

# Creating and Coding an Expert System Using Visual Basic.NET

This exercise is designed to give students exposure to how computer programs are created using a computer programming language (VB.NET). In this exercise, students will create an expert system in the Visual Basic.NET programming environment 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. **This activity must be completed in the CBA computer lab or on some other computer that already has Visual Studio/VB.NET installed.**

According to Webster's dictionary, 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 area is used to reason through the problem, with the end 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 types of problems include medical diagnosis, credit and loan evaluation, help desk analysis, and compliance analysis.

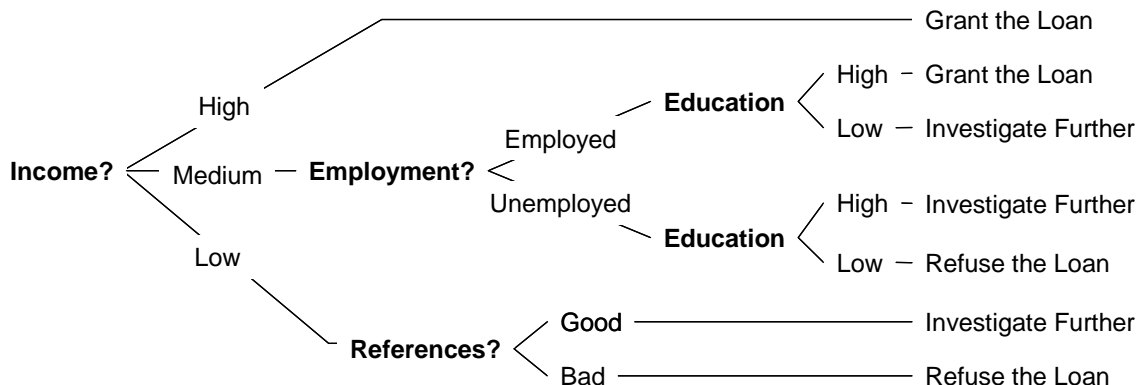
Although there are many different ways for expert systems to solve a problem or make a recommendation, most often expert systems follow some sort of decision making rules. Complex expert systems are usually created using a specialized type of development environment called expert system shells. However, the rule-based expert system that we are going to make is going to be created using Visual Basic.NET. Visual Basic.NET is a programming language designed to create applications for the Windows operating system environment.

The purpose of this exercise is create an application (an expert system) and to get a better understanding of exactly 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 related to computer programming.

Visual Basic is one of the easier programming languages to learn, and Visual Basic lets us create and code GUIs, or graphical user interfaces. In a graphical user interface environment (such as Windows and most of today's other operating systems), we have windows, icons and other pictures/elements that we work with. We can click or double click on a picture or icon, and the computer does something. When we create a computer program/application, such as our expert system, there are two primary components that we create:

1. The graphical user interface (what the user sees and works with)
2. The actual program code, which is a set of instructions that tells the computer how to perform a given task. A computer does not know what to do unless there is some sort of program or code that tells it exactly what must be done and how it is to be done.

In this exercise, we are going to create a loan evaluation expert system. The general decision making rules can be seen in the decision tree seen below, which shows conditions, outcomes, and appropriate actions. The decision tree diagram is read from left-to-right, with the far right-hand side of the diagram showing the final action to be taken (i.e. the expert system's recommendation regarding how to handle that particular loan).



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, levels of education attained, and questions about the applicant's references.

It is now time to open Visual Basic and begin creating our expert system.

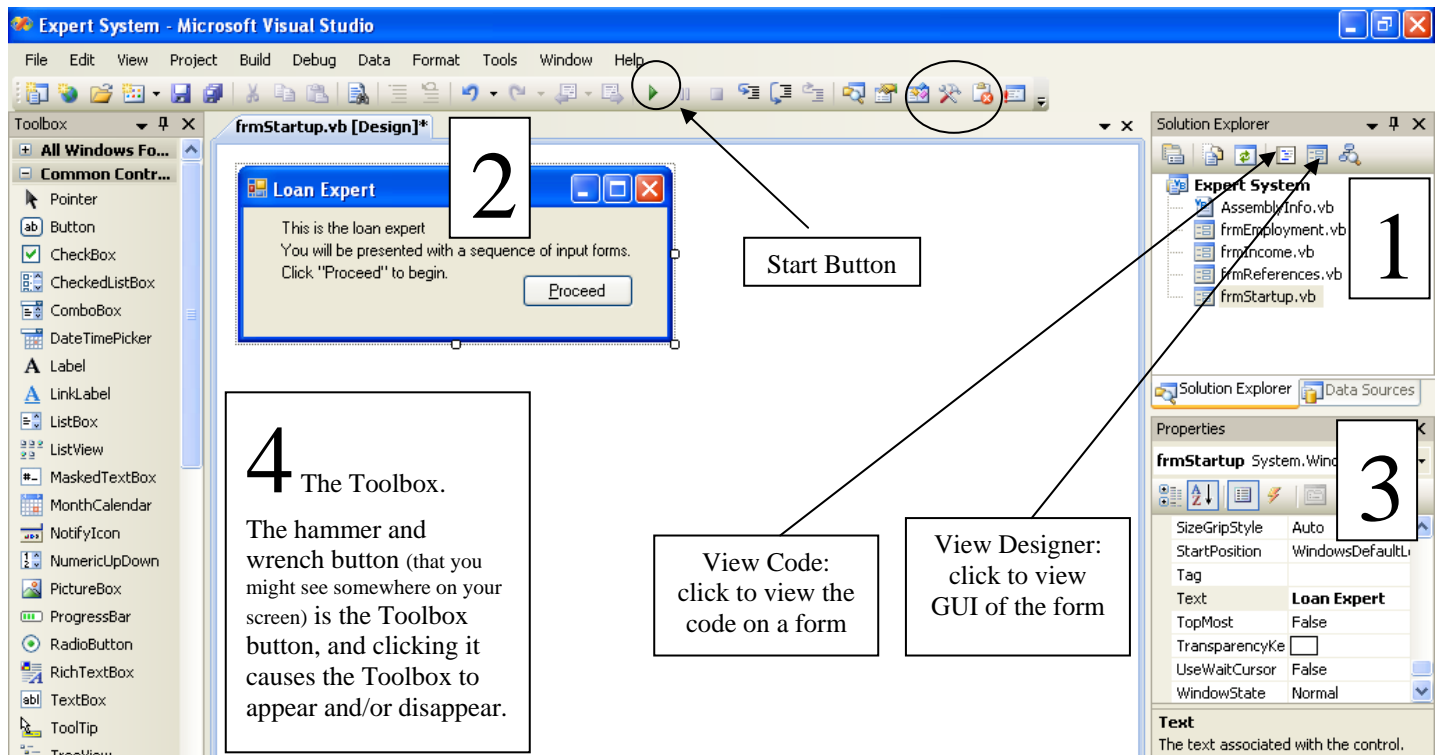


Please open the folder that contains the expert system files obtained prior to beginning this activity).

You should see two items in Windows Explorer (see picture)

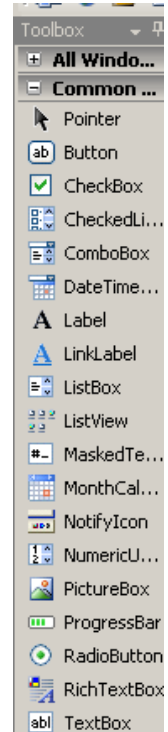
Notice the file called **Expert System, Microsoft Visual Studio Solution**. A Visual Basic Solution file is simply a container file that stores all of the projects and files for an entire application. **Double click** the **Expert System solution file**. If you ever close down the Visual Basic program 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 gives you access to all of the other elements that make up your application.

Double clicking the/a solution file allows a programmer to enter the Visual Basic.NET programming environment, as seen below. Your screen should look similar to the picture below, although there is the possibility that items that you see in the picture might not be visible, or they may appear in other locations. (What you do or don't see on your screen is largely dependent on the Visual Basic screen setup used by the last person using the computer that you now are using)



Notice the large 1 (one) on the right hand side of the picture above. 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 to see the Solution Explorer Window.) The Solution Explorer Window displays a listing of all of the items contained within your current solution (Expert System). AssemblyInfo is a file that Visual Basic 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 interface/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” on picture). If you do not see the form labeled “2” on your screen, then double click frmStartup.vb in the Solution Explorer.

The large 3 (three) on lower right hand side of the picture identifies the Properties Window. (If you do not see a Properties Window on your screen, go to the View Menu, and select Properties Window to see the Properties Window.) The each object/control 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. The picture on the prior page shows the properties for the form called frmStartup. If the "Proceed" button was the currently selected item, then its properties would display instead.



