard DLL is a useful approach.

## Another Way: Object-Oriented COBOL

However, calling a DLL has limitations. If you want to:

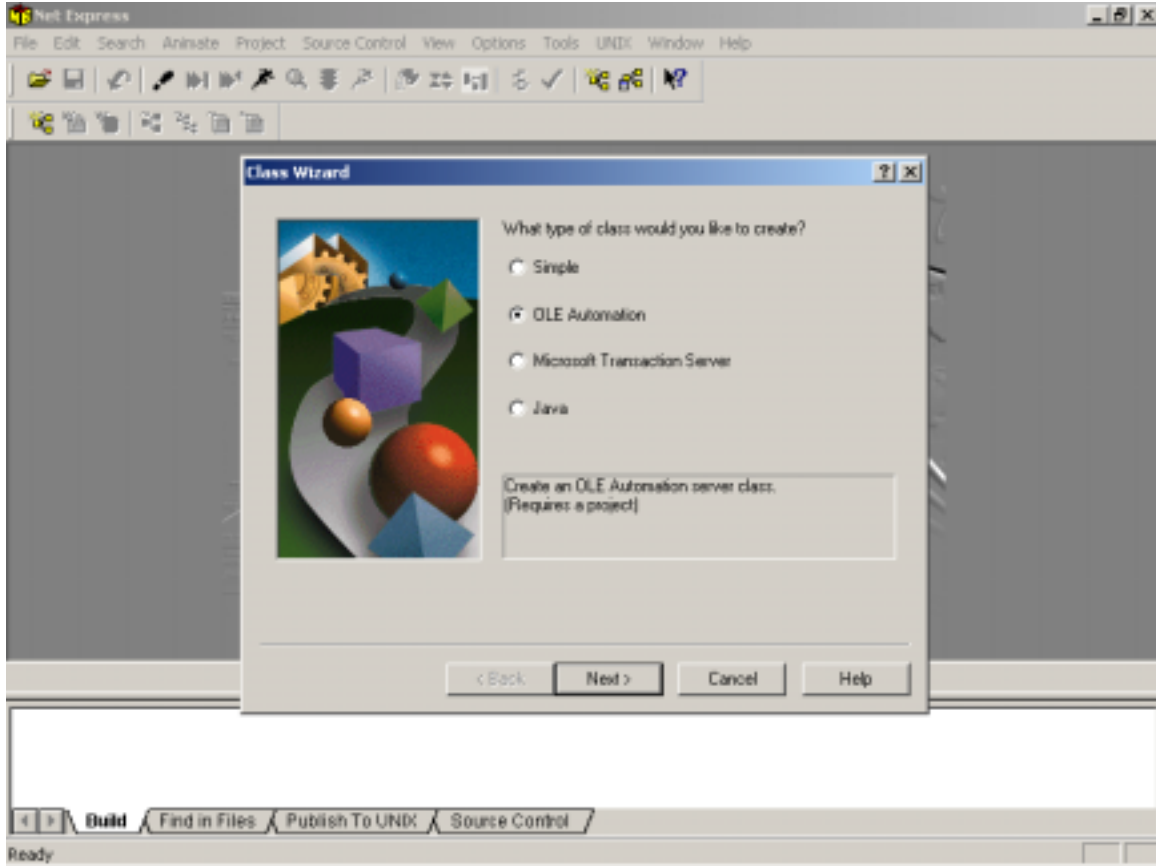
- use new functionality such as Microsoft Transaction Server,
  - eventually use a Web environment, or
  - create distributed applications where VB code runs on one machine and COBOL code runs on another,
- you should turn your COBOL code into an Automation object. Automation (also known as ActiveX objects, formerly called OLE Automation) from Microsoft is based on the Component Object Model (COM). Using Automation enables applications to expose their functionality to scripting languages and other applications. COBOL compiler vendors take different approaches to supporting COM and Automation. This article focuses on the direction taken by Micro Focus as supported in Net Express.

Micro Focus introduced OO COBOL support four years ago. Automation describes everything in terms of objects; the OO extensions to COBOL therefore are a natural fit. Even though Micro Focus COBOL uses the OO extensions, you do not have to learn OO programming techniques to create Automation objects in Cobol. Net Express provides wizards that generate skeleton code for the object—you just insert the code that will perform the functionality required.

