

# LotusScript for the Terrified — An Introduction for Non-Programmers (Updated for R5.X)

Gary Devendorf

The YMCA offers a course called "Swimming for the Terrified." It not only assumes a lack of prior skill, but some degree of apprehension. "Terrified" is also an apt description for a great many potential LotusScript users.

In this age of advancing technology, we operate ATMs, use calling cards, do our work on computers, and program our VCRs. As Notes and Domino developers, we have the power to create applications that make everyone's job easier without any "programming". However, by adding LotusScript programming to your set of tools, you jump to a whole new level of capability.

Writing LotusScript is not difficult. The hardest part is getting started. All the training material I've seen assumes too much, leaving beginners behind. This article is meant as a primer. Like several other Lotus products, Notes and Domino contains LotusScript, a built-in programming language for adding functionality to applications. LotusScript is a *structured language*, which means that, compared with macros and simple actions, it gives you maximum control over an application. But for many, programming with a structured language like LotusScript seems too difficult — maybe even beyond our abilities. This leaves us dependent on others to provide the special Notes functionality we need to do our jobs.

If you fit into this group, or feel you just don't have time to learn something new, let me assure you that LotusScript *is* for you. It is a BASIC language (actually a superset of BASIC), so it's very English-like and easy to read and understand.

In this article, I'll help you to jump in and get your feet wet programming with LotusScript (and that's the last you'll hear of my swimming analogy — promise!). Without assuming any prior programming knowledge on your part, I'll show you how to write simple LotusScript programs that can act as building blocks for more advanced applications. I will keep our LotusScript examples and discussions solely in a Notes context, even though LotusScript is also available in many other Lotus products, such as Approach, 1-2-3, Freelance, and Word Pro. So let's begin to begin.

## **Our LotusScript Application: Some Assembly Required**

An application, such as Notes, is made up of programs. The programs are made up of subroutines. Subroutines consist of commands, functions, methods, variables, constants, and many other elements — much as a house consists of wood, nails, cement, doors, windows, and so on. A house builder doesn't create these discrete parts of a house. He or she just puts them together. As you "build" your first application, you too will just be putting parts together.

There are more "parts" available than we can cover in our introduction to programming. However, we can look at enough of them to build the software equivalent of a doghouse or a swing set. We'll build an application that converts degrees Fahrenheit to degrees Celsius by creating a form, dialog box, and agent that present an interactive UI (user interface). Users will be able to enter degrees Fahrenheit into a field and then click on a Convert button to see the equivalent Celsius degrees.

The building site for our sample application is Notes. You will need a full Notes 4.5 client (or later) to execute the steps I present. For an optimal learning experience, I encourage you to follow along and build your own version of the application from scratch. The exercises are very short. If you have any problem with them or if you want a shortcut, you can download the completed demo database and screen cams from THE VIEW's Web site ([www.eVIEW.com](http://www.eVIEW.com)) to your Notes/data directory; and then add a database icon to your Workspace.

## Your First Program

You will need Domino Designer installed to try these exercises. You can download the finished examples from [www.eVIEW.com](http://www.eVIEW.com).

There are many places where you can use LotusScript in a Notes or Domino application. At this beginner stage, we will keep it simple. Let's start by creating a button and coding the activity that will take place when the user clicks on that button. This button will not ultimately be part of our conversion application; creating it will just help you to get a feel for the tools.


To do this, we must create a database and several small programs.

### To create a local database:

1. Go to your Notes Workspace, and select **File-Database-New** from the menu.
2. Enter the title "My Programs." Notes automatically creates a file named "My Programs.nsf". Leave the template selection as "-Blank-" and click OK.

Now we have a handy place to build our programs.

### To create a program in the new database:

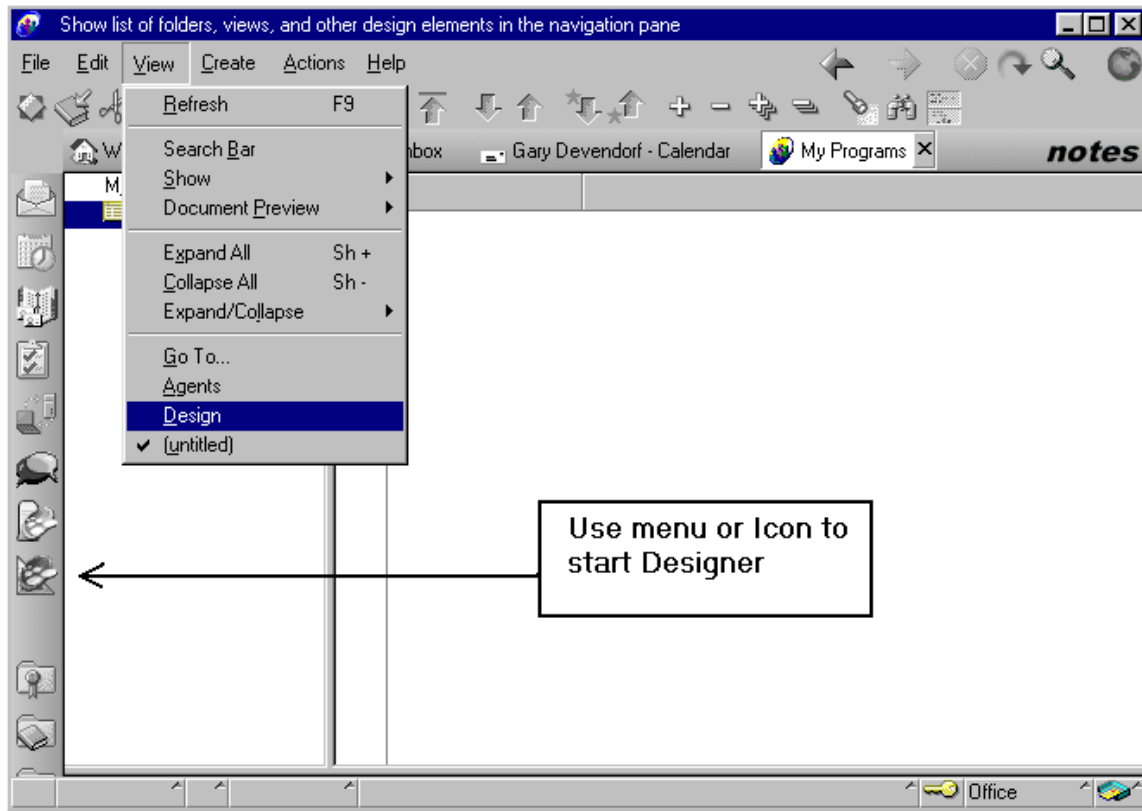
1. Open the My Programs database if it is not already open. It should be empty.
2. Start Domino Designer from the menu **View-Design** or click the Designer Icon (Figure 1).
3. Expand the "My Programs" database section under "Recent Databases" by clicking on the database name. A tree listing of parts of the database are now displayed. Select **Forms**. There are no forms in the database yet.
4. To create a form, click on the  button or select **Create-Design-Form** from the menu. You are now in the Form Designer. The mostly empty screen with an upper and two lower windows that now appears is a blank form in design mode.
5. The upper pane is the *form design area*. It works like a word processor with special fields. Type "This is my form" in the form design area. Your screen should now look like the one shown in **Figure 2**.
6. Next, select **Create-Hotspot-Button** from the menu. A Button Properties box appears. Enter "My Button" as the button label, and close the Properties box.

7. With the button selected, pick **Script** from the **Run** pull down . (You'll find it between the panes.) Code now shows up in the *programmer's pane*, which is the lower window. We're now ready to code the Click event.

8. In the programmer's pane, between the Sub Click and End Sub lines, type the following:

```
Msgbox "Here is my output"
```

The code should look like what you see in **Figure 3**.



