Common VB Code Patterns for Automation Rules

Alastair Foreman 07 February 2005 FOR INTERNAL USE ONLY

The code views of the automation Rule Builder allow you to write Visual Basic (VB) code that can do almost anything. A full VB tutorial is beyond the scope of this document, but there are a few patterns that are especially common.

1. Simple Condition

A condition is a small piece of VB code that checks if something is true. It usually contains an IF-THEN statement and a “RETURN TRUE” statement and a “RETURN FALSE” statement.

The following condition checks to make sure that the “NewApprovalState” of a page is set to “Approved”. If so it will let the condition pass by returning TRUE.

```
'Check to make sure the page was APPROVED
IF FiredEvent.NewApprovalState = ApprovalState.Approved THEN
       'This will make the condition pass
       return TRUE
END IF

'This will make the condition fail
Return FALSE
```

2. Simple Parameter Assignment

2.1 Assigning static values

A “static” is a value that is the same every time a rule is fired. Examples are: the process template to use, the email address to send to, the priority to use, etc.

```
'--- Assign the value of the Action Parameter "Comment" ----
Action.Comment = "This is a test comment"

'--- Assign the value of the Action Parameter "Priority" ----
Action.Priority = 2
```

2.2 Assigning event properties

Event properties are dynamic values – they are different every time a rule fires. In this case we assign a property of the event to a parameter of the action.

```
'--- Assign the value of the Action Parameter "Job" ---
Action.Job = FiredEvent.Job

'--- Assign the value of the Action Parameter "UserName" ---
Action.UserName = FiredEvent.Process.UserName
```

2.2 Assigning strings with embedded properties

Sometimes it is useful to construct a string that combines some static words with event property value(s) mixed in (especially useful in emails). This requires a combination of the 2 patterns described above.

```
'--- Assign the value of the Action Parameter "Subject" ---
Action.Subject = ("Failed refine in Job " + FiredEvent.Process.Job.Name + ".")
```

3. Working with Arrays

Most of the really useful action parameters in Prinergy require lists (or arrays) of objects. For example the Refine action needs a list of Input Files to refine, and the Loose Proof action needs a list of Pages. Likewise, many event properties are lists. For example the Page Approval Changed event provides a list of pages that had their approval changed, Working with these arrays in VB is considerably more complex than simple assignments.

3.1 Simple array assignment

In the simple case, where you want to pass ALL the members of an event property array to an action parameter array, the code looks just like a simple assignment:

```
'This takes ALL the input files from the event and gives them to the action
Action.InputFiles = FiredEvent.InputFiles
```

3.2 Flattening nested arrays

There are sometimes cases where the data that you want to assign to an action parameter is "spread out" over the members of an array in the event properties. For example, the results of a "Refine" action are available from an "Input Files Refine OK" event. If you look into that event you will see that it has the following properties:

Input files	The list of input files which refined successfully
Process	The refine process
Event time	The time the event was fired
Root Event	The first event in the sequence
Parent	The parent event for this event

Where are the pages? In fact each of the refined input files has its own list of pages that were created from it. In other words, each member of the InputFIles array contains an array of Pages.

If we want to pass a list of all the refined pages to an action (eg Loose Proof) then we need a way to reach into each Input File and build a collection of all the Pages.

```
'First we need to create a holder for the new list we will be building
Dim newList As System.Collections.ArrayList = New System.Collections.ArrayList

'Now we create a counter variable and initialize to the value 0
Dim i As Integer = 0

'Next we create a loop that will go through all the input files (until
'the counter i reaches the end of the list)
Do While (i < FiredEvent.InputFiles.Length)

      'We reach into the ith Input File and add its Pages array to our list
      newList.AddRange(FiredEvent.InputFiles(i).Pages)

      'Increment the value of i
      i = (i + 1)
Loop

'Finally, we convert our list into an Array of Pages and assign it to the
'Pages parameter of the action
Action.Pages = newList.ToArray(GetType(Creo.PWS.Automation.PrinergyDataModel.Page))
```

3.3 Filtering an array

Another common requirement is the ability to “weed out” parts of a list that you are not interested in. For example, let’s say that you want a simple way to create Imposition Proofs of only the “Front” Surfaces of every Signature in your job. What we would like to be able to do is pass the Imposed Proof action a list of Surfaces that has all the “Back” surfaces filtered out.

```
'First we need to create a holder for the new list we will be building
Dim newList As System.Collections.ArrayList = New System.Collections.ArrayList

'Now we create a counter variable and initialize to the value 0
Dim i As Integer = 0

'Next we create a loop that will go through all the input files (until
'the counter i reaches the end of the list)
Do While (i < FiredEvent.Surfaces.Length)

	'We look at the ith Surface and add it to our list if it is "Front"
	If FiredEvent.Surfaces(i).Side = "Front" Then
		newList.AddRange(FiredEvent.Surfaces(i))
	End If

	'Increment the value of i
	i = (i + 1)
Loop

'Finally, we convert our list into an Array of Surfaces and assign it to the
Surfaces parameter of the action
Action.Surfaces = newList.ToArray(GetType(Creo.PWS.Automation.PrinergyDataModel.Surface))
```

3.4 Filtering and flattening arrays

Combining the two patterns for flattening and filtering gives us the most power of all. In this example we want a list of all the refined pages that are larger than 8.5 x 11. Since we need to look at every page in order to determine its size, this requires us to introduce a second loop. The outer loop goes through all the input files, while the inner (nested) loop goes through all the refined pages for each input file.

```
'First we need to create a holder for the new list we will be building
Dim newList As System.Collections.ArrayList = New System.Collections.ArrayList

'Now we create a counter variable and initialize to the value 0
Dim i As Integer = 0

'Next we create a loop that will go through all the input files (until
'the counter i reaches the end of the list)
Do While (i < FiredEvent.InputFiles.Length)

	'Create a second counter and make a second loop to go through all
	' the pages for input file i
	Dim j As Integer = 0
	Do While (j < FiredEvent.InputFiles(i).Pages.Length)

		'Grab the jth Page from the ith Input File
		Dim thePage As Page = FiredEvent.InputFiles(i).Pages(j)

		'See if the page is larger than 8.5 x 11 – if so add it to our list
		If (thePage.TrimSize.x > 8.5 AND thePage.TrimSize.y > 11) Then
			newList.add(thePage)
		End If

		'Increment the value of j
		j = (j + 1)
	Loop

	'Increment the value of i
	i = (i + 1)
Loop

'Finally, we convert our list into an Array of Pages and assign it to the
'Pages parameter of the action
Action.Pages = newList.ToArray(GetType(Creo.PWS.Automation.PrinergyDataModel.Page))
```