TRANSITION FROM TEACHING VB6 TO VB .NET

Azad Ali, Butler County Community College – azad.ali@bc3.edu
David Wood, Robert Morris University – wood@rmu.edu

ABSTRACT

The upgrade of Microsoft Visual Basic from version 6 (VB6) to Visual Basic .Net (VB .Net) environment represents a substantial change in the ways that programs are structured. Furthermore, additional features have been added and others deleted as a result of this upgrade. For faculty transitioning from teaching VB6 to VB .Net, it will be beneficial to have knowledge of the changes that have taken place between the two versions of the software and how it affects their course contents. This paper is to help with this transition from teaching programming courses using VB6 to the same ones teaching VB .Net. It will attempt to explain the changes that have occurred between the two versions of VB as they pertain to the topics taught in a typical first level-programming course that teaches the Visual Basic Programming Language.

Keywords: VB6, VB .Net, Visual Basic, Visual Programming, Visual Basic Programming

INTRODUCTION

Faculty teaching computer courses frequently must change their course contents to reflect update of technologies. The level of change in these contents is dependent on the extent at which the technology is updated. If the updates are minimal; then simply changing required textbooks might help accommodate the technology update. But if the change in technology is substantial, then the same faculty might have to change lesson plans, illustrated examples, exam questions, required textbooks, delivery methods or any other course materials.

The switch from the old Visual Studio version 6 to the newer Visual Studio .Net to create Visual Basic (VB) programs represents a substantial update in technology. It represents far more than changing of a version name. When upgrading from Visual Studio version 6, instead of naming it version 7, Microsoft used the term .Net as the name of the new version to denote the difference that it carries from the older technology. Faculty members teaching programming courses in Visual Basic have to learn about the changes in the new version of Visual Basic in order to switch from teaching the older VB6.

This paper is intended to give a description of such changes between the two versions of VB as they pertain to the teaching of a first level course in Visual Basic Programming. The paper will examine textbooks that are widely used in both VB6 and VB .Net courses at colleges and universities (see the reference section of this paper). It will note the difference in features illustrated, methods of delivery, and means of execution between the books that teach VB6 versus the books that teach VB .Net. The paper will explain the differences between the two versions of the software and tabulate them at the end in a summary sheet so it may be used as a reference when transitioning in teaching between the two versions of VB in a typical first level course in Visual Basic Programming at colleges or universities.
FROM PROJECTS TO SOLUTIONS

The changes between the two versions of VB are apparent from the first step that is usually taken to create a VB program. In VB6, to start creating VB programs, a project needs to be created and then additional steps may require adding a form and adding programming code to control the functionality of the program. In VB.Net, instead of creating a project, a “solution” must be created first, and the project then added.

The changes in creating VB programs are not only ones of terminology. Instead they reflect structural changes in how files are created and grouped and how they are then divided to provide the final functionality of the program. In order to explain these “structural changes” between the two versions of VB, we first created a simple project in VB6 to see the different files and folders it creates. We called this project “Proj1” and then checked the files created from this program. Three files were created from this simple program in VB6: “Proj1.vbp” for the project file, “Proj1.frm” for the form file and “Proj1.vbw” for the workspace file. We then proceeded to create the same program in VB.Net and found that it created 14 files and stored them in 6 folders. Table 1 below lists the files that were created as a result of this:

<table>
<thead>
<tr>
<th>File-Folder name</th>
<th>Description</th>
<th>File-Folder name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assemblyinfo.Vb</td>
<td>Assembly info</td>
<td>Bin\Proj1.exe</td>
<td>Runtime Executable</td>
</tr>
<tr>
<td>Form1.Resx</td>
<td>Binary resource file</td>
<td>Bin\Proj1.pdb</td>
<td>Program Database</td>
</tr>
<tr>
<td>Form1.Vb</td>
<td>Form file</td>
<td>Obj (Folder)</td>
<td>Object folder</td>
</tr>
<tr>
<td>Proj1.Sln</td>
<td>Solution file</td>
<td>Obj\Debug (Folder)</td>
<td>Holds debug files</td>
</tr>
<tr>
<td>Proj1.Suo</td>
<td>Solution User options</td>
<td>Proj1.Exe</td>
<td>Debug Executable</td>
</tr>
<tr>
<td>Proj1.Vbproj</td>
<td>Project file</td>
<td>Proj1.Resources</td>
<td>Resources file</td>
</tr>
<tr>
<td>Proj1.Vbproj.user</td>
<td>User profile</td>
<td>Proj1.Pdb</td>
<td>Program Database</td>
</tr>
<tr>
<td>Bin (folder)</td>
<td>Holds executable and link files</td>
<td>Temp (folder)</td>
<td>Temporary empty folders</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TempPe (Folder)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 – List of files and folders created from a simple project in VB.Net

