

Raise Events from Visual Basic 6 Collection objects

I recently received an email concerning some code in my Learn to Program Objects with Visual Basic book.

In chapter 6 of the book, I illustrated how you can create a Class Module (the Dish Class) whose objects (Dish Objects) can 'raise' events back to the code that instantiates the object. Events, when used in this manner, are an important means of communication between an object and the code using the object.

What's the question then?

In Chapter 7, when we created a Collection Object called Order which itself created the Dish Object from Chapter 6, we no longer raised an event from the Dish object. Is this possible? Is it possible to raise an event from the Dish Object to the Order Collection Object?

The answer is 'Yes', but before we do not, let's first review what we did to raise the event out of the Dish Object.

A Quick Review of Events

All that's required is that you declare the Event in the General Declarations section of your class module, and then 'raise' the event using the RaiseEvent statement. Let's review the code from the China Shop project. This statement was placed in the General Declarations Section of the Dish Class Module

Public Event DataChanged(Item As String, Value As Variant)

What we've done here is to declare an event called DataChanged which will pass two arguments, Item and Value, back to the program that creates an instance of the Dish Object from this class. This code

RaiseEvent DataChanged("Quantity", m_strBrand)

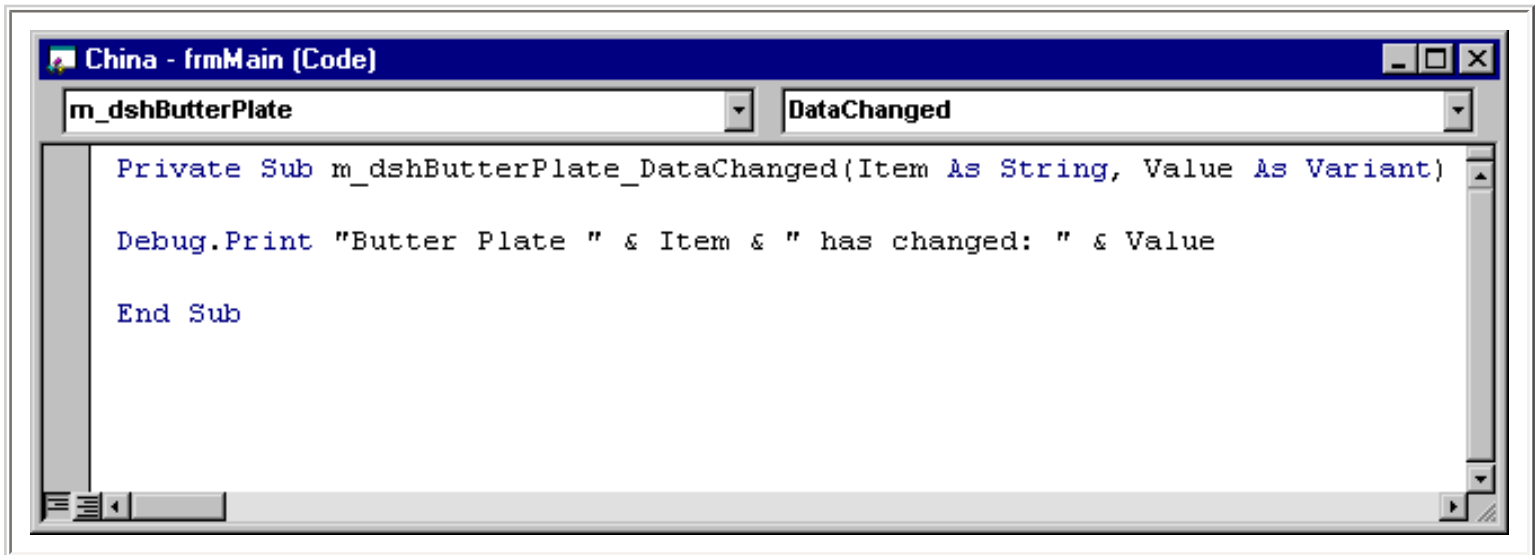
was placed in the Property Let procedures of the Brand Property (and similar code was placed in other Property Let procedures as well)--the idea being that when one of the properties of the Dish object was changed, we alert the calling program to this fact.