**Figure 1 - View Selection for Displaying Design under Folders and Views**

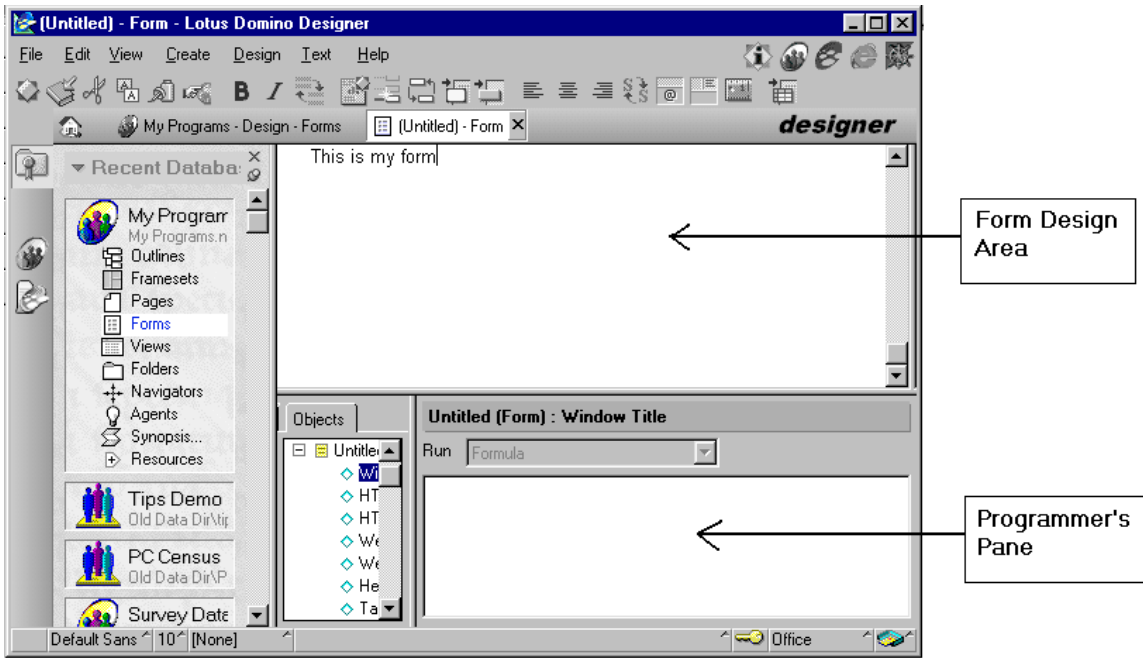


Figure 2 - A Form in Design Mode

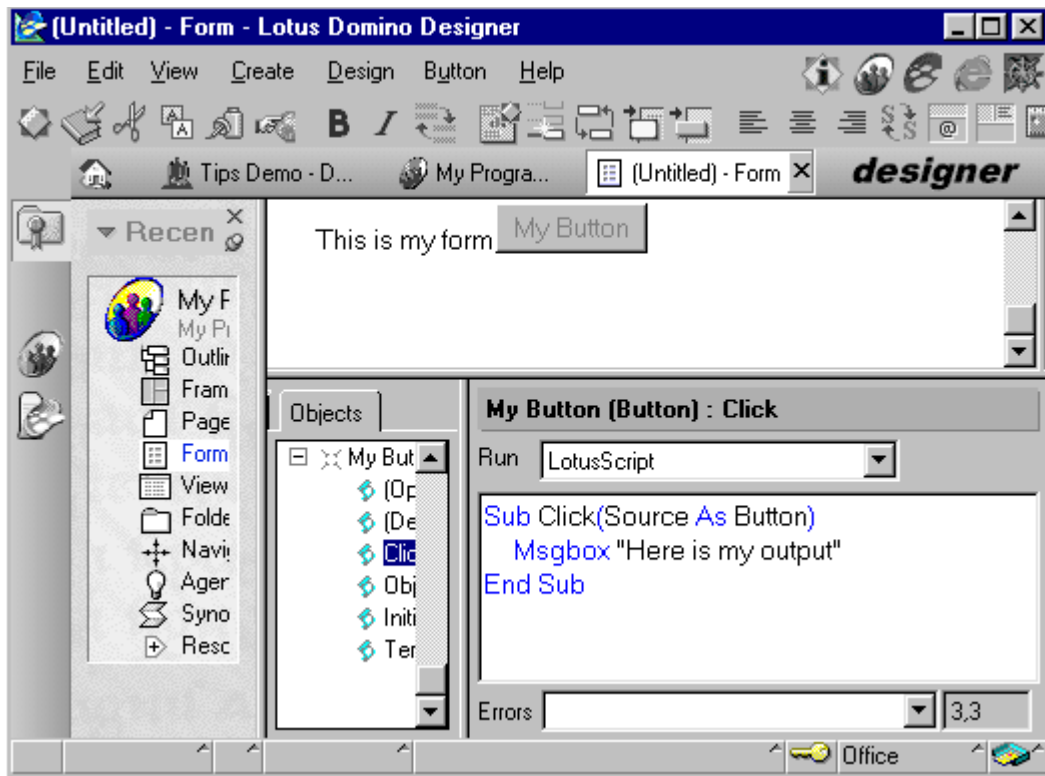


Figure 3 - Button with Scripted Click Event

## To run your program:

1. From the menu, select **Design-Preview In Notes**.
2. When you are asked to save the form, give it the name "My Form." Your form is now running.
3. Click on **My Button** and see the message box. Congratulations! You've just run your first LotusScript program.
4. Click **OK** on the message box. Then press **ESC** to exit the form test and return to the form design screen.

So far, we have created a database, a form, a button, and one line of code that tells the user, "Here is my output." We're now ready to add some of the functionality of our conversion program. We'll add a button that users can click on when they want to enter a temperature in Fahrenheit and have it converted to Celsius.

## To add another button and program:

1. Back in the form design area, click on the white space directly after My Button, which is the one we added in the first exercise, enter a few spaces, and type, "A program by degrees."
2. Using the method you just learned, create another button (**Create - Hotspot-Button**). Label this button "Celsius," and close the Properties box.
3. With the new button selected, pick **Script** from the run pull down. Type the code shown in **Figure 4** between the Sub Click and the End Sub lines.
4. Run your program. (Use **Design - Preview In Notes**.)
5. When asked, "Do you want to save this form?" click on **Yes**.
6. With your form now showing, click on the **Celsius** button. You are told to "Enter a temperature in Fahrenheit." Enter "65." Click **OK**.

A message box tells you that 65 degrees Fahrenheit equals 18.333333 degrees Celsius. Click **OK**.

7. Press **ESC** to get back to the Form Designer.

### Sub Click(Source As Button

```
x = InputBox("Enter a temperature in Fahrenheit")
celsius = (x-32)*(5/9)
Msgbox (x & " degrees Fahrenheit equals " & celsius & "
degrees Celsius")
```

End Sub

Figure 4 - Button Code

### How this program works line by line:

- **Sub Click (Source As Button):** Runs the lines of code shown above when a button is clicked.
- **x = Inputbox("Enter a temperature in Fahrenheit"):** Displays an input box with the prompt text "Enter a temperature in Fahrenheit," and puts the entered value in a variable called x.
- **celsius = (x-32)\*(5/9):** Converts Fahrenheit degrees to Celsius degrees, and puts the result in the variable **celsius**. The **x** in this equation refers to the value of variable x. So, in our program the equation would evaluate to:  $celsius = (65 - 32) * (5/9)$  or: (65 minus 32) times (5 divided by 9). The answer (18.33333) is put in the variable **celsius**.
- **Msgbox (x & " degrees Fahrenheit equals " & celsius & " degrees Celsius"):** Displays a message box with the conversion. The x means "show the value of the variable x." The ampersand (&) that comes after the variable x is a special character used to append one thing to another (i.e., stick things together). Here we append the value of variable x with the words, or string, "degrees Fahrenheit equals" and the value of the variable celsius with the string "degrees Celsius." When the input is "65," the result is "65 degrees Fahrenheit equals 18.3333 degrees Celsius."
- **End Sub:** Stops the running of this program.