Faculty may need to understand this structure for two purposes. First, students may have to be shown how to move solutions from one computer system to another. Secondly, when looking for errors, it might help to examine different source files to see the scope of the entire project.

CLASSES AND MORE CLASSES

Object Oriented Technology (OOP) is not new to the VB environment. It is used in the older VB6 environment quite a bit. However, most textbooks did not cover this subject at the beginning of their books and instead either deferred such explanation to later chapters in the book (1), (4), or did not cover it in any major depth (7), (8). VB.Net can be taught without explaining classes and objects in detail (9). However, it is hard to introduce a program that has a heading like the following without explaining about classes and inheritance:

    Public Class frmRentals ....
    Inherits System.Windows.Forms.Form
The above two lines are usually displayed at the beginning of Visual Basic programs that are created in the .Net environment. This reflects the nature of the move toward object oriented approach in VB .Net. Moreover, VB .Net includes a code generator, which adds many lines to a program. It uses the concept of “Region” to hide this generated code, but students can see it, and may cause errors if they modify it. When users want to view this hidden code, they can click at the plus sign to view the code generated by the system. If users click on the plus sign that is listed at the top of VB programs, they can view more of the code generated by the system (2). This code defines the form, the controls and the properties. This generated code lists more object oriented terms like classes and inheritance. Therefore, some explanations of the major OOP terms are deemed essential to better understand programs in VB .Net. Faculty need to understand the following terms: Class, Object, Instance, and Inheritance.

DATA TYPES AND CONVERSIONS

Data types have faced some changes in VB .Net. More data types were added while some used in VB6 were eliminated and combined with others. Variant and Currency data types are no longer available in VB .Net. VB .Net includes three newer data types (Char, Short and Decimal) that were not available in VB6. The Char data type stores a single Unicode character in a two-byte variable. This is used to store a single character in typical examples that allow users to enter one character (such as entering middle initial).

Additional changes between the two versions include the Currency and Date data types. The use of the Currency data type that was supported in VB 6 is eliminated in VB .Net and instead can use the Decimal data types as a replacement. The Date data type is available in VB .Net but is not stored in the same format as it was in VB6. In VB6, the date format data is stored in a Double while in VB .Net; the Date variables are stored in 64-bit integer. The changes of data types include also the elimination of the fixed-length format that was used in Vb6. This type of format was intended to create variables of the same length. Users can still create same length variables, but instead have to use other methods. They can for example simulate the same behavior of fixed length string variables by padding a string to the desired length with spaces as shown in the following example: Dim lname as String = Space(10)

VB .Net introduced the CType function to convert data from any one type to another compatible one. This can be used in examples where users enter data into text boxes (which are stored in text formats) to ones with numeric format so they can be used for calculations. The CType command can also be used for converting data from any type to another one of in another format. This allows programmers to deal directly with Collections. Collections own a list of objects, which, to be used, must be converted into their original type. VB6 textbooks tended to allow Collections to be an ancillary topic. They are more prominent in VB .Net.

ARITHMETIC, ASSIGNMENT AND COMPARISON OPERATORS

The arithmetic and assignment operators have changed in VB .Net. In VB 6, the assignment operator is limited to using the equal sign to make one variable equal to another. In VB Net, the assignment operators include more than just a simple equal string; they include a number of simplified assignment operators as show in the following:
For example a statement like (Total = total + 25) in VB6 might be writing in VB.Net something like (Total += 25). This shortcut; although it saves some keystrokes; it needs further explanation than the previous one. Some students have a difficulty when putting the same variable name on both sides of the equal sign. Yet, it seems more complicated to some when including both the equal sign and the plus sign in one operation.