That's it for the code in the Dish Class Module--we then needed to take steps, within the Main form of the China Shop project to react to the event. This is a two step process.

First, we need to declare objects from the Dish Class module, using the 'WithEvents' keyword to tell Visual Basic that the Dish object raises events, and we wish to handle them. We placed this code in the General Declarations Section of the form...

Private WithEvents m_dshPlate As Dish
Private WithEvents m_dshButterPlate As Dish
Private WithEvents m_dshSoupBowl As Dish
Private WithEvents m_dshCup As Dish
Private WithEvents m_dshSaucer As Dish
Private WithEvents m_dshPlatter As Dish

This needs to be done in the General Declarations Section of the form so that 'event procedures' for each of the Object variables that we declare appear in the Dropdown list of objects within the code window. Once we find the appropriate event procedure, we can then place code of our choice in the event procedure to react to the event when it is raised out of the Dish object...

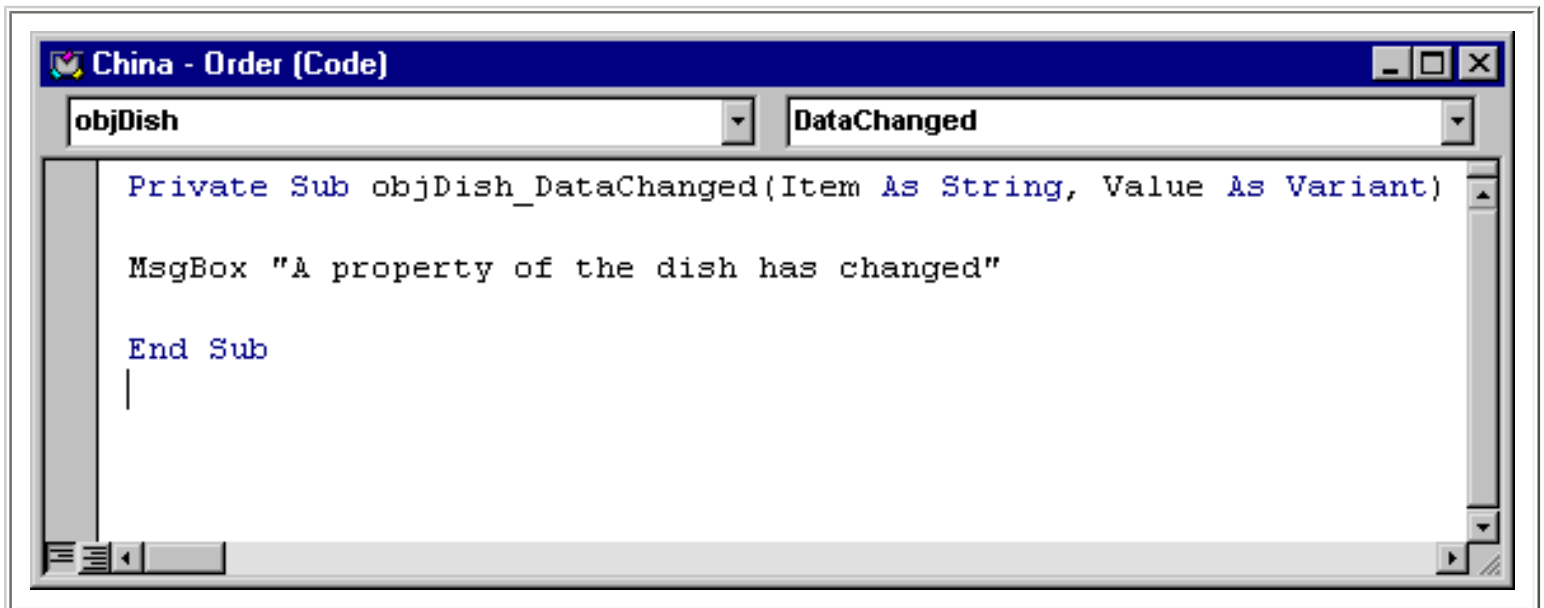


Raising an event from the Dish object to the Collection Object

Raising an event from the Dish Object to the Order Object is no problem--and we don't need to change the Dish Object at all--all of the work is done using the Collection Object. The first thing we need to do is change the declaration of the Dish Object in the Collection Object so that Visual Basic knows we wish to accept events raised from the Dish Object. Let's take the declaration of the Dish Object out of the Add Method of the Collection Object, and move it to the General Declarations Section instead (remember, for the WithEvents keyword to produce an Event Procedure in the dropdown ListBox, we need to declare the object in the General Declarations Section.)