The large 4 (four) appears to the right of the Toolbox. The Toolbox itself appears to the right of this paragraph. The Toolbox is typically found on the left-hand side of the screen, but it can be located elsewhere as well. (If you do not see a Toolbox on your screen, go to the View Menu, and select Toolbox to see the Toolbox. Clicking on the button (shown in the previous picture) that looks like a hammer and wrench also makes the Toolbox appear and disappear.) The Toolbox contains a collection of tools that you can use when designing the GUI/interface for an application. You use the Toolbox to place objects, or controls, on your form.

The tools/objects/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**, also called the **option button**, allows the user to select only one item from a group.

The Start Button, which is used to run your application from the Design screen, is circled.

- ☞ Notice the arrow leaving from the textbox where it says: "View Code: click to view the code on a form." Click the button that the arrow points to, and you should be able to see the code that is currently on that form. Go ahead and see how that works.
- ☞ Notice that the next button (on the right of the View Code button) is the View Designer button. You can click on that to see the interface/GUI of the form. Go ahead and click on that button.
- ☞ When you are done clicking on the View Code and View Designer button, you should see the GUI/interface, as seen in the large picture on the prior page.

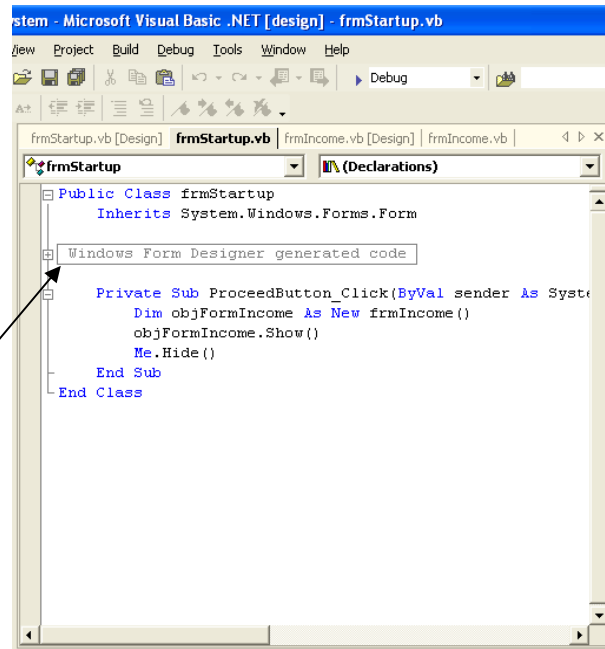
We will first take a look at the form and the user interface. The words "This is the loan expert system. You will be presented with a sequence of input forms. Click "Proceed" to begin" are displayed in a label (a label without borders, which makes it somewhat difficult to see). Labels are used to simply display text to the user. The form also contains a button, or command button, labeled as "Proceed." When the user clicks the button, its associated event procedure (i.e. coding) will begin to execute. Let's take a look at the code for that button/event procedure.

- ☞ **Please click the View Code button** (seen on the prior page).

After clicking the **View Code button**, you should see the code window (part of which is seen in the picture to the right).

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.

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 Visual Basic to keep the current form active (frmStartUp), but it will be hidden (invisible) to the user.

In essence, when the "Proceed" command button is clicked, Visual Basic creates a new object, representing the Income form (frmIncome), displays the new Income form, and then hides the prior form the user was seeing/using (frmStartUp).

Now, after looking at the coding on the first form, it is now 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 earlier VB programming environment picture). **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).

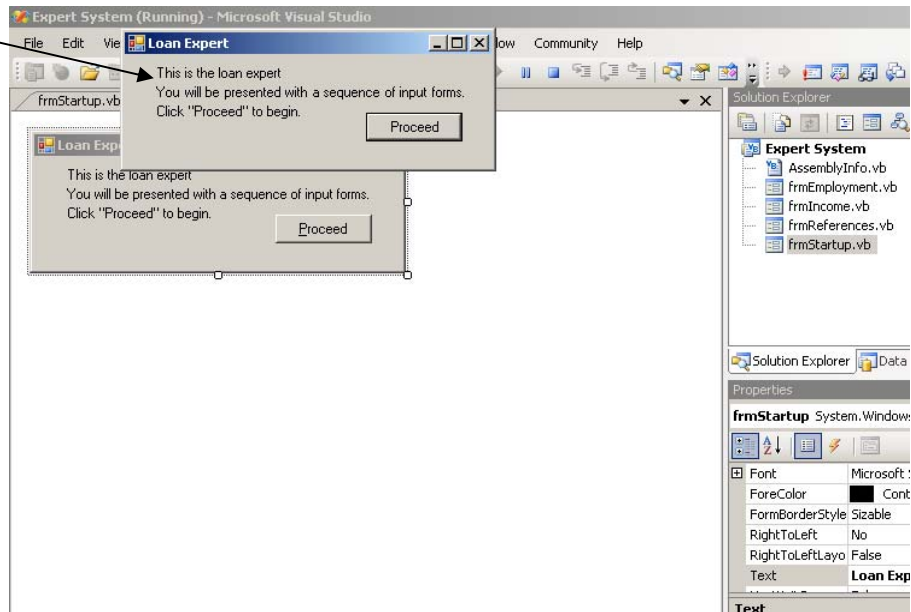
Application running in runtime mode. Everything else that you might see is part of the design screen.

*Next, I will mention a few items that might appear on your screen at some future point, but don't worry about them if you don't see them now.*

You might also see a new window opening at the bottom of your screen. That window is the Task List.

It could say "0 Build Error tasks shown" or it could say "**Build: 1 succeeded, 0 failed, 0 skipped**". That means that you did not make a syntax or typing error, and your application will work as

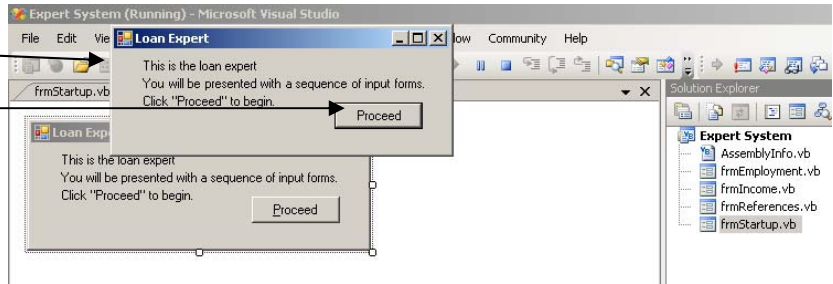
it is written/coded. If you ever make a syntax error or other type of error, then the application will not run and you will see some sort of message box indicating that you have build errors, which you will then need to stop and fix before the application is able to run. However, since we have not coded or typed anything in yet, we should be in good shape (with no error messages appearing).



We now want to work with our currently running application.

Click on the Proceed button.

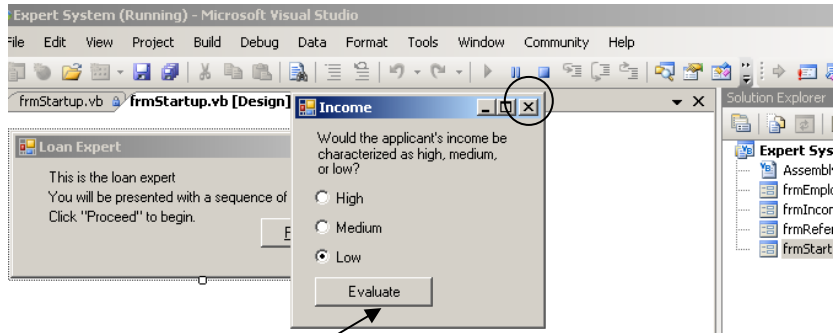
Visual Basic is referred to as an “event driven” language because when the user does something (in this case clicks on the Proceed button), its associated event procedure (code we looked at earlier) begins to execute.



When the user clicks on the Proceed command 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, resulting in what you see in the next picture.

As you can see, the next form in our expert system asks about the applicant’s income (refer to far-left of earlier decision tree diagram on page #1).

The expert system user would next characterize the applicant’s income as high, medium, or low. After selecting the appropriate option, it would be time to evaluate, or move on to the next form, and its corresponding question.

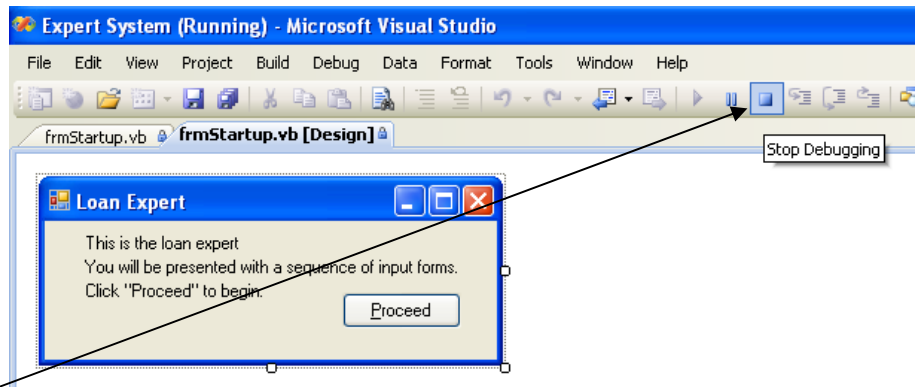


Go ahead and click on the Evaluate button now.

Notice that nothing happens when you click on this Evaluate button. Why??

Objects/controls in Visual Basic only respond to events that have been coded, and we have not yet created the code that will execute when the user clicks on the Evaluate button. Visual Basic is an event-driven language, and Visual Basic applications only respond to events/actions they have been programmed/coded to respond to. 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 programmer creates/writes the appropriate code 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 (circled in the prior picture) in the upper right hand corner of the application window.



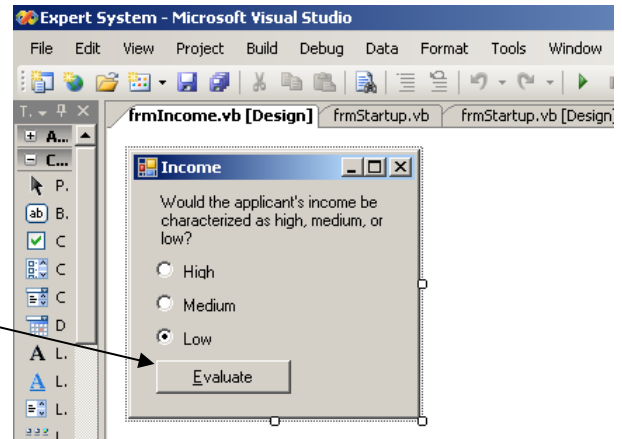
At this point, the Expert application should have stopped running, but you are still in the “debug” mode, which you will next need to stop. To stop debugging, click the **Stop Debugging** button.

Go ahead and click the **Stop Debugging** button, and you should now be back on 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.

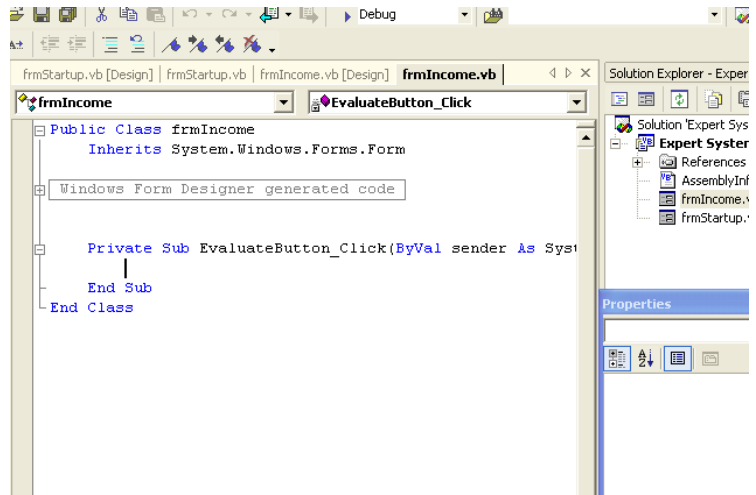
After double clicking the Evaluate button, you should now see the code window, as seen in the next picture below. **Do not move your cursor from its current position.** As you can see, Visual Basic has already started to create the event procedure (also called a sub-routine) for when the user clicks on the Evaluate button.




**Private Sub** and **End Sub** are keywords automatically included by VB. **Private Sub** marks the beginning of the event procedure/sub-routine, while the keywords **End Sub** mark the end of the event procedure.

**EvaluateButton\_Click** is the name of the event procedure that you are now working on.

As mentioned earlier, programmers typically indent lines of code within event procedures, and the VB code editor has already made the initial indent for you.



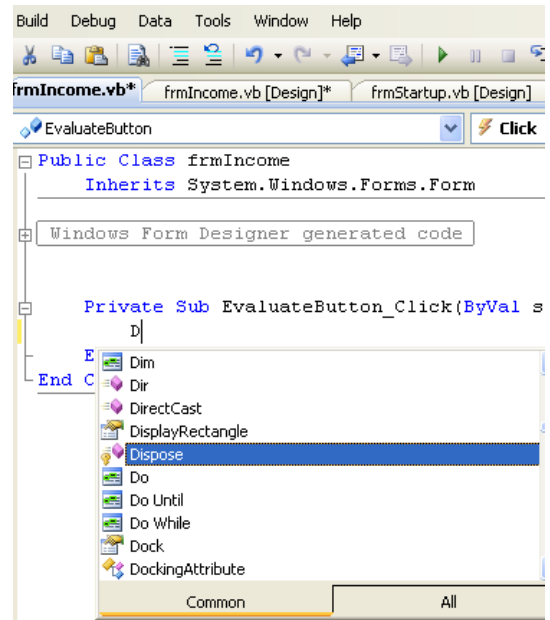
 Type in the following code:


**Dim objFormEmployment As**

As soon as you type in “D”, 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)

**Dim objFormEmployment A**

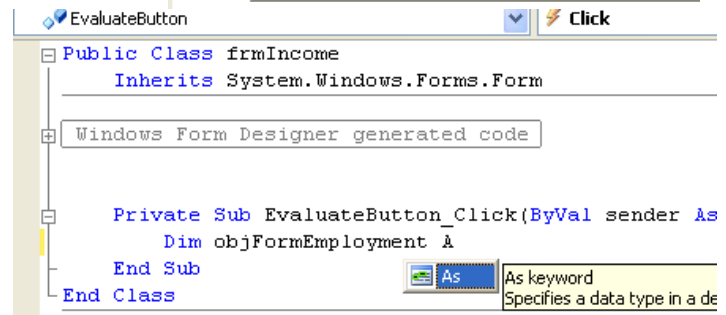
As the capital A is typed, notice IntelliSense has highlighted “As”. The VB keyword “As” is the next item that you want to appear in your code. While it is acceptable to type in the rest of the keyword or relevant object name, you can also use IntelliSense to add that object to your code by simply pressing the spacebar while the appropriate item (As) is highlighted in the IntelliSense listing, and the keyword **As** is automatically added to your code.



 Continue typing:

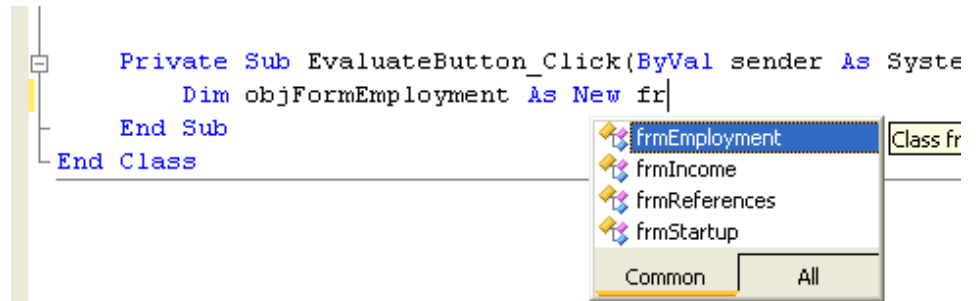
**Dim objFormEmployment As New fr**

While typing that code, IntelliSense continues guide you and offer potential assistance. Once you type “As New fr”, IntelliSense shows 4 possible form object options that you might wish to use (see next picture). Once you have typed enough for frmEmployment to be highlighted by IntelliSense,



you can now have IntelliSense add the object to your code, without having to type out the entire object name in your code.

- To add frmEmployment to your coding through the use of IntelliSense, simply press the spacebar while the object (frmEmployment, in this case) is highlighted and frmEmployment is automatically added to your code. Go ahead and use this feature now.



- 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**

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 or if one thing happens, then I will take one course of action. If another condition occurs or if another thing happens, 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, on a business level, an accounts payable program must be able to handle overpayments. If an overpayment does occur, the AP program might issue a credit to that customer's account, or it might even produce a refund check for that particular customer. 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 on the first page of this document (please take a look at the Expert System decision tree and refer to it during the following discussion).

As we have seen already, 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.

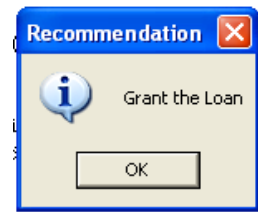
Before coding the next part of our expert system, we first need to take a look at 3 other features used by programmers: Message Boxes, Comment Code, and Radio/Option Buttons. Those 3 features/elements will appear inside our upcoming selection statements.

## Message Boxes, Comment Code, and Radio/Option Buttons

The **Message Box statement** causes a new window to open on the screen. The window contains 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")

Notice that the message box syntax contains 2 commas. The "things" before and after the commas are called arguments. The first argument is simply the message that you want displayed to the user. The message 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 title text that will appear in the title bar of the message box.

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

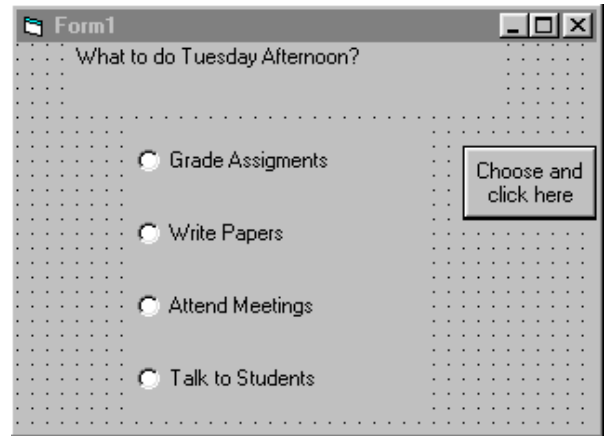


**Comment code**, also called **remark statements**, is used by the programmer as program documentation. Comments are not considered “executable” and have no effect when the program runs. The purpose of comments is to make the program easier to read so you can tell what the programmer is trying to do with a block of code. Comments are used to clarify the coding. 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, and 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 below:

' **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**, also known as **option buttons**, are used to ensure that the user will select **only one option** from a group of options. The picture to the right shows an example of an application that uses 4 radio/option buttons on the same form. Programmers typically use radio/option buttons in conjunction with Selection statements.




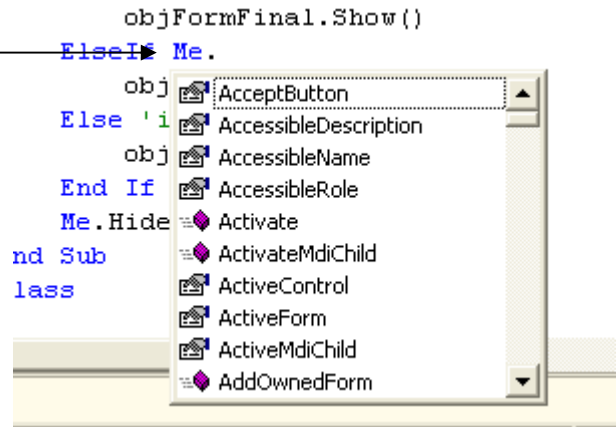
*Another control that is used in conjunction with Selection statements is the Checkbox control, which we will not be using in this project. While the radio button allows only one option to be selected from a group of options, the checkbox allows the user to select multiple items from a group of options.*

Before we get back to coding this application, one more feature of the VB code editor deserves mention. When writing code in VB, a programmer will often use **Me** to refer to the current form.

**Me.** (Me dot/period)

can also be used to invoke the IntelliSense drop-down list of object names. Using IntelliSense (*in conjunction with the spacebar*) can allow the programmer to add object names and actions to the programming code, and avoid making syntax errors due to incorrect spelling.

-  Now, it is time to begin coding again. Your cursor should still be on the next line, underneath the “D” (still indented) of the word “Dim.” We are going to start a new line of code. 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, THEN THAT MEANS THAT YOU HAVE MADE A TYPING/CODING/SYNTAX ERROR** that you will need to fix before your program works correctly.

The next page also shows a picture that shows the completed event procedure, as it should appear on your screen. You can use that as an additional guide, if you wish.



Now type:

**If Me.HighRadioButton.Checked = True Then** (pressing Enter next 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/checked the medium or low radio buttons).

Now, 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.

Now type:

**ElseIf Me.MediumRadioButton.Checked = True Then** (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/checked 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/checked the low radio button).

Completed  
Code →  
For  
Evaluate  
Button  
Event  
Procedure

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

Now, press your backspace key so that the cursor backs up and lines up with the "E" of the ElseIf and "E" of the End If.

Now 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/checked 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).

```
        MsgBox("Grant the Loan", Msg  
    ElseIf Me.MediumRadioButton.Chec  
        objFormEmployment.Show()  
    Else 'income low  
        objFormReferences.Show()  
    End If  
    Me.Hide()  
End Sub  
End Class
```

Now, with our IF statement finished, we now 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()**

The line **Me.Hide ( )** tells Visual Basic to keep the current form active (frmIncome), but it will be hidden (invisible) to the user.

We have now finished coding the event procedure for when the user clicks on the evaluate button on the Income form. Before leaving this form, we have one additional line of code that we are going to write.

- ☞ Please place your cursor in-between the Windows Form Designer generated code and the event procedure that you just created (as seen in the picture to the right)

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

    Windows Form Designer generated code

    'Programmed by (your own name goes here)|

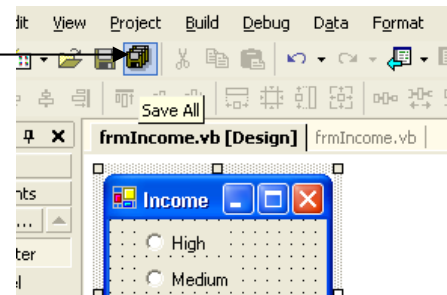
    Private Sub EvaluateButton_Click(ByVal sender
        Dim objFormEmployment As New frmEmployment
        Dim objFormReferences As New frmReferences
```

- ☞ Type in the comment code, as seen below. Don't forget that the apostrophe signals the beginning of a comment.

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

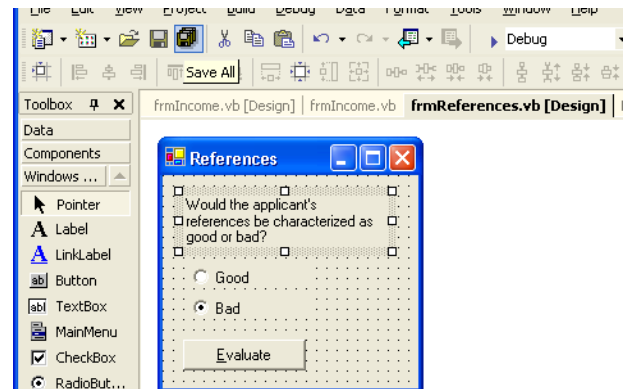
- ☞ Now, we have finished coding the Income form. Please click on the **View Designer** button so that you can see the actual form/GUI (and not the code). If you have forgotten where the View Designer button is located, then please see the labeled picture on page #2.

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.



- ☞ **Please click the SAVE ALL button now.**

- ☞ Now, find 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.



Let's go back to our Expert System decision tree found on the first page of this document (please take a look at the Expert System decision tree and refer to it during the following discussion).

As we have seen already, 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 being 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/denying the loan.

In an event-driven application such as we are creating, 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, **we need to double click the Evaluate button found on the References form**, and you should now be in the code window for the References form, ready to create the code for `Private Sub EvaluateButton_Click`

- ☞ Type in the following lines of code:

**If Me.GoodRadioButton.Checked = True Then** (then press Enter, make sure that you are indented, and then type):

**MsgBox("Investigate Further", MsgBoxStyle.Information, "Recommendation")**

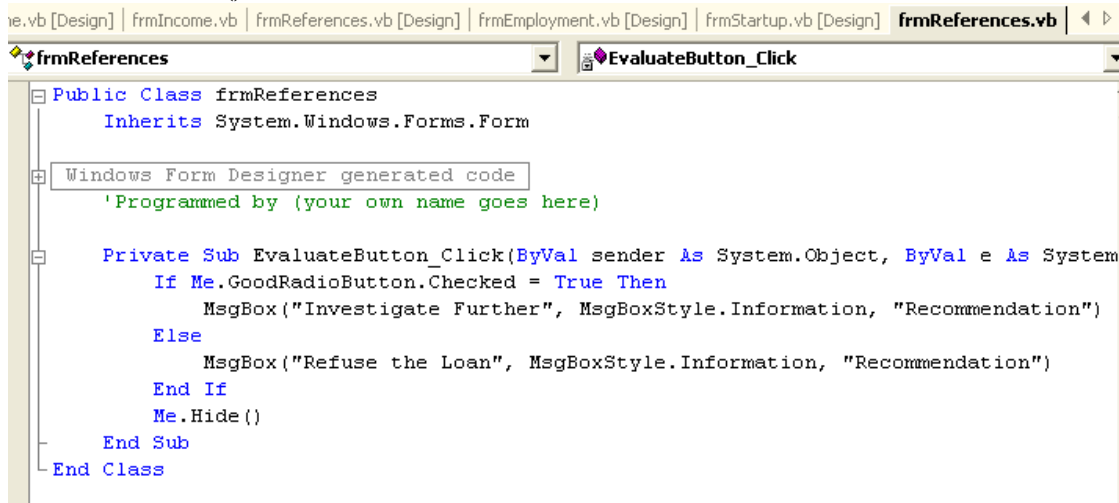
Now, press your backspace key so that the cursor backs up and lines up with the "I" of the If and "E" of the End If.

- ☞ Now type: **Else** (then press Enter, make sure that you are indented, and then type):

**MsgBox("Refuse the Loan", MsgBoxStyle.Information, "Recommendation")** (this message box will appear when the Bad radio button is checked)

Now, with our IF statement finished, we now 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()**



```
Public Class frmReferences
    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 System.EventArgs) Handles EvaluateButton.Click
        If Me.GoodRadioButton.Checked = True Then
            MsgBox("Investigate Further", MsgBoxStyle.Information, "Recommendation")
        Else
            MsgBox("Refuse the Loan", MsgBoxStyle.Information, "Recommendation")
        End If
        Me.Hide()
    End Sub
End Class
```

Next, we need to create the comment that appears before our event procedure (as shown above).

- Please place your cursor in-between the Windows Form Designer generated code and the event procedure that you just created. Type in the same comment that we typed in on the prior form (as seen below and in the picture).  
**'Programmed by (replace the parenthesis and these words by your own first name and last name)**

**Before continuing on, please make sure that your code for the References form matches the sample/picture seen above.**

The coding on this form says the following: If the Good radio button is checked by the user, then the application should display a message box recommending that the user investigate the applicant a little bit 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 by the user, 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 your Enter key, 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 of code that we will eventually replace with something else, as indicated in the accompanying comment code, but we want the **End** command to appear so that we can have the application stop/end itself automatically once this form is done.

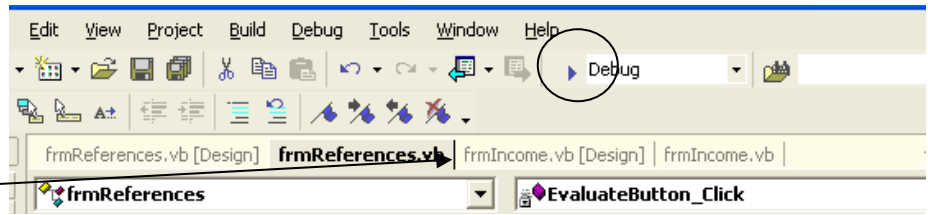
- Now, copy the entire line of code that you just typed.

- 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

(seen in the picture) to see the code on frmIncome.



- 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 seen in the next picture.

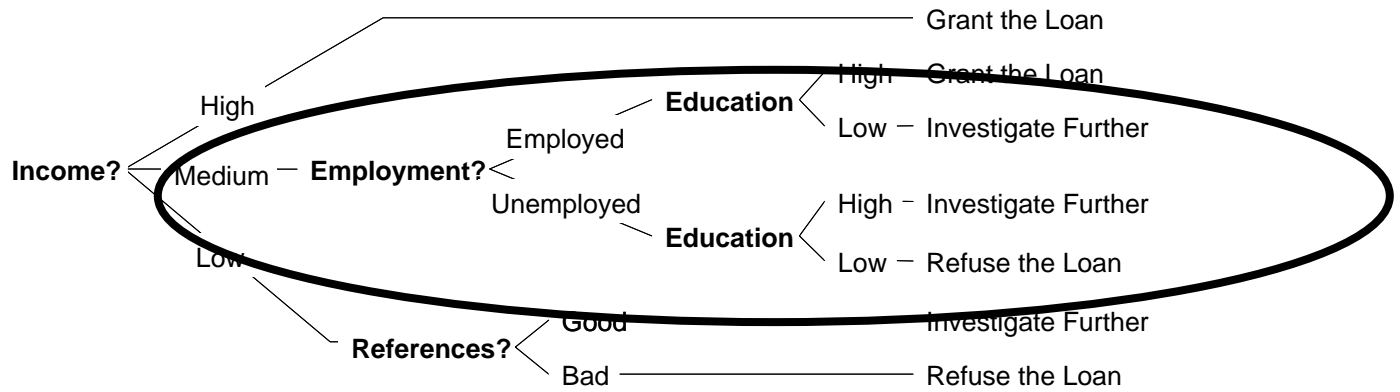
- Now, click the **Save All** button to save all of the changes that you have made to all of the forms (and their code).

```
'Programmed by (your own name goes here)
Private Sub EvaluateButton_Click(ByVal sender As System.Object, By
Dim objFormEmployment As New frmEmployment()
Dim objFormReferences As New frmReferences()
If Me.HighRadioButton.Checked = True Then
    MsgBox("Grant the Loan", MsgBoxStyle.Information, "Recomme
End 'this line of code will eventually be deleted and repl
ElseIf Me.MediumRadioButton.Checked = True Then
    objFormEmployment.Show()
Else 'income low
    objFormReferences.Show()
End If
Me.Hide()
End Sub
Class
```

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/run the application. The start/run button is circled in the picture on the previous page (to the left of the word Debug). **Click the start/run 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 go back to the design screen** (since the end command tells the application to quit running.)
- Now, for our 2<sup>nd</sup> test, **click the start/run button again**. Once again, the Startup form should appear. **Click the Proceed button**. Again, 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 appropriate message box for the selected option should appear. **Click ok, and you should go back to the design screen once again**.
- Now, for our 3<sup>rd</sup> test, **click the start/run button again**. Once again, the Startup form should appear. **Click the Proceed button**. Again, the Income form should appear. **Click the low option, and then click the Evaluate button**. Just like we saw in the 2<sup>nd</sup> test, the References form should now appear. **Select either the Bad option and then click on the Evaluate button**. The appropriate message box for the selected option should appear. **Click ok, and you should go back to the design screen once again**.

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 the expert system decision tree), and we have ignored the medium income path. We will work on that path next. However, before coding the medium income path, we need to look at the role and scope of variables in a VB program.

### Variables and Variable Scope

Recall that **variables** are simply memory locations that we use to store data. These memory locations are called variables because the data stored in these memory locations **can change (or vary)** as the program is running. When a data type for a variable is chosen, one must first 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, words, and numbers not used in calculations). If a variable is to be used in a mathematical calculation, one of the numeric data types (integer, decimal, single, double, short, long) should be chosen. Object data types, which take up the greatest amount of memory space, can be used to store anything, and up to this point, all of the variables that were created for you or that you created have been of data type object.

When creating variables, a programmer must think about the scope of the variable. A variable's scope 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. For our discussions of variable scope, there are three levels of variable scope. Going from smaller scope to larger levels of scope, these levels of scope are called procedure-level (local) scope, module-level scope, and global scope.

Any variable that is declared inside an event procedure has **procedure-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 procedure-level/local scope) that other event procedures are unable to "see" or 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 of those variables have the same name, they can hold/contain 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 variable called Total, 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. A variable declared within an event procedure is only visible within that event procedure. It is not visible outside of the procedure where it is declared. Dim statements are used to create procedure-level variables. All of the variables that we have created in our expert system (up to this point) have been procedure-level variables.

In fact, the VB keywords of **Private Sub** are indicators of private subroutines used only by individual controls (such as a button). No other control or element of a VB program can use a private subroutine or the variables and other elements that exist only in that subroutine.

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 of an event procedure, in what is known as the general

```

frmEmployment.vb [Design] | Form1.vb [Design]* | Form1.vb*
Form1
Public Class Form1
    Inherits System.Windows.Forms.Form
    Windows Form Designer generated code
    'This is the general declarations section.
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    End Sub
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    End Sub
End Class
  
```

declarations section. 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.**

The next screen shot illustrates local procedural-level variables, as well as module-level variables which are placed in the general declarations section.

However, as you have already seen in this project, oftentimes a computer application contains 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 **that can be shared across all forms have global scope**, and those types of 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 that is shared with all of the forms, and that repository is called a code module. Another 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 ever sees a code module is the application's programmer and the each code module has its own code window that contains the data 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.

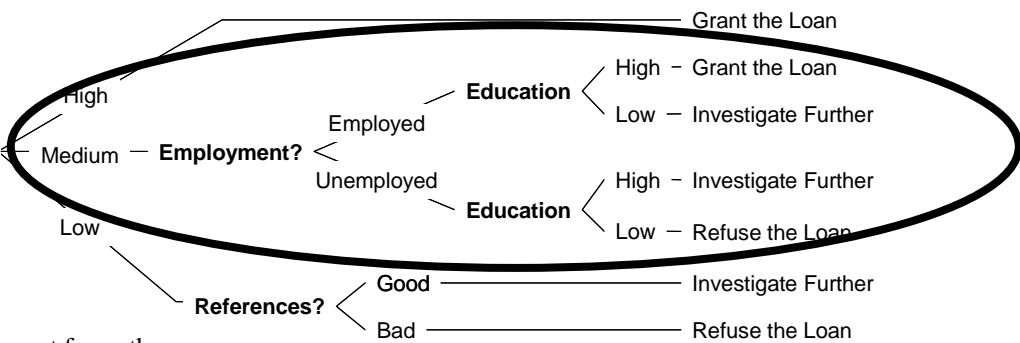
```

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
    
```

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.



As seen in the decision tree, if the applicant's income is characterized as being medium, then the Employment form is displayed. Once on the Employment form, the user then 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, then 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, then 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 and that capture mechanism will be global variables. As seen previously, global variables can be shared between all forms, and global

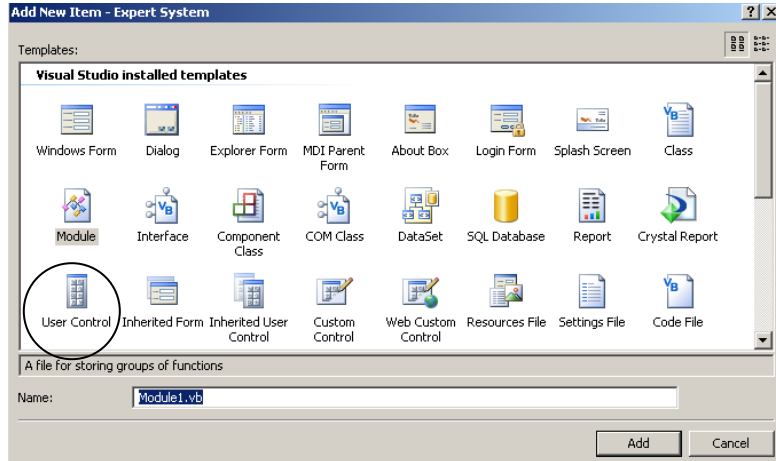
variables are stored in a code module. Therefore, the next thing that needs to be created and added to our loan expert system application is a code module.

### Creating the Code Module

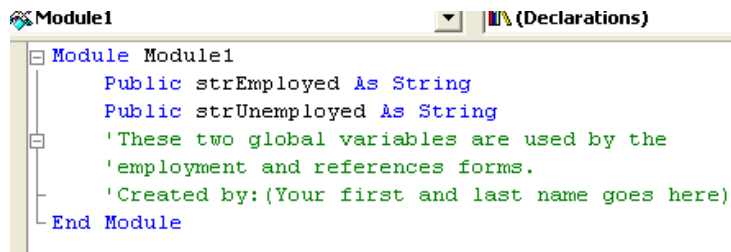
- From your current position in Visual Basic, go to the **Project Menu**, and click **Add Module**. Scroll through the options until you find module. Click once on **Module**, and then click **Add**.

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 pictured next, including the comment code (5 lines total are typed).



Two variables are created in the code module. Both are global variables that can be used by all forms in this project, as indicated by the Public keywords. The words "As String" are used in the variable declaration statements to create two variables that are of string data type. The string data type is used to store a sequence of characters as text (letters, words, and numbers not used in calculations). The two variables that are created are called strEmployed and strUnemployed. The final three lines that you typed are comment code, and your actual first name and last name should replace the *(Your first and last name goes here)*.



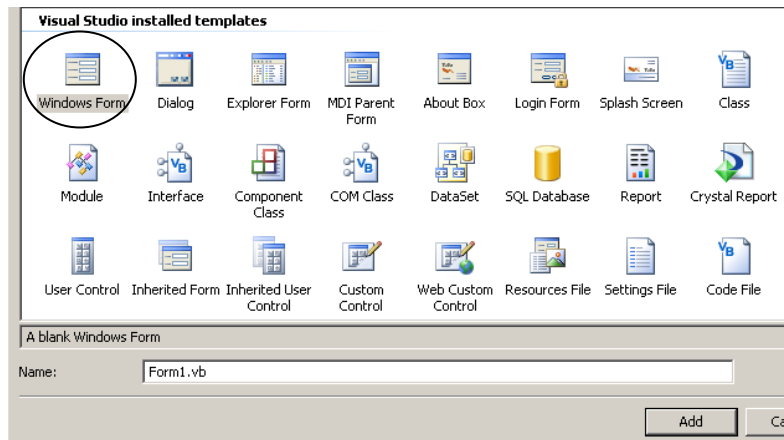
- After creating the code module, click the **Save All** button to save your work on all forms and the code module.

According to the expert system decision tree, after the user indicates the applicant's employment status, the application should next display the Education form. However, our application does not have an Education form. That is what we will create next.

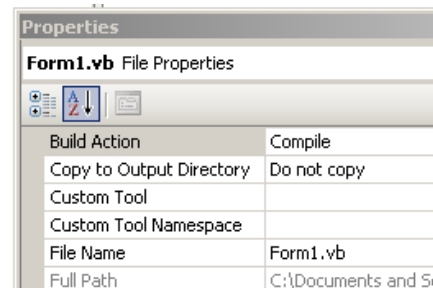
### Adding a new form to a VB application

- From your current position in Visual Basic, go to the **Project Menu**, and click **Add Windows Form**. Scroll through the options until you find Windows Form. Click once on **Windows Form**, and then click **Add**.

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.




Up to this point, we have not worked with the Properties window (usually on the lower right-hand side of your screen). Since we are making a new form from scratch, we are going to have to specify some of the properties for this new form that have been previously given to us in the files that accompanied this exercise.

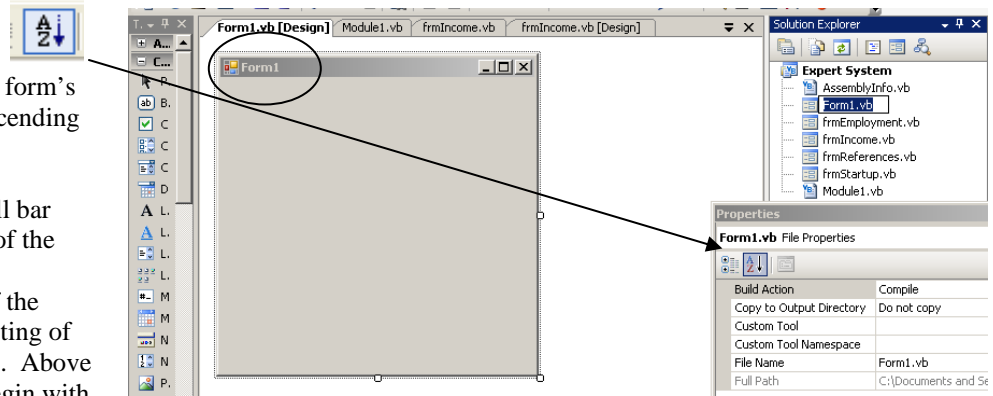


- Click once on Form1.vb in the **Solution Explorer Window**. Look at the **Properties window**, usually located underneath the Solution Explorer window, on the right. You should see **six, and only six properties**, similar to the picture to the right. For the File Name property, highlight **Form1** (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 area). Although we just named the **file**, we must also name the **form control**. When you click in the gray area, a longer listing of properties now appears in the Properties

windows. Click the  button to arrange the form's field properties in ascending (alphabetical order).

- 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, and you can see that the (Name) property contains **frmEducation**.



- Now, scroll down through the alphabetical listing of the properties until you find the Text property. Notice that the form control currently displays Form1 in the title bar of the form (*circled in the picture above*). That is the same wording that is displayed in the text property. Change the text property of the form to **Education**, and then press the Enter key. You now should see Education appearing 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 see the size of the form change.

The GUI of all of your other forms contains controls such as radio/option buttons, a label, and a button to click on. We will next add those controls to the Education form.

Pictured to the right is the Toolbox, which is probably on the left-hand side of your screen.

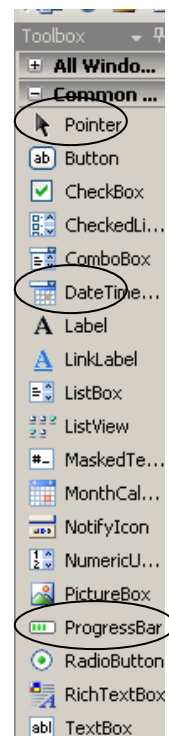
- A label is used to display text to the user. Click the Label button (circled in the middle). After you click the Label button, move your cursor onto the form, and you should see your cursor changing into a plus sign and an A. Position the cursor toward the top left-hand side of the gray-dotted section of the form. Hold down your left mouse button, and while holding the left mouse button down, drag the mouse down, and then to the right and when you release the left mouse button, a box should appear. Then, click somewhere else on the form. At this stage, do not worry about the size or position of the label. We will set those properties next.

- Click on the label, and the label/box should now be highlighted. The Properties window should now display Label1 (one). Change the following label properties:

- AutoSize property: change to **False**
- Size property: change to **128, 48** (128 width, 48 height)
- Text property: change Label1 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/option 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 (see bottom circled item in the picture above). After you click the Radio button, move your cursor onto the form, and you should see your cursor changing into a plus sign and a button. Position the cursor under the label, hold down your left mouse button, and while holding down the left mouse button, drag the mouse down, and then to the right and when you release the left mouse button, a





radio button should appear. Do not worry about the size or appearance of the radio button. We will set those properties next.

- ☞ 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**
  - `AutoSize` property: change to **False**
  - Location property: change to **16, 64**
  - Size property: **96, 17**
  - 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**
  - `AutoSize` property: change to `False`
  - `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 radio/option buttons appear in computer programs, programmers commonly have a default option commonly selected when the user must choose one option (and only one option) from a list of options. If a default option is created, and if the user does not select an option (at all) before initiating processing (when an option is expected to be selected), then the program might crash or lock up. To avoid this from happening, programmers commonly select a default option and/or use error checking IF statements to make sure that the user has selected an 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 being written for. For instance, 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.
  - Location property: change to 16, 88
  - Size property: 96, 16
  - Text property: change to **Low**

One control still remains to be added to our form, the Evaluate button. When the user clicks on the button, the program should be begin to process the instructions in the button's event procedure.

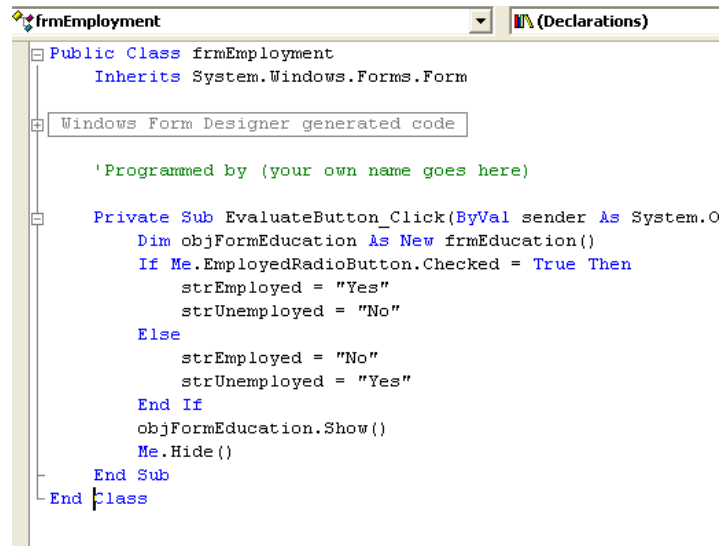
- ☞ Click the button option in the Toolbox (top circled button in the picture above). After you click the button option, move your cursor onto the form, and you should see your cursor changing into a plus sign and a boxed **ab**. Position the cursor toward the bottom of the form. Hold down your left mouse button, and while holding the left mouse button down, drag the mouse down, and then to the right and when you release the left mouse button, a button should appear. This button will be used to initiate processing, and it should have 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. When the E is underlined, you are creating a keyboard access key that the user can use to initiate processing instead of initiating processing the usual way 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.*)

After having created the Education form, it is now time to get back to coding `frmEmployment`.

## Coding the Employment Form

Double click on frmEmployment.vb in the Solution Explorer window. You should see the frmEmployment's design screen showing the form's GUI. Double click the Evaluate button, and you should now be in the code window, ready to create the event procedure for EvaluateButton\_Click.

Type in the following lines of code:  
**Dim objFormEducation As New frmEducation**  
*(saying to create a new variable (of data type object) and store a copy of the form called frmEducation in that object variable)*