**DISPLAYING AND FORMATTING OUTPUT**

One of the methods that are practiced to explain computer program execution sequence is to enter data, perform certain processing (such as calculating) then displays the entered and resultant processed data. These methods are commonly shown in beginner programming textbooks. In some cases, certain formats, such as adding decimals or tabulating the data are applied to enhance the displayed data or to make them more understandable.

VB6 includes the Picturebox control that is used to display such output. This control uses the old Print command to display output along with the Tab feature to tabulate it. VB6 has formatting options that are embedded within the print command. The following statement for example displays an output in a picture box named “picout”. The output contains three data, last name, date of hire and salary. The output is displayed in columns 1, 15 and 30. The date is displayed using the Formatdatetime option while the salary is formatted using the FormatCurrency option.

```
picout.print tab(1); lastname; tab(15) formatdatetime(datehire); tab(30) _
    formatcurrency(salary)
```

Print is not supported in VB.Net so one cannot display output in the same old way in VB Net projects. Printing is much more complicated and involves creating Graphics objects and using the Graphics.DrawString method. Because of the complexity, current VB.Net textbooks use the ListBox control commonly (5) to display output similar to what has been used in the old Picturebox control. Output can be displayed in a multi-line listbox by combining it as a string and then including the different columns in curly brackets. VB.Net also uses some characters to distinguish the output to be displayed. The following example shows the same output illustrated in the previous example in VB6, but this time it is displaying them in a list box as practiced in VB.Net.

```vbnet
dim fmtstr as String = "{0, -8}{1, 12:n}{2, 15:c}"
lstout.items.add(String.Format(fmtstr, lastname, datehire, salary))
```

**ARRAYS**

Arrays are also different in the new version. To begin with, a slightly different syntax is used to declare and initialize arrays in VB.Net. This syntax allows specifying not only the size of the array but also the initial values for it. In VB6, the redimension statement can only be on arrays that are not dimensioned when it is declared. In VB.Net, the redimension statement can be
executed only if the arrays are dimensioned when declared. In VB .Net, the “preserve” keyword can be used to copy original arrays into new ones. Another change of the use of arrays is the deletion of control arrays in VB .Net. VB6 used to allow the use of control arrays so to be able to add controls dynamically. This option is not available in the newer version of VB. Instead, typical VB controls can be used for creating arrays at runtime.

Arrays in VB .Net are classes that have methods and properties. One of the methods that are supported in the array class is the sort method. Thus, instead of using the old bubble sort and binary sort, a simpler method can be used to accomplish the same thing through the use of the sort method within the array class.

**ERROR HANDLING**

The changes of runtime error handling represent one of the most extensive changes between the two versions of VB. Both versions of VB implement error handling, but the implementation is quite different between the two. In VB6, the implementation of error handling is mainly through the use of On Error … GOTO statements. But the GOTO statement is dreaded in a structured programming environment and is replaced in VB .Net with the newer exception handling syntax routine. The basic syntax for including error handling in VB6 programs is to include the GOTO statement forces the program to go to a particular label or program segment if a runtime error occurs. The label or program segment then is supposedly displays an error message and some helpful information to locate and fix the error later. The following figure shows an example written in VB6 that let the user enter values in two text boxes: Unit Cost, Quantity. It will then calculate the total and display the result in the appropriate text box. The program will branch to the cmd_calc_error if a runtime error occurs to display an error message alerting the user of such error in the program.

```vbnet
Public sub cmd_calc
    On error goto cm_calc_error
    Dim uncost as single, qty as integer, tot as double
    uncost = val(txtuncost.text)
    qty = val(qty.text)
    tot = uncost * qty
    Ttxtot = cstr(tot)
    Exit sub
    Cmd_calc_error:
        MsgBox(“Error in calculating total cost”)
End Sub
```

