Presentation
Paper
Bio
Return to Main Menu

P R E S E N T A T I O N

T16

Thursday, May 4, 2000
2:30PM

AUTOMATION TESTING USING VISUAL BASIC

Mary R. Sweeney
Data Dimensions, Inc.

Automation Testing Using Visual Basic 6.0

Mary R. Sweeney

Data Dimensions, Inc.
Software QA and Test Training

Overview

- Introduction
- Why Visual Basic?
- Limitations of Visual Basic
- Where Visual Basic fits in the Test Process
- Getting Started with Visual Basic - Tools/Code
- Beyond the Basics: Using Visual Basic Code
- **Demonstrations**
- Pitfalls
- Recommendations - Lessons Learned

Introduction

- Increasing need for testers with some programming capabilities
- Visual Basic is generally testers first choice
- There are *some* things major automation tools can’t do
- VB resources and courses don’t have the QA perspective

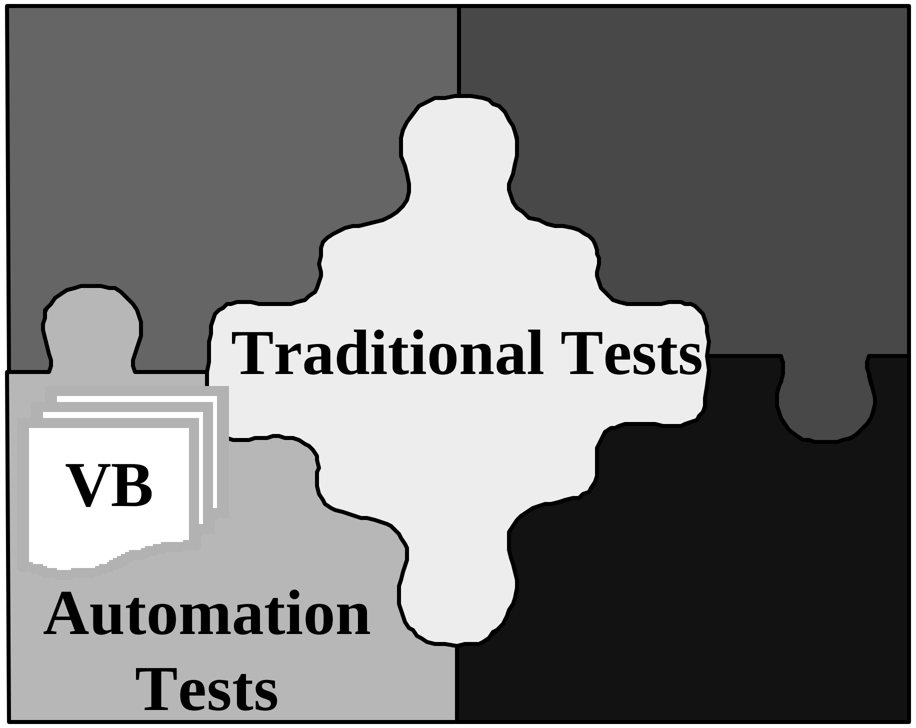
Why Visual Basic?

- Popularity
  - Many companies already have it
  - Many employees already know it
- Proliferation of Resources
  - No lack of books, training, user's groups
- Relatively Simple and Powerful tool
  - to manipulate and return info on the system and the Application Under Test (AUT)
- You own the source code!

Limitations of Visual Basic

- Not designed as an automation tool so:
  - No inherent bug reporting or test design support
  - No recording feature
  - Requires testers/programmers to write code
  - Very few resources available for using VB for testing
- Requires a shift in perspective

Where Visual Basic fits in the Test Process



- Won’t replace major automation tools
- An excellent adjunct to the overall test process - fills in the gaps

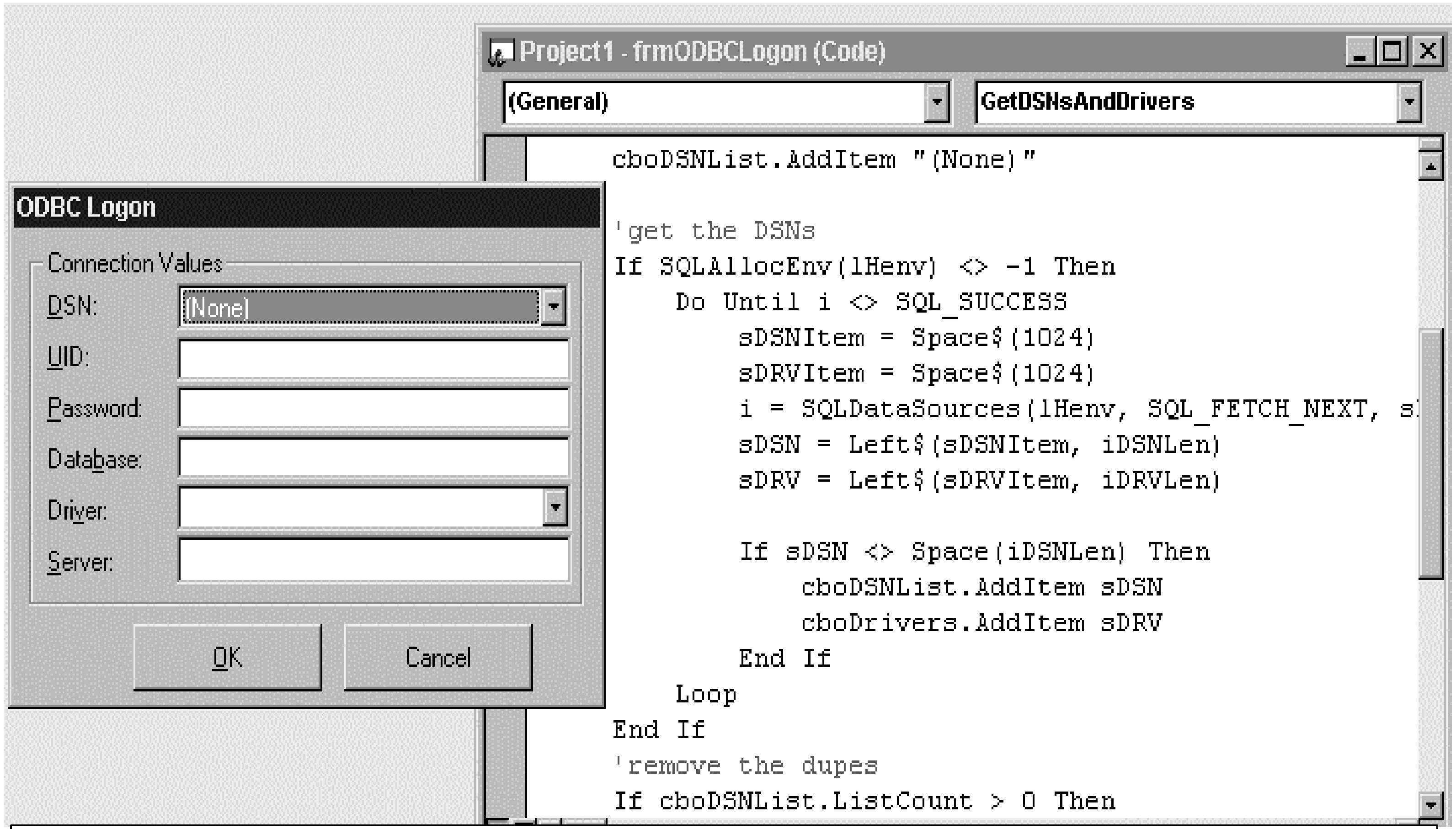
Getting Started with Visual Basic

- Management Issues
  - Automation Experience?
  - Personnel: Acceptable level of Experience
    - Entry level vs. COM objects
    - Level of experience determines level of automation
  - Correct Edition/Version: VB6 Enterprise

Getting Started with Visual Basic

- Tools:
  - Wizards and Templates
    - Data Form
  - Visual Data Tools - Database Testing - White box testing using SQL
  - T-SQL Debugger
  - Object Browser
  - Enterprise Tools
    - APE (Application Performance Explorer), OLE/COM object viewer