## Debugger Keeps You on Track

The first draft of a program typically has errors, or bugs. Finding and fixing them is part of the programming process. (I've often wondered about the etymology of the word debugger. It turns out that early computers actually had problems with moths flying into their electronic tubes, just like a porch light, shorting out the circuit and causing programs to act unpredictably!)

LotusScript comes with a debugger tool that helps you find "bugs," problems, in a program. The Debugger works by stepping through a program, running it line by line, and showing what happens. It's like running videotape frame by frame to understand all the information presented in a home improvement show. We'll use it to test our program now.

### To run the Debugger:

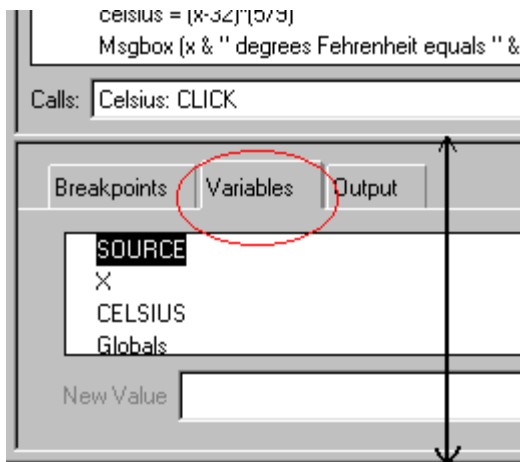
1. At the form design screen, select **File-Tools-Debug LotusScript** from the pulldown menu to start the Debugger.
2. Once again, run your program (**Design - Preview In Notes**). This time, when you click on the **Celsius** button, the Debugger shows up. See that your first line of code is highlighted and that the Debugger is waiting for your instructions (**Figure 5**).

3. In the Debugger's output section in the bottom half of the screen, select the **Variables** tab.

If you don't see the Variables tab, you may have to expand the hard-to-see window. Hidden at the bottom of the window is another window.



With your mouse pointer, drag this bar up so it looks like this:



Click on the **Variables** tab.

4. Click on the **Step Into** button. The line of code is executed and an input box appears.
5. Enter the number 65, and click **OK**. Notice that the number you entered (65) now appears next to the variable x.
6. Click on the **Step Into** button again. A value for the variable celsius appears (18.333333).
7. Click on the **Step Into** button again to display the message box with the results.
8. Click **OK** and then on the **Step Into** button one more time to complete the execution.
9. Press **ESC** to get back to the Form Designer.

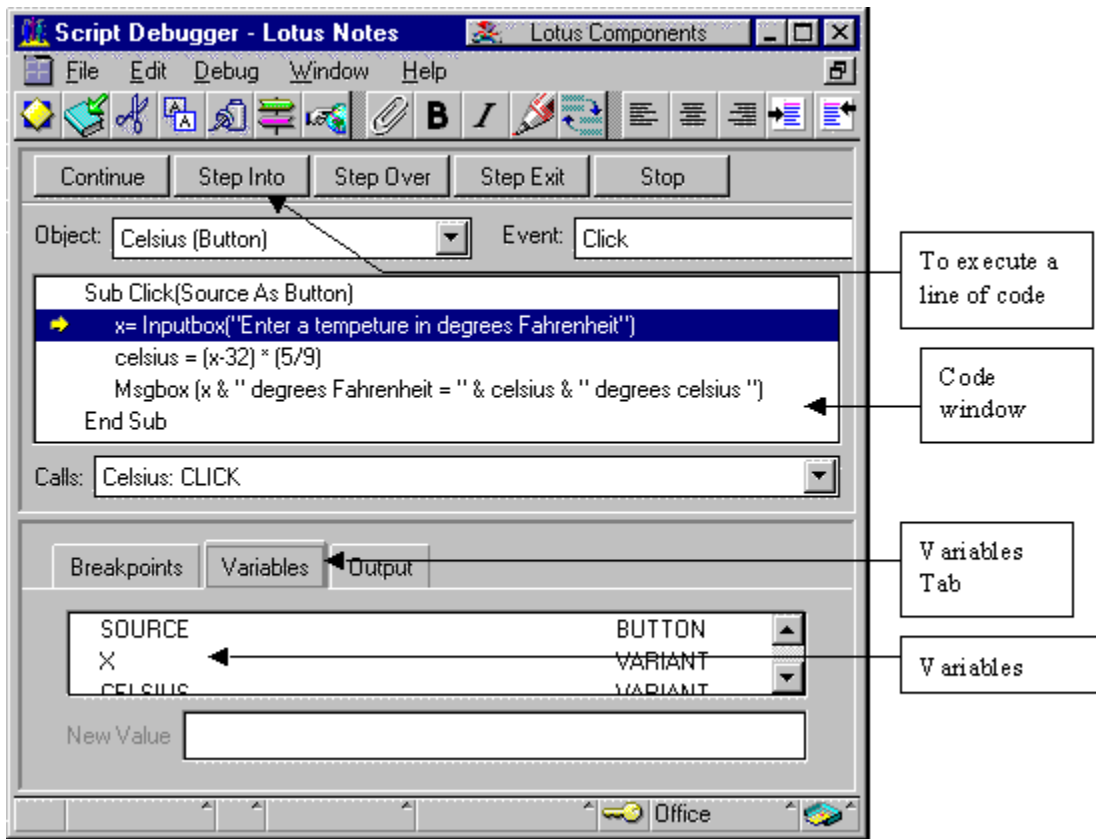


Figure 5 - The Debugger

So far, so good, but now let's see if we can find a bug. Just like before, run your program (**Design - Preview in Notes**). Remember, the Debugger is still on because we didn't turn it off. When the Debugger pops up, select the **Variables** tab, and click on the **Step Into** button. This time, instead of entering "65," enter "really hot" in the input box. Click **OK** and then **Step Into**. An error message pops up stating, "Type mismatch." What happened?

Look at the value for the variable x (really hot). Now look at the line of highlighted code "celsius = (x-32)\*(5/9)." If we substitute the value of the x variable in that line, we get "celsius = ("really hot"-32)\*(5/9)." The part of the code that tries to subtract 32 from the text "really hot" doesn't make sense; therefore, the error occurs. This error is fairly obvious. Many errors are harder to see.

Let's stop this run-through (Click **OK**, then **Stop**, and press **ESC**) and run the test again. This time, enter "123", being sure to use the quotation marks.

An error says "Type mismatch" again. What happened this time? If we look at the execution in the Debugger, we can see that we tried to subtract 32 from what we thought was a number. But quotation marks around a value means the value is a string of characters, while no quotation marks means it's a number.

Numbers and characters are different "Types" of data that are stored differently in computer memory and are treated differently in code. So "123" really means the character 1, then the character 2, and the character 3. It doesn't represent the integer one hundred and twenty-three. To subtract 32 from "123" is equivalent to subtracting 32 from words or "strings," which doesn't

make sense. Hence the error. Our program should not allow the user to enter the wrong kind of the data in the field.

To force LotusScript to treat a variable as a particular Type, thus ensuring that users entering the wrong type of data receive an instant error message, we do something known as "declaring" a variable by using the Dimension (Dim) statement. It is good programming to Dim variables to keep LotusScript from guessing what Type was intended: string, numeric, or variant (any type of data). To fix the problem we found here, we need to add a Dim statement to our code that declares the variable x to be a type of number (Dim x As Single). We won't do that now, in the interest of keeping our sample short. In the exercises that follow, you'll see that we Dim just enough to make the samples work.

**To force LotusScript to treat a variable as a particular Type, we do something known as "declaring" a variable by using the Dimension (Dim) statement. It is good programming to Dim variables to keep LotusScript from guessing what Type was intended: string, numeric, or variant (any type of data).**

## Logical Errors:

Some errors are not seen as errors by the program, yet yield wrong results when the program runs. These are *logical* errors. The Debugger helps catch logical errors as well as program errors. We can see how by using a *comment* to create a logical error.