Figure 1 – Sample of Error Handling in VB6

VB Net handles error messages differently. It uses the try … catch block to handle error displays. The simplest syntax in this version for runtime error handling begins by using the Try statement, which is then followed by program statements. After that the Catch statement is listed to handle the error by displaying a message to alert the occurrence of such error (2). The following figure 2 shows an example of handling runtime error in VB .Net using the same example that was displayed in figure 1 above:
Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCalculate.Click
    Try
        Dim unitcost As Decimal, qty As Single, totcost As Double
        unitcost = Convert.ToDecimal(txtunitcost.Text)
        qty = Convert.ToSingle(txtqty.Text)
        totcost = unitcost * qty
        txtTotalCost.Text = Convert.ToString(totcost)
    Catch ex As Exception
        MsgBox("The following error number occured during calculation " & Err.Number, MsgBoxStyle.Information)
    End Try
Figure 2 – Example of error handling in VB .Net.

DATA FILES VS IO STREAM

Some textbooks still cover the topic of sequential file to illustrate the retrieval of data from text files. VB6 uses the Open, Do ... While, Input and close statements to open a sequential file, read it and then close the file. Figure 3 below shows a sample of program segment that opens a sequential file, read the data line by line, and then close the file after completing processing it.

Open "A:Customer.Txt" For Input As #1
    Do While Not EOF(1)
        Input #1, Custnum
        Input #1, CustName
        Input #1, Custaddr
    Loop
Close #1
Figure 3 – Sample of sequential file retrieval in VB6

VB .Net does not handle retrieval of data files similar to VB6. Instead of opening an input file and then reading the input files, VB .Net uses terms like IO Streamreader, Readline and peek in their process of text files. The following figure 4 shows the same program segment that was written in shown in figure 3 above but is implemented within the VB .Net syntax.

Dim cs As IO.StreamReader = IO.File.OpenText("Customer.TXT")
Do While cs.Peek() <> -1
    custnum = cs.ReadLine
    custname = cs.ReadLine
    custaddr = cs.ReadLine
Loop
cs.Close()
Figure 4 – Sample of processing IO stream in VB .Net

VB .Net Version Incompatibility

A major problem surfaced from teaching VB .Net is the version incompatibility between VB .Net 2002 and VB .Net 2003. These two versions are not compatible and if a program is written in one version, it will not work on the other version without some modifications. This issue is
specifically problematic if the college has one version installed on their computers while some students have the other version. Therefore, in order to convert VB .Net projects from the 2002 version to the newer 2003 version, the following steps must be followed:

1. Create a blank Solution.  
2. Add a new project to the solution.  
3. Delete the new blank form created.  
4. Add the source files from the project to be converted to the new project.  
5. Reset the Startup Object for the new project.

**SUMMARY**

Table 2 below shows a summary of the differences between the two versions of VB, which may be useful when transitioning from teaching VB6 to VB .Net.

<table>
<thead>
<tr>
<th>Type of Change</th>
<th>VB6</th>
<th>VB .Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure and files created</td>
<td>Only 3 files created</td>
<td>9 files and 4 folders created</td>
</tr>
<tr>
<td>OOP Terminology</td>
<td>Less emphasis on OOP</td>
<td>Substantial emphasis on OOP</td>
</tr>
<tr>
<td>Data Conversion</td>
<td>Different conversion methods</td>
<td>CType introduced</td>
</tr>
<tr>
<td>Arithmetic operators</td>
<td>Use of = as assignment operators</td>
<td>Use +=, -=, *=, /=</td>
</tr>
<tr>
<td>Display output</td>
<td>May use picture box and Print</td>
<td>May use list box or Graphics.DrawString</td>
</tr>
<tr>
<td>Arrays</td>
<td>Use of control arrays</td>
<td>Control array not used</td>
</tr>
<tr>
<td>Functions/ Procedures</td>
<td>ByVal is default in passing</td>
<td>ByVal is default in passing</td>
</tr>
<tr>
<td>Error handling</td>
<td>Use On Error …. GOTO</td>
<td>Uses Try … Catch</td>
</tr>
<tr>
<td>Sequential File Handling</td>
<td>Uses Input File Terms</td>
<td>Uses IO Stream Terms</td>
</tr>
</tbody>
</table>

Table 2 – Summary of changes teaching VB6 to VB .Net

**REFERENCES**