Business Students Sharpen C# Programming Skills with Visual Studio Tools for Microsoft Office

Jerry Chin
Sheryl Brahnam
Mary Chin
College of Business Administration
Southwest Missouri State University
Springfield, MO 65804, USA

ABSTRACT

In October of 2003 Microsoft released a new application package to create project templates for document-centric solutions for the host applications Word and Excel. Using Microsoft Visual Basic or C#, code modules can be created in much the same way as using Visual Basic for Application (VBA). This paper discusses the development of a C#/Excel student assignment. In addition, this paper provides a systematic view of the relationships between source modules, internal data structures, and the worksheet.

Keywords: C# application, code modules, Excel worksheet, classes.

1. Introduction

The first course in programming can be and usually is a frustrating experience for many MIS/CIS undergraduates. The typical MIS curriculum tends to lean more towards a consulting/analysis career path. The application oriented CIS curriculum is generally suited for future four-year programmer/analysts. In this paper, we assume a beginning class in a business school – CIS environment. Most CIS majors in a school or business administration school will be computer literate, having completed a microcomputer application course which features word processing, spreadsheets, database, and basic Web development. Microsoft's Office suite with Word, Excel, Access and FrontPage is one such software package that could be easily used for such a fundamental course.

For this project the student should have some background in programming and have made the fundamental jump to object-oriented programming. In addition to CIS students, power-users, MIS specialists, and programming specialists in accounting or financial functions of the organization would have a particular interest in the resulting synergy of C# and Excel.

The arrival of a package such as Microsoft Visual Studio Tools for Office (MSVSTO) is a reflection of the evolving technology into other parts of an organization that is not directly related to the MIS/Technology department. It is not a question of VBA being limited or falling out of favor, but a matter of using the existing programming staff which will be a more Java-centric work force. The C-flavored syntax is the foundation of not only C++ and C, but many other programming languages such as Javascript and shell scripting languages. C# creates a familiar environment for programmer/analysts. As desktop applications return or originate in an organization's functional areas such as accounting or finance, some programming from non-programming staff will be seen as a highly valued skill.

In a VBA/Excel application, the user enters data directly on the worksheet and creates VB procedures as macros or the Visual Basic editor. Any data processing results usually are deposited directly on the worksheet. Similarly, in our C#/Excel environment, data entry is accomplished via a form and worksheet activity is a result of C# procedures.

At this time there are few "how-to" books concerning the VSTO software package. The book, "Using Microsoft Visual Studio Tools for the Microsoft Office System**:** Msm2052acppb" has a January 2004 print date. Students and interested parties should consult Java and C# texts and references. Examples are cited in the Reference section. Microsoft documentation and working examples are available in the Help feature of VSTO.

2. THE MARKETING PROBLEM

Assume that QueValue managers desire to measure the sales activity in the eight US distribution centers conveniently located at or near hub sites of national ground/air package service companies. Each regional manager reports the activity of pre-selected items based upon corporate projections and market analysis.

A quick analysis by the Applications Group has determined that one of the forms of the project is a four-part input form for the Center, Number of Items, Item Number, and the transaction date. The Cleveland OH Center form is shown in FIGURE 1.

In addition, also suppose that the Marketing department has a number of marketing directives triggered by individual managers using the system. As an illustration, Marketing Directive 321 (MD321) is implemented in the student exercise. If the number of Units exceeds 1000, then the order is automatically increased by 20% and diverted to the center located in Canton, OH.

Viewing FIGURE 1, using Microsoft's Visual Studio Tools for Office (VSTO), the following components were generated:

- QueForm.cs
- QueForm.cs[Design]
- Center.cs
- dataOhio
- dataTableOhio and
- ThisWorkbook.cs.

In C# all functions and data members must be accessible from a class. Therefore, a class can be added as a separate code module. Likewise we add the form, named QueForm. This generates two C# components, a code module, called "QueForm.cs" and the corresponding graphical representation, "QueForm.cs[Design]". The third component, "ThisWorkbook.cs", is a code module created for access to the Excel spreadsheet, which becomes visible at runtime.

"Center.cs" is a module that contains the class holding Center, Date, Units, Units_Num and a static member to track the actual number of entries :

```
public class Center
      { public string center;
        public string date;
        public string units;
           public string units_num;
       public static int recordcnt;
           public Center()
                      { …..
```

We also use a dataset, essentially cache or in-memory copy of data, called dataOhio. We bind our form's datagrid, named dataTableOhio, to the dataset. This automatically creates a visual gateway to the data.

3. SETTING UP THE MACHINERY

In FIGURE 2, it can be seen that there exists relationships between the code modules and the form. These relationships are constructed by declarations and function calls. We address the construction and refer to them via numbered arcs or arrows in FIGURE 2.

The Form and its code. Creating a form called "QueForm", generates a corresponding C# code module called, "QueForm.cs" as a class with a similar name in a namespace QueForm. Any data in the form can be accessed by reference to the appropriate textbox. We use the class called Center and load its class members with form data. This mechanism to pass form data to ThisWorkBook.cs is denoted in Figure 2 by arc 4. The QueForm.cs C# code is the following:

```
Center c = new Center();  //a new center object
c.center = textBox1.Text;
c.date = textBox3.Text;
c.units = textBox2.Text;
c.units_num = textBox4.Text;
Center.recordcnt ++; // Bump Record Counter

CheckMD123(ref c);
```

As advertised, we implement MD321 as part of QueValue's processing requirements. The call CheckMD321 passes the Center object to the following procedure:

```
private void CheckMD123(ref Center x )
{
 double work1, work2;
 work1 = Convert.ToDouble(x.units_num);
            if (work1 > 1000.00)
               {work2 = work1 * 1.20;
              x.center = "Canton";}
              else work2 = work1;

 x.units_num = Convert.ToString(work2);}
```

At this point, the data on the form is ready to be written to the Excel worksheet. In our scenario, we first write to a data table linked to the datagrid on the form:

```
DataRow myrow = dataTableOhio.NewRow();
      myrow["Center"] = c.center;
      myrow["Item Number"] = c.units;
```

```
    myrow["Date"] = c.date;
    myrow["Units"] = c.units_num;
    dataTableOhio.Rows.Add(myrow);
// add a row to table
```

Finally, to accomplish arc 2 in Figure 1, we pass the Center object and the current record count as parameters to the code module called ThisWorkBook.cs :

```
int reccount = Center.recordcnt;
this.excelCode.GoToExcel(c, reccount);
```

The procedure GoToExcel in ThisWorkBook.cs has the following structure:

```
public void GoToExcel(Center c, int count )
{
 Excel.Worksheet s1 =
(Excel.Worksheet)this.ThisApplication.
Sheets.get_Item("Sheet1");
    ((Excel.Range)s1.Cells[count+1,1]).Value2 =
c.center;
    ((Excel.Range)s1.Cells[count+1,2]).Value2 =
c.date;
    ((Excel.Range)s1.Cells[count+1,3]).Value2 =
c.units;
    ((Excel.Range)s1.Cells[count+1,4]).Value2 =
c.units_num;
((Excel.Range)s1.Cells[1,1]).Value2 = count;
}
```

The "Cells[row,col]" syntax indicates that the data is being assigned to cells in the Excel worksheet. This is arc 3 in Figure 2. The Form and the user's view of the Excel worksheet can be seen in FIGURE 3.

Half of the task is now complete. The user has entered data on the form, data has been processed, and data has been entered on a worksheet.

On the form in Figure 1 there is a button labeled, "Excel To Data Table". Can we access data from the worksheet? The code connected to that button event is

```
public void ExcelToTable(ref DataTable d)
    {
        Excel.Worksheet s1 =

(Excel.Worksheet)this.ThisApplication.Sheets.get_Ite
m("Sheet1");
        Excel.Range rng2;
        rng2 = (Excel.Range)s1.Cells[1,1];
        int MaxRow =
Convert.ToInt16(rng2.Value2);
        // add 1 to Max
        for (int Rindex = 2; Rindex <= MaxRow +1
; Rindex++)
          {
DataRow r = d.NewRow();
   rng2 = (Excel.Range)s1.Cells[Rindex,1];
r["Center"] = rng2.Value2;
   rng2 = (Excel.Range)s1.Cells[Rindex,2];
r["Item Number"] = rng2.Value2;
   rng2 = (Excel.Range)s1.Cells[Rindex,3];
r["Date"] = rng2.Value2;
   rng2 = (Excel.Range)s1.Cells[Rindex,4];
r["Units"] = rng2.Value2;
d.Rows.Add(r);
  }
}
```

Any data in the data table can be accessed and sent back to the appropriate textboxes in QueForm. Using code very much like code associated with arc 3, any information in the datatable can be inserted back into the Excel worksheet. In the following code "d" can be interpreted as a reference to datatableOhio passed to a procedure in ThisWorkBook.cs.

```
((Excel.Range)s1.Cells[ExcelIndex,1]).
Value2 = d["Center"];
((Excel.Range)s1.Cells[ExcelIndex,2]).
Value2 = d["Item Number"];
((Excel.Range)s1.Cells[ExcelIndex,3]).
Value2 = d["Date"];
((Excel.Range)s1.Cells[ExcelIndex,4]).
Value2 = d["Units"];
```

The process flow designated by arc 5 in Figure 1 is now established. We have completed traversing the flow arcs in Figure 1.

4. STUDENT HELP

The Try-Catch block is a powerful method in development and overall design. The usual Debug facility for VSTO is available to the student at development time to isolate logic or syntax errors. In addition, the student may elect to use Try-Catch code to pinpoint errors. The form of the Try-Catch code structure is reminiscent user print routines to trace through code modules. These two error tracking approaches are helpful to the student. In addition, the usual expected Debugging facilities using Breakpoints and Watch Expressions are available with VSTO.

5. CONCLUSION

The new software Microsoft Visual Studio Tools for Office provides the tools to create document-centric solutions in the Microsoft Office environment using C# or Visual Basic. The student developer selects Word or Excel as the host application. This is an excellent way for students, not necessarily CIS majors, to combine their familiarity of Word or Excel with their C# programming / analysis development. Microsoft will continue to target nonprofessional programmers. Microsoft has

announced it will create "Express" editions of SQL Server 2005, Visual Studio 2005, Visual Web Developer 2005, and its C# 2005, C++ 2005, and Visual J# 2005 computer-programming languages. An accompanying textbook as a reference for VSTO is strongly suggested.

6. REFERENCES

Barnes, David J. and Michael Kolling (2003) Objects First With Java. Pearson Limited Education. Edinburgh England.

Davis, Stephen Randy (2002) C# For Dummies. Hungry Minds Inc. New York.

Horton, Ivor (2003) Ivor Hortons' Beginning Java 2 SDK 1.4 Edition. Wiley Publishing, Indianapolis, IN.

7. APPENDIX

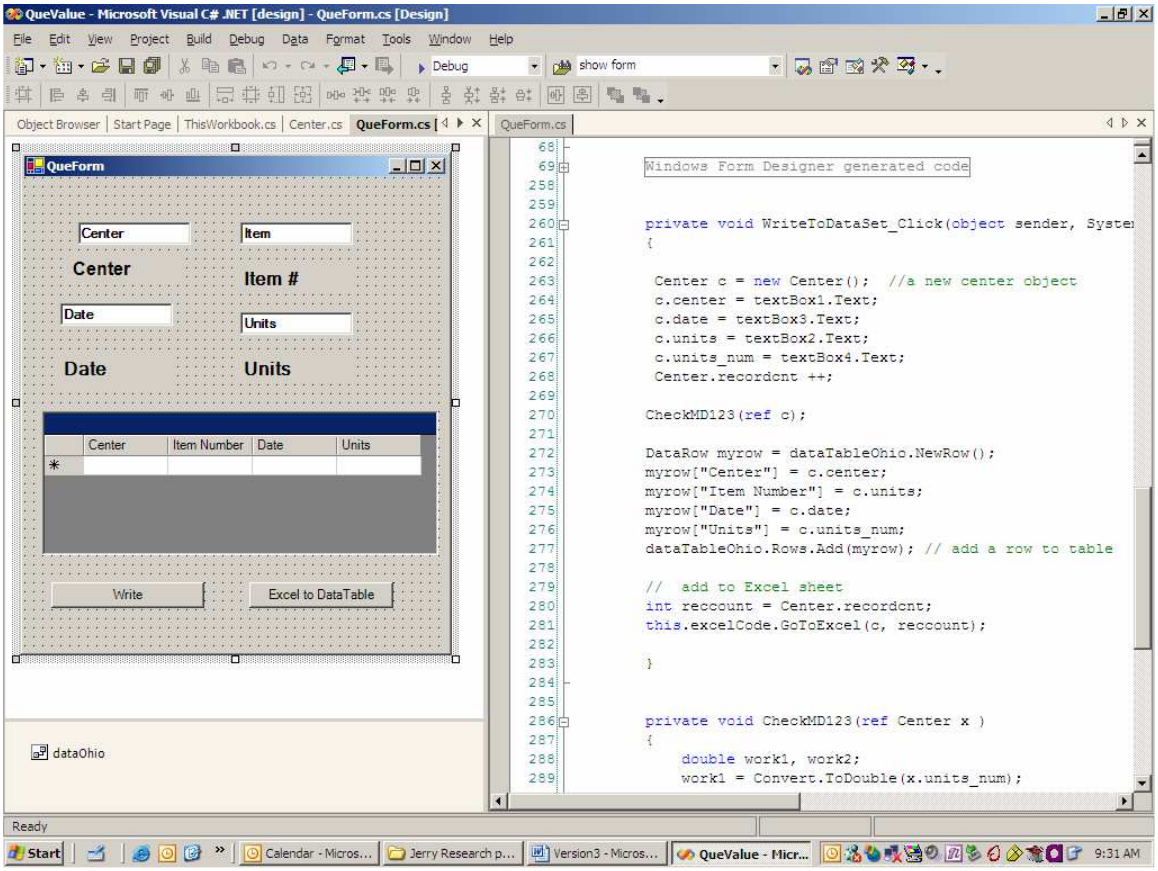

FIGURE 1

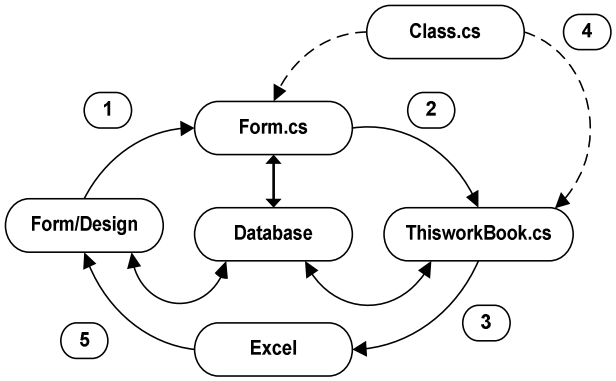

FIGURE 2

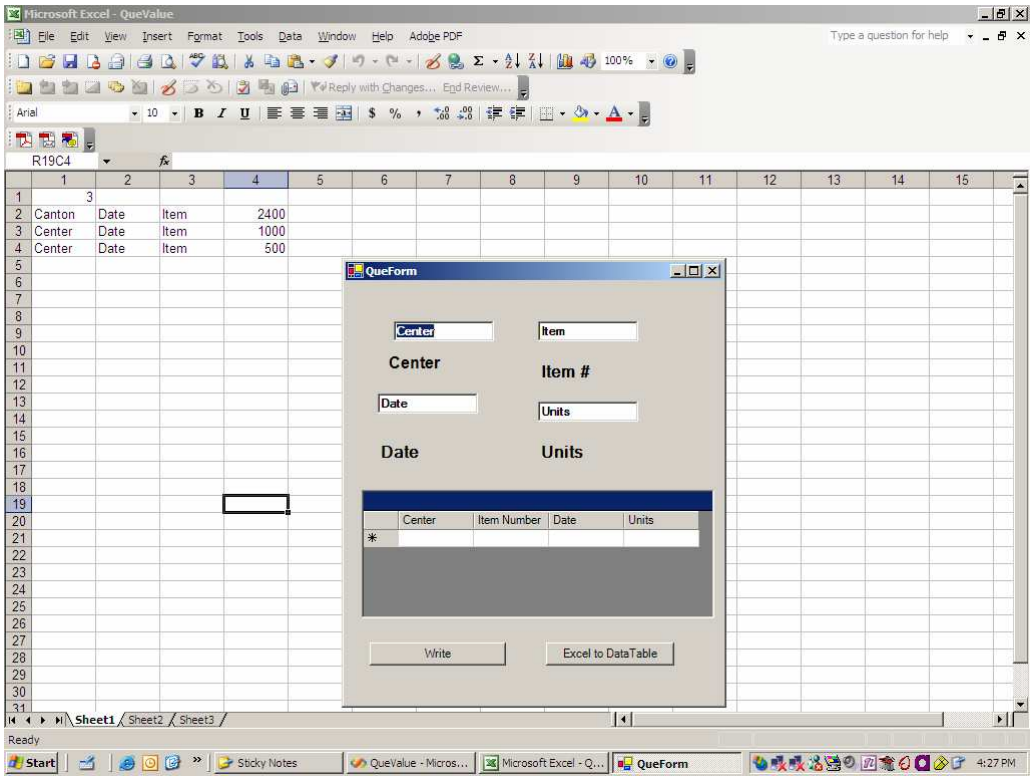

FIGURE 3