Private WithEvents objDish As Dish

Once we do that, we should see an event procedure listed in the dropdown ListBox of the Collection Class...



If we run the program now, and click on the Brands ListBox, we'll receive a message telling us that the brand of china has been changed to Corelle.



Similarly, if we select a Plate, we'll receive a message telling us that the item of change has been changed to 'Plate'..



Taking this a step further

It's possible for the Dish Object to pass even more information back about itself to the Collection object by raising an event---the event can be raised an instead of passing back two arguments consisting of the item that has been changed and its value, we can also have the Dish object pass back a reference to the Dish object itself. What this means is that the Collection object would have direct access to each of the Properties of the Dish Object.

To do that, we'll need to change the way the DataChanged event is declared and raised within the Dish Object. Let's look at the Declaration first...

Public Event DataChanged(Item As String, Value As Variant, objDish As Dish)

As you can see, this declaration has three arguments, with the third argument being a Dish Object. This tells Visual Basic that the DataChanged event, when raised, will be passing a reference to a Dish object along with it.

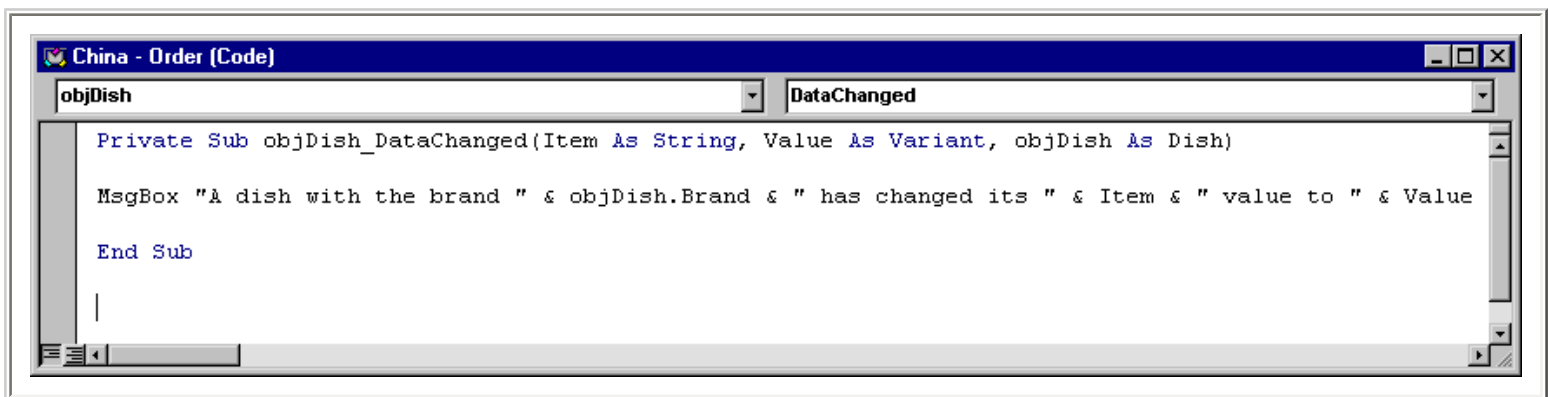
Not surprisingly, having changed the declaration for the event, we now need to supply a third argument when we raise it. Just to refresh your memory, we raise the event whenever any one of the Dish Object Properties change. Here's the new look of the line of code to raise the DataChanged event in the Property Let procedure for the Brand...