```
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
```

Press the Enter key, and then type the following code seen in the pictured event procedure:

**If Me.EMPLOYEDRadioButton.Checked = True Then**  
**strEmployed = "Yes"**  
**strUnemployed = "No"**

Press Enter, then use the backspace key to line the cursor up with the I (of If Me.) and E (of End If) and type:

**Else**  
**strEmployed = "No"**  
**strUnemployed = "Yes"**

(the End If should already appear in your code). The IF statement features two sets of assignment statements. An assignment statement is used to assign/give a value to a variable. In this case, values were assigned to the global/public 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 given/assigned the value of "No" and strUnemployed is given/assigned the value of "Yes." Both strEmployed and strUnemployed were declared to be of data type string, and when data is assigned to a string variable, it is surrounded by quote marks.

Because strEmployed and strUnemployed are global variables, that means that those variables (declared and stored in the code module) can be used by all of the forms in the expert system application. The Education form, when it evaluates user responses in its If statement, will use the user's responses to its levels of Education question in conjunction with the values stored in the two employment variables, to make one of four recommendations.

Go to the line below the End If and type:  
**objFormEducation.Show()**  
**Me.Hide()**

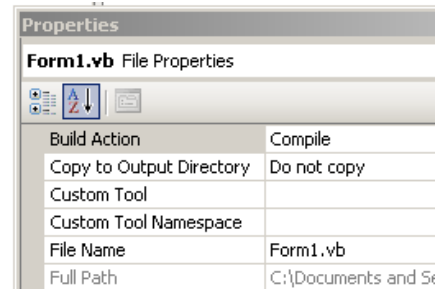
Regardless of the employment condition specified, after values have been assigned to strEmployed and strUnemployed, the application will then display the Education form and hide the Employment form.

To finish up the coding on the Employment form, add the comment **'Programmed by (replace the parenthesis and these words by your own first name and last name)** to the form in the general declarations section of the Employment form.

Now, we have finished coding the Employment form. Before we continue our coding, our last form, the Final form, must be created. The Final form does not appear on the expert system decision tree diagram that you have seen in this tutorial, 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

- 📄 From your current position in Visual Basic, go to the **Project Menu**, and click **Add Windows Form**. Scroll through the options until you find Windows Form. Click once on **Windows Form**, and then click **Add**. 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 **six, and only six properties**, as seen in the picture to the right. 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). 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 area).
- 📄 Give the form object the following properties:
  - Size property: change to **256, 143**
  - Text property: change Form1 to **Closing**
- 📄 Now, draw a label on your form. Change the following label properties:
  - AutoSize property: change to **False**
  - Size property: change to **211, 56**
  - Text property: change Label1 to **Do you wish to run the loan expert system again, or would you like to exit?**
  - Location property: change to **25, 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 a keyboard access key when used in conjunction with the Alt. key (Alt. and the R pressed at the exact same time) that will execute the button's event procedure.*)
- 📄 One final button is needed. Draw a 2<sup>nd</sup> 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** (*don't forget that the & sign is used to underline the X, which creates a keyboard access key when used in conjunction with the Alt. key (Alt. and the X pressed at the exact same time) that will execute the button's event procedure.*)
- 📄 At this time, go ahead and click the **Save All button**. Then, double click on frmEducation.vb in the Solution Explorer window. You now should 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, ready to create the event procedure for EvaluateButton\_Click.

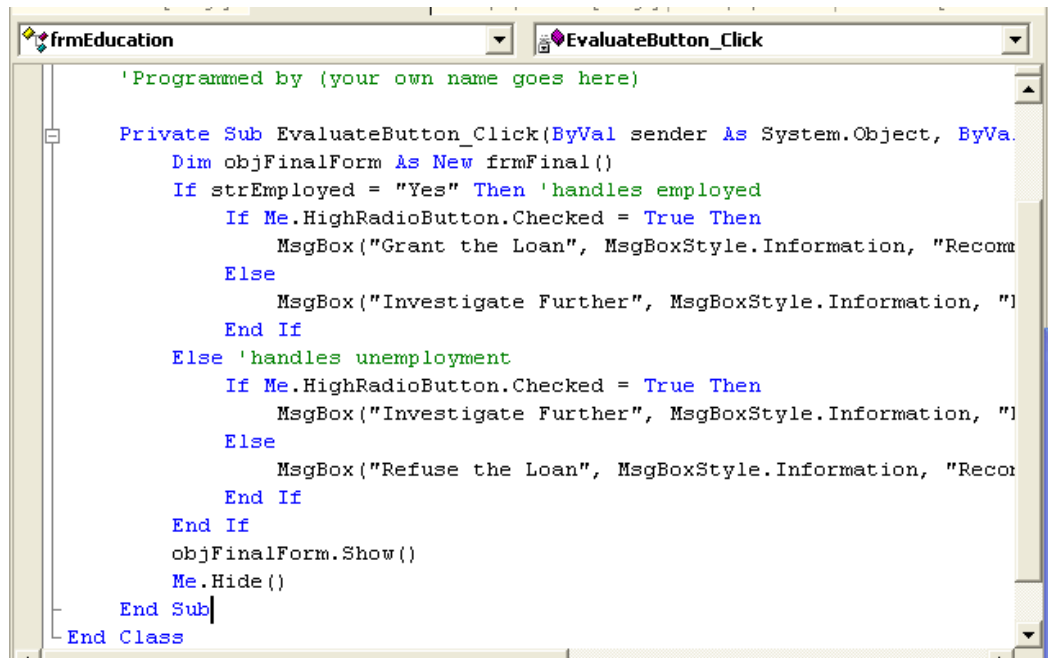


## Coding the Education Form

As seen in the code on the right, the Education form uses a nested IF statement. A nested IF statement is simply one IF statement inside another IF statement. Our use of the indent feature makes it easier for us to see the beginning and ending of each IF statement.

We will first write the code for the form

(as seen in the picture) and then a description of how the code works and what it does will be given.