Comments are notes you write in the program to explain what your code is supposed to be doing in a particular part of the program. The apostrophe character (') in LotusScript code means that the rest of the line is a comment. In other words, the line is only commentary. Putting an apostrophe in front of a line of actual code is a handy way to skip that line while testing.

Try putting an apostrophe before "celsius = (x-32)\*(5/9)" in your code. You will need to be in the Form Designer with the Celsius button selected to edit your code. You will see the line turn color (usually to green), indicating that the line is a comment. Now run the program and enter "65" in the input box.

The program runs without an error. But look at the message box. It says "65 degrees Fahrenheit equals degrees Celsius." This is not the result we want. If we step through the program in the Debugger, we see that the variable **celsius** never gets a value. Appending nothing, which is the value of **celsius**, to the string does not cause an error or change the string. The program is doing just what it was told to do.

The fix, of course, is to remove the apostrophe, so that the line that gives the variable celsius a value is read as code once more.

## What You've Learned So Far

At this point, you've learned how to use several of the major LotusScript building blocks. You created a program with input, output, variables, and calculations. You watched the Debugger find an error, and you checked the values of variables while stepping through the program's execution. These are extremely important skills. In terms of our house-building analogy, you've learned the equivalent of using a hammer, saw, and nails to pound, cut, and fasten. You can

accomplish a great deal using just these tools and skills. There are some other tools, however, that you can use to build your projects faster and better.

## LotusScript Classes - Getting a Head Start with Prebuilt Parts

The LotusScript language provides a great deal of functionality, but its real strength comes from the "classes" found in its hosting application (e.g., Notes, Approach, 1-2-3, Freelance). A *class* is used to build-in a function or feature of a Lotus product to an application. Think of classes as ready-made parts you can use in your program, like the parts you bring to a construction site — doors, windows, sinks, bathtubs, lights, water heater, and so forth. Each of these objects serves a particular purpose, and each is pre-made.

Domino Objects, AKA Notes classes or LotusScript classes, are the set of tools for adding Notes and Domino features and functions to your applications. The classes available in another Lotus product, like Approach, are different. Notes 5.0 has 40 classes to help write Notes-centric applications. We'll use two of them in our sample application — the UIWorkspace and the UIDocument class.

Before beginning to code, let's take a closer look at the elements used with classes. Classes have properties, methods, and events.

- **Properties** describe an object (which is an "instance" of a class). Just as size, color, type of lock, and handle are properties of a door, the Notes Document class has the properties Author, Date Created, Signer, and Size, among others.
- **Methods** are actions or verbs. For a door, some methods could be Open Door, Lock Door, and Close Door. The NotesDocumentCollection class has the methods GetFirstDocument, GetNextDocument, PutAllInFolder, and more.
- **Events** are actions. Consider the Button class. One of its events is the Click event (i.e., a user clicks on the button); the Click event is where we have been putting our programs so far.

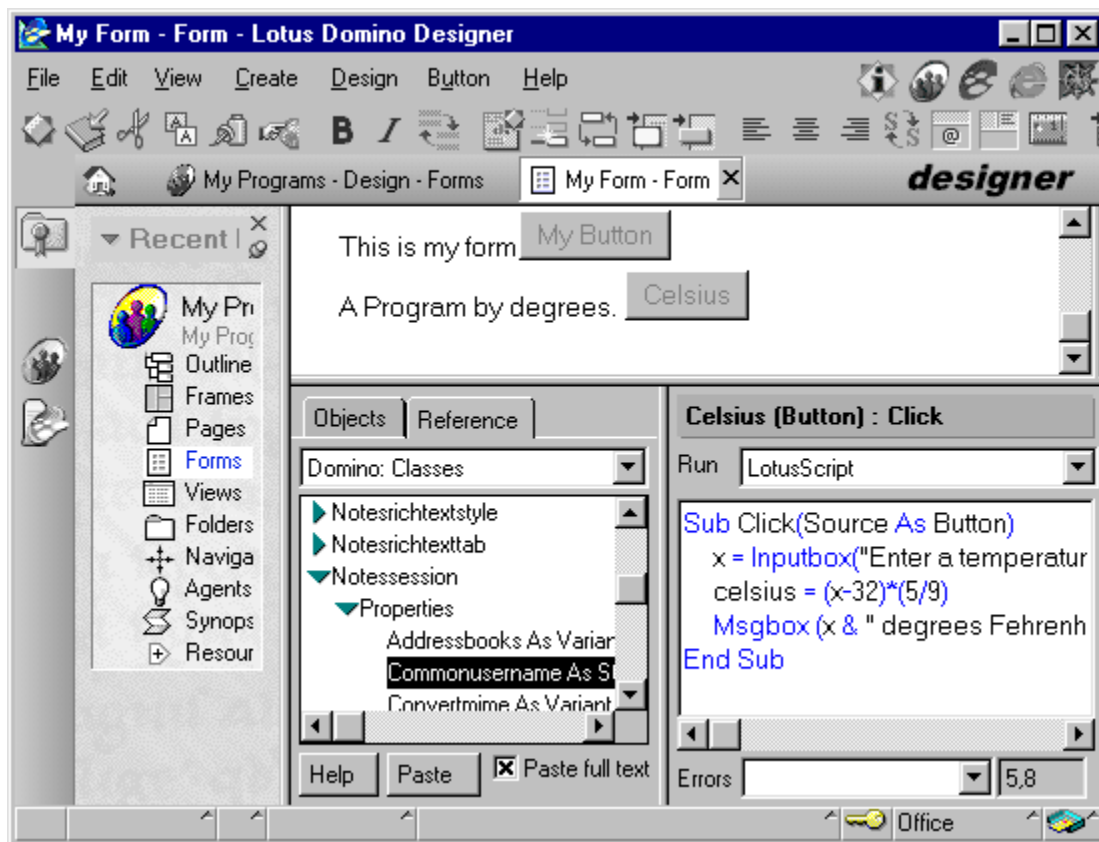
**A class is used to build-in a function or feature of a Lotus product to an application. Think of classes as ready-made parts you can use in your program, like the parts you bring to a construction site — doors, windows, sinks, bathtubs, lights, water heater, and so forth. Each of these objects serves a particular purpose, and each is pre-made.**

Not all classes have all three elements. For example, the Button class has no properties and no methods; it has two events.

How do you learn all the hundreds of Notes classes, methods, properties, and events — not to mention the LotusScript commands used with them? As all programmers know, you don't. Just as our house builder does, you look for the parts you need in a catalog. Thankfully, several tools exist to help you obtain complete information on each class.

To see one of these tools, go to the Form Designer, where we edited our code, and highlight one of the buttons so that code appears in the programmer's pane. Now click on the **"Reference tab"**

and pick **“Domino: Classes”** from the pulldown list. A tree list of all the Domino Objects now appears below. See **Figure 6**. Select the class method, property, or event you are interested in. I selected **“Commonusername”**. If I now click the **Help** button or press the **F1** key, I will quickly see the corresponding help documentation. After looking at this documentation, I simply press **ESC** to get back to the Form Designer.