RaiseEvent DataChanged("Quantity", m_strBrand, Me)

You may be wondering what 'Me' is?

'Me' refers to the actual instance of the Dish Object that is raising the event. At the time the Dish Class is coded, there's no way to know what the Object variable name used to refer to it will be. 'Me' tells Visual Basic not to worry about the name---just to pass a reference to the Dish Object to the Collection Object.

Once the event is raised to the Collection object, the additional information supplied by the Dish Object necessitates a change to the Event Procedure Header. Look at the difference...

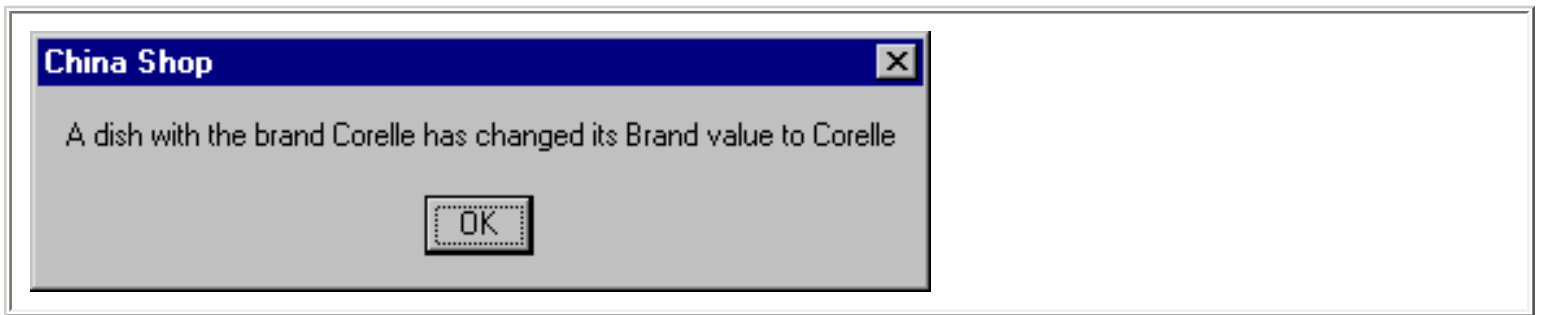


```
Private Sub objDish_DataChanged(Item As String, Value As Variant, objDish As Dish)

MsgBox "&A dish with the brand " & objDish.Brand & " has changed its " & Item & " value to " & Value

End Sub
```

Notice that within the event procedure, we can refer to individual properties of the Dish Object by prefixing the name of the argument---objDish---to a property name. If we execute the program now, and click on the Brands ListBox, we'll receive this message...



Passing a reference to an entire object like this can save you a bunch of work. In fact, if we wanted to, we could modify the event so that the only argument we pass is a reference to the Dish object.

Suppose?

Suppose you want your form to be able to react to events within the Dish Object. Can that be done?

The answer is 'No'---since we are now instantiating the Dish Object via a method of the Order Collection Object, only the Collection Object can directly react to events of the Dish Object. However, there's nothing to stop you from writing code in the Collection Object that raises an event of its own in response to an event in the Dish Object.

Confused?

If you're confused by anything I've shown you here, feel free to download a copy of the China Shop project I've coded up which includes these changes. You can download it from this location...

<http://www.johnsmiley.com/downloads/marilyn.zip>

Be sure to unzip the files into a directory called

C:\VBFILES\CHINA

so that the program runs properly.

Summary

I hope that this article will give you an insight into the enormous power that object to object communication can give you.