## An Example

Consider a company that has an existing payroll package written in COBOL that calculates employees' net pay after deducting the cost of benefits, taxes, and so on. The existing package uses character-based screens, and the company wants to add a graphical front-end to make it easier for the human-resources staff to use it. The company decides to use VB to develop the package's front-end, leaving the existing COBOL logic to handle all the calculations. The company also wants to let employees use its intranet to select health plans and other benefits and then to immediately see the net impact on their paychecks. This application must use the same back-end logic that the first application uses. The company's intranet site uses Microsoft Active Server Pages, so the logical mechanism is to create an Automation object containing the key payroll routines. Then both the VB application and the intranet application can use the object.

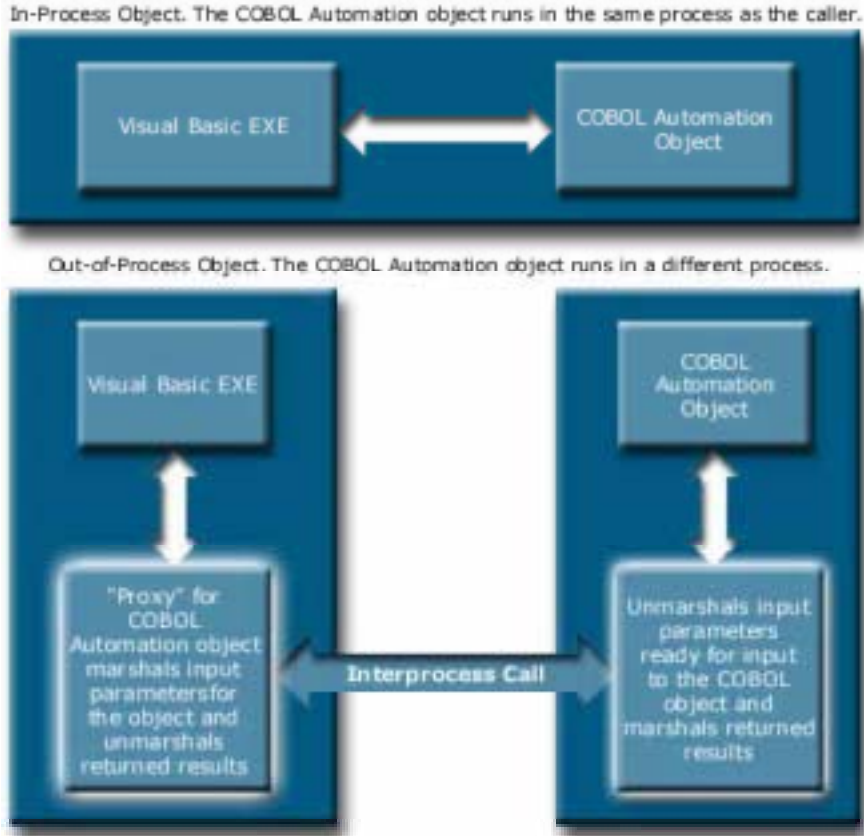
Using Micro Focus Net Express first, you create the COBOL class for the object using the Class Wizard. Figure 1 shows the wizard's initial dialog box. Select the option labeled OLE Automation.



**Figure 1. The Micro Focus Net Express Class Wizard.**

This Wizard lets you create other types of objects. The next few screens ask you the name of the project you want to use to build the class, the name of the class, whether you want to build it as an EXE file or DLL (in this case, choose DLL), and the type of threading option you want to use (choose Apartment).

