FOLDER SWEEP

OVERVIEW

A common use of EFT Server's Folder Monitor rule is to detect files and move them to a different location on the network for processing. Mission critical operations require all files to be processed, even those missed due to a temporary network outage or other error. Below is a diagram showing how EFT Server's folder monitor works and the potential for missing a file:

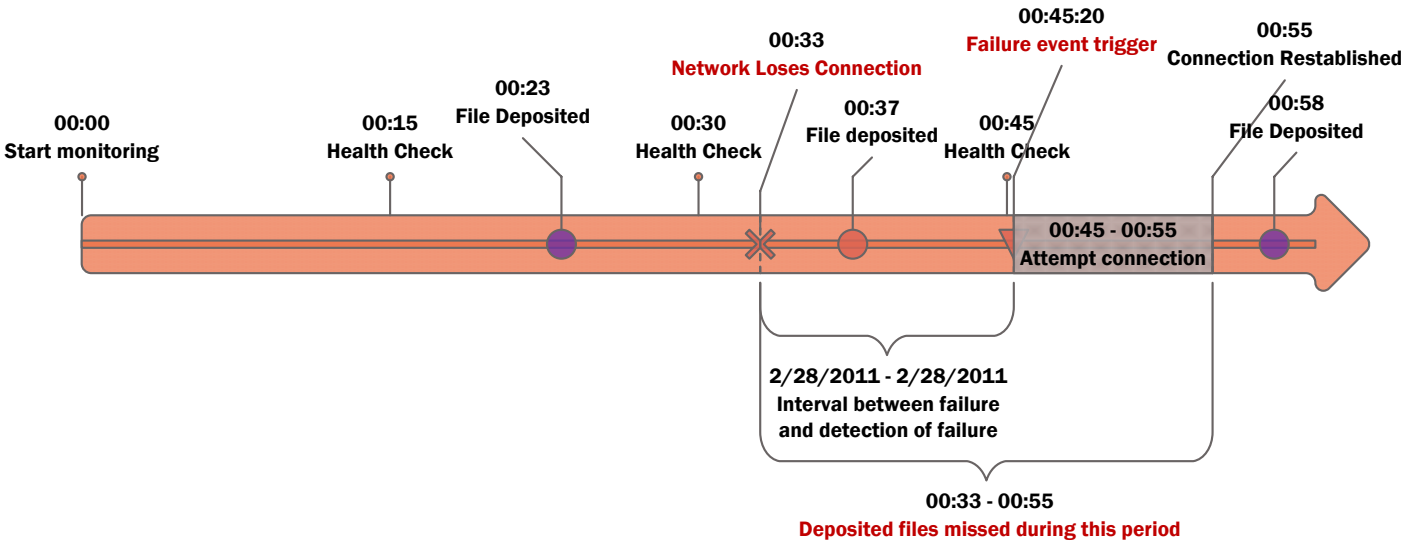


CHART EXPLAINED:

1. Folder monitor started at 0:00.
2. First health check[1] at 0:15.
3. File deposited at 0:23 and is processed by EFT Server.
4. Network connection lost at 0:33.
5. File deposited (and missed!) at 0:37.
6. Second health check at 0:45.
7. Failure event thrown at 0:45:20
8. A second attempt to re-establish connection succeeds at 0:55.
9. File deposited at 0:58 gets processed.

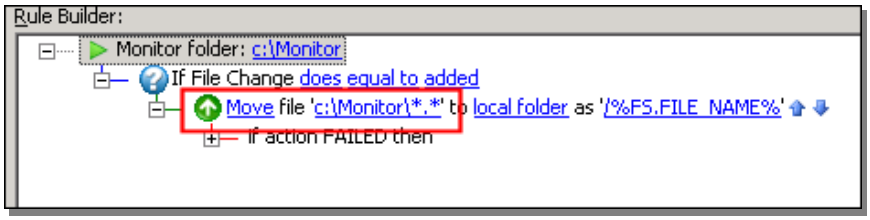
GOAL:

Sweep all files in the monitored folder regardless of when deposited (i.e., process ALL files).