Wizards and Templates



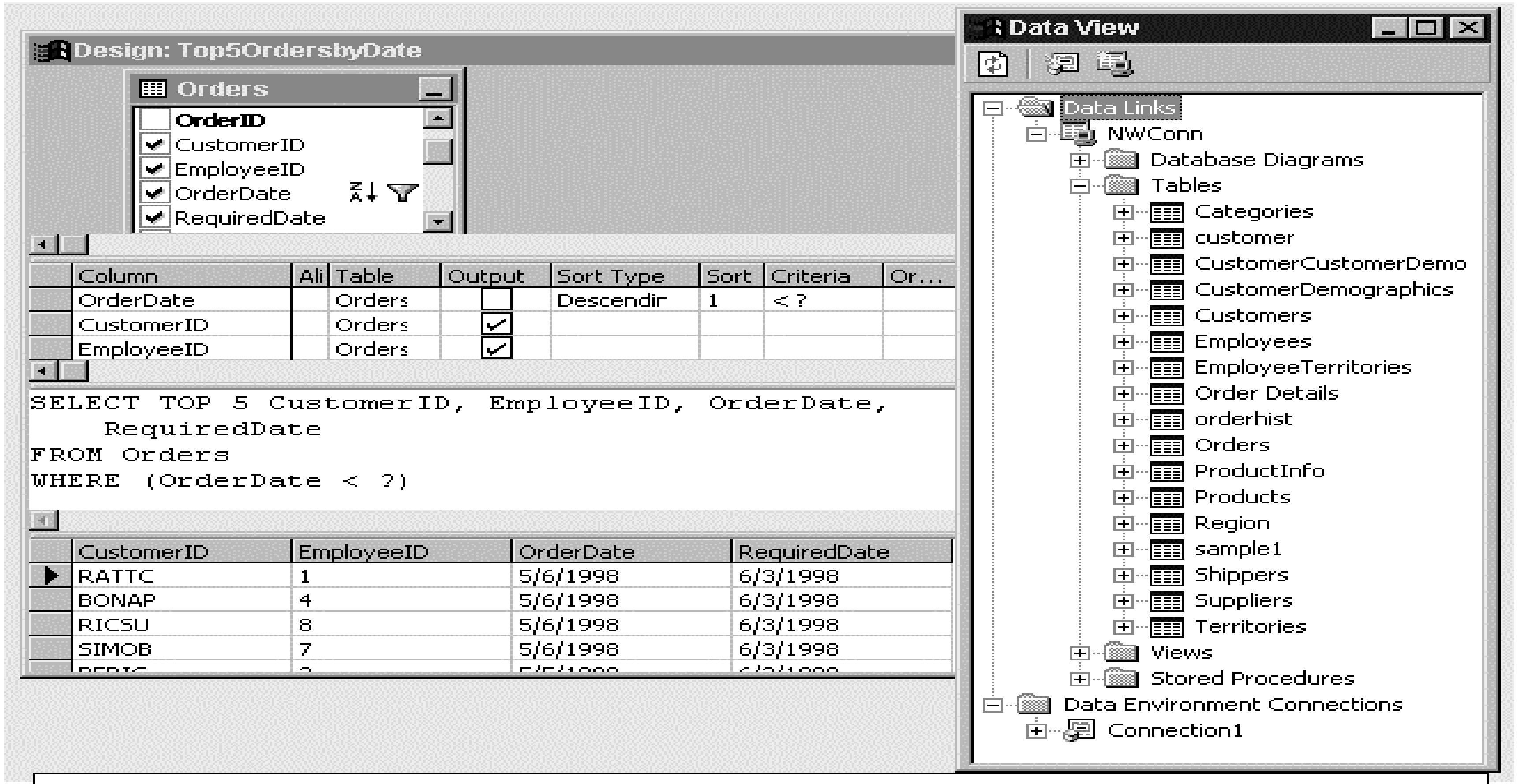
```
    cboDSNList.AddItem "(None)"

    'get the DSNs
    If SQLAllocEnv(lHenv) <> -1 Then
        Do Until i <> SQL_SUCCESS
            sDSNItem = Space$(1024)
            sDRVItem = Space$(1024)
            i = SQLDataSources(lHenv, SQL_FETCH_NEXT, s
            sDSN = Left$(sDSNItem, iDSNLen)
            sDRV = Left$(sDRVItem, iDRVLen)

            If sDSN <> Space(iDSNLen) Then
                cboDSNList.AddItem sDSN
                cboDrivers.AddItem sDRV
            End If
        Loop
    End If
    'remove the dupes
    If cboDSNList.ListCount > 0 Then
```

Figure 1. An ODBC log on form generated from a template with associated code.

Visual Database Tools



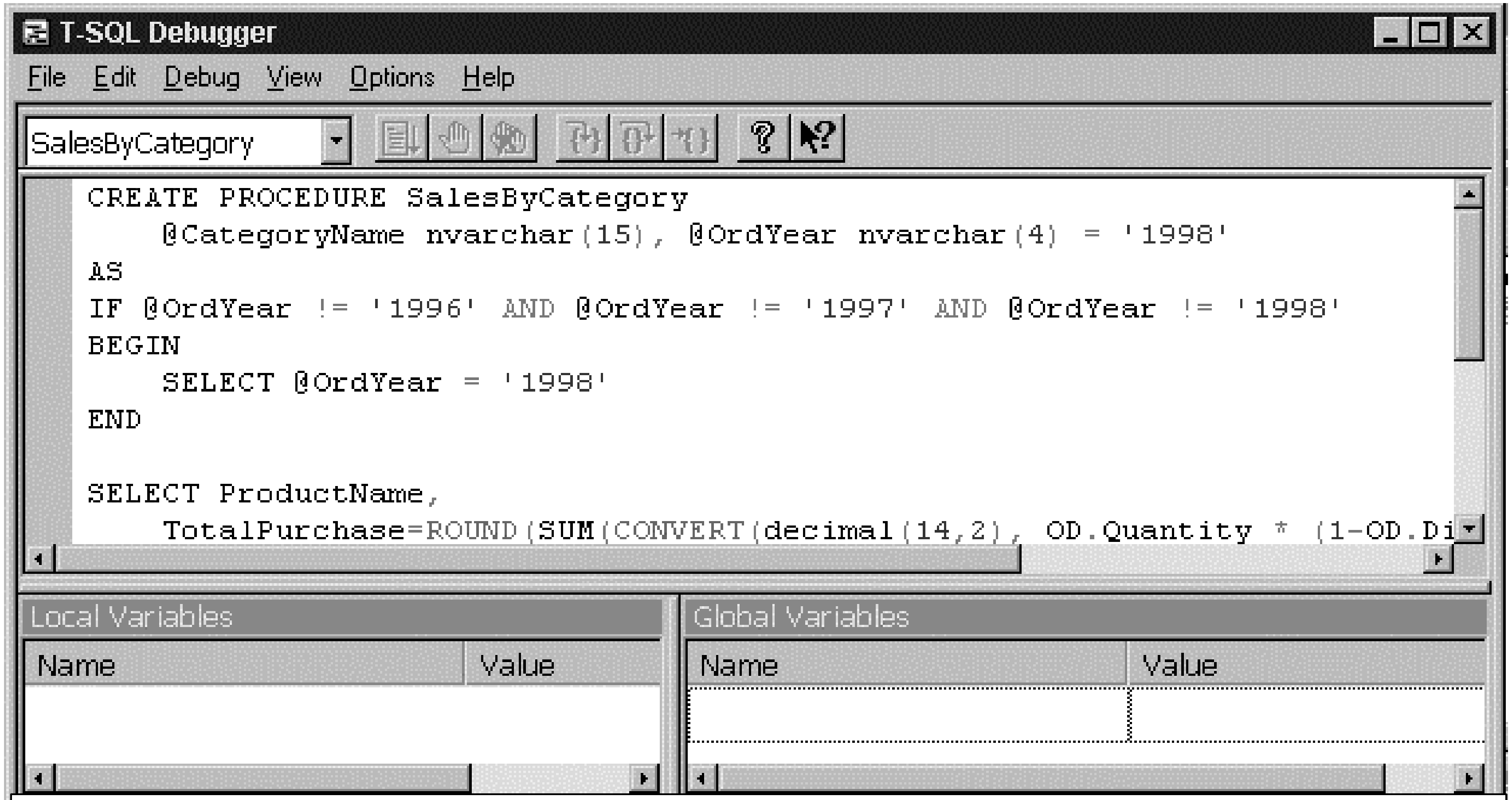
Column	Ali	Table	Output	Sort Type	Sort	Criteria	Or...
OrderDate		Orders		Descendir	1	< ?	
CustomerID		Orders	✓				
EmployeeID		Orders	✓				

```
SELECT TOP 5 CustomerID, EmployeeID, OrderDate,
    RequiredDate
FROM Orders
WHERE (OrderDate < ?)
```

CustomerID	EmployeeID	OrderDate	RequiredDate
RATTC	1	5/6/1998	6/3/1998
BONAP	4	5/6/1998	6/3/1998
RICSU	8	5/6/1998	6/3/1998
SIMOB	7	5/6/1998	6/3/1998

Figure 1. The Visual Data Tools windows include the Data View window for linking to a database and the Query Design window for creating and testing SQL statements.

T-SQL Debugger



```
CREATE PROCEDURE SalesByCategory
    @CategoryName nvarchar(15), @OrdYear nvarchar(4) = '1998'
AS
IF @OrdYear != '1996' AND @OrdYear != '1997' AND @OrdYear != '1998'
BEGIN
    SELECT @OrdYear = '1998'
END

SELECT ProductName,
    TotalPurchase=ROUND(SUM(CONVERT(decimal(14,2), OD.Quantity * (1-OD.Di
```

Figure 2. The T-SQL Debugger can be used to debug SQL Server stored procedures.