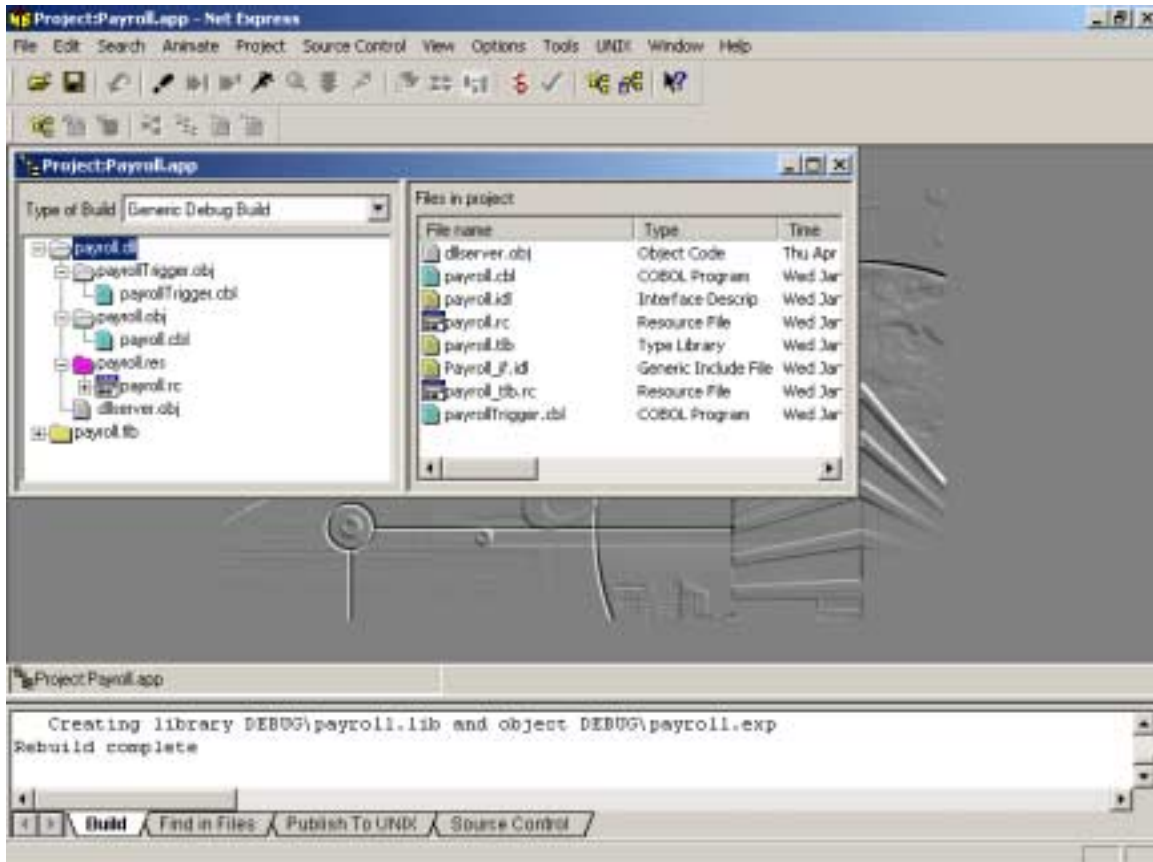
Net Express lets you build either an **in-process** object (a DLL) or an **out-of-process** object (an EXE). An in-process object runs in the same process as the module that calls it. Calls to the object execute much faster because they do not need to cross process boundaries, but if the object crashes for any reason, it is likely to take down the entire application. An out-of-process object, on the other hand, runs in its own process space. This involves a much higher call overhead. Usually, out-of-process objects are complete applications such as Microsoft Word or Microsoft Excel. Figure 2 shows this in more detail. In our example, we are more interested in the higher call performance, so we choose to build an in-process object. Note that the code to call the object is the same regardless of whether it is in-process or out-of-process—the underlying COM layer handles the cross-process communication issues.



**Figure 2. In-process and out-of-process objects.**

The subject of threading models is beyond the scope of this paper. For more information, refer to the Microsoft MSDN Web site at <http://msdn.microsoft.com> and search on “threading, Apartment model.” However, in most cases, you will not need to worry about this since the Class Wizard will insert all of the appropriate code for you.

Once you’ve selected all the appropriate options, Net Express creates the code for the class as well as the project files needed to create the object. Figure 3 shows the resulting Net Express project.



