Using External Values in a VB.NET Script

Application Note

Date	April 23, 2009
Applies To	Kofax Capture 8.0 VB.NET Scripting
Summary	This application note provides information regarding how to implement a persistent value stored in a text file and a database and using that value in a VB.NET Validation Script.
Revision	1.0

Overview

Kofax Capture 8.0 introduced a feature in the Text Release Script where released image files can be based on the Standard Filename, a Decimal Based Filename or an Index Field value. This document will demonstrate two easy methods for maintaining a formatted numerical value externally. These examples will demonstrate both using a text file and a database using ADO.NET.

Using a Text File

1. First, you need to add file handling support. In your VB.NET script, you will need to add the following Imports statement:

   ```
   Imports System.IO
   ```

2. For the purpose of this article, we are going to perform all the file access in the DocumentPreProcess event. However, this same code could also be placed in the DocumentPostProcess or any of the FieldPre or PostProcessing events.

3. Below is our DocumentPreProcessing event:

```
Private Sub TestDoc_DocumentPreProcessing(ByVal sender As Object, ByVal e As
Kofax.AscentCapture.Scripting.PreDocumentEventArgs) Handles Me.DocumentPreProcessing

    'First check if the file exists, create it if missing.
    'There is no need to execute the other read/writes so
    'just set the Index value and move on.
    If Not File.Exists("c:\Number.txt") Then
        Using fs As New StreamWriter("c:\Number.txt", False)
            fs.WriteLine("00000000")
        End Using
        TestVal.IndexField.Value = "00000000"
    Else
        'File exists. Open it and get the value.
        Using sr As New StreamReader("c:\Number.txt")
            TestVal.IndexField.Value = sr.ReadLine
        End Using
        'Write out the new incremented value.
        Using sw As New StreamWriter("c:\Number.txt", False)
            TestVal.IndexField.Value = _
               (Integer.Parse(TestVal.IndexField.Value) + 1).ToString().PadLeft(8, "0")
            sw.WriteLine(TestVal.IndexField.Value)
        End Using
    End If

End Sub
```

We first check if the file exists. If it doesn't exist, the file is then created using a StreamWriter which writes a beginning value of "00000000" to the file. In this case, the Index value is simply set to that same beginning value and we move on.

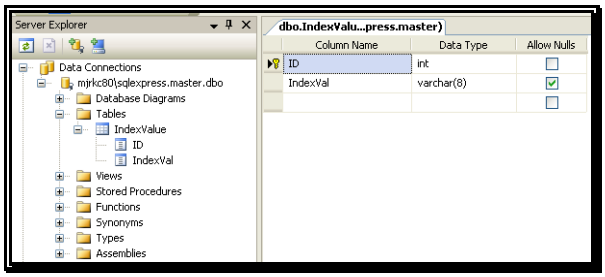
If the file does exist, a StreamReader is used to open a file, read a single line from the file then close the file. A Using block is implemented here, which implicitly closes the file in the End Using statement. The Index Field value is set to the value of the incoming string read in from the file.

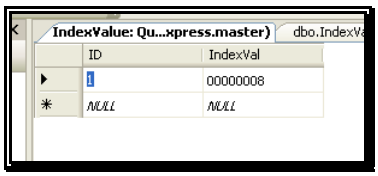
The second Using block implements a StreamWriter. The StreamWriter opens the file and writes an 8-character string value, padded with zeros, overwriting the original value. The Index Field value is set to this new value.

NOTE: Only one instance of the file can be open at a time. When there are multiple users running Validation, this could become a problem. Also, on systems running Microsoft Vista, we recommend not placing the file in the root c: directory. Vista implements user security for the root directory restricting write access only to the Administrator.

Using a Database

For this demonstration, we created a table in a local SQL Server database named IndexValue with two columns as shown here.





1. At the top of the script add this Imports statement:

   ```
   Imports System.Data.SqlClient
   ```

2. Again, for the purpose of this article, we are going to perform all the database access in the DocumentPreProcess event. However, this same code could also be placed in the DocumentPostProcess or any of the FieldPre or PostProcessing events.

3. Here is our DocumentPreProcessing event:

```
Private Sub TestDoc_DocumentPreProcessing(ByVal sender As Object, _
                                          ByVal e As Kofax.AscentCapture.Scripting.PreDocumentEventArgs) _
                                          Handles Me.DocumentPreProcessing

    Try
        'Create a database connection and open the database
        Dim oConn As New SqlConnection(_
                              "Data Source=MJRKC80\SQLEXPRESS;Initial Catalog=master;Integrated Security=True")
        oConn.Open()

        'Start a transaction process
        Dim oTran As SqlTransaction = oConn.BeginTransaction

'Retrieve the value out of the database and set the Index Field to the value
        Dim oCmdSel As New SqlCommand("SELECT TOP 1 IndexVal FROM IndexValue", oConn)
        oCmdSel.Transaction = oTran
        TestVal.IndexField.Value = "" + oCmdSel.ExecuteScalar

        'Write the incremented value into the database
        Dim oCmdUpdate As New SqlCommand("UPDATE IndexValue SET IndexVal='" + _
                    (Integer.Parse(TestVal.IndexField.Value) + 1).ToString().PadLeft(8, "0") + "'", oConn)
        oCmdUpdate.Transaction = oTran
        oCmdUpdate.ExecuteNonQuery()

        'Commit the transaction
        oTran.Commit()

        'Close the database connection
        oConn.Close()
        oConn = Nothing
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try

End Sub
```

As common with ADO.NET applications, we first create a connection to the database and open it using the Open() method. We have implemented this access in a transaction. This helps alleviate the issue with multiple users running Validation at the same time.

The transaction is created using the BeginTransaction method on the connection object.

The SELECT statement simply retrieves the current value from the table, which in this case, is a formatted string containing a number padded with zeros to eight characters.

For each command object used in the various queries and updates, each is added to the Transaction object in turn, to place them as part of this Transaction process.

```
oCmdSel.Transaction = oTran
```

```
oCmdUpdate.Transaction = oTran
```

To retrieve the single record, we use the ExecuteScaler method. The update is done in an ExecuteNonQuery method.

The Transaction is then executed and closed with the Commit method.

Finally, the database is closed and set to Nothing.

Summary

This technique is one of many for maintaining a persistent value. Other methods include using a stored procedure in a database or even using a Registry Key if Validation is to be done on a single machine. On systems where there will be a large number of users running Validation, it is a good practice to implement a stored procedure that will perform the actual incrementing and updating of the persistent value in the database.