Getting Started with Visual Basic

- Coding - Starting Simple
  - Intrinsic Functions

```
FileDateTime(mStrAppname) 'returns date and time of a file
FileLen("c:\windows\calc.exe")  'returns length of a file
CurDir                       'returns the current directory
Environ("Windir") 'returns the current Windows directory
Now                   'returns current system date and time
```

Getting Started with Visual Basic

- Coding - Starting Simple
  - Open and verify text files (e.g. log files)
  - Return and set Registry information
  - GUI Automation using Shell and Sendkeys

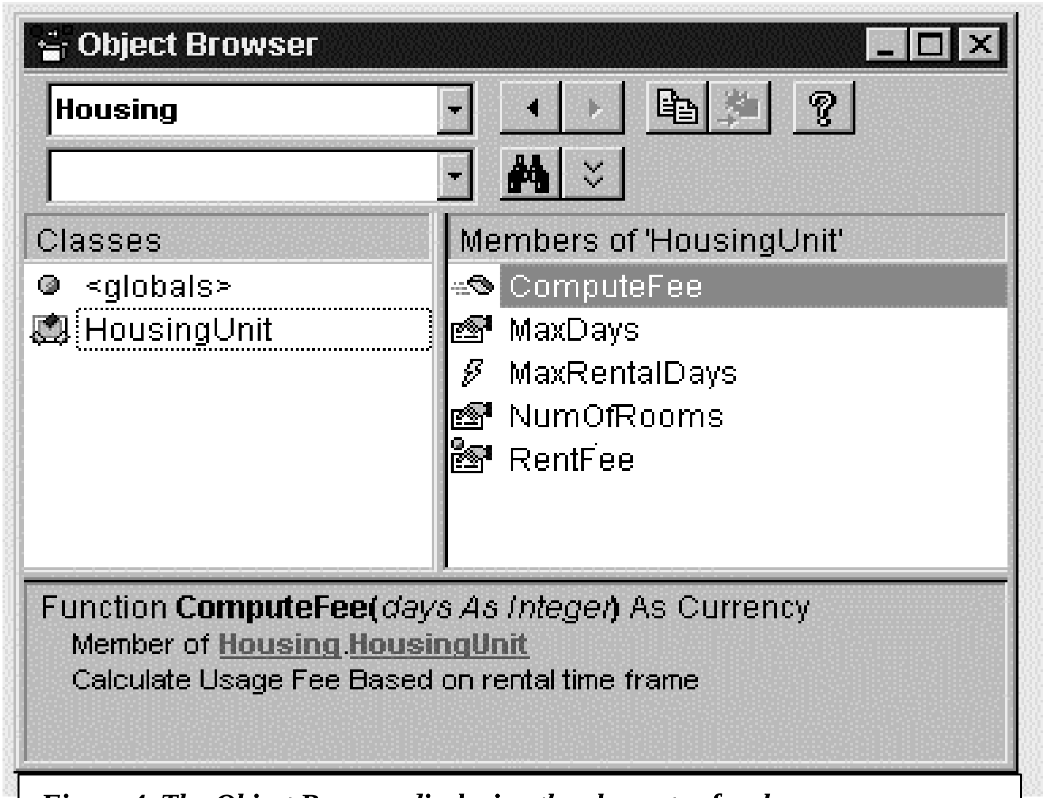
Using Visual Basic Code

- API --
  Application Programmer's Interface
  - –Windows API
  - –Application API
  - –Routines to access all open Registry keys

Using Visual Basic code

- COM Objects
  - Setting the reference
  - Testing COM objects in VB Code
  - Creating COM objects for testing
  - Testing Database applications
  - Testing middle-ware components

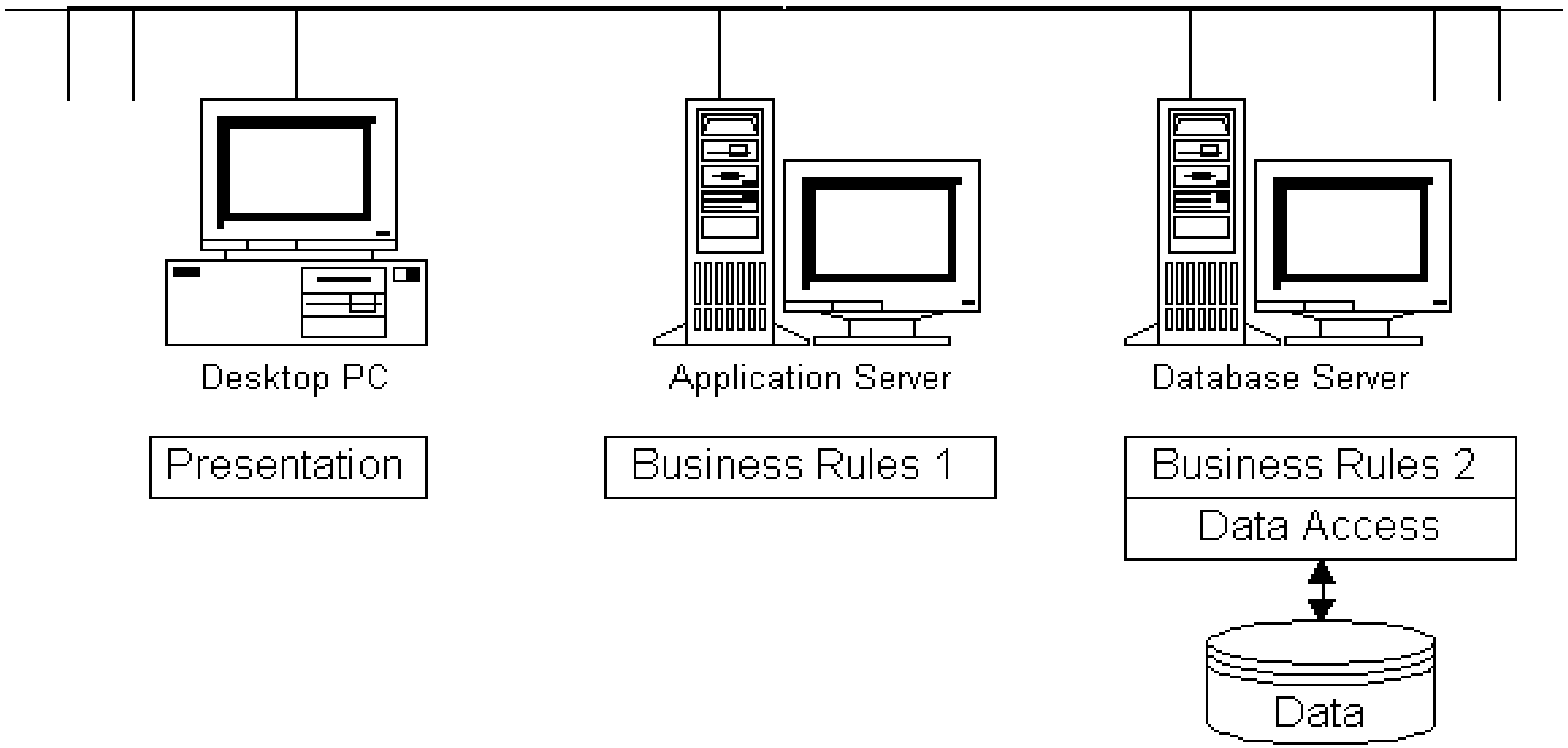
Object Browser



Function **ComputeFee(***days As Integer***)** As Currency
Member of Housing.HousingUnit
Calculate Usage Fee Based on rental time frame

Figure 4. The Object Browser displaying the elements of a class

Testing COM in n-tier architecture applications



DEMONSTRATIONS!

Recommendations/Lessons learned

- Appropriately Experienced testers/programmers
- Implement effective source code control
- Allocate sufficient time for planning, design and integration into test plan
- Plan for code reuse
- Much capability with the proper QA perspective

Automation Testing Using Visual Basic 6.0

Mary R. Sweeney

Data Dimensions, Inc

Introduction

I have used and taught classes in Microsoft Visual Basic ®[1] since its Version 2.0, in addition to teaching and consulting on test automation. Since I work for a Quality Assurance company, Data Dimensions, Inc., most of the students I have trained have been testers. The tester's goal, usually, was to gain programming experience and to perhaps build test utilities in VB. Yet, the courses I taught were initially the same as the courses taught to developers. Quite naturally, these courses began to evolve to fit what a tester needs to know. To support this, I spent some time researching what was being done with Visual Basic for test automation in our own company and compared that to the other major test automation tools we use. I learned more as I spoke with testers around the country about their use of Visual Basic for automation testing.

From these experiences, our course development team and I put together a course called *Visual Basic for Testers*. This course has since been taught on both coasts and so has afforded us the opportunity to be on-site at companies speaking to testers about their use of VB. There are some exciting things going on with Visual Basic in testing these days. Visual Basic can be, and is being, used very effectively on automation projects in a variety of ways.