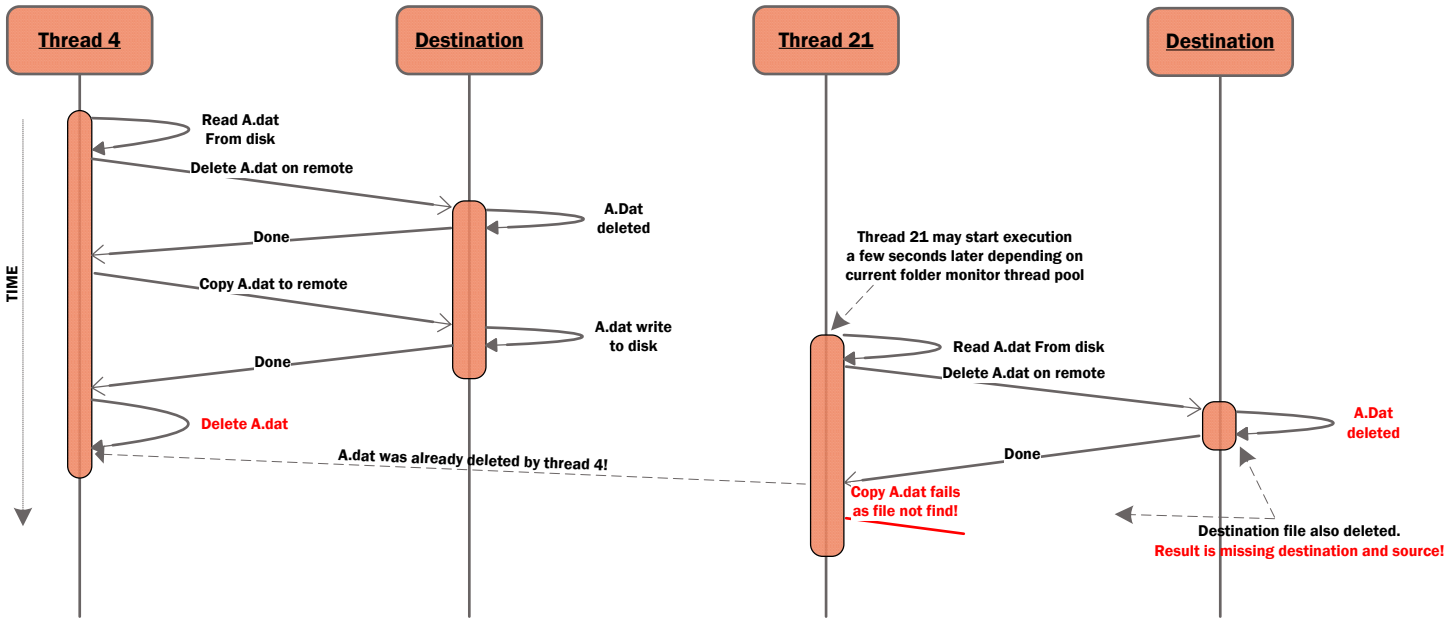
[1] All files deposited between minute 0:33 and 0:55 are missed. The potential gap is broader; that is, immediately after the last health check, until the next health check, plus 20 seconds.

MOVE USING WILDCARDS APPROACH – NOT RECOMMENDED

This approach is commonly attempted, but may result in unpredictable behavior. The approach consists of doing a MOVE operation and using a wildcard mask, such as *.* for the local file source path, rather than the default %FS.PATH% argument provided when creating the Move action:



This approach doesn't work consistently due to the asynchronous nature of how Folder Monitor creates threads for each file deposited into the monitored folder. For example, if you drag 100 files into a monitored folder, up to 100 threads will attempt the move *.* operation, all within milliseconds of each other.



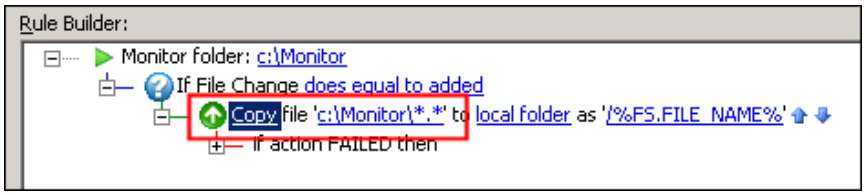
Before the Move action copies the source file to the destination, the action determines whether the source file exists in the destination and, if so, deletes the file in the target location. Next, the action copies the source file to the destination, and then deletes the source (original) file.

If the timing of multiple threads is just right, it is possible for one thread (thread 21 above) to delete the destination file that corresponds to another thread's source file (thread 4 above) that was just uploaded. In that case, when thread 21 attempts to copy the source file to the destination, it fails, because Thread 4 already deleted the file at the end of its operation. The disastrous result is the complete removal of both source AND destination files!

In EFT Server 6.3, you could mitigate this risk by choosing "Skip" rather than "Overwrite" for matching filenames overwrite logic, but then you wouldn't be able to use the overwrite logic choices as they were originally designed.

COPY USING WILDCARDS APPROACH – NOT RECOMMENDED

This approach is sometimes attempted instead of the Move operation:

