View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Clute Institute: Journals

Introducing Programming In The AIS Course: Creating And Coding An Expert System Using Visual Basic.NET

Brian R. Kovar, (E-mail: bkovar@ksu.edu), Kansas State University

ABSTRACT

The primary purpose of this paper is to provide a stand-alone, introductory tutorial exercise that can be used to give AIS students a very basic understanding of Visual Basic.NET. The tutorial demonstrates Visual Basic.NET in the context of developing a simple expert system to evaluate loans, thus giving the student exposure to expert systems concepts and design as well. This paper provides a discussion of the potential reasons for including an introduction to programming in the AIS course, gives more detail about the tutorial and provides guidance for its use in the classroom.

INTRODUCTION

When designing the curriculums for Accounting Information Systems (AIS) programs and AIS courses, a question is often raised. What role, if any, does the skill of computer programming have in the course or program? Individual educators have their own beliefs and opinions regarding the role of computer programming in the AIS curriculum, and within the AIS educational community, those opinions are varied. Practitioners also have varied opinions regarding the role of computer programming in AIS.

While prior studies (Jackson, 1999, Callaghan, Peacock & Savage, 2001) have found that programming and similar technical skills should not be a primary emphasis of AIS programs, other studies have suggested that technical skills, such as programming, should remain in the AIS curriculum because they help students build important competencies. Beard and Smith (2002) state that exposure to computer programming concepts helps students develop and improve problem solving and critical thinking skills. In describing their use of programming in their curriculum, Beard and Smith (2002) noted that students credited their programming exposure to "an increased ability to handle more complex accounting issues and problems." This is further supported by the American Institute of Certified Public Accountant's (AICPA) *Core Competency Framework for Entry into the Accounting Profession*. More specifically, one core competency is to leverage technology to develop and enhance functional competencies. The AICPA (2004) states:

"*Technology is pervasive in the accounting profession. Individuals entering the accounting profession must acquire the necessary skills to use technology tools effectively and efficiently. These technology tools can be used to develop and apply other functional competencies.*"

In addition to helping build important skills, programming knowledge may have more direct relevance for accounting students. As Calderon et al., (2002) point out, the AICPA recognizes both that information technology (IT) is an integral part of accounting and that accounting education should incorporate increasing amounts of IT so that accountants can provide quality information systems services. Calderon et al. further point out that "the lack of computer programming skills may handicap accountants in working effectively as IS auditors." (2002). The

Information Systems Audit and Control Association (ISACA), in its curriculum models developed in 1998, also stresses the need and importance of computer programming skills for information systems auditors.

Calderon et al. (2002) also create the analogy of fundamental programming skills for the AIS graduate being very similar to the need for introductory financial accounting courses in accounting programs. Just as the concepts of introductory financial accounting prepare students for more advanced financial accounting concepts and the remainder of their accounting curriculum, a working knowledge of computer programming lays the foundation and helps establish the thought processes for future work in information systems. Furthermore, Calderon et al. state:

"In short, a working knowledge of programming offers AIS students a strong foundation for developing the types of skills and thought processes needed by future accounting information systems consultants and auditors."

EXPLANATION OF THE TUTORIAL

To serve as a starting point for AIS students to develop a working knowledge of computer programming concepts, Appendix A contains a tutorial, "Creating and Coding an Expert System Using Visual Basic.NET**,"** designed to give students exposure to how computer programs are created using a computer programming language (Visual Basic.NET or VB.NET). In this tutorial, students will create an expert system as they work through a step-by-step tutorial that explains the Visual Basic.NET programming environment and the elements used to create the expert system program, such as IF statements, variables, multiple forms, and other elements of computer programming. Students gain exposure to how computer programs are created as they create the necessary code and user interfaces of an expert system.

In addition to providing an initial starting point to develop some of the programming and IS skills recommended by the AICPA and ISACA, this tutorial also incorporates features recognized as being of value to the accounting profession and accounting education. In the tutorial, while students are exploring and practicing computer programming skills and concepts, they are also building an expert system used to evaluate loan applications, based upon a number of decision-making criteria. As students create the computer code, they also see the inside mechanics ("nuts-and-bolts") of the decision-making rules that make up the expert system. While coding the expert system's IF statements, students also begin to see how expert systems evaluate criteria, with the eventual output being a recommendation regarding that particular loan application. In addition, the Visual Basic.NET language is an example of an "object-oriented/event-driven language" (Zak, 2002), with the student being exposed to object-oriented terminology and concepts such as attributes (characteristics that describe an object), behaviors (that the object is capable of performing), and inheritance (creating one class or object from another class). (Zak, 2002)

In the IFAC's Guideline Number 11, *Information Technology in the Accounting Curriculum*, noteworthy IT trends were noted, including object-oriented programming as a new system development technique and "continuing development of intelligent support systems incorporating expert systems" and other problem solving aids (1998). The ISACA curriculum models (1998) also recognize the importance of exposing students to expert systems and object-oriented programming concepts. In its *Core Competency Framework for Entry into the Accounting Profession*, the AICPA (2004) mentions the "recognition of commonly used information architectures" as a broad business perspective competency and "exploration of new technologies and their application to business and accounting scenarios" as a personal competency of leveraging technology.

USING "CREATING AND CODING AN EXPERT SYSTEM USING VISUAL BASIC.NET" IN THE CLASSROOM

Due to its nature, the "Creating and Coding an Expert System Using Visual Basic.NET" tutorial found in Appendix A can be used as part of an in-class unit where computer programming concepts are presented and discussed and an assignment is made, or it can simply be a stand-alone, self-paced unit that the students are required to complete.

The tutorial provided in Appendix A is a shortened version of the complete tutorial. It is recommended that instructors using the tutorial in class download the full version from http://info.cba.ksu.edu/bkovar/publications/. This version contains more complete instructions and embedded figures. In order to complete the tutorial, students also must be provided with the Visual Basic.NET files, also found in a zipped file on the website listed above.

Students self-report an average time of 3.25 hours to complete the tutorial, ranging from a minimum completion time of 1.5 hours (working straight through with no interruptions), to a maximum completion time of 5.5 hours. The tutorial has been used at both the graduate and undergraduate levels to introduce business students to the basics of computer programming. As written, the tutorial does not contain any assignment exercises, but an instructor can assign the tutorial for students to complete as practice and a learning exercise, with the instructor then using the tutorial as the basis for developing focused exercises that might meet the instructor's own unique needs.

Students have rated the "Creating and Coding an Expert System Using Visual Basic.NET" highly in terms of its value, the amount that they learned, and its ability to help them understand the topic area. Table 1 contains a selected sample of student comments concerning the tutorial.

Table 1
Selected Student Comments Regarding the Creating and Coding an Expert System Using Visual Basic.NET Tutorial

Comment
I found that this exercise gave me a basic understanding of how the Visual Basic programming language works. It also helped me to better understand how an expert system goes about its "thought process" and comes to the conclusion that it does.
Overall, I found the assignment to be very well thought out and explained. It was a very good tutorial for anyone that is unfamiliar with programming and a good introduction to Visual Basic for those that know basic coding concepts.
By completing this tutorial, I developed a better understanding not only for expert systems but for the work involved in making them. It is a lot more complicated than it looks.
This exercise gives students a chance to get involved in the actual development of a program or system, not just its use. I think this gives people more respect and understanding towards computers and software.
I would recommend that you continue to offer this as an exercise because I felt that this exercise really teaches well how conclusions are based on logic. I think that this is something that everyone taking this class would benefit from when dealing with expert systems.
I think that it is a great way for a person who knows nothing about computer programming, like me, to get a taste of what programming is like, without overwhelming them. It is a great introductory experience for those who are just starting their journey into computer programming.

One challenge in using the tutorial can be assuring that the Visual Basic.NET software is available to students. However, many management information systems departments belong to the Microsoft Software Developer Network Academic Alliance (MSDNAA) which provides a variety of software, including Visual Studio.NET which contains VB.NET, to educational departments at a cost of $799 (current cost as of 8/17/04). With this license, the institution can install the software on its own computers and distribute it to students for educational purposes, within the guidelines of the program. For more information, visit the MSDNAA at http://www.msdnaa.net/program/.

REFERENCES

1. American Institute of Certified Public Accountants. 2004. *AICPA Core Competency Framework for Entry into the Accounting Profession-Functional Competencies*, Available online: http://www.aicpa.org/edu/corecomp.htm (Accessed April 13, 2004)
2. Beard, David V. and Smith, Kenneth A. 2002. "A Beginning Programming Course for the CPA," *Proceedings of the 2002 Accounting Information Systems Educator Conference.*
3. Burrows, William E. and Langford, Joseph D. 2000. *Programming Business Applications with Microsoft Visual Basic 6*.0. Boston: Irwin McGraw-Hill.
4. Callaghan, Joseph, Peacock, Eileen and Savage, Arline. 2001. "Feedback On Developing An AIS Curriculum," *The Review of Business Information Systems,* Vol. 5, No. 4, pp. 51-60.