Why Visual Basic?

The first time I sat down to seriously think about what you could do with Visual Basic as a test tool, I wondered then what you may be wondering right now. What can Visual Basic do that cannot be done better by test automation tools that are actually built for that purpose? And, is there really a way to use Visual Basic effectively on an automation project?

It's Popular; It's Powerful; You already have it

Visual Basic is not a testing tool. Visual Basic is a programming language for software development. Of course, a big advantage to using Visual Basic is that it's a very popular language. It's popular because it is easy to learn and it happens to be the macro language for the widely-used Microsoft Office products. Many other software companies use a form of Basic for their own products. So, it follows that there is a wide base of people with a knowledge of some form of Basic. The fact that Visual Basic is popular also means there are a lot of books and resources available for it. Chances are you already have Visual Basic and people on staff who are familiar with it.

[1] **Trademarks:**
Windows®, Visual Basic®, SDK and C++® are registered trademarks of the Microsoft Corporation. Use of trade names within this document is not intended to convey endorsement or affiliation**.**

Since Visual Basic is not really a testing tool, how is it possible to adapt it for use in testing? It turns out that Visual Basic has many features that can support the testing process. For example, it has a host of intrinsic functions that can return important information about the test platform and the application under test (AUT). Visual Basic's `Shell` function and `SendKeys` statement can also be used to run an application and manipulate its Graphical User Interface (GUI). Visual Basic's Visual Database Tools will allow you to link to a database and view its data structure. And that is all just for starters. You can get very sophisticated with Visual Basic and write essentially anything you want, like a load testing application. Of course, the tradeoffs for a more sophisticated programming endeavor are that you will need the programmers and the time.

Visual Basic can also be used to test many behind the scenes operations of the application. For example, scripts can be written to access the .ini (initialization) files and the Windows Registry. Accessing the Windows API (Application Programmer's Interface) from Visual Basic is a very powerful way to both manipulate an application and return important information. Therefore, the very fact that Visual Basic is a powerful development tool makes it a promising tool for testing.

But that brings up another question: why not use C++? There really isn't anything you can do in Visual Basic that you cannot do in C++. The answer is, for what we need to do, Visual Basic takes less time to learn, is in many ways a lot easier to learn, and quicker to write than any other language currently available. Visual Basic was designed to support rapid application development. Even though you will need to provide time for proper software development of your test utilities and scripts, Visual Basic can give you a high return on test coverage and automation in key test areas.

Limitations of Visual Basic for testing

Before we get going on the things that Visual Basic can do on a testing project, first we should explore what it won't do. Since it's not a test tool, Visual Basic does not include most of the bells and whistles that the high-end automation tools do. For example, Visual Basic has no inherent support for bug reporting or test design and documentation, as many testing tools have. It also lacks a recording feature and any automatic test settings. If you want these kinds of things in Visual Basic code, your test team will have to write them.

The bottom line here is that although Visual Basic is powerful enough to accomplish some useful testing tasks you must have knowledgeable testers and programmers to write the Visual Basic code. There isn't a lot of information out there yet to help them adapt Visual Basic for testing, either. Most all of the Visual Basic resources are geared for developers, not testers.

Using Visual Basic for testing requires a shift in perspective. A tester can come out of a standard Visual Basic course still wondering how it could ever be used on a test project. Once you change perspective and explore the possibilities, you unearth a myriad of uses for Visual Basic for QA.

Getting Started with Visual Basic on an Automation Project

Visual Basic is a powerful, flexible tool that can do much for testing, so how do we begin? If you are familiar with automation, that is, have at least one automation project to your credit, then you should consider the following:

- **Personnel experience:** Do you have experienced Visual Basic developers? The level of experience will determine the level of automation you will be able to undertake. One introductory course in Visual Basic will not be enough to enable your testers to undertake a huge automation project. However, they could use some of VB’s tools and wizards to support a testing project, and perhaps create and use some simple test utilities.
- **Correct Edition/Version:** You might look around and find your version of Visual Basic is 4.0 or less. You are better off with at least Version 5; significant changes were made in this edition and you’ll get the full benefit of the new ADO (ActiveX Data Objects) access if you have at least version 5.0 or higher. If you are looking at purchasing it, of course, go with the current release, 6.0. The Visual Database Tools are included in this edition. The listings in this paper were written using Visual Basic 6.0.

If you do not have experience with test automation in your company, it does not mean you cannot use Visual Basic at all; you can use Visual Basic tools and simple programming techniques to support your current testing process. However, if you do plan to use Visual Basic as your first foray into a major automation project, then you will need to do more investigation first on the issues involved in setting up a new automation project. All the standard issues apply: budget, time frame, management, and so on. That is an important discussion but is outside the scope of this paper. A good resource for this is *Automated Software Testing: Introduction, Management and Performance* by Elfriede, Rashka, and Paul. Although it doesn’t discuss the use of programming-level automation tools, it will help you compare other tools and the process of setting up an automation project. Another excellent resource is *Software Test Automation: Effective Use of Test Execution Tools* by Graham and Fewster.[2]

Getting Started with Visual Basic Tools:

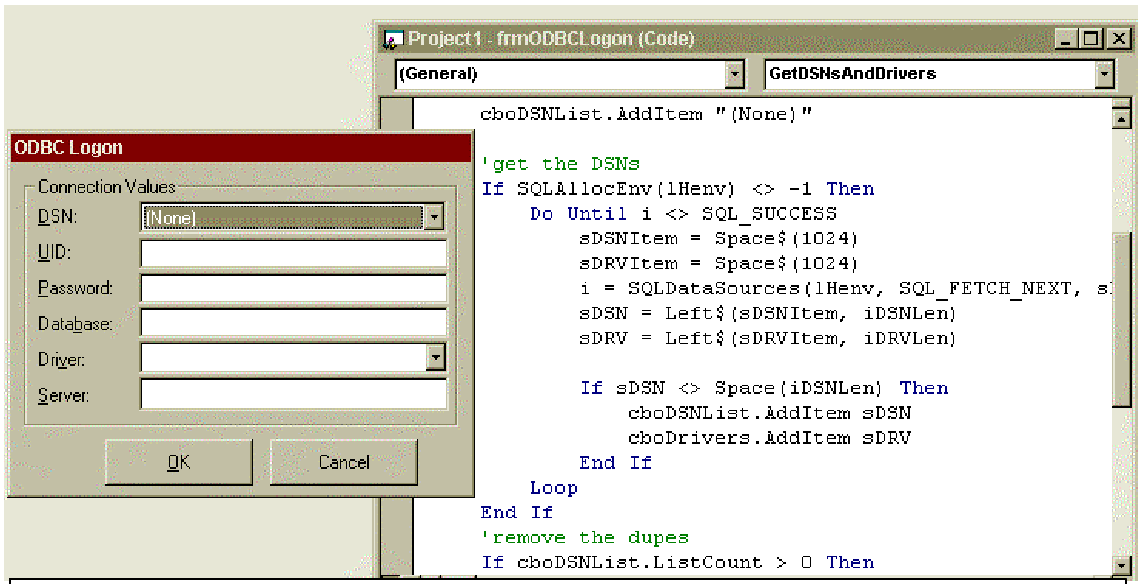
- Wizards and templates
- Visual Database Tools
- T-SQL Debugger
- Object Browser

Visual Basic 6.0 has a tool set that can be used to support testing without doing any coding. Included in this are an abundance of Wizards, the Visual Database Tools, the Object Browser, and the T-SQL Debugger.

[2] For the complete references see *Suggested Reading* at the end of this document.

Wizards and Templates

There are many Wizards available in Visual Basic. Wizards are added via the Add-in menu item. Not all of them are useful for testing; for instance, the Application Wizard generates a lot of code, but needs so much customization that it's easier just to start from scratch. One that can be useful for testers is the Data Form Wizard, which creates a form that will link to an Access or ODBC database. The form can be set up to view records individually or in a grid format. This can then be compiled into a very quickly set-up and easy-to-use test tool for inspecting data.



```
    cboDSNList.AddItem "(None)"

    'get the DSNs
    If SQLAllocEnv(lHenv) <> -1 Then
        Do Until i <> SQL_SUCCESS
            sDSNItem = Space$(1024)
            sDRVItem = Space$(1024)
            i = SQLDataSources(lHenv, SQL_FETCH_NEXT, s
            sDSN = Left$(sDSNItem, iDSNLen)
            sDRV = Left$(sDRVItem, iDRVLen)

            If sDSN <> Space(iDSNLen) Then
                cboDSNList.AddItem sDSN
                cboDrivers.AddItem sDRV
            End If
        Loop
    End If
    'remove the dupes
    If cboDSNList.ListCount > 0 Then
```