```
'Programmed by (your own name goes here)

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

- ☐ Please indent your own code exactly as is seen in the picture. Type the following:

```
Dim objFinalForm As New frmFinal()
```

```
If strEmployed = "Yes" Then 'handles employed (this marks the beginning of the outside IF statement)
```

```
    If Me.HighRadioButton.Checked = True Then (this marks the beginning of the first inside IF statement)
```

```
        MsgBox("Grant the Loan", MsgBoxStyle.Information, "Recommendation")
```

```
    Else
```

```
        MsgBox("Investigate Further", MsgBoxStyle.Information, "Recommendation")
```

The *End If* should already appear. It marks the end of the first inside IF statement and you should not have to type it since it should already appear, so then move your cursor down to the next line, press your Backspace key, and then type:

```
Else 'handles unemployment
```

```
    If Me.HighRadioButton.Checked = True Then (this marks the beginning of the second inside IF statement)
```

```
        MsgBox("Investigate Further", MsgBoxStyle.Information, "Recommendation")
```

```
    Else
```

```
        MsgBox("Refuse the Loan", MsgBoxStyle.Information, "Recommendation")
```

Make sure that the next two *End If*'s appear, as seen in the above picture. The first *End If* this marks the end of the second inside IF statement, while the second *End If* marks the end of the outside/original IF statement. Move the cursor to the end of the 2<sup>nd</sup> *End If*, press the Enter key, and then press the backspace key so that the cursor lines up with the E of the last *End If*, and then type the following:

```
objFinalForm.Show()
```

```
Me.Hide()
```

- ☐ To finish up the coding on the Education form, add the comment **'Programmed by (replace the parenthesis and these words by your own first name and last name)** to the form in the general declarations section of the Education form. Make sure that your code on frmEducation matches the picture above.

Now, we have finished coding the Education form. Before continuing on, we need to look at the decision making portions of our code, found in the IF statements of the Education form. To better understand the description that follows, it also might be a good idea to refer to one of the expert system decision tree diagrams found earlier in this tutorial.

Remember that on the Employment form, the strEmployed and strUnemployed variables were either 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 then 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 issue a recommendation of granting the loan. However, if the applicant is employed, BUT the applicant has a low level of education, then the expert system should issue a recommendation of investigating the applicant further.

If the applicant is unemployed (strEmployed="No"), then the application enters the second nested IF, which also 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 issue a recommendation of 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 issue of recommendation of refusing the loan.

- At this time, go ahead and click the **Save All button**. Then, double click on frmFinal.vb in the Solution Explorer window. You now should see the frmFinal's design screen showing the form's GUI.

## Coding the Final Form

Notice that the Final form has two buttons (text properties of the buttons appearing as Run Again and Exit). Since there are two buttons that the user can click on, there are also two event procedures on the Final 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 it 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 performs the same way that a loop would by allowing the program to run again.*

When the user clicks on the Exit button, the program ends (i.e. is terminated).

- Double click the Run Again button, and you should now be in the code window, ready to create the event procedure for RunAgainButton\_Click.
- Type in the following lines of code:  

```
Dim objFormIncome As New frmIncome()
objFormIncome.Show()
Me.Hide()
```

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

Private Sub RunAgainButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RunAgainButton.Click
    Dim objFormIncome As New frmIncome()
    objFormIncome.Show()
    Me.Hide()
End Sub

Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ExitButton.Click
    End
End Sub
End Class
```

Those three lines of code create an object variable, store a copy of frmIncome in the object variable called objFormIncome, execute the Show method of the variable in order to display the Income form, and then the existing form (Final form) is hidden from view.

- Next, go back to the form's design screen so that you can see the GUI and the two buttons.
- Double click the Exit button, and you should now be in the code window, ready to create the event procedure for ExitButton\_Click. Type in the following line of code:  
**End** (the End command is used to exit the application)
- To finish up the coding on the Final form, we must also add the comment **'Programmed by (replace the parenthesis and these words by your own first name and last name)** to the form in the general declarations section of the Final form.
- Make sure that your code matches the picture seen on the prior page. Once your picture matches the sample, click the **Save All** button to save the work found on all of your forms.

## The Final Coding

There are a few additional modifications/additions that need to be made to some of the existing forms before the application is completed. These modifications are related to the Final form, which you created just a short time ago. 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 Visual Basic code editor think we were making an error by referring to an object that did not exist at the time. Now, since the Final form does exist, we will add that final bit of code.

- Display/open the code window for the Income form. The first two lines of event procedure EvaluateButton\_Click are the variable declaration Dim statements. Place your cursor after the 2<sup>nd</sup> variable declaration 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. Inside the IF statement, 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()`

Your completed code on the Income form should now match the picture to the right.

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

Private Sub EvaluateButton_Click(ByVal sender As System.Object, ByVal
    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
```

- Click the **Save All** button to save your work.
- Next, display/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/empty line. On that empty line, type in the following variable declaration statement:

`Dim objFormFinal As New frmFinal()`

- Then, place the cursor right behind **End If** and press the Enter key to create a new line. Type in the following code:  
`objFormFinal.Show()`

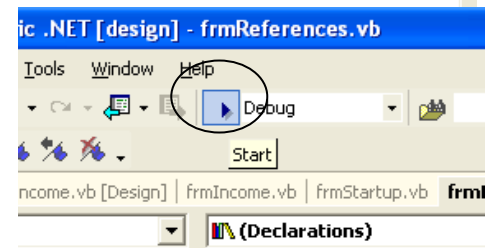
- Finally, find and delete the following entire line of code: `End 'this line of code will eventually be deleted and replaced with something else`

Your completed code on the References form should now match the picture to the right.

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

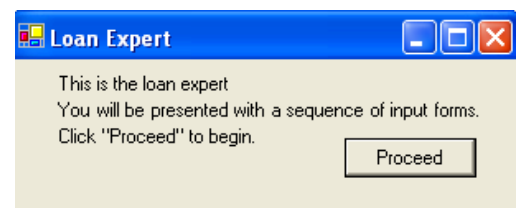
Private Sub EvaluateButton_Click(ByVal sender As System.Object, ByVal
    Dim objFormFinal As New frmFinal()
    If Me.GoodRadioButton.Checked = True Then
        MsgBox("Investigate Further", MsgBoxStyle.Information, "Reco
    Else
        MsgBox("Refuse the Loan", MsgBoxStyle.Information, "Recommen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

- Click the **Save All** button to save your work.
- Now, for the moment of truth. It is now time to run your completed application. To start/run the loan expert system, click on the Start button (circled in the picture on the right).



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 somewhere on your screen, as seen in the next picture. If you have any syntax or coding errors anywhere in your application, the output window should indicate the existence of your error(s) and also where in your code the error(s) is/are found. Correct any errors that you might find, and then 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, it is time to fully test your completed application, again looking for any sort of error that you might have made. If an



error occurs or the program crashes, you will then need to find the source of that error and correct it. Included in the folder called Expert System is a file called **Answer Key**. You can run that file to compare your results with the answer key.

Please look at one of the expert system decision tree diagrams found earlier in this tutorial. Notice that there are seven recommendations that can be made by the expert system, depending upon how the user responded to the different income, employment, references, and education questions that were asked by the expert system. **Test each path/possible recommendation found in the decision tree diagram to make sure that the expert system works properly.** After each recommendation is presented to the user (in the form of a message box), the user should then be presented with the final form, where another consultation session can be run, or the user can decide to quit and stop running the application.

Once you have tested each possible path/recommendation, and everything works, then you are done with this tutorial.