5. Calderon, Thomas G., Cheh, John J. and Chatham, Michael D. 2002. "An Examination of the Current State of Accounting Information Systems Education," *The Review of Business Information Systems,* Vol. 6, No. 2, pp. 29-41.
6. Information Systems Audit and Control Association. 1998. *Model Curricula for Information Systems Auditing at the Undergraduate and Graduate Levels*, Available online: http://www.isaca.org (Accessed April 13, 2004).
7. International Federation of Accountants. December 1995, revised June 1998. *International Guideline No. 11: Information Technology in the Accounting Curriculum*, IFAC.
8. Jackson, William K. 1999. "Assessment of Entry Level Computer Skills by Potential Employers of Accounting Graduates," *Proceedings of the 1999 Accounting Information Systems Educator Conference.*
9. *Webster's Ninth New Collegiate Dictionary*. 1985. Springfield: Merriam-Webster Inc.
10. Zak, Diane. 2002. *Programming with Microsoft Visual Basic.NET*. Boston: Thomson Course Technology.

APPENDIX A

CREATING AND CODING AN EXPERT SYSTEM USING VISUAL BASIC.NET

This exercise is designed to give students exposure to how computer programs are created. In this exercise, students will create an expert system in the Visual Basic.NET (VB.NET) programming environment as they work through a step-by-step tutorial that explains the VB.NET programming environment and the elements used to create the expert system, such as IF statements, variables, multiple forms, and other elements of computer programming. **This activity must be completed in a computer lab or on some other computer that already has Visual Studio/VB.NET installed.**

Webster's dictionary says an expert is "one with the special skill or knowledge representing mastery of a particular subject." When a human expert solves a problem, their knowledge of that subject is used to reason through the problem, with the result being a recommended solution. Expert systems are computer applications that can be used to capture expertise and then make that expertise available for others to use. Expert systems are often used to solve diagnostic (figuring out what is wrong) and prescriptive (figuring out what should be done) types of problems. Examples of diagnostic and prescriptive problems include medical diagnosis, credit and loan evaluation, help desk analysis, and compliance analysis. Although there are many different ways for expert systems to solve a problem or make a recommendation, most expert systems follow some sort of decision-making rules. Complex expert systems are usually created using a specialized type of development environment called expert system shells. However, the rule-based expert system described here can be created using the VB.NET programming language.

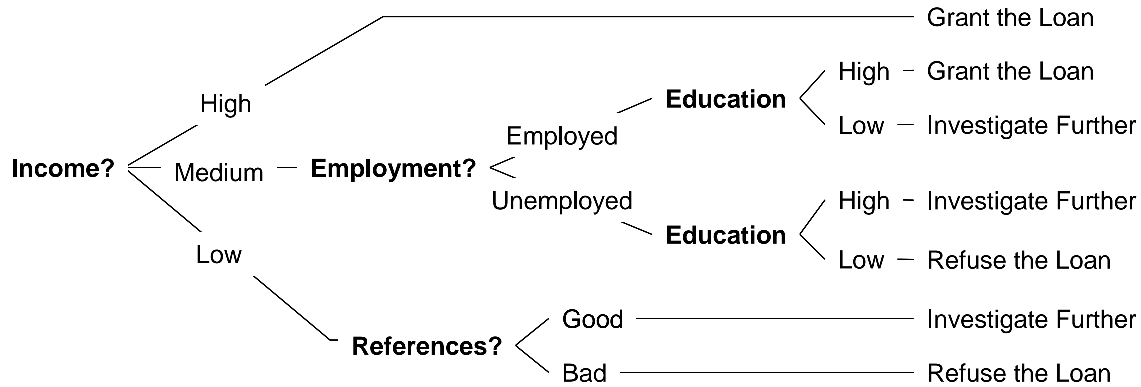
The purpose of this exercise is create an application (an expert system) and to get a better understanding of what expert systems do, how they are set up, and how expert system rules tend to work. At the same time, we are also going to learn some of the concepts and principles of computer programming.

VB.NET is a relatively easy programming language to learn, and VB.NET lets us create and code GUIs, or graphical user interfaces, containing windows, icons and other elements. Icons can be clicked or double clicked, and the computer does something. Computer programs have two primary components:

1. The graphical user interface (what the user sees and works with)
2. The actual program code or set of instructions that tells the computer how to perform a task.

In this exercise, we are going to create a loan evaluation expert system[i]. The decision-making rules can be seen in Figure 1's decision tree. The decision tree diagram is read from left-to-right, with the far right-hand side of the diagram showing the expert system's recommendation for handling that particular loan.

Figure 1
Expert System Decision Tree



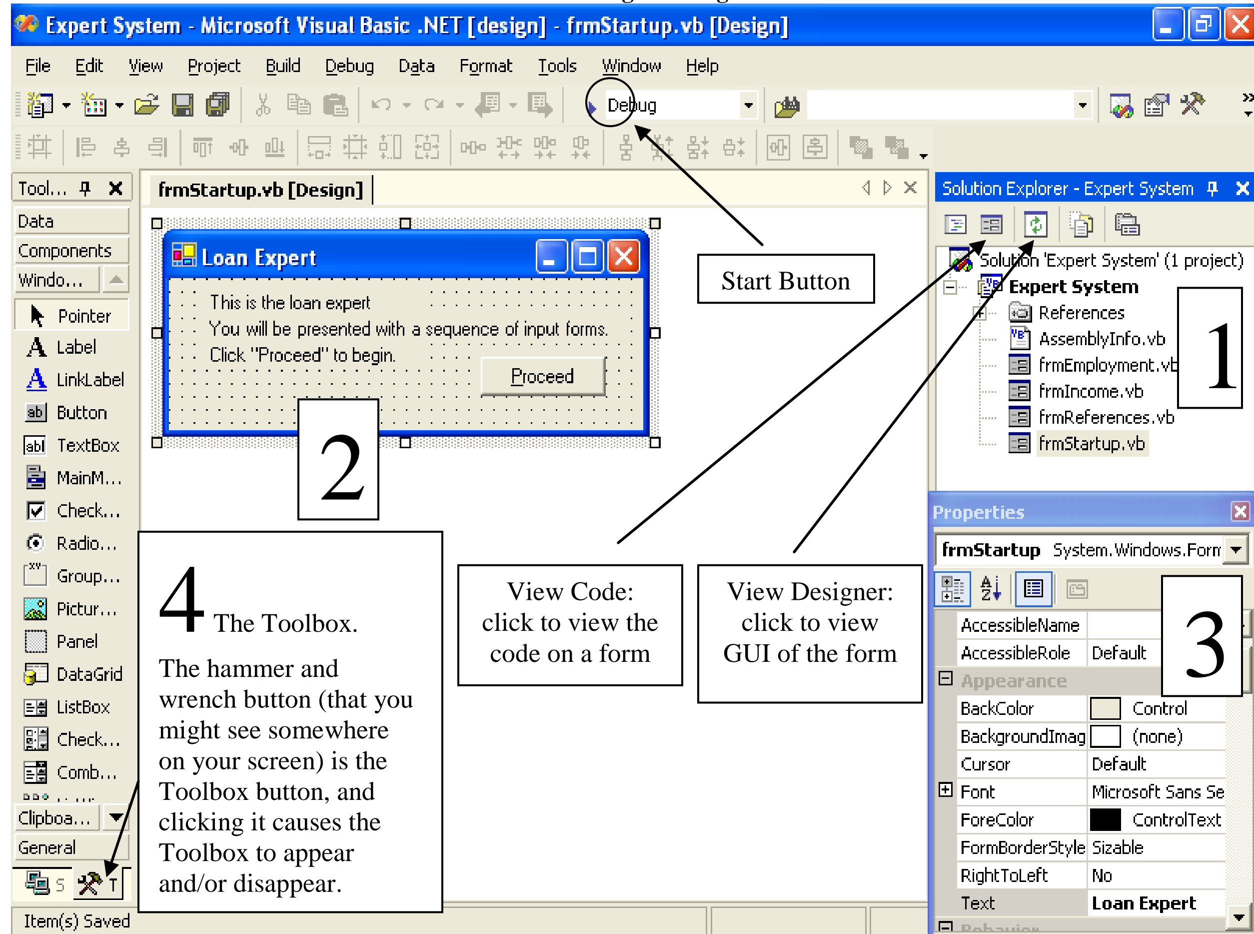
For the expert system to properly work, the user must provide input related to different questions asked by the expert system. Questions asked prior to a recommendation being made include questions about income, employment status, education level attained, and the applicant's references.

It is now time to open VB.NET and begin creating our expert system. Open the folder that contains the expert system files (obtained prior to beginning this activity). Notice the file called **Expert System**, **Visual Studio Solution**. A VB.NET Solution file is a container file that stores all of the projects and files for an application. **Double click the Expert System solution file.** If you ever close down the VB.NET program in the middle of a project, when you wish to resume your work, you will need to double click the solution file. It is always the file that you open, and opening that file provides access to all the other elements that make up an application. Double clicking the solution file allows a programmer to enter the VB.NET programming environment seen in Figure 2. Your screen should look similar to Figure 2, although it is possible that items seen pictured might not be visible, or they may appear in other locations.