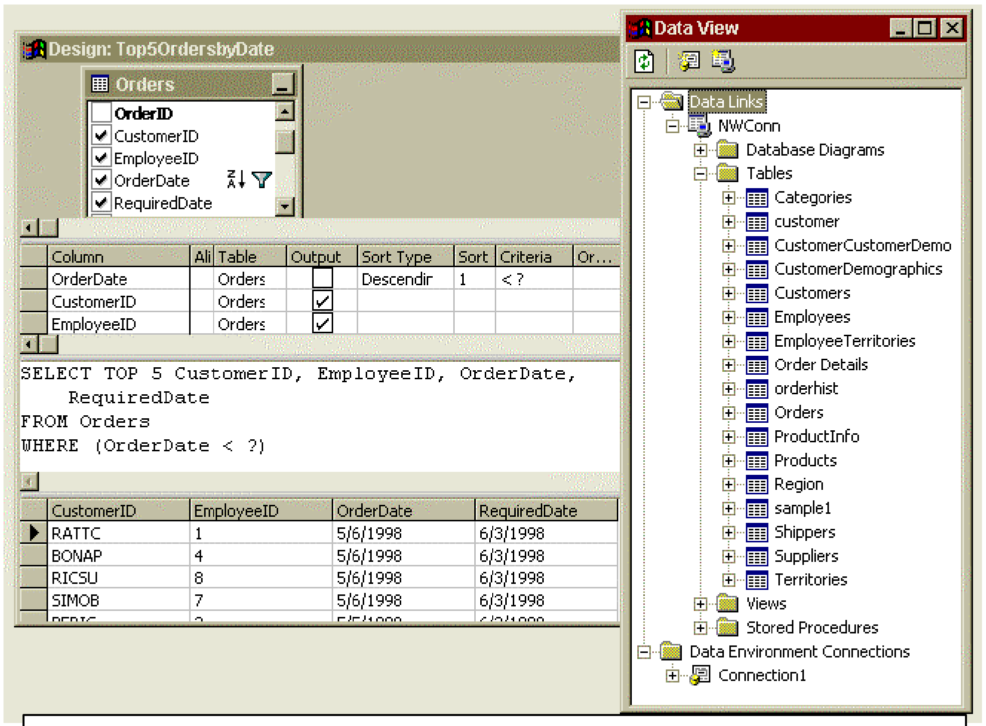
Figure 1. An ODBC log on form generated from a template with associated code.

Form Templates are provided to speed creation of standard types of forms. Not only can you use these to generate common forms like a splash screen, ODBC log on, About box, or a web browser, but the code is generated along with the form. This generated code can be very useful for cutting and pasting into your own customized forms.

As your Visual Basic experience grows, other Wizards, like the Data Object Wizard and the ActiveX Control Interface Wizard, can help you set up and deploy useful test objects with a minimum amount of coding.

Visual Database Tools: Database application testing and White Box testing using SQL

The Visual Database Tools allow you to link to an ODBC (Open Database Connectivity) or OLEDB compliant database. You can view the database structure, that is, tables, views, and other basic objects. These tools, including the Data View window and Data Environment Designers, support database application testing by allowing you to examine the database backend with a common interface. This means that if your application has data in SQL Server, Oracle and Access you can examine all of these sources using the Visual Database Tools rather than have to log into each DBMS interface separately. This allows Visual Basic to be a common interface front-end to a database back-end that is accessible via ODBC or OLE DB and that can save you some testing time and perhaps



- OrderID
- ✓ CustomerID
- ✓ EmployeeID
- ✓ OrderDate
- ✓ RequiredDate

Column	Ali	Table	Output	Sort Type	Sort	Criteria	Or...
OrderDate		Orders	☐	Descendir	1	< ?	
CustomerID		Orders	✓				
EmployeeID		Orders	✓				

```
SELECT TOP 5 CustomerID, EmployeeID, OrderDate,
    RequiredDate
FROM Orders
WHERE (OrderDate < ?)
```

CustomerID	EmployeeID	OrderDate	RequiredDate
RATTC	1	5/6/1998	6/3/1998
BONAP	4	5/6/1998	6/3/1998
RICSU	8	5/6/1998	6/3/1998
SIMOB	7	5/6/1998	6/3/1998

- Data Links
  - NWConn
    - Database Diagrams
    - Tables
      - Categories
      - customer
      - CustomerCustomerDemo
      - CustomerDemographics
      - Customers
      - Employees
      - EmployeeTerritories
      - Order Details
      - orderhist
      - Orders
      - ProductInfo
      - Products
      - Region
      - sample1
      - Shippers
      - Suppliers
      - Territories
    - Views
    - Stored Procedures
- Data Environment Connections
  - Connection1

Figure 2. The Visual Data Tools windows include the Data View window for linking to a database and the Query Design window for creating and testing SQL statements.

even training time in those database products.

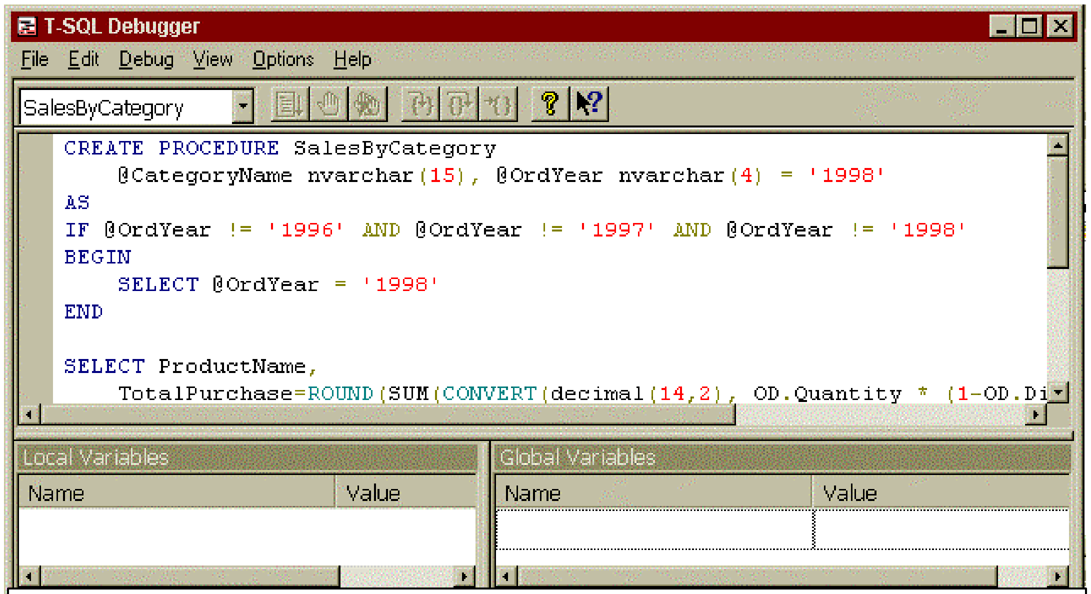
The Visual Database Tools support White box testing since you can use them to enter and test SQL statements against the database. Figure 2 shows the Data View window open with a connection to the Northwind database. Notice that you can see the tables in the database by expanding the Tables folder. Double clicking on a table will yield a window that returns all data. In SQL Server, if you log on with the proper permissions, you can modify the data and even create new data objects, like tables and views. The Data View window can also be used to inspect database objects like Views and Stored Procedures.

Also shown in Figure 2 is the design window for a command object created in a Data Environment designer. In the Data Environment designer, a connection to a database can be established and then you can create commands that can be saved and executed either from code or manually. The command specifications can be created graphically, as shown in Figure 2. Some common SQL statements for testing data, i.e., retrieving duplicate rows and uncovering referential integrity leaks, can be built here and executed when needed.

The T-SQL Debugger

The T-SQL debugger is an add-in to Visual Basic that allows you to open, inspect, run, and debug a stored procedure from a SQL Server database.

You can use the Visual Database Tools to link to a database and list all of the stored procedures; then you can invoke the T-SQL Debugger to step through the code. This can be useful if you are drilling down into the data to find the source of a bug. It may be that the problem is in the code for a stored procedure rather than in application code. The T-



```
CREATE PROCEDURE SalesByCategory
    @CategoryName nvarchar(15), @OrdYear nvarchar(4) = '1998'
AS
IF @OrdYear != '1996' AND @OrdYear != '1997' AND @OrdYear != '1998'
BEGIN
    SELECT @OrdYear = '1998'
END

SELECT ProductName,
    TotalPurchase=ROUND(SUM(CONVERT(decimal(14,2), OD.Quantity * (1-OD.Di
```

Figure 3. The T-SQL Debugger can be used to debug SQL Server stored procedures.

SQL debugger can help you determine the answer. Figure 3 shows the T-SQL Debugger with the contents of a stored procedure. The debugger allows you to insert test data for parameters and then step through the code line by line.

Interrogating COM objects with the Object Browser

Another useful tool in Visual Basic is the Object Browser. Using Visual Basic's References dialog, you can set a reference to a COM (Component Object Model) object. Once the reference has been set, you can use the object browser to inspect the properties and methods that the object exports. The Object Browser also exposes the kind of parameters required and, if the developers have been thorough, can bring up customized help on the objects. Testers can utilize this information to create verification and functionality tests on these objects in Visual Basic code.

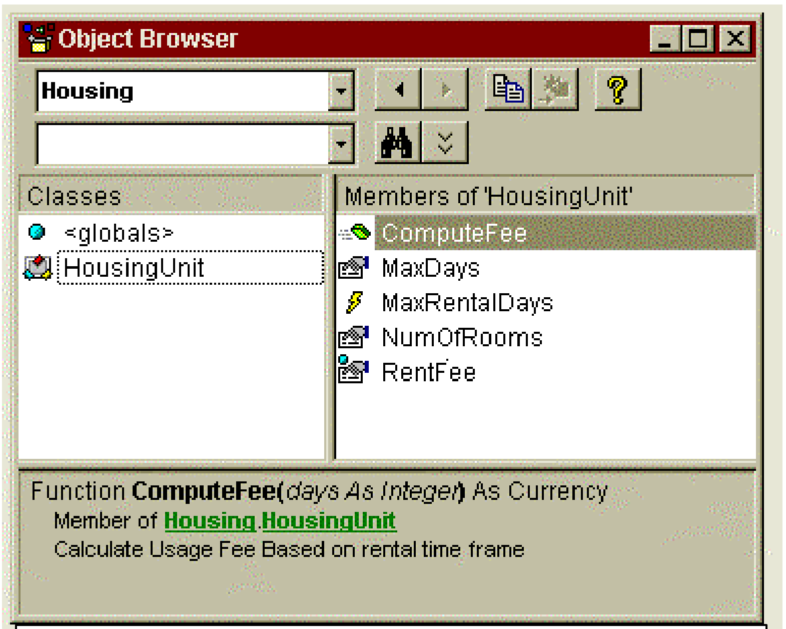
Figure 4 shows the Object Browser displaying the library of a custom COM object called *Housing*. The Housing library displays the events, properties, and methods of this object. Notice the definition of the ComputeFee function with its parameter and return value as well as a description of its purpose.

The Housing object could have been created in any language that supports the COM object model. Setting a reference to an object and viewing the information in the Object

Browser can be done in a couple of minutes by a tester with very little training. Of course, to set up Visual Basic test scripts to exercise the properties, methods, and events takes some coding.

Enterprise Tools

There are a whole lot of useful tools included in the Enterprise Edition. Some can be very valuable for testing purposes, for example, the Application Performance Explorer (APE) and the OLE/COM Object Viewer. I won't cover these tools here since they belong to the Enterprise package rather than specifically Visual Basic. However these are worth investigating.



Function **ComputeFee**(*days As Integer*) As Currency
Member of **Housing.HousingUnit**
Calculate Usage Fee Based on rental time frame

Figure 4. The Object Browser displaying the elements of a class

All of the above can be done without a line of code. Next, we'll look at simple coding examples to return key test information.

Coding – Starting Simple

Let's assume that you have some Visual Basic experience in-house and a few testers who are experienced in automation, but with a different tool. If you would like to get them started using Visual Basic without going into a full-scale automation project with VB, there are some fairly simple ways to do so. The following gives some examples of ways you can get started using simple code including using Visual Basic's intrinsic functions, manipulating text files, performing simple GUI tests, and accessing VB's special Registry key.

Visual Basic’s Intrinsic functions

Visual Basic, like all programming languages, needs to be able to return certain basic kinds of system and application information, such as current directory and path information, memory state, current file date and time, etc. Visual Basic has a very rich set of fairly easy-to-use functions. The following is a small sampling:

```
FileDateTime(mStrAppname)           'returns date and time of a file
FileLen("c:\windows\calc.exe")      'returns length of a file
CurDir                        'returns the current directory
Environ("Windir")             'returns the current Windows directory
Now                           'returns current system date and time
```

Open and Verify text files

A very common task for testing is to examine application log files for error messages. These log files are often simply text files and are very easy to open and read in any programming language using basic file handling functions. Another common testing task on an automation project, using essentially the same functions, is the creation of logging routines to record test results. A simple method of logging would be to open and write to a text file. So basic file handling is important to an automation tester.

Listing 1 shows a Visual Basic module containing simple logging routines for a testing project. These routines demonstrate opening and closing text files for logging test results. These same types of statements could be used to open, read, and write to any text file.

```
''*************************************
'*  Name:       LogUtil.bas
'*  Purpose:    This standard module contains
'*              procedures for logging
'*  Author:     M. Sweeney
'*  Date Created: 3/1/99
'*  Inputs:It presumes a constant called "Appname" exists
'*         in a general purpose module for AUT utilities
'*  Modification History:
'*  Name:       Date:       Purpose:
'*
'*************************************
Public Const LOGFILE = "c:\testresults.txt"

Sub LogIt(strLogText As String)
'Initial logging
    Static lLogNum As Long
    lLogNum = lLogNum + 1
    Debug.Print lLogNum, strLogText, Now
End Sub
Sub LogToFile(strLogText As String)
'Logging to an output file
    Static lLogNum As Long
    Dim fHand As Long
    fHand = FreeFile
    lLogNum = lLogNum + 1
    If lLogNum = 1 Then
        Open LOGFILE For Output As fHand
        Print #fHand, "Starting Tests on " & Appname, Now
    Else
        Open LOGFILE For Append As fHand
    End If
    Print #fHand, lLogNum, strLogText, Now
    Close #fHand
End Sub
Sub ReadLog()
'*  The following routine writes both to the debug window
'*  and also opens the notepad to display the output
    Dim fHand As Long
    Dim strHold as String
    fHand = FreeFile
    Debug.Print "Output from File: "
    Open LOGFILE For Input As fHand
    Do While Not EOF(fHand)
        Line Input #fHand, strHold
        Debug.Print strHold
    Loop
    Close fHand
    'open notepad and display the current output
    Shell "notepad.exe " & LOGFILE, vbNormalFocus
End Sub
```

Listing 1. The Util.bas module contains 3 simple routines for logging test information. The Logit routine writes a string to the Immediate window with a time stamp. The LogtoFile does the same to a text file. The ReadLog routine opens a text file and prints it to the Immediate window. The same commands can be used to open and inspect other kinds of text files. VB's file system object can also be used to examine file information.

These routines are simple, but having been written once, they can be upgraded to include operations that are more sophisticated, say, logging out to a database file, without changing the test code that references them. It's a good idea to be proactive on an automation project and create basic routines and then expand them as your test code becomes more complex. This will create a basic structure for your tests and can ultimately save you a lot of time. One of the first things I recommend doing on an automation project is to create a set of basic routines for starting and closing the application under test and to create logging and other relevant utilities. Doing this generates code that can be useful on other projects, sometimes with only minor modification.

GUI functionality tests

Many automation testing projects involve writing automation code to simulate user input for common tasks. I don't claim that this would be a good use for Visual Basic since GUI testing can be very tricky and time-consuming. Nevertheless, it can be accomplished to some degree and you may choose to do limited GUI functionality tests using Visual Basic.

The code in Listing 2 opens a Windows program, the Calculator, and sends keystrokes to add a set of numbers. The `StartProgram` and `CloseProgram` routines are created elsewhere and call Visual Basic's `Shell` function to start the application. The `SendKeys` statement is used to send keystrokes to the application to simulate user input.

```
Private Sub cmdTest_Click()
  Dim StartTime As Double
  Dim lngTaskID As Long
  Dim I As Integer

  lngTaskID = StartProgram(Appname)   ' Run Calculator.

  AppActivate lngTaskID       ' Activate the Calculator.
  For I = 1 To 100            ' Set up counting loop.
     SendKeys I & "{+}", True ' Send keystrokes to Calc
  Next I                      ' to add each value of I.
  SendKeys "=", True          ' Get grand total.

  CloseProgram Appname
End Sub
```

Listing 2. The cmdTest subroutine calls the StartProgram routine (not shown) to launch an application, in this case, Calculator. The SendKeys statement is used to send keystrokes to the application window.

Return and set Visual Basic Registry information.

At one point or another all testers find themselves investigating the Windows Registry; it's such a valuable place for viewing application installation settings, options and statistics. Visual Basic 6.0 includes some new functions for retrieving information from a special registry key reserved for use by VB applications. The functions simply set and retrieve information for this key, and are especially useful when testing Visual Basic applications.

Listing 3 displays code to retrieve all settings under the "HKEY_CURRENT_USER\VB and VBA Program Settings\" registry key and load them into a list box control named *lstSettings*:

```
Dim astrSettings() as string
lstSettings.Clear
astrSettings = GetAllSettings(txtAppname, txtSection)
For iCount = 0 To UBound(astrSettings)
     lstSettings.AddItem astrSettings(iCount, 0) & ": " _
                           & astrSettings(iCount, 1)
Next iCount
```

Listing 3. A code fragment that calls VB's GetAllSettings function to return an array containing a list of all settings underneath the Visual Basic registry key.

To access other Registry keys, you will need to write calls to the Windows API. These are discussed in the next section.

Beyond the basics: Using Visual Basic Code

You could spend a lot of time and resources building Visual Basic objects to test an application. Of course, then you might begin to wonder if you are entering the test tool business! Some companies are currently investing a significant amount of time and resources into building Visual Basic test harnesses, load tests, test utilities, and so on. There are some advantages to this in addition to what we have already discussed. One major advantage is that you own all the code and can modify it without worrying about licensing problems and high support fees. It's hard to underestimate the commitment in time and resources that kind of project entails, however. Short of building a full-featured test tool in Visual Basic, there are a number of useful things you can do on a test project using Visual Basic, assuming you have experienced programmers.

API – Application Programmer's Interface

API stands for Application Programmer's Interface and usually refers to a set of Dynamic Link Libraries (DLLs) containing useful procedures for an application. Windows contains a set of core libraries that developers can use as an interface between their application and the operating system. These core libraries are referred to as the Windows API. Automation testers will usually find themselves making calls to these functions since they essentially contain the keys to the system. If there is anything you can't do with standard Visual Basic commands, you can usually accomplish the task with an API call. That may include assessing the current memory state, finding the screen resolution, or manipulating a control in the GUI.

You don't find much about using the Windows API in the Microsoft Official Curriculum materials. This is because that is really not the way Visual Basic is intended to be used. The idea is to use COM objects and VB's intrinsic functions to access all necessary information. Remember, though, that Microsoft is considering the developers and not us, the testers. It's best to go through COM whenever possible and we will certainly do that, but we will also frequently need to go in through the back door with calls to the Windows API. Actually developers end up doing that a lot too; Microsoft did not expose operating system functionality using the COM model in Windows releases prior to Windows 2000.

The following code is a piece of a Visual Basic "Spy" program, written by WorldMaker.com, that uses a number of calls to Windows API routines. It assesses the current location of the cursor and returns information about the window it happens to be over, including class, window handle, parent window information, window style, and so on.

```
Private Sub Timer1_Timer()
    Dim r As Long
    Dim pt32 As POINTAPI
    Dim ptx As Long
    Dim pty As Long
    Dim sWindowText As String * 100
    Dim sClassName As String * 100
    Dim hWndOver As Long
    Dim hWndParent As Long
    Dim sParentClassName As String * 100
    Dim wID As Long
    Dim lWindowStyle As Long
    Dim hInstance As Long
    Dim sParentWindowText As String * 100
    Dim sModuleFileName As String * 100
    Static hWndLast As Long
    Const GWL_STYLE = 0
    Call GetCursorPos(pt32)               ' Get cursor position
    ptx = pt32.x
    pty = pt32.y
    hWndOver = WindowFromPointXY(ptx, pty) ' Get window cursor is over
    If hWndOver <> hWndLast Then           ' If changed update display
        hWndLast = hWndOver                ' Save change
        Cls                                      ' Clear the form
        Print "Window Handle: "; hWndOver ' Display window handle
        r = GetWindowText(hWndOver, sWindowText, 100)     ' Window text
        Print "Window Text: " & Left(sWindowText, r)
        r = GetClassName(hWndOver, sClassName, 100)       ' Window
Class
        Print "Window Class Name: "; Left(sClassName, r)
        lWindowStyle = GetWindowLong(hWndOver, GWL_STYLE) ' Window
Style
        Print "Window Style: "; lWindowStyle
        ' Get handle of parent window:
        hWndParent = GetParent(hWndOver)
        ' If there is a parent get more info:
            If hWndParent <> 0 Then
                ' Get ID of window:
                wID = GetWindowWord(hWndOver, GWW_ID)
                Print "Window ID Number: "; wID
                Print "Parent Window Handle: "; hWndParent
                ' Get the text of the Parent window:
                r = GetWindowText(hWndParent, sParentWindowText, 100)
                Print "Parent Window Text: " & _
                        Left(sParentWindowText, r)
                ' Get the class name of the parent window:
                r = GetClassName(hWndParent, sParentClassName, 100)
                Print "Parent Window Class Name: ";
                        Left(sParentClassName, r)
            Else
                ' Update fields when no parent:
                Print "Window ID Number: N/A"
                Print "Parent Window Handle: N/A"
                Print "Parent Window Text : N/A"
                Print "Parent Window Class Name: N/A"
            End If
        ' Get window instance:
        hInstance = GetWindowWord(hWndOver, GWW_HINSTANCE)
        ' Get module file name:
        r = GetModuleFileName(hInstance, sModuleFileName, 100)
        Print "Module: "; Left(sModuleFileName, r)
    End If
End Sub
```

Listing 4. A routine that will return information about application windows such as classname, ID number, Window handle and Window text. The routine uses a number of calls to the Windows API. The Windows API routines used are declared in a separate module.

The code fragment in Listing 4 demonstrates calls to the Windows API routines GetWindowWord, GetModuleFileName, GetClassName, GetWindowText, GetCursorPosition, WindowFromPoint, and GetParent. None of these is a Visual Basic function so you won't find any information about them in the Visual Basic documentation. You can find out about these functions in the Windows Platform SDK and C++ documentation. Additionally, there are a few good resources on using the Windows API in Visual Basic. By far the best one is a book by Daniel Appleman called *The Visual Basic Programmer's Guide to the Windows API.* This is a must-read text for testers. The Windows API is written for C++ programmers, but Appleman precisely describes how to effectively use these routines in Visual Basic.

Inspecting the entire Registry using Windows API

In a previous section, we talked about accessing the Registry but our discussion was limited to a specific Visual Basic Registry key. Most applications insert valuable application information in other places in the Registry during installation and while the application runs. Interrogating the registry for application settings is another common testing task on an automation project.

Principal Consultant, Jonathan Griffin, from DDI's San Mateo offices, found himself so frequently accessing the Windows Registry that he wrote a Visual Basic class module to simplify the task. Listing 5 shows a function from that module that accesses the HKEY_DYN_DATA Registry key to retrieve information about the local CD-ROM. The `cmdTestIt_Click` routine shows the code to call this function which returns a string bearing the name of the CD manufacturer as listed in the Registry.

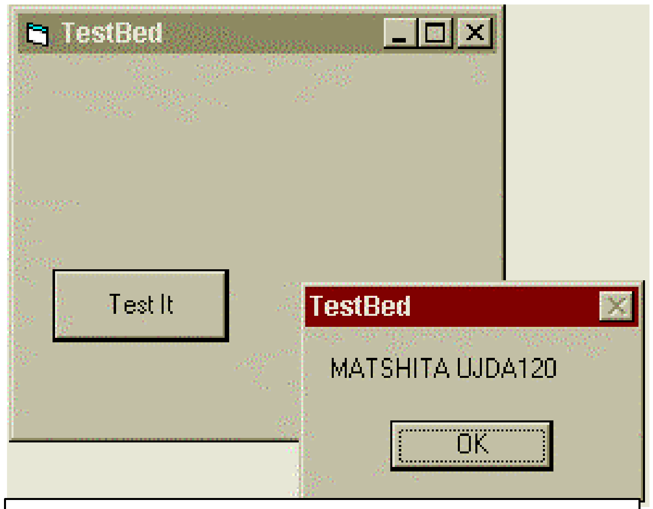
```
Public Function GetCDROM() As String
    Dim lRetValue As Long
    Dim hKey As Long
    Dim lIndex As Long
    Dim strName As String
    Dim strDeviceID As String
    Dim strClass As String
    Dim strDriverLocation As String

    lIndex = 0
    strName = String$(60, " ")
    lRetVal = RegOpenKeyEx(HKEY_DYN_DATA, "Config Manager\Enum", _
                  0, KEY_ALL_ACCESS, hKey)
    While (lRetVal = 0)

        strDeviceID = RegistryQueryValue(HKEY_DYN_DATA, _
            "Config Manager\Enum\" + strName, "HardWareKey")
        strClass = RegistryQueryValue(HKEY_LOCAL_MACHINE, _
            "Enum\" + strDeviceID, "Class")
        If (strClass = "CDROM") Then
            GetCDROM = RegistryQueryValue(HKEY_LOCAL_MACHINE, _
                "Enum\" + strDeviceID, "DeviceDesc")
        End If

        lIndex = lIndex + 1
        strName = String$(60, " ")
        lRetVal = RegEnumKey(hKey, lIndex, strName, Len(strName))
    Wend
    RegCloseKey (hKey)
End Function
Private Sub cmdTestit_Click()
    MsgBox GetCDROM
End Sub
```

Listing 5. The GetCDROM routine searches the registry for the local CD-ROM. The



MATSHITA UJDA120

Figure 5. Displaying results of a call to the GetCDROM routine.

cmdTestIt_Click event procedure issues a call to the GetCDROM routine. GetCDROM is part of a class module containing multiple routines for accessing the Windows Registry.

Creating this routine as part of a class module allows it to be compiled into a DLL. Subsequent VB test programs can then set a reference to this DLL and have the full power of these routines as though they were part of the language.

COM Objects

Component Object Model (COM) objects are used by Windows application developers to represent key elements of their application model. These objects export an interface, including properties, methods, and events, that can be accessed in other Windows-based applications that also take advantage of COM. So a COM object created in C can be accessed from a Visual Basic application and vice versa. In testing, we can take advantage of this by accessing an application's COM objects using Visual Basic regardless of the language in which they were written. In a previous section, we showed the Object Browser displaying a custom COM object. The code in Listing 6 accesses this object.

```
'Test case HSG01:  Validate value of ComputeRentFee routine
    Dim curRentfee As Currency
    Dim DaystoRent As Integer
'set object
    Dim Cabin1 As HousingUnit
    Set Cabin1 = New HousingUnit
'prompt for test values
    curRentfee = Val(InputBox("Enter Rental Fee"))
    Cabin1.RentFee = curRentfee
    DaystoRent = Val(InputBox("Enter # of Rental Days"))
'perform test
    If DaystoRent <= 7 And _
        (Cabin1.ComputeFee(DaystoRent) = DaystoRent * curRentfee) Then
        LogToFile ("HSG01: Test passed; correct calculation")
    ElseIf DaystoRent > 7 And _
        (Cabin1.ComputeFee(DaystoRent) = _
            DaystoRent * curRentfee - _
            (DaystoRent * curRentfee * 0.01)) Then
        LogToFile ("HSG01: Test passed; correct calculation")
    Else
        LogToFile ("HSG01: Test Failure; incorrect calculation")
        LogToFile ("HSG01: " & DaystoRent & " " & " Rental Fee " _
             & curRentFee)
    End If
'end test
```

Listing 6. The code above accesses the custom COM object displayed in the Object Browser in Figure 4. The ComputeFee method of the Housing object is tested.

In the same way that a class module was created and instantiated to access the Registry, your test team can create a class module with routines to access an application’s custom COM objects.

Database Testing using COM

Once a reference to an object has been set, including a database reference, Visual Basic can then access it. SQL Server exposes its COM architecture through a library called the SQL-DMO. Listing 7 displays code to access a SQL Server through the COM objects exposed by setting a reference to the SQL-DMO library.

```
Option Explicit
Dim oServer As SQLDMO.SQLServer

Private Sub Form_Load()
    Dim oDB As SQLDMO.Database
    Set oServer = New SQLDMO.SQLServer
    oServer.Connect "wsb4662", "sa"
    lstDatabases.Clear
    For Each oDB In oServer.Databases
        lstDatabases.AddItem oDB.Name
    Next
End Sub

Private Sub cmdPingIt_Click()
Dim strMessage As String
On Error GoTo errhand
    Select Case oServer.PingSQLServerVersion("TestServer", "sa")
        Case SQLDMOSQLVer_70
            strMessage = "SQL Server 7.0"
        Case SQLDMOSQLVer_65
            strMessage = "SQL Server 6.5"
        Case Else
            strMessage = "unable to determine version"
    End Select
    MsgBox strMessage
Exit Sub
errhand:
    MsgBox "Unable to connect to server; try again later!"
End Sub
```

Listing 7. Once a reference is set to the SQL-DMO object library, an object can be created to connect to a specific server, as in the Form_Load subroutine in the listing above. The code can then access any of the exposed properties and methods of the object. The above code uses the databases collection in the connected SQL Server and loops through it, loading each database name (using the Name property) into a list control. The procedure cmdPingIt_Click accesses the PingSQLServerVersion method of the server object to return the version of the Server.

The code in Listing 7 is part of a program that was eventually expanded to create a testing front-end to the SQL Server database. The value of using code to test the database backend is the verification of stored procedures, views and the correctness of data. The advantage of code over the Visual Database Tools is that we can get very specific in testing an application's functionality with the database. Of course, once the code is written it can be used repeatedly.

COM and N-Tier applications

In applications using N-tier architectures, clients/customers now frequently access data through Internet applications via browsers, while employees access data management applications for the same data through traditional front-ends written in C or other languages. Business rules are implemented as services through COM components using software like Microsoft Transaction Server (MTS) or Java Beans. Since the users access the data through these components, it is critical to test them thoroughly. Visual Basic can be used to create a test harness to bypass the application and directly access the component and put it through its paces, that is, test its interfaces, and expose its properties, methods, and events for testing. That way, when the user's front-end changes and we encounter bugs, we can test the COM component to either eliminate or verify it as the source of the bug. . This can be done in much the same way as in the examples above.

It also can be beneficial to create a middle-tier test object as well to verify the functionality of the middle-tier objects themselves. Visual Basic can be used to create middle-tier objects, as well. The bottom line is, if an application exposes COM objects, we can create a Visual Basic front-end to test those objects.

Recommendations/Lessons Learned

There are some common pitfalls in any automation project. I asked people involved in Visual Basic automation projects to tell me the things they wished they had done to make the project go easier. Here is the top four:

1. **Acquire appropriately experienced personnel.** One 3 to 5 day Visual Basic course will not automatically turn a tester into a Visual Basic professional. Hire knowledgeable automation testers with Visual Basic experience if possible. And, if

you plan an aggressive Visual Basic automation project, make sure your staff is fully trained in advanced topics like COM and API calls.

2. **Implement effective source control**. Keeping track of the test code, versioning it, and knowing what's current can be an enormous administrative task that will sink an automation project quickly if not done well. Plan for and enforce effective source code control.
3. **Allocate sufficient time.** Allocate sufficient time to bring Visual Basic automation code into accord with your current test processes. Development of Visual Basic test scripts should follow the basic rules of software development since that’s exactly what it is. This means allowing sufficient time for requirements analysis, design, code, testing and maintenance of your test scripts.
4. **Plan for reuse.** Building your Visual Basic code with a plan for reuse means creating standard and class modules with generic, reusable code. Without this, the team ends up writing the same things repeatedly. This implies effective project management and source control.

Conclusion

"(Automation) tools should be selected based on your requirements as opposed to making your requirements meet the functions of a tool." Graham Titterington, Senior Analyst at Ovum Inc., (a London-based consulting firm).

I worked with a company that has a very expensive test tool in-house; still, they were creating large Visual Basic programs to test their application. When I asked them why, the response was “We couldn’t get the tool to test the COM objects; it does a lot of thing but it won’t do that.” So they were writing the code to do so themselves. The expensive test tool was being put to good use to test the things it was designed to test, but Visual Basic was being used to fill in the gaps. That is where Visual Basic belongs in the testing arena: filling in the gaps to provide a broader, more thorough test effort.

To receive a zip file with the full code examples used in this paper, e-mail Mary Sweeney at:

Mary-Sweeney@Data-Dimensions.com

Acknowledgements

I would like to acknowledge and thank:

Ron Talmage, author of the *SQL Server Administrator's Handbook* and my colleague at Data Dimensions for his insight into SQL Server database testing.

Walt Rischer, President of the PNW Visual Basic Developer's Association and my colleague at Bellevue Community College for his excellent Visual Basic courses, which I sat in on, and all his help and advice.

Roger Sessions, for the valuable insight into COM and the middle-tier I gained from his February 2000 talk at the local VBDA.

My friends and colleagues at Data Dimensions who provided me with information on their Visual Basic automation projects especially Jonathan Griffin, Harvin Queen, and James Hancock.

The clients and customers of Data Dimensions, Inc., for whom I have consulted. I thank them for the opportunity to assist in their efforts and for all the hard work, it's the best way to learn.

Suggested Reading:

Appleman, Dan. **The Visual Basic Programmer's Guide to the Window's API**. MacMillan Computer Publishing. 1997.

Elfriede, D., Rashka, J., Paul, J. **Automated Software Testing: Introduction, Management and Performance**. Addison-Wesley Publishing. 1999.

Graham, Dorothy and Fewster, Mark. **Software Test Automation: Effective Use of Test Execution Tools**. Addison-Wesley Publishing. 1999

Sessions, Roger. **COM+ and the Battle for the Middle Tier.** John Wiley and Sons Publishing. 2000.

Visual Basic Programmer's Journal. Monthly Issues. Occasionally has articles from the QA perspective.

Mary Sweeney

Mary Sweeney is a Senior Consultant and Instructor for Data Dimensions, Inc., specializing in automation testing. She has taught courses and consulted on automation topics across the United States and in London, England. In 1999, Mary developed the Visual Basic for Testers course for DDI and has since taught it at major companies on both coasts. She has been a speaker at the Software Quality Assurance User's Group and the Seattle Visual Basic Developer's Association. Mary holds degrees in Computer Science and Mathematics from Seattle University and a certificate in Software Engineering Management from the Software Engineering Institute (SEI) at Carnegie-Mellon University. Mary is on the faculty at Bellevue Community College in Bellevue, Washington, teaching Database Design, SQL, Advanced Visual Basic, Access and SQL Server courses. She resides in Issaquah, Washington, with her husband and two great kids.