**Figure 6 - Accessing Help documentation for Notes Classes**

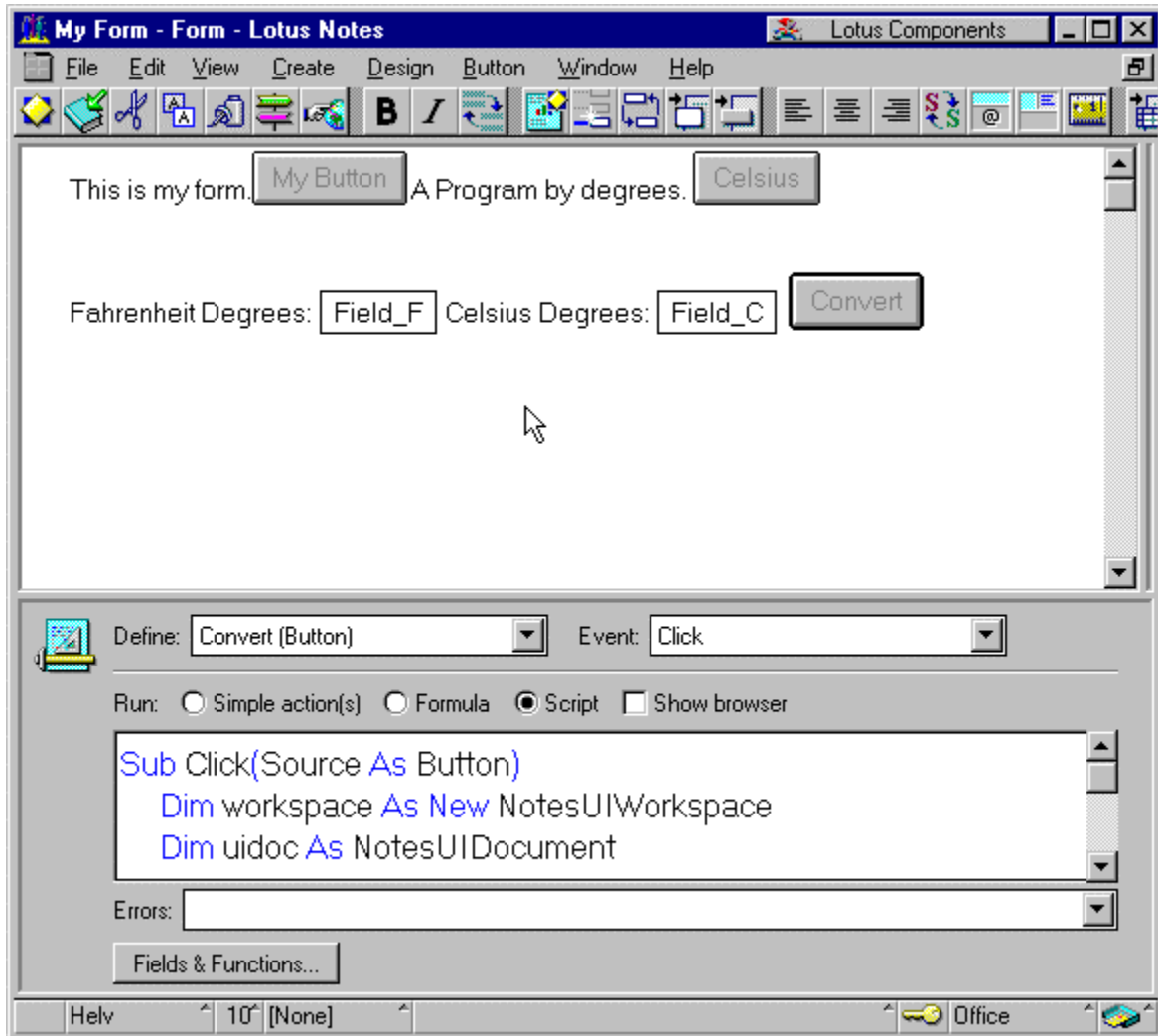
**How do you learn all the hundreds of Notes classes, methods, properties, and events? As all programmers know, you don't. Just as a house builder does, you look for the parts you need in a catalog.**

You can also easily find help on all the LotusScript commands. The process is the same as for Notes classes. In the Form Designer, with code showing in the programmer's pane, click on the **Reference tab** and select **“LotusScript Language”** from the pulldown list. Scroll through the LotusScript sections of the browser, expand the **All** section by clicking on it, scroll down to the command **“InputBox”**, and click on it. Now a click on the **Help** button or pressing **F1**, help documentation for the command appears. Press **Esc** to return to the form editor.

Another source for the latest Notes classes is the Lotus Web site:

<http://doc.notes.net/lotusscript/lotusscript.nsf>

We'll see how classes add functionality to an application as we add two new fields to our form: a field where the user can enter the Fahrenheit degrees and a field where the equivalent Celsius temperature displays after the user clicks on a Convert button, which we'll also add now. This is an alternative to how our application currently functions. The new fields and the Convert button we will be adding are shown in **Figure 7**.



**Figure 7 - Form with Temperature Fields and Convert Button**

**We'll see how classes add functionality to an application as we add two new fields to our form: a field where the user can enter the Fahrenheit degrees and a field where the equivalent Celsius temperature displays after the user clicks on a Convert button, which we'll also add now. This is an alternative to how our application currently functions.**

## To add fields and a Convert button to MyForm:

1. Go to the form design area and enter two carriage returns (new lines) after the Celsius button.  
Type:

"Fahrenheit Degrees: "

2. Create a field after the text (**Create-Field**). Name the field "Field\_F" and set its type to "Number." Close the Properties box.
3. Add some spaces after Field\_F and type:

"Celsius Degrees: "

4. Create a field after this text (**Create-Field**). Name the field "Field\_C" and set its type to "Number." Close the Properties box.
5. Create a new button (**Create-Hotspot-Button**) and label it "Convert." Close the Properties box.
6. With the Convert button selected, click on the **Script** (for LotusScript) radio button. Now add the following code between the Sub Click and End Sub lines:

```
Sub Click(Source As Button)

Dim workspace As New NotesUIWorkspace
Dim uidoc As NotesUIDocument
Dim celsius As String
Set uidoc = workspace.CurrentDocument
x = uidoc.FieldGetText( "Field_F" )
celsius = (x-32)*(5/9)
Call uidoc.FieldSetText("Field_C", celsius)

End Sub
```

## Running your program in debug mode:

1. Run the program (**Design-Preview In Notes**). Select **Yes** when asked if you want to save the form. Remember, the Debugger is still on -- you can turn the debugger on and off with the menu pick **File-Tools-Debug LotusScript**
2. In the Fahrenheit field, enter "65" and click on the **Convert** button. The Debugger screen will appear.
3. Now click on the **Variables** tab of the Debugger and Step Into each line of code just as you did in the earlier exercise.

Here is a breakdown of what happens:

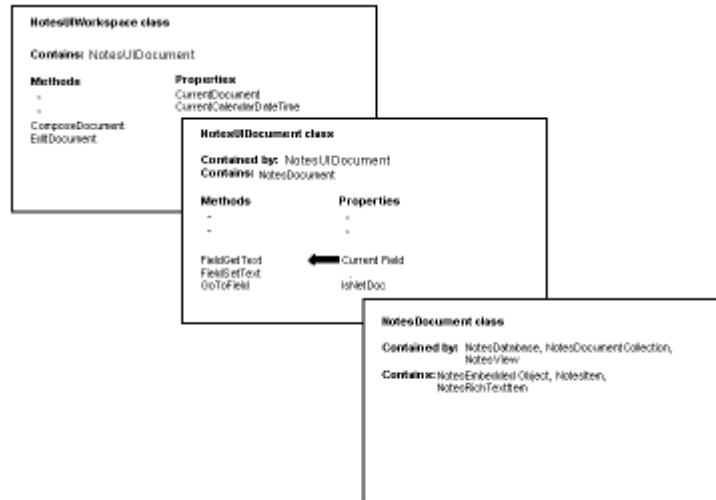
- **Sub Click (Source As Button):** Runs this code when the button is clicked. The button Click event is triggered.
- **Dim workspace As New NotesUIWorkspace:** Creates an object called Workspace that is of type NotesUIWorkspace. What the heck does that mean? Well, going back to our house building example, you can think of NotesUIWorkspace as something like a foundation. Just as a builder must first create a foundation and then a wall before installing windows, we need to work with a hierarchy of Notes classes. The NoteUIWorkspace is the foundation for Notes classes that control the Notes user interface (UI). Notes fields are part of the UI, so we must create a NotesUIWorkspace object on which to build field objects. In our building example, Field\_F and Field\_C would be the windows.