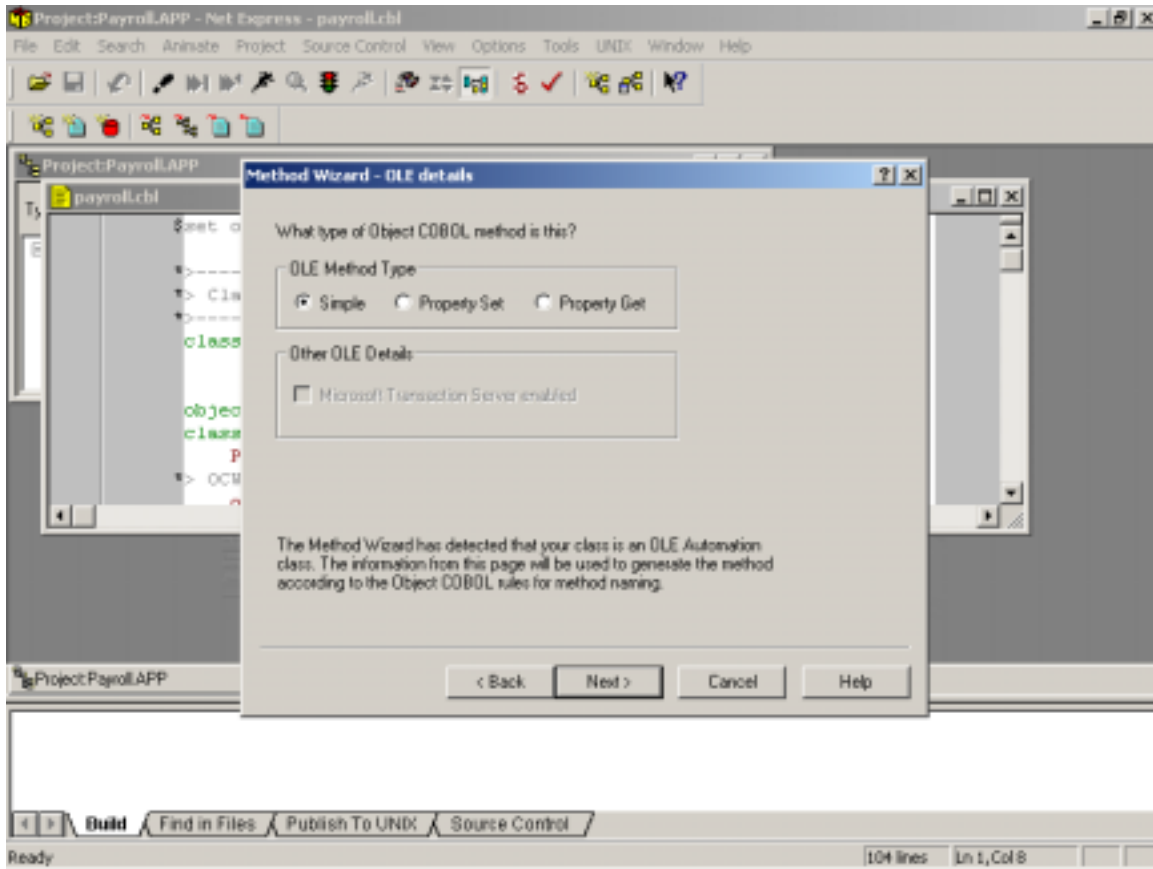
**Figure 3. A project created by the Class Wizard.**

The Wizard has added eight files to the project. The eight files, and the only one you are likely to modify `payroll.cbl`, the main source file for the class, can be viewed in the complete paper found at [www.cobolportal.com](http://www.cobolportal.com) under Technical Resources.

The Class Wizard has added two methods to the file, `queryClassInfo` and `queryLibraryInfo`. These housekeeping routines should not be changed.

At this point, the object contains all the code needed to create a valid Automation object that will start and finish. To add code that performs useful functions, we must use the Net Express Method Wizard. For this example, we will create one method, `RetrieveTotalPaycheck`, which calculates and returns the net paycheck amount. This will take two input parameters, the employee ID and the benefit year, and return the total net paycheck amount.

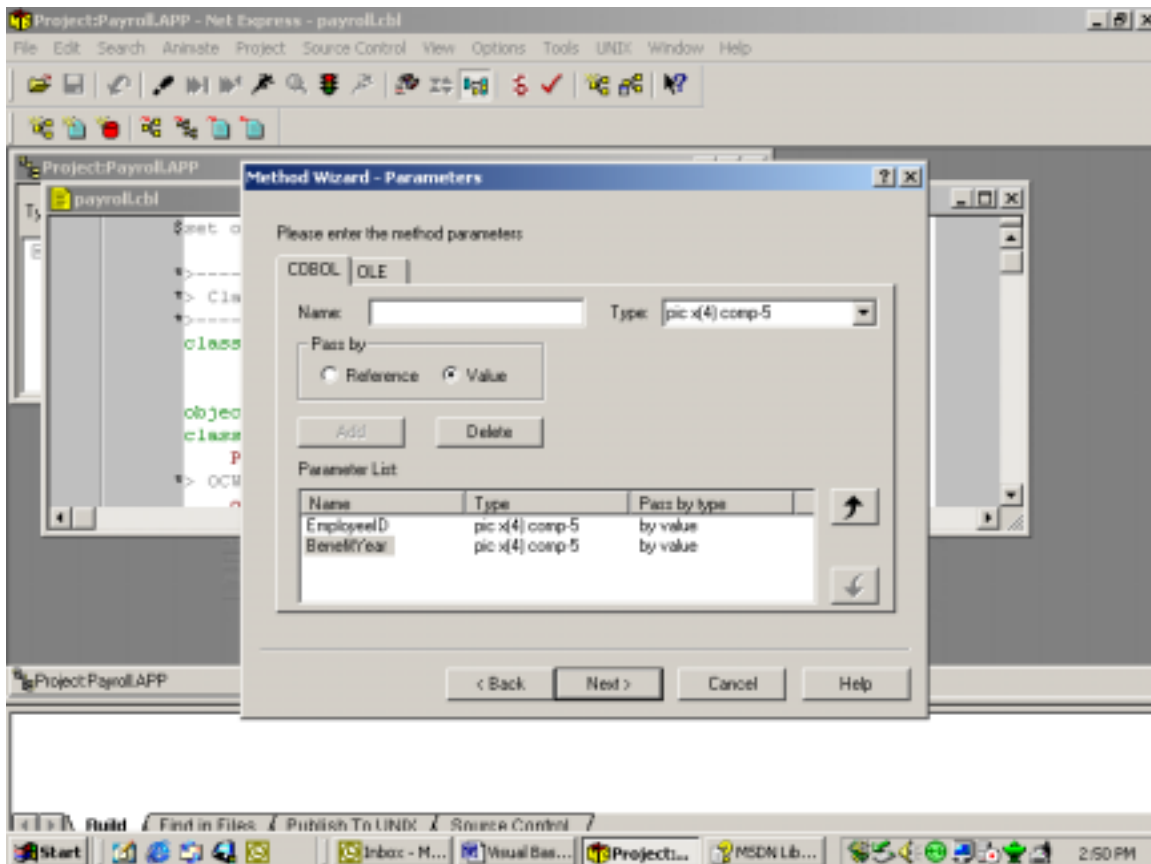
When you run the Method Wizard, it automatically detects that you are working on a class that will form an Automation object. It will prompt you to select the type of method being created, as Figure 4 shows.