Notice the large 1 on the right hand side of the picture in Figure 2. The "1" represents the location of the Solution Explorer Window. (*If you do not see a Solution Explorer Window on your screen, go to the View Menu, and select Solution Explorer*). The Solution Explorer Window displays a listing of all of the items contained within your current solution (Expert System). References and AssemlyInfo are two files that VB.NET created automatically related to your present project. frmEmployment.vb, frmIncome.vb, frmReferences.vb and frmStartup.vb are four container controls that hold other controls (Items found on the GUI such as buttons, labels, textboxes, etc.) and the event procedures (i.e. coding) that tells the application how to respond when certain events occur. Notice that frmStartup.vb is highlighted in the Solution Explorer Window, with the design screen/GUI for the Startup form appearing to the left of the Solution Explorer Window (See large "2" in Figure 2). *If you do not see the form labeled "2" on your screen, then double click frmStartup.vb in the Solution Explorer.*

The large 3 on lower right hand side of Figure 2 identifies the Properties Window. (*If you do not see a Properties Window on your screen, go to the View Menu, and select Properties Window*). Each control, or object, in Visual Basic.NET has a set of characteristics, called attributes or properties, which are assigned to it. The properties which determine an object's appearance and behavior are listed in the Properties Window. Figure 2 shows the properties for the form called frmStartup. If the "Proceed" button was the currently selected item, then its properties would display instead.

Figure 2
Visual Basic.NET Programming Environment



The large 4 in Figure 2 appears to the right of the Toolbox. The Toolbox is typically found on the left-hand side of the screen, but it can be located elsewhere. (*If you do not see a Toolbox on your screen, go to the View Menu, and select Toolbox. Clicking on Figure 2's hammer and wrench button also makes the Toolbox appear and disappear.*) The Toolbox contains a collection of tools that you can use when designing the GUI for an application. You use the Toolbox to place controls on your form.

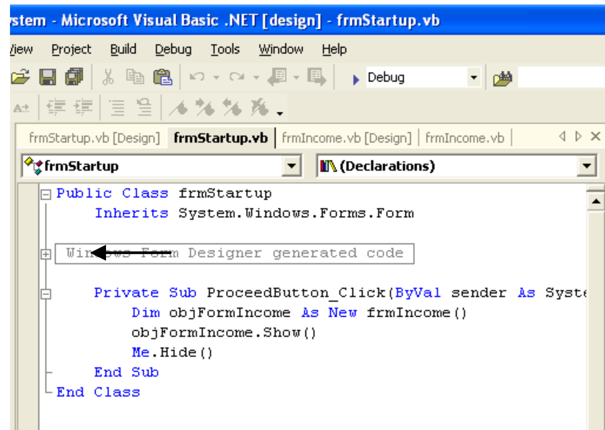
The controls that we will use in this exercise include:

- The **label**: used to simply display text.
- The **button**: used to initiate processing tasks (the user clicks on the button).
- The **radio button/option button**, which allows the user to select only one item from a group.

In Figure 2, notice the arrow leaving the textbox where it says: "View Code: click to view the code on a form." Click the button that arrow points to, and you should be able to see the code currently on that form. Notice that the next button (on the right of the View Code button in Figure 2) is the View Designer button. You can click on that to see the GUI of the form. **Click on both of those buttons to see what they do**. When you are done clicking those buttons, you should go back to the GUI, as seen in Figure 2.

Take a look at the form and the user interface. The words "*This is the loan expert system. You will be presented with a sequence of input forms. Click "Proceed" to begin*" are displayed in a label (*without borders, making it difficult to see*). The form also contains a button labeled "*Proceed*." When the user clicks the button, its event procedure will begin execution. Let's take a look at the code for that button/event procedure. **Please click the View Code button** (seen in Figure 2) and you should see the Code Window (part of which is seen in Figure 3). The Code Window shows the code that is on the form that you are currently working with. The code above the box that says *Windows Form Designer generated code* is created automatically by VB.NET. **Never delete code in this section.** Any code you create will appear below this box. The code below the box on this form is the event procedure that executes when the "Proceed" button is clicked.

Figure 3
The Code Window



Private Sub and End Sub are words automatically added to the event procedure by VB.NET. Words that appear in blue font on your screen, such as these words, are considered to be keywords by VB.NET. Notice the three lines that are indented (in between the lines beginning with Private Sub and End Sub). Programmers typically indent their coding to make the code easier to read, and we will do that also.

In the first indented line, the blue keyword **Dim** appears. This is called a Dim statement (Dimension statement). Dim statements are used to create variables, memory locations that are used to store data. They are called variables because the data stored in these memory locations can change (or vary) as the program runs. The **obj** following the keyword Dim indicates the type of variable created is an object variable. Object data types, which take up the greatest amount of memory space, can be used to store anything. When a variable's data type is chosen, one must ask how that variable is going to be used (what type of data will it store?). The string data type is usually assigned when you want to store a sequence of characters as text (letters and numbers not used in calculations). If a variable is to be used in a calculation, one of the numeric data types (integer, decimal, single, double, short, long) should be chosen.

The object variable created by our Dim statement is **objFormIncome.** The **As New** coding indicates we want to create a new object, and **As New frmIncome ( )** indicates we want to pattern this new object based upon our existing form file of frmIncome (basically, a copy of the form file frmIncome will be stored in that object variable). Therefore, the line **Dim objFormIncome As New frmIncome ( )** says to create a new object variable and store a copy of the form called frmIncome in that variable.

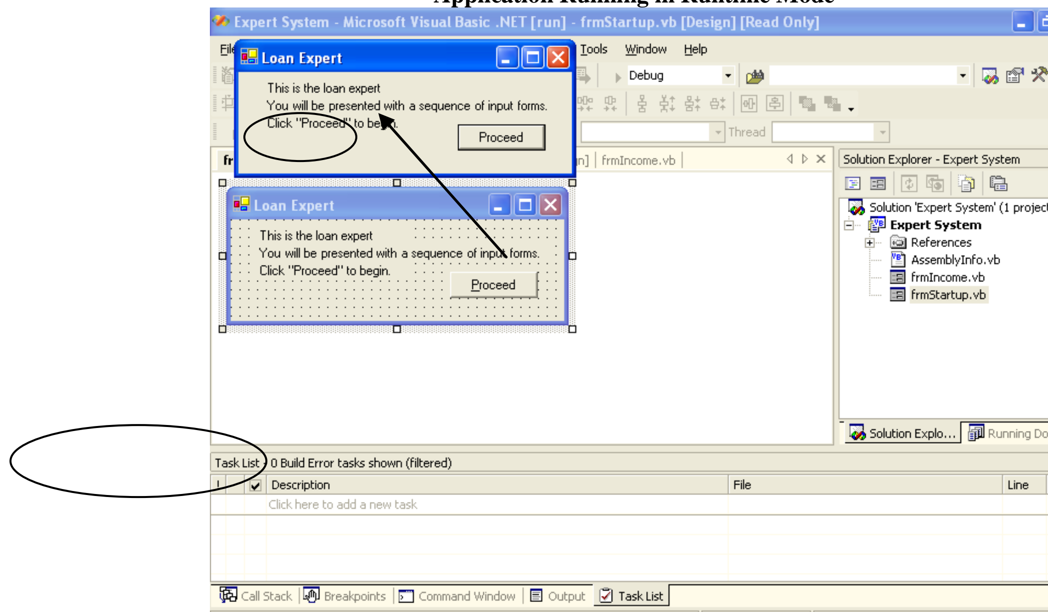
The line **objFormIncome.Show ( )** asks the new object variable (objFormIncome) to execute its Show method, which will result in the object variable (its current contents) being loaded into memory and then displayed. The line **Me.Hide ( )** tells VB.NET to keep the current form active (frmStartUp), but it will be hidden (invisible) to the user (**Me** is used to refer to the current form). In essence, when the "Proceed" button is clicked, VB.NET creates

a new object, representing the Income form (frmIncome), displays the new Income form, and then hides the prior form the user was seeing (frmStartup). Now, after looking at the coding on the first form, it is time to run our application (as it currently exists) and see what it does.

Switch back to the design screen. To start the application while in the design screen, click the Start button found on the toolbar (labeled in Figure 2). **Go ahead and start/run the application.** Running the application changes your screen as VB.NET compiles the program code (checks for errors). If no compile/syntax errors are found, VB.NET enters runtime mode, where the application actually runs (but you also see the design screen).

The box highlighted with an arrow in Figure 4 is the application running in runtime mode. Everything else is part of the design screen. You might also see a new window opening at the bottom of the screen. That window is the Task List and is circled in Figure 4. Notice it says "0 Build Error tasks shown." That means that you did not make a syntax or typing error, and your application will work as it is coded. If you ever see **Build: 1 succeeded, 0 failed, 0 skipped** in a window labeled output, then you are also in good shape. If you ever make a syntax error or any other error, then the application will not run and you will see a message box indicating that you have build errors, which must be fixed before the application will run. However, since we have not typed anything in yet, we should not see any error messages.

Figure 4
Application Running in Runtime Mode



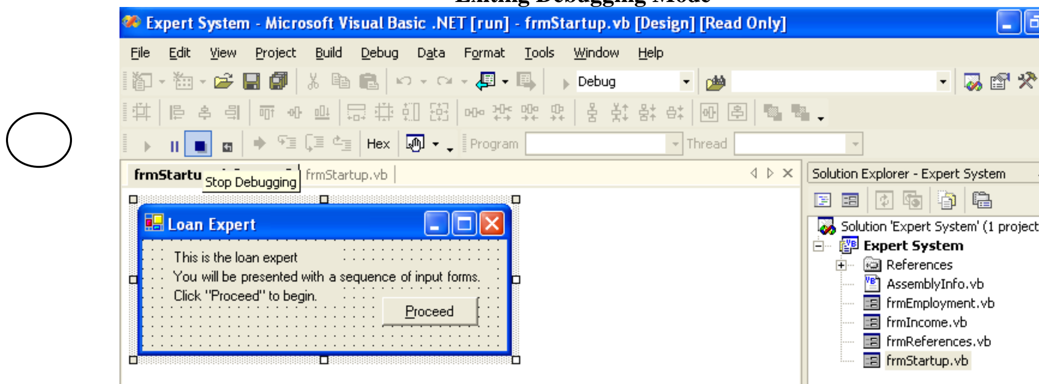
VB.NET is an "event-driven" language because when the user does something (clicks the Proceed button), its associated event procedure (code seen earlier) begins to execute. When the user clicks on the Proceed button, the next form that you want to see (Income) is loaded into memory, displayed on the screen, and the initial form (StartUp) disappears from view, as the application asks about the applicant's income (refer to far-left of Figure 1). The expert system user would next characterize the applicant's income as high, medium, or low. After selecting the appropriate option, the next form is displayed.

We now want to work with our currently running application, (the window highlighted by an arrow in Figure 4). **Click on the Proceed button** (circled in Figure 4). The Income form should appear. **Click the Evaluate button** on the Income form**.** Nothing happens when you click on this button. Why??

Objects in VB.NET only respond to events they have been programmed to respond to, and we have not yet created the code for when the Evaluate button is clicked. The programmer determines which user events the application needs to respond to, such as clicking (most common event) or typing in a certain location, and then the appropriate code is created so that the application performs as desired.

Since nothing is happening when we click on the Evaluate button, we need to stop running the application. **To stop running the application, click the X** (commonly used to close windows programs) **in the upper right hand corner of the application window**. At this point, the application should have stopped running, but you are still in the "debug" mode (your screen should look like Figure 5), which you will need to stop. To stop debugging, click the **Stop Debugging** button (circled in Figure 5). Go ahead and click the **Stop Debugging** button, and you should now be back on the Design screen.

Figure 5
Exiting Debugging Mode



You might see a new window toward the bottom of your screen. That window is called the output window, and at this point, you can close it if you wish. Now, you should now be back in the design screen. As stated earlier, we have not yet created the event procedure for the Evaluate button. To create the event procedure for the Income form's Evaluate button, you first need to double click frmIncome.vb in the Solution Explorer Window. Then, while in design mode, you will need to double click the Evaluate button. **Please double click the Evaluate button.** You should see the code window, as seen in Figure 6.

VB.NET has already started creating the event procedure (also called a sub-routine) for when the user clicks on the Evaluate button. **Private Sub** and **End Sub** are keywords automatically included by VB.NET. **Private Sub** marks the beginning of the event procedure, **End Sub** marks the end of the event procedure, and **EvaluateButton_Click** is the name of the event procedure that you are now working on.

Programmers usually indent lines within event procedures, which the code editor has done for you. Type in the following code: **Dim objFormEmployment As**

As soon as you type in "As" and press the spacebar, the VB IntelliSense list appears; showing possibilities for what might come next in your coding. IntelliSense helps the program developer by checking spelling and providing suggestions on what to include in a statement. Continue on that same line, typing in the following (and notice how the IntelliSense list changes with each letter that you type) **New frmE**

As the capital E is typed, notice IntelliSense has highlighted frmEmployment (see Figure 7). frmEmployment is the next item that you want to appear in your code. While it is acceptable to type in the rest of the object's name, you can also use IntelliSense to add that object to your code by simply pressing the spacebar

while the appropriate item (frmEmployment) is highlighted in the IntelliSense listing, and frmEmployment is automatically added to your code.

Figure 6
The Code Window

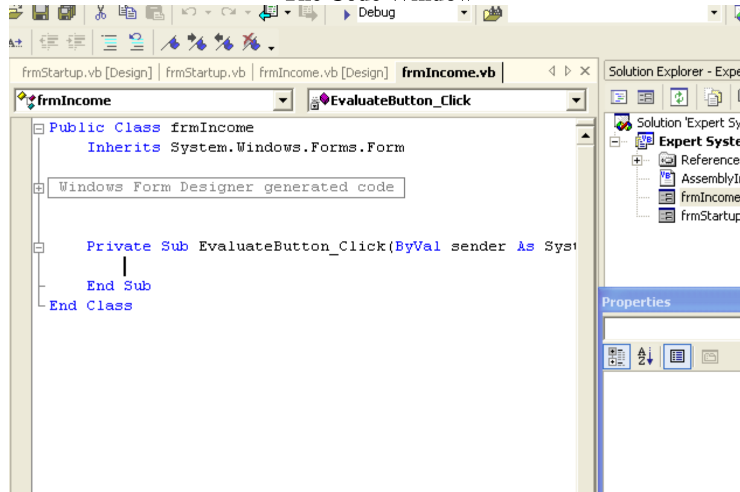
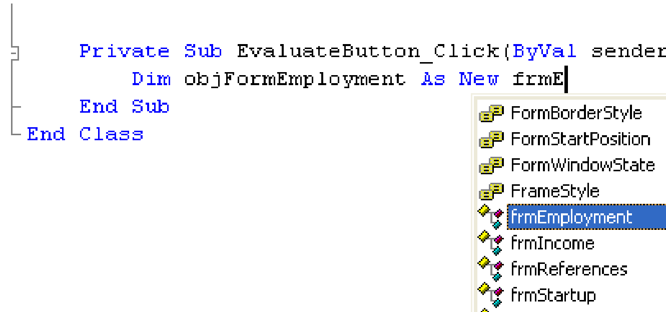


Figure 7
IntelliSense highlights the Desired Item



Since the first line of your code is now complete, **go ahead and press the Enter key** to move to the next line. You should now see the following line of code that you just typed:

Dim objFormEmployment As New frmEmployment()

Notice that the line ends with (). Although you did not type in the brackets, you can see that the VB.NET code editor automatically finished the line for you. Now, what does this line of code do? The line **Dim objFormEmployment As New frmEmployment ( )** says to create a new variable (of data type object) and to store a copy of the form called frmEmployment in that object variable.

Type in the next line of code, using IntelliSense and the spacebar to have words appear in your code:

Dim objFormReferences As New frmReferences() This line says to create a new object variable and to store a copy of frmReferences in that variable.

The Selection Statement

In our daily life, we have to make decisions. If one condition occurs, then I will take one course of action. If another condition occurs, then I will do something different. Just as we select an action to take based on certain criteria, computer programs must also select an appropriate course of action from several alternatives, depending on certain conditions in the data or depending on user input. For instance, if an accounts payable program encounters an overpayment, the program might issue a credit to that customer's account, or if the amount is small, it might produce a refund check otherwise. If….Then…..Else statements are one of the most common forms of selection statements. Let's go back to our Expert System decision tree found in Figure 1 and refer to it during the following discussion.

The user (on frmIncome) characterizes the applicant's income as high, medium, or low. If the applicant's income is high, then the expert system's recommendation should be to grant the loan. If the applicant's income is medium, then the expert system should ask a question about the applicant's employment (by displaying frmEmployment), which may result in additional questions. If the applicant's income is low, then the expert system should ask a question about the applicant's references (by displaying frmReferences) before making a recommendation. Three different courses of action can be taken, depending on the applicant's income. Before coding the next part of our expert system, we first need to take a look at three other features used by programmers: Message Boxes, Comments, and Radio/Option Buttons. Those three elements will appear inside our upcoming selection statements.

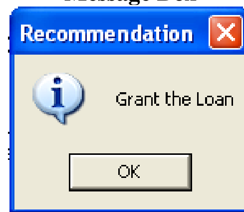
Message Boxes, Comments and Radio/Option Buttons

The Message Box statement causes a new window to open on the screen, displaying a message that stays on the screen until the user has acknowledged it by clicking on one of its buttons. The syntax of a message box statement is **MsgBox ("**The message to display", Buttons, "A title for the message box")

The message box syntax contains 2 commas. Before and after the commas are arguments. The first argument is the message to be displayed to the user, which is surrounded by quote marks. The second argument displays information regarding what types of icons and buttons that should be displayed within the message box. The third argument represents the text that will appear in the message box's title bar.