Now that we've created our NotesUIWorkspace object, we can use any of the 26 methods and 4 properties found in its class. We'll use the CurrentDocument property.

- **Dim uidoc As NotesUIDocument:** Declares (Dims) a variable, called uidoc, of type NotesUIDocument. We are just making a "pocket" in memory big enough to hold an object called NotesUIDocument. The NotesUIDocument is like the wall in our building example.
- **Dim celsius As String:** As in the earlier example, Dims a variable called celsius that can hold strings of characters.
- **Set uidoc = workspace.CurrentDocument:** Sets the uidoc variable to "workspace.CurrentDocument." Notice the "workspace" part that is our foundation. We are saying, *Take the current document, our form now on screen, and store it in the object variable uidoc.* CurrentDocument is a property of the NotesUIWorkspace. Now that we have a NotesUIDocument, we can use any of the methods and properties in the NotesUIDocument class.

**Just as a builder must first create a foundation and then a wall before installing windows, we need to work with a hierarchy of Notes classes. The NoteUIWorkspace is the foundation for Notes classes that control the Notes user interface (UI). Notes fields are part of the UI, so we must create a NotesUIWorkspace object on which to build field objects.**

- **x = uidoc.FieldGetText( "Field\_F" ):** Variable x gets the value that was typed into Field\_F. To get the text contained in Field\_F, use the FieldGetText Method. To be able to use this method, we first had to create a NotesUIWorkspace object so that we could then create a NotesUIDocument. Then we could use this method on Field\_F, which is on the current document that is running in Notes. **Figure 8** shows the hierarchy. To see a full graphic representation of all Notes classes, with access information and examples, go to this URL on the Web: [www2.lotus.com/notesdev/lotusscript.nsf](http://www2.lotus.com/notesdev/lotusscript.nsf).
- **celsius = (x-32)\*(5/9):** Takes the value of x, calculates the value of Celsius, and puts the result in the variable celsius.
- **Call uidoc.FieldSetText("Field\_C", celsius):** Calling the FieldSetText method puts the value of variable celsius into the field Field\_C. A "call" is required to execute a method that doesn't return a value. The "uidoc" part of UidocFiedSetText specifies the object running this method. Earlier we set uidoc to the current document. So, uidoc.FieldSetText ("Field\_C", celsius) puts the value of variable celsius into Field\_C on the current document (uidoc).
- **End Sub:** Stops the running of this program.



**Figure 8 - Hierarchy of Notes UIWorkspace and NotesUIDocument Classes**

That's it. Using class objects is the most difficult concept presented here. But once you understand it, programming becomes very easy.

It's time to exit the Debugger and complete the design of our form. The LotusScript Debugger stays on until you turn it off. Since we don't want it to pop up every time Notes tries to run some LotusScript, let's turn it off now (**File-Tools-Debug LotusScript**). When the program has completed, press **ESC** and don't save the document when asked. (In this application we are using a form to interact with the user, not to create documents.)

## **Adding a Layout Region, Fields, and Field Exit Events**

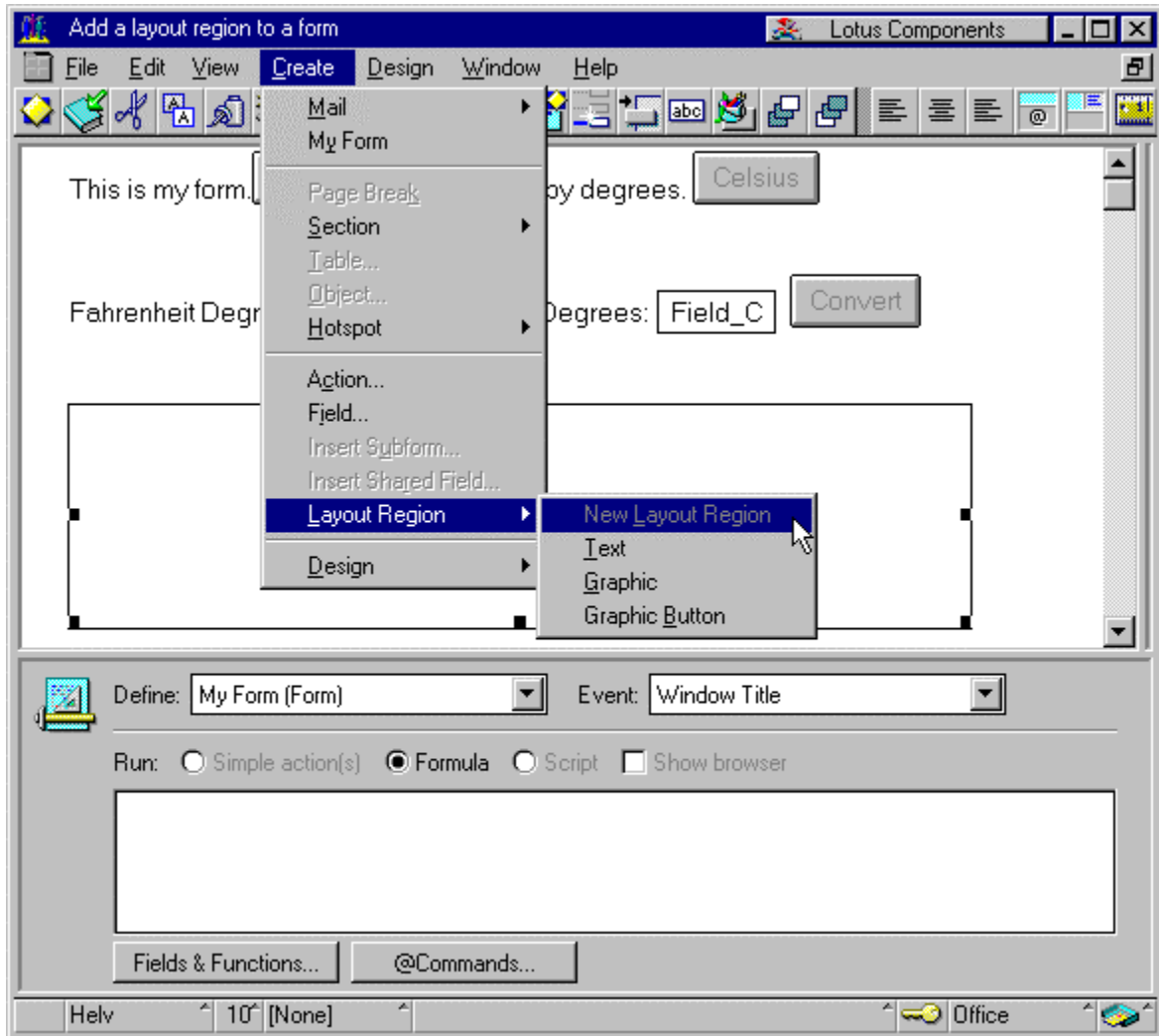
Our application is coming along nicely. Our conversion program works. We can make it better, however. Instead of a one-way conversion from Fahrenheit to Celsius, we'll enable users to enter a temperature in either Fahrenheit or Celsius and have it converted to its equivalent. A user will enter a temperature in either field; the conversion will take place automatically as the field is exited, and the converted temperature will display in the other field. The user will not have to click a button to initiate the conversion. Instead of the bracketed fields we added in the last exercise, we'll create boxed, 3D-style fields to display the temperatures.

To add the two new fields with text, we need to add a layout region. A layout region is a special graphical area that lets you position objects, such as fields and text, anywhere within it. To make the conversion take place automatically, we need to add field exit events to the new fields. The field exit events are triggers that cause our LotusScript program to run when the user exits a field after making an entry to it.

Begin by opening My Form in the Form Designer.