**Figure 4. Method Wizard’s method type selection dialog.**

Here, the terminology can get a bit confusing. **Methods** are actions that an object can perform. For example, `RetrieveTotalPaycheck` provides a method that calculates the net paycheck total and returns it to the caller. **Properties** are functions that access information about the state of something in the Automation object. For instance, you might store the state income tax rate as a property in your object and provide routines to Set and Get that value.

In COBOL, we implement both as COBOL methods. The difference is that a method to set a property is prefixed with the word “set” and the method to get a property is prefixed with the word “get.” The Method Wizard next prompts for the method’s name and then for the names and types of the input and output parameters. Because the object is being called via COM, we must restrict our data types to those supported by COM. Figure 5 shows the input parameter entry dialog. Both parameters are four-byte integers, so we enter the parameters’ names (`EmployeeID` and `BenefitYear`) and select the type as `PIC X(4) COMP-5`.



**Figure 5. Method Wizard's input parameters entry dialog.**

The next dialog box lets you choose a return value. To return the net paycheck amount (`TotalPaycheck`), we use an item defined as `COMP-1` to indicate a floating-point value.

The Method Wizard now builds the skeleton code for the method and inserts it into the class. It also updates any other file affected in the project. To see what code has been added to `payroll.cbl`, please see the complete paper found at [www.cobolportal.com](http://www.cobolportal.com) under [Technical Resources](#).

The final step is to insert the code to do the actual calculation, using the Micro Focus Net Express editor or any other text editor. To see the full example code that retrieves information on the cost of benefits from a database and performs the calculation, see the complete paper mentioned above. All code inside the method uses standard procedural COBOL programming techniques. You need no additional OO knowledge. Additional methods can be added to the object using the same techniques.

Once you've built the project, you can deploy your Automation object on any system running a 32-bit Windows operating system using the Windows utility `regsvr32`—for example, using `Regsvr32 payroll.dll`.

To access this object from either VB or an Active Server Page, you would use very similar code. In both cases, you create an instance of the Payroll object using a call to `CreateObject` and then execute the `RetrieveTotalPaycheck` method in that instance. Refer to the complete paper mentioned earlier for code examples.

If you expose your application's functionality via Automation objects, that functionality becomes accessible from any language or application that supports Automation. This includes Micro Focus COBOL, Visual Basic, Visual C++, Borland Delphi, and Sybase PowerBuilder, as well as applications that use Visual Basic for Applications as their macro language. These include all the applications in the Microsoft Office suite as well as other non-Microsoft applications.

\* A more complete version of this paper, which includes COBOL code fragments, can be found on [www.cobolportal.com](http://www.cobolportal.com).

Wayne Rippin is a self-employed consultant. Previously, he worked for Micro Focus for 16 years, first as a systems programmer and later as a product manager. His most recent role there was director of product management, leading a team of product managers responsible for Net Express, Mainframe Express and Unix compiler products.