Typing in the code of: MsgBox("Grant the Loan", MsgBoxStyle.Information, "Recommendation") creates the message box that appears in Figure 8.

Figure 8
Message Box



The programmer uses comments as program documentation since they are not considered "executable" and have no effect when the program runs. The purpose of comments is to make the program easier to read so you can tell what the programmer is trying to do with a block of code. Comments typically include information identifying the name of the programmer, the purpose of the program, or the purpose of a line of code. The apostrophe signals the beginning of a comment. Comments can be a separate line of code or they can be at the end of a line of code, as seen in the examples that follow:

This is an example of a comment that is a separate line of code.

objFormIncome.Show ( )' **This is an example of a comment at the end of a line of code.**

Radio buttons / option buttons are used to ensure that the user will select **only one option** from a group of options. Radio buttons are typically used in conjunction with selection statements.

Now, it is time to begin coding again. Make sure your cursor is on new line, underneath the "D" (still indented) of the word "Dim." Don't forget that you can use IntelliSense and the spacebar to finish completing parts of your code. If you see a space appearing after a word that you added using IntelliSense, and if that space is not supposed to be there, simply use your backspace key to backup and remove that unwanted space, and then continue on with your typing.

You must be VERY EXACT with your typing. Typing errors (also known as coding errors) will cause your program not to work properly. IF YOU EVER SEE A LINE OF CODE UNDERLINED with a blue or green jagged line, THAT MEANS THAT YOU HAVE MADE A TYPING OR CODING ERROR that you will need to fix before your program works correctly.

Now type:

If Me.HighRadioButton.Checked = True Then (pressing Enter results in your cursor moving down to the next line and making another indent. You should also see the words End If appearing on the next line below your cursor. The words End If are VB keywords that signal the end of an IF statement).

Now type:

MsgBox("Grant the Loan", MsgBoxStyle.Information, "Recommendation") (when you type in these first two lines of code, you are telling the application that if the user has clicked/checked the radio button indicating that the applicant's income is characterized as high, then you want a message box to appear, telling the user that the applicant's loan request should be granted. The next few lines of code that you type will tell the application what to do if the user has clicked the medium or low radio buttons).

Press Enter to end the current line. Then, press your backspace key so that the cursor backs up and lines up with the "I" of the If and "E" of the End If. Then type:

ElseIf Me.MediumRadioButton.Checked = True Then

(Press Enter, make sure that you are indented, and then type):

objFormEmployment.Show() (these two lines of code tell the application that if the user has clicked the radio button indicating that the applicant's income is characterized as medium, then you want to have the Employment form appear so that an employment status question can be asked. The remaining code will tell the application what to do if the user has clicked the low radio button).

Make sure that your cursor is on a new line. Press the backspace key so that the cursor backs up and lines up with the "E" of the ElseIf and "E" of the End If. Then type:

Else 'income low (then press Enter, make sure that you are indented, and then type): **objFormReferences.Show**()(these two lines of code tell the application that if the user has clicked the radio button indicating that the applicant's income is characterized as low, then you want to have the References form appear so that a question can be asked about the applicant's references).

Now, with our IF statement finished, we need to add one more line to this event procedure. Place your cursor right after the F (of the End If). Press your Enter key, and then type in the following:

Me.Hide() **Me.Hide** ( ) tells VB.NET to keep the current form active (frmIncome), but it will be invisible to the user.

We have now finished coding the event procedure used when the user clicks on the evaluate button on the Income form. Next, place your cursor in-between the Windows Form Designer generated code and the event procedure that you just created. Type in the comment that follows. (*including the apostrophe.*)

’Programmed by (replace the parenthesis and these words by your own first name and last name)

We have finished coding the Income form. Make sure that your code matches the code seen in Figure 9. Then, click the **View Designer button** so that you can see the actual form/GUI (and not the code).

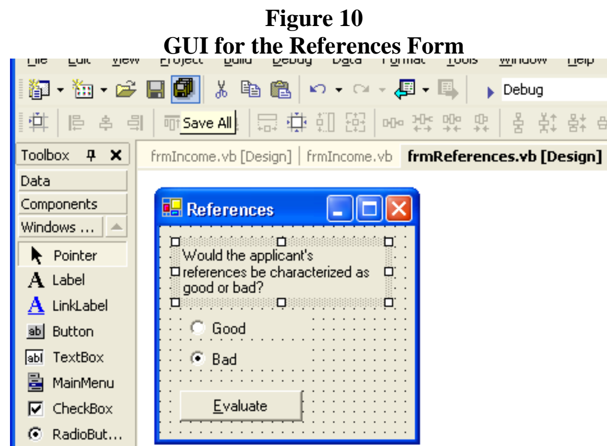
Figure 9
Completed Code for the Income Form

```
Public Class frmIncome
    Inherits System.Windows.Forms.Form

 Windows Form Designer generated code 

    'Programmed by (your own name goes here)

    Private Sub EvaluateButton_Click(ByVal sender As System.Object, ByVal e As Sys
        Dim objFormEmployment As New frmEmployment()
        Dim objFormReferences As New frmReferences()
        If Me.HighRadioButton.Checked = True Then
            MsgBox("Grant the Loan", MsgBoxStyle.Information, "Recommendation")
        ElseIf Me.MediumRadioButton.Checked = True Then
            objFormEmployment.Show()
        Else 'income low
            objFormReferences.Show()
        End If
        Me.Hide()
    End Sub
End Class
```

One unique feature of Visual Basic is that it has a **Save All** button, in addition to the traditional Save button. Clicking the Save All button saves your work (code and changes to the GUI that you might make) on ALL forms that make up your application. Clicking the traditional Save button just saves the work on the current form that you are working on. The Save All is located to the immediate right of the traditional Save button. **Please click the SAVE ALL button now.** Using the Solution Explorer on the right-hand side of your screen, double click frmReferences.vb, and you now should see the GUI for the references form on your screen, as shown in Figure 10.

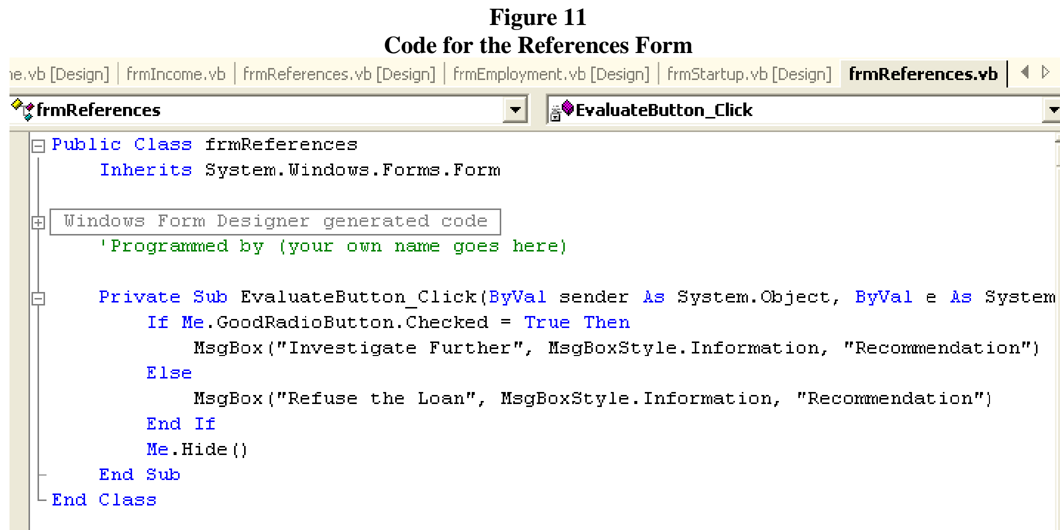
Referring to the Figure 1 decision tree, the user (on frmIncome) characterizes the applicant’s income as high, medium, or low. References are checked only if the applicant’s income is characterized as low. If the applicant’s references are characterized as being good, then further investigation should be done before a recommendation is made; however, if the applicant’s references are bad (along with the low income), then the expert system should make a recommendation of refusing the loan.

Figure 10
GUI for the References Form



In an event-driven application, the application only responds to user events that have been coded, and the Evaluate button on the References form does not have an event procedure written for it yet. In order to create the needed event procedure, **double click the Evaluate button**, and you should now be in the code window for the References form. Now, type in the code for the event procedure Private Sub EvaluateButton_Click, as seen in Figure 11. After coding this event procedure, please type the comment code that appears prior to Private Sub EvaluateButton_Click.

Figure 11
Code for the References Form



The coding on frmReferences says the following: If the Good radio button is checked, then the application should display a message box recommending that the user investigate the applicant more before deciding whether to grant or refuse the loan (remember that for this form to even be displayed, the user had to indicate that the applicant had a low level of income). If the Bad radio button is checked, then the application should display a message box recommending that the loan should be refused. After the user clicks the “Ok” button for either message box, then the current form should be hidden from view.

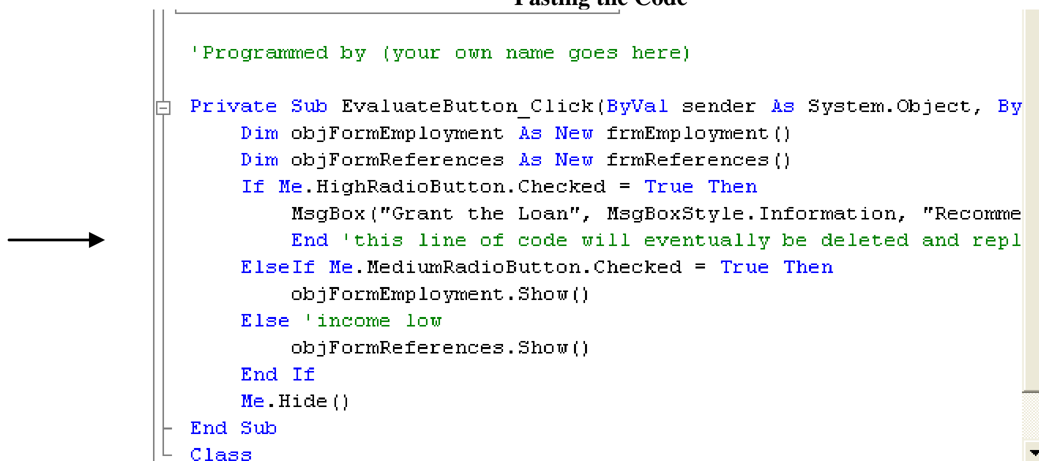
Now, we will add one more line of code to this form. Place your cursor after **Me.Hide().** Press Enter, and your cursor should line up right underneath the M of Me.Hide. Then, type the following line of code

End 'this line of code will eventually be deleted and replaced with something else

This is a temporary line that will eventually be replaced, but we want the **End** command to appear so that we can have the application stop itself automatically once this event procedure is completed.

Now, copy the entire line of code that you just typed. Move back to the code window for frmIncome by using one of the two following methods: 1) In the Solution Explorer, click on frmIncome.vb, and then click on the View Code button or 2) Click on the frmIncome.vb tab (found beneath the top button bars). Once you can see the code window for frmIncome, you then need to paste the line of code that you just copied into the first part of the IF statement (the branch for HighRadio.Checked = True), as shown by the arrow in Figure 12. Now, click the **Save All** button to save all of the changes that you have made to all of the forms (and their code).

Figure 12
Pasting the Code



Next, we will run our application, testing out selected branches from our decision tree diagram. **To avoid having problems,** please conduct the tests exactly as specified (since we are only going to test the completed branches of our application). First, we need to start the application. The start button is labeled in Figure 2. **Click the start button**, and if there are no coding errors, the application should start to run, and you should see the Startup form. **Click the Proceed button.** Next, you should see the Income form. **Click the high option, and then click the Evaluate button.** A message box saying "Grant the Loan" should appear. **Click ok,** and you should exit the application (since the end command tells the application to stop running.)

For our 2nd test, **click the start button.** The Startup form should appear. **Click the Proceed button**. The Income form should appear. **Click the low option, and then click the Evaluate button.** Instead of a message box like we saw in the first test, you should instead see the References form. **Select the Good option** and **then click on the Evaluate button.** The Investigate Further message box should appear. **Click ok,** and you should exit the application.

For our 3rd test, **click the start button.** The Startup form should appear. **Click the Proceed button**. The Income form should appear. **Click the low option, and then click the Evaluate button.** The References form should appear. **Select the Bad option** and **then click on the Evaluate button.** The Refuse Loan message box should appear. **Click ok,** and you should exit the application.

If your tests went well, then everything should have worked and no errors were detected (and no errors appeared on your screen). If there were errors, then your application probably quit running, and you will need to go

back to your code to identify and correct the problem. If errors were made, you can probably see the error and its location in the Output window, which appears at the bottom of your screen.

Up to this point in our exercise, we have coded both the high and low income paths (as seen in Figure 1). Before coding the medium income path, we need to look at the role and scope of variables in a program.

Variables, Variable Scope and Creating the Code Module

Recall that variables are memory locations that we use to store data. When creating variables, a programmer must think about the scope of the variable, which is the set of all code that can refer to, or use, the variable. A variable's scope is determined by where and how the variable is declared. There are three levels of variable scope. Going from smaller scope to larger levels of scope, these levels of scope are called procedural-level (local) scope, module-level scope, and global scope.

Any variable that is declared inside an event procedure has procedural-level scope. Variables that have procedural-level scope are said to be local to the procedure in which they are declared. That means that a variable that is declared within an event procedure is visible only to that event procedure. Other event procedures on that form, or within that project, are unable to "see" or use a procedural-level variable. Because we can create variables (with procedural-level scope) that other event procedures are unable to use, that means that you can have a variable called Total in event procedure A and another variable called Total in event procedure B. Even though both variables have the same name, they can hold entirely different values, simply because their scope is different. Event procedure A can change its variable called Total, and event procedure B can change its Total variable, all without affecting the Total variable stored by the other event procedure. Although both variables have the same name, they are actually two different variables, representing two different memory locations. Dim statements are used to create procedural-level variables. All of the variables that we have created in our expert system (up to this point) have been procedural-level variables. Procedural-level variables are created within private subroutines, denoted by the keywords **Private Sub.** Private subroutines and the variables and other elements created within them are used only by individual controls (such a button) and cannot be shared by other controls or elements of a VB.NET program.

Module-level variables are also declared using a Dim statement. However, instead of placing the Dim statement inside an event procedure, the Dim statement is placed outside an event procedure in the general declarations section (see Figure 13). The general declarations section is the area between the Windows Form Designer generated code box and the first event procedure on the form. Any event procedure on the form can access module-level variables. Figure 13 illustrates local procedural-level variables, as well as module-level variables, which are placed in the general declarations section.

As you have already seen in this project, a computer application may contain more than one form. Is it possible to share variables between different forms? Obviously, procedural-level variables cannot be shared between forms since a variable declared within one event procedure can't be seen or used by other event procedures. Module-level variables cannot be shared between different forms either. Each form has its own general declarations section, and variables declared within a form's general declarations section belong only to that form. However, there is a third type of variable scope, global scope.

Variables shared across all forms have global scope, and those variables are called global variables. Global variables are not declared on any of the forms within an application. Instead, global variables are declared in a repository called a code module, which is shared with all of the forms. A unique feature of the code module is that it does not have a user interface (which means it does not contain any controls and it does not have any event procedures), which also means that the user never sees it. The only individual who sees a code module is the programmer and the code module has its own code window that contains the variables being shared between different forms. While procedural-level and module-level variables are declared using the Dim statement, global variables (stored in the code module) are declared using a Public statement. The Public statement is identical to the Dim statement except it uses the keyword Public instead of Dim. Now, since we have finished our more in-depth

look at variables and variable scope, it is now time to examine the medium income path of our expert system decision tree.

Figure 13
Illustration of Local and Module Level Variables

```
frmEmployment.vb [Design] | Form1.vb [Design]* | Form1.vb*
Form1                                 (Declarations)

Windows Form Designer generated code
    'Placing a Dim statement here (in the general declarations
    'section, which is outside of an event procedure)creates a
    'module-level variable that can be used by all event
    'procedures on this form.


    Private Sub Button1_Click(ByVal sender As System.Object, ByVal
        'Placing a Dim statement here creates a local variable
        'that can only be used by this event procedure
        '(Button1_Click.  No other event procedure can see or use
        ' this variable.
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal
        'Placing a Dim statement here creates a local variable
        'that can only be used by this event procedure
        '(Button2_Click.  No other event procedure can see or use
        ' this variable.
    End Sub
```

As seen in Figure 1, if the applicant's income is characterized as being medium, then the Employment form is displayed. On the Employment form, the user indicates whether the applicant is employed or unemployed. Regardless of their employment status, the user is next asked to characterize the applicant's level of education (on frmEducation). If the applicant is employed and has a high level of education, the expert system recommends granting the loan. If the applicant is employed and has a low level of education, or if the applicant is unemployed, but has a high level of education, then the expert system recommends investigating further. If the applicant is unemployed and has a low level of education, the expert system recommends refusing the loan. As seen in the expert system decision tree, the response to the employment question (employed vs unemployed) leads to four possible paths/recommendations. The employment question response is used on both the employment form and on the education form in deciding what recommendation to make. We need something in our program to capture the employment response so that it can be used on more than one form in our expert system application. That capture mechanism will be global variables, which can be shared between all forms. Therefore, the next thing that needs to be created and added to our loan expert system application is a code module.

From your current position in VB.NET, go to the **File Menu**, and click **Add New Item.** Click once on **Module**, and then click **Open.** A code module called Module1.vb should appear in the Solution Explorer window, and the code module's code window should appear on the screen. Type the lines of code that you see in Figure 14, including the comment code (5 lines total are typed). After creating the code module, click the **Save All** button to save your work on all forms and the code module.

Figure 14
Code Module Code

```
Module1                                    (Declarations)
Module Module1
    Public strEmployed As String
    Public strUnemployed As String
    'These two global variables are used by the
    'employment and references forms.
    'Created by:(Your first and last name goes here)
End Module
```

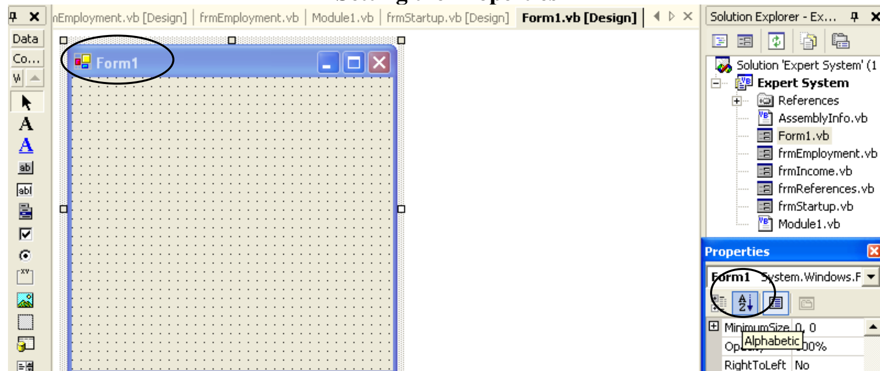
Adding a new form to a VB.NET application

According to Figure 1, after the applicant's employment status is indicated, the application should display the Education form. However, our application does not have an Education form. From your current position in VB.NET, go to the **File Menu**, and click **Add New Item.** Click once on **Windows Form**, and click **Open**. A new form (Form1.vb) should appear on your screen (in design view) and in the Solution Explorer window listing. Up to this point, we have not worked with the Properties window (usually on the lower right-hand side of the screen). Since we are making a completely new form, we will specify some of the properties that have been previously given to us in the files that accompanied this exercise.

Click once on Form1.vb **in the Solution Explorer Window.** Look at the **Properties window**. You should see **five, and only five properties**. For the File Name property, highlight **Form1** (and not the .vb). Type in **frmEducation** so that the new File Name will display as **frmEducation.vb** Press the Enter key, and you should see **frmEducation.vb** appearing in the Solution Explorer window (instead of the previous Form1.vb). You have just named the **Form File**, just like you would name a spreadsheet or word processing document.

Next, click inside the form object itself (in the gray doted area). Although we just named the **file**, we must also name the **form control**. When you click in the gray doted area, you should see that the object in the Properties window changes to **Form1**, and a longer listing of properties now appears. Click the A/Z button (circled in Figure 15) to arrange the form's field properties in ascending/alphabetical order.

Figure 15
Setting the Properties



Use the vertical scroll bar (located to the right of the Properties window) to scroll to the top of the alphabetical order listing of the form's properties. Above the properties that begin with "A", you will find the **(Name)** property. Highlight the existing name (probably **Form1**). Type in **frmEducation, and p**ress the Enter key

to change the name of the **form control** to frmEducation. Now, scroll through the listing of properties until you find the Text property. Notice that the form control currently displays Form1 in the title bar of the form *(circled in Figure 15)*. That is the same wording displayed in the text property. Change the form's text property to **Education**, and then press the Enter key. Education should now appear in the form's title bar. Next, scroll through the alphabetical listing of the properties until you find the Size property. Change the Size property to **208, 184**. Press Enter to change the form's size.

On the left-hand side of Figure 2 is the Toolbox, which is used to add controls (radio buttons, labels, buttons, etc.) to your form. A label is used to display text to the user. Click the Label button. Then, move the cursor onto the form, and you should see your cursor changing into a plus sign and an A. Position the cursor toward the top left-hand side of the gray-dotted section of the form. Hold down your left mouse button, and while holding the left mouse button down, drag the mouse down and to the right and when you release the left mouse button, a box should appear. Then, click somewhere else on the form. Do not worry about the size or position of the label. We will set those properties next.

Click on the label, and the label should now be highlighted. The Properties window should now display Label1. Change the following label properties:

- Size property: change to **128, 48**
- Text property: change to **Is the applicant's level of education characterized as high or low?**
- Location property: change to **16, 8**

Next, you need to add radio buttons to the Education form. Radio buttons allow the user to select only one item from a group. You will begin with the High radio button. Click the RadioButton button, located in the Toolbox. After clicking the Radio button, move the cursor onto the form. Position the cursor under the label and draw a box to represent the radio button (using the same technique that you used to create the label), and a radio button should appear. Click on the radio button to highlight it. The Properties window should now display RadioButton1. Change the following radio button properties:

- (Name) property: change to **HighRadioButton**
- Location property: change to **16, 64**
- Size property: **96, 16**
- Text property: change to **High**

After you create the High radio button, you next need to make the Low radio button. Go ahead and draw it on your form. The Low radio button should have the following properties:

- (Name) property: change to **LowRadioButton**
- Location property: change to **16, 88**
- Size property: **96, 16**
- Text property: change to **Low**
- Checked property: change from False to **True**.

Changing the checked property from False to True causes the Low radio button to be automatically selected by default. When using radio buttons, programmers commonly have a default option selected when the user must choose only one option from a list of options. If a default option is not created, and if the user does not select an option before initiating processing (when an option is expected to be selected), then the program might crash. To avoid this from happening, programmers may select a default option. Programmers typically set the default option to be the one that will cause the least harm (monetarily or otherwise) to the company that the program is written for. From the perspective of the financial institution that is using our expert system and making a loan, it would be better for them to deny a loan to someone who actually qualifies for a loan, as opposed to granting a loan to someone who probably won't be able to repay the loan. That is why the Low radio button has been checked to be the default button.

One control still remains to be added to our form, the Evaluate button. When the user clicks on the button, the program should begin to process the instructions in the button's event procedure. Click the button option in the Toolbox. Then, move the cursor onto the form, and you should see your cursor changing into a plus sign and a boxed **ab**. Create a rectangle to represent this button, which is used to initiate processing, and assign it the following properties:

- (Name) property: change to **EvaluateButton**
- Location property: change to **24, 112**
- Size property: **96, 24**
- Text property: change to **&Evaluate** (the & sign is used to underline the E, signaling the existence of a keyboard access key that the user can use to initiate processing instead of clicking on the button with the mouse. Pressing the Alt. key at the same time the underlined key is pressed (in this case, Alt. and E) will execute the EvaluateButton_Click event procedure).

With the Education form now in existence, it is time to code frmEmployment.

Coding the Employment Form

Double click frmEmployment.vb in the Solution Explorer window. You should see the form's design screen. Double click the Evaluate button. Now you should be in the code window, ready to create the event procedure for EvaluateButton_Click. Type the lines of code seen in Figure 16.

Figure 16
Event Procedure for frmEmployment

```
frmEmployment                                  (Declarations)
Public Class frmEmployment
    Inherits System.Windows.Forms.Form

 Windows Form Designer generated code

    'Programmed by (your own name goes here)

    Private Sub EvaluateButton_Click(ByVal sender As System.O
        Dim objFormEducation As New frmEducation()
        If Me.EmployedRadioButton.Checked = True Then
            strEmployed = "Yes"
            strUnemployed = "No"
        Else
            strEmployed = "No"
            strUnemployed = "Yes"
        End If
        objFormEducation.Show()
        Me.Hide()
    End Sub
End Class
```

The IF statement features two sets of assignment statements, which are used to assign/give a value to a variable. In this case, values were assigned to the global variables of strEmployed and strUnemployed. If the Employed radio button is checked, then strEmployed is given the value of "Yes", while strUnemployed is given the value of "No." However, if the Employed radio button is not checked (which means the user checked the Unemployed radio button), then strEmployed is assigned the value of "No" and strUnemployed is assigned the value of "Yes." Both strEmployed and strUnemployed were declared to be string data types, and when data is assigned to a string variable, it is surrounded by quote marks. Because strEmployed and strUnemployed are global variables, those variables (declared and stored in the code module) can be used by all of the forms in the expert system. The Education form, when it evaluates user responses in its IF statement, will use the user's responses to its level of

Education question in conjunction with the values stored in the two employment variables to make one of four recommendations.

We have finished coding the Employment form. Our last form, the Final form, must now be created. The Final form does not appear in Figure 1, but the Final form serves as the means for the user to either exit the program or run the expert system again.

Creating the Final Form and Coding the Education Form

From your current position in VB.NET, go to the **File Menu**, and click **Add New Item.** Click once on **Windows Form**, and then click **Open**. A new form should appear on your screen (in design view) and you should see a new form called Form1.vb appearing in the Solution Explorer window. Click once on Form1.vb **in the Solution Explorer Window.** In the **Properties Window**, you should see **five, and only five properties**. For the File Name Property, highlight **Form1** (and not the .vb). Type in **frmFinal** so that the new File Name will display as **frmFinal.vb** Press the Enter key, and you should see **frmFinal.vb** appearing in the Solution Explorer window (instead of the previous Form1.vb).

Next, click inside the form object itself (in the gray doted area). Although we just named the actual file itself, we must also name the form object. When you click in the gray doted area, you should see that the object in the Properties window changes to **Form1**, and a longer listing of properties appears. Click the A/Z alphabetic order button to arrange the form's field properties in alphabetical order.

Give the form object the following properties:

- (Name) property: change to **frmFinal**
- Size property: change to **256, 143**
- Text property: change Form1 to **Closing**

Now, draw a label on your form. Change the following label properties:

- Size property: change to **168, 48**
- Text property: **Do you wish to run the loan expert system again, or would you like to exit?**
- Location property: change to **48, 8**
- Font property: change to **Microsoft Sans Serif, 10pt, style=Bold**

Next, draw a button on your form. Change the following button properties:

- (Name) property: change to **RunAgainButton**
- Location property: change to **40, 64**
- Size property: change to **80, 24**
- Text property: change to **&Run Again (***don't forget that the & sign is used to underline the R, which creates keyboard access keys of Alt. & R.)*

Draw a 2nd button on your form, and then change the following button properties:

- (Name) property: change to **ExitButton**
- Location property: change to **136, 64**
- Size property: change to **80, 24**
- Text property: change to **E&xit**

Once you have set the required properties, click the **Save All button.** Then, double click on frmEducation.vb in the Solution Explorer window so you can see the frmEducation's design screen showing the

form's GUI. Double click the Evaluate button, and you should now be in the code window. Type the event procedure code for EvaluateButton_Click using Figure 17. Click **Save All** when finished.

Figure 17
Code for Education Form

```
Windows Form Designer generated code

    'Programmed by (your own name goes here)

    Private Sub EvaluateButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Dim objFinalForm As New frmFinal()
        If strEmployed = "Yes" Then 'handles employed
            If Me.HighRadioButton.Checked = True Then
                MsgBox("Grant the Loan", MsgBoxStyle.Information, "Recommednation")
            Else
                MsgBox("Investigate Further", MsgBoxStyle.Information, "Recommendation")
            End If
        Else 'handles unemployment
            If Me.HighRadioButton.Checked = True Then
                MsgBox("Investigate Further", MsgBoxStyle.Information, "Recommendation")
            Else
                MsgBox("Refuse the Loan", MsgBoxStyle.Information, "Recommendation")
            End If

        End If
        objFinalForm.Show()
        Me.Hide()
    End Sub
End Class
```

As seen in Figure 17, the Education form uses a nested IF statement (one IF statement inside another IF statement). Indenting makes it easier to see the beginning and ending of each IF statement. On the Employment form, the strEmployed and strUnemployed variables were given the values of Yes/No or No/Yes respectively. Those variables were also used in the IF statements found on the Education form. If the applicant is employed (strEmployed="Yes"), then the application enters the first nested IF, which checks the applicant's level of education, and issues one of two possible recommendations. If the applicant is employed AND if the applicant has a high level of education, then the expert system should recommend granting the loan. However, if the applicant is employed, BUT the applicant has a low level of education, then the expert system should recommend investigating the applicant further. If the applicant is unemployed (strEmployed="No"), then the application enters the second nested IF, which checks the applicant's level of education, and issues one of two possible recommendations. If the applicant is unemployed, BUT the applicant has a high level of education, then the expert system should recommend investigating the applicant further. However, if the applicant is unemployed AND if the applicant has a low level of education, then the expert system should recommend refusing the loan.

Coding the Final Form

Double click frmFinal.vb in the Solution Explorer window to see the form's design screen. The Final form has two buttons (with text properties of Run Again and Exit). With two buttons that the user can click on, there are also two event procedures on this form (one for each button).

- The Run Again button causes a looping action to occur. When the user clicks on this button, the expert system displays the Income form and allows the user to evaluate a loan for another applicant. *A programming loop allows a process to be done several times. While an actual loop is not used in this program, our Run Again button serves this same purpose.*
- When the user clicks on the Exit button, the program ends (i.e. is terminated).

Double click the Run Again button, and you should be in the code window, ready to create the event procedure for RunAgainButton_Click. Type the lines of code for that event procedure, as shown in Figure 18. Once you have finished that event procedure, return to design view, and double click the Exit button, and type its event procedure code (shown in Figure 18). The keyword **End** is used to exit the application. Click **Save All**.

Figure 18
Code for Final Form

```
Windows Form Designer generated code

    'Programmed by (your own name goes here)

    Private Sub RunAgainButton_Click(ByVal send
        Dim objFormIncome As New frmIncome()
        objFormIncome.Show()
        Me.Hide()
    End Sub

    Private Sub ExitButton_Click(ByVal sender A
        End
    End Sub
End Class
```

The Final Coding

There are additional modifications that need to be made to some of the existing forms before the application is completed. These modifications are related to the Final form. Since the Final form did not exist when the Income and References forms were coded, code referring to that form was not written because we wanted to avoid having the VB.NET code editor think we were making an error by referring to an object that did not exist. Now, since the Final form does exist, we will add that final bit of code.

Open the code window for the Income form. The first two lines of event procedure EvaluateButton_Click are the Dim statements. Place your cursor after the 2nd Dim statement {right after frmReferences()}. Press the Enter key once to create a blank line. On that empty line, type in the following line of code:

Dim objFormFinal As New frmFinal() Then, move your cursor inside the IF statement, where you will find another line of code, seen below:

End 'this line of code will eventually be deleted and replaced with something else

Delete that entire line of code and replace it with: **objFormFinal.Show()** Once your completed code on the Income form matches Figure 19, click the **Save All** button.

Next, open the code window for the References form. Place the cursor right before the I of the IF statement. Press the Enter key to create a blank line, and then type in the following:

Dim objFormFinal As New frmFinal() Next, place the cursor right behind **End If.** Press the Enter key to create a new line and type in:

objFormFinal.Show() Finally, find and delete the following entire line of code:

End 'this line of code will eventually be deleted and replaced with something else

Once your completed code on the Income form matches Figure 20, click the **Save All** button.

Figure 19
Completed Code for Income Form

```
Windows Form Designer generated code

    'Programmed by (your own name goes here)

    Private Sub EvaluateButton_Click(ByVal sender As System.Object, ByVa
        Dim objFormEmployment As New frmEmployment()
        Dim objFormReferences As New frmReferences()
        Dim objFormFinal As New frmFinal()
        If Me.HighRadioButton.Checked = True Then
            MsgBox("Grant the Loan", MsgBoxStyle.Information, "Recommend
            objFormFinal.Show()
        ElseIf Me.MediumRadioButton.Checked = True Then
            objFormEmployment.Show()
        Else 'income low
            objFormReferences.Show()
        End If
        Me.Hide()
    End Sub
End Class
```

Figure 20
Completed Code for References Form

```
Windows Form Designer generated code
    'Programmed by (your own name goes here)

    Private Sub EvaluateButton_Click(ByVal sender As System.Object, ByVa
        Dim objFormFinal As New frmFinal()
        If Me.GoodRadioButton.Checked = True Then
            MsgBox("Investigate Further", MsgBoxStyle.Information, "Recor
        Else
            MsgBox("Refuse the Loan", MsgBoxStyle.Information, "Recommen
        End If
        objFormFinal.Show()
        Me.Hide()
    End Sub
End Class
```

Now, it is time to run your completed application. Click on the Start button. If you have not made any syntax or coding errors, then the application should begin running, and the Loan Expert System's Startup form should appear on the screen. If you have any syntax or coding errors in your application, the output window should indicate the existence of your error(s) and also the location in your code the error(s) is/are found. Correct any errors that are found, and try to run the application again. Ultimately, once all the errors are corrected, then you should see the Loan Expert System's Startup form on your screen.

Now, test your completed application, looking for any sort of error that you might have made. If an error occurs or the program crashes, you will need to find and correct the source of the error. The Expert System folder contains a file called **Answer Key**, which you can run as a point of comparison.

There are seven recommendations that can be made by the expert system (see Figure 1), depending upon how the user responded to the income, employment, reference, and education questions that were asked. **Test each path found in Figure 1 to make sure the application works properly.** After each recommendation is presented to the user (in the form of a message box), the user should be presented with the final form, where another consultation session can be run, or the user can decide to stop running the application. Once you have tested each possible path, and everything works, the tutorial is complete.

---

[i] The expert system created in this tutorial is an adaptation of an exercise written for VB 6.0, which appeared in *Programming Business Applications with Microsoft Visual Basic 6*.0, by William Burrows and Joseph Langford.