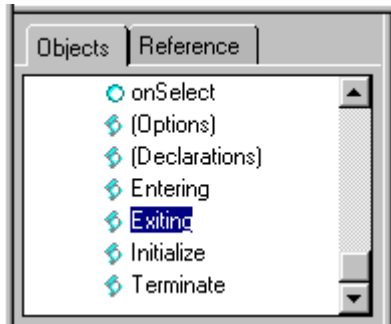
### **To add a layout region, 3D-style fields, and field exit events:**

1. Place your cursor to the right of the Convert button and add a few new lines.
2. From the menu, select **Create-Layout Region-New Layout Region**, as shown in **Figure 9**.



**Figure 9 - Creating a Layout Region**

3. Right mouse click on the new layout region and select **Layout properties** to bring up the Properties box. Put a check in the box next to **3D style** and close the Properties box. With the Layout region still highlighted, select **Create-Field** from the menu.
4. When the new field appears, double click on it to bring up its Properties box. Enter the name "F\_F" and close the Properties box. Move the field to the left side of the layout region. Now create a second field the same way and name it "F\_C."
5. Highlight field **F\_F** and select the **Exiting** event; under the **Objects** tab in F\_F tree.



- Put the following code between the Sub Exiting and End Sub lines. It is very similar to code we entered before, so I won't explain it line by line.

```
Sub Exiting (Source As Field)
Dim workspace As New NotesUIWorkspace
Dim uidoc As NotesUIDocument
Dim celsius As String
Set uidoc = workspace.CurrentDocument
x = uidoc.FieldGetText( "F_F" )
celsius =Format$(((x-32)*(5/9)),"Fixed")
Call uidoc.FieldSetText("F_C", celsius)
End Sub
```

- Do the same for field F\_C. Highlight field **F\_C** and select the **Script** radio button; in the **Event** pulldown, select "Exiting."

- Put the following code between the Sub Exiting and End Sub lines.

```
Sub Exiting(Source As Field)
Dim workspace As New NotesUIWorkspace
Dim uidoc As NotesUIDocument
Dim fahrenheit As String
Set uidoc = workspace.CurrentDocument
x = uidoc.FieldGetText( "F_C" )
fahrenheit =Format$((x*(9/5)+32),"Fixed")
Call uidoc.FieldSetText("F_F", fahrenheit)
End Sub
```

- To add the text that will appear above the fields, select **Create-Layout Region-Text** from the menu.
- Double click on the new text to bring up its Properties box and enter the text "Fahrenheit." Drag this text above the F\_F field.
- Now create text for the F\_C field the same way. Select **Create-Layout Region-Text**; double click on the new text to bring up its Properties box, and enter the text "Celsius." Move this text above the F\_C field.

The form now looks like the one shown in **Figure 10**.

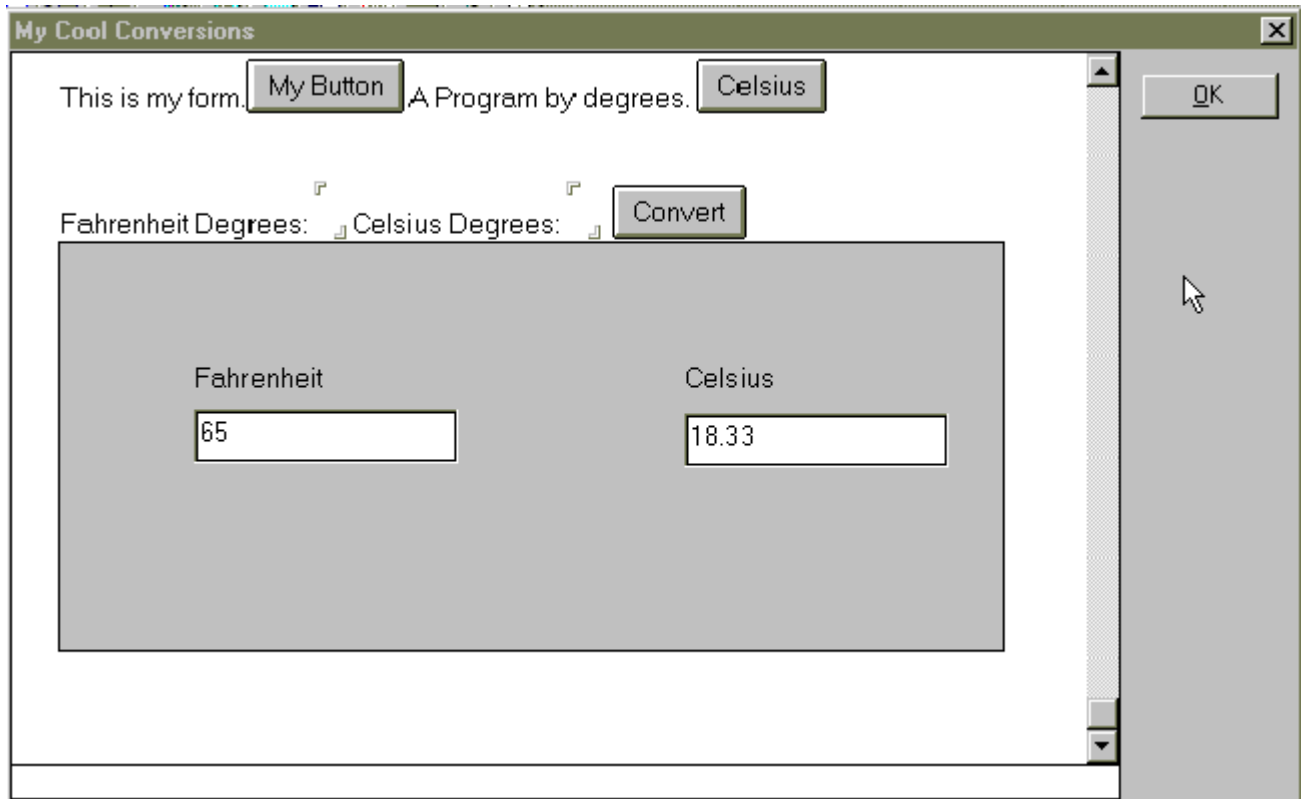


Figure 10 - Completed Conversion Form with Layout Region

### Test the form:

1. Select **Design – Preview In Notes**.
2. Enter "65" in the **Fahrenheit** field and click on the **Celsius** field. By clicking on the Celsius (F\_C) field, you are exiting the Fahrenheit (F\_F) field. This causes the code you put in the Exit event for the F\_F field to run. That code reads the F\_F field value, converts it to Celsius, and writes it to the F\_C field.
3. Now enter "18" in the **Celsius** field and click on the **Fahrenheit** field — thereby running the code in the F\_C field's Exit event. This code reads the value in field F\_C, converts it to Fahrenheit, and writes the value (64.40) into the Fahrenheit (F\_F) field.
4. Press **ESC** to exit the test. Don't save the document when prompted. We will fix this prompt later.

## It's Time for a Final Coat of Paint

The functionality of our application is now complete, but we can make it prettier. Our conversion form looks and works fine except for the way it starts and ends. To start it, we've been going to

the Form Designer and selecting **Design – Preview In Notes**. We could run it by selecting **Create-MyForm** instead, but we'd still have that annoying, "Do you want to save this document" message when we exit the form. These problems can be fixed by creating an agent that uses the dialogbox method. It would also be nice to have a different background color.

### To change the background color:

1. In the Form Designer, right mouse click on the form design area and select "Form Properties" from the pop-up list.
2. Select the **center** tab and then a color from the color pulldown list. (Light colors work best.)



3. With the Properties box still showing, click on the Layout Region to see its properties.



4. Close the Properties box and the Form Designer. We have finished working with My Form.

**Our conversion form looks and works fine except for the way it starts and ends. To start it, we've been going to the Form Designer and selecting Design-Test Form. We could run it by selecting Create-MyForm instead, but we'd still have that annoying, "Do you want to save this document" message when we exit the form. These problems can be fixed by creating an agent that uses the dialogbox method.**

### To create an agent (action) to call "My Form" using the dialogbox method:

1. To bring up the Agent Designer screen, Go back to the "My Programs – Design – Form" window and click on the **Create** menu and select "**Agent...**" in the My Programs database (**Figure 11**).

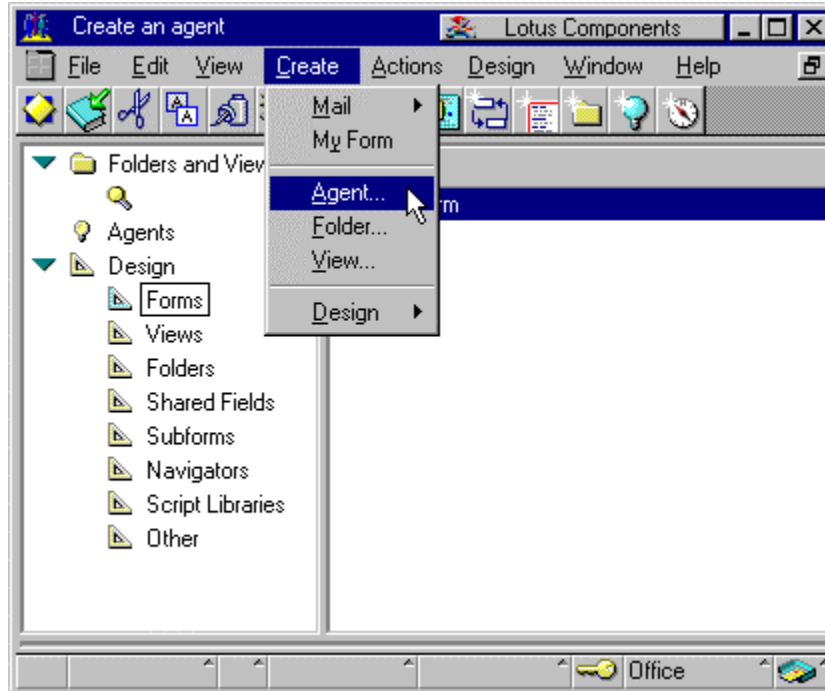


Figure 11 - Selecting the Agent Designer

2. In the Name field, enter "Conversions" and check **Shared Agent** (Figure 12).

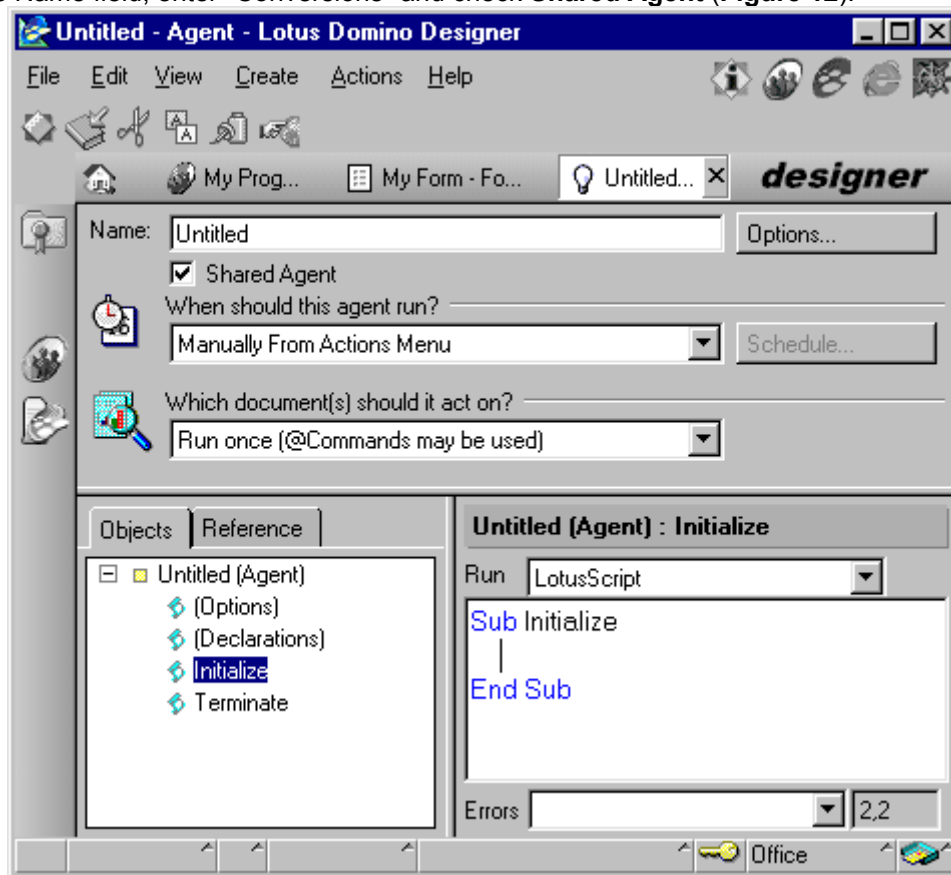


Figure 12 - Agent to Call Form Using Dialogbox Method

3. The **When should this agent run?** pulldown should be set to "Manually From Actions Menu."

The **Which document(s) should it act on?** pulldown should be set to "Run once (@Commands may be used)."

4. Select "LotusScript" from the **Run** pull down and select the Initialize event.

5. Enter the LotusScript code presented in **Figure 13** between the Sub Initialize and End Sub lines.

```
Sub Initialize  
Dim workspace As New notesUIWorkspace  
Dim session As New NotesSession  
Dim db As NotesDatabase  
Dim doc As NotesDocument  
Set db = session.currentDatabase  
Set doc = db.createDocument  
Call workspace.dialogbox( "My Form" ,,, True,True,True,False,  
"My Cool Conversions" ,doc )  
End Sub
```

**Figure 13 - The Initialize Code**

6. Save your agent. Try the agent by selecting **Actions-Conversions** from the menu.

7. Click **OK** to exit the Conversion application.

Our conversion form still has the text and button from our first exercise, the original set of Fahrenheit and Celsius fields, and the Celsius button — all created before we improved our application.

If you were presenting this application to users, you'd want to clean up the conversion form by removing these elements. I suggest keeping them, however, because you may want to write another application sometime that utilizes one of these elements, and you'll be able to see here how you created it.

## **Moving to the Next Level of LotusScript Programming**

In these exercises you created a database, form, buttons, fields, a layout region, and an agent. You wrote code for Click events, Exiting events, and an agent. The application you just built can be used as the starting point for other applications.

One programmers' secret is that all programmers copy pieces of their previous programs to help with new programs. As a bonus exercise, try adding the following fields and conversions to My Form:

Miles = (Kilometers \* 0.62)  
Kilometers = (Miles \* 1.61)

Pounds = (Kilograms \* 2.21)  
Kilograms = (Pounds \* 0.45)

Fluid\_ounce = (Milliliters \* 0.03)  
Milliliters = (Fluid\_ounce \* 29.57)

Another programmers' secret is that we don't know everything, but we know how to look it up. There is much more to LotusScript than what I've shown you here. I encourage you to look things up in Help and also to check out these two sources on the Web:

<http://doc.notes.net/lotusscript/lotusscript.nsf>

[www.lotus.com/idd](http://www.lotus.com/idd)

Finally, Notes templates, such as the Mail template, contain tons of LotusScript. Look at the built-in forms and agents in the Form Designer to see how Lotus uses LotusScript. You'll get important clues on how to advance your LotusScript programming skills.

*Gary Devendorf has worked for Lotus and Iris for more than 6 years. He is the Product Manager for all things App Dev in the Notes and Domino space (LotusScript, Domino Objects, XML, COM toolkit, Domino Collaboration Objects, Java, Agent Manager, Web services, and more). Gary is a Notes developer and provides code demos of a large variety of areas. You can find his demos and presentations at "Gary's Page" (<http://lotuspro.e-promag.com/garyspage>). Gary holds bachelor's degrees in Industrial Engineering and Geography. In addition, he is a pilot and sky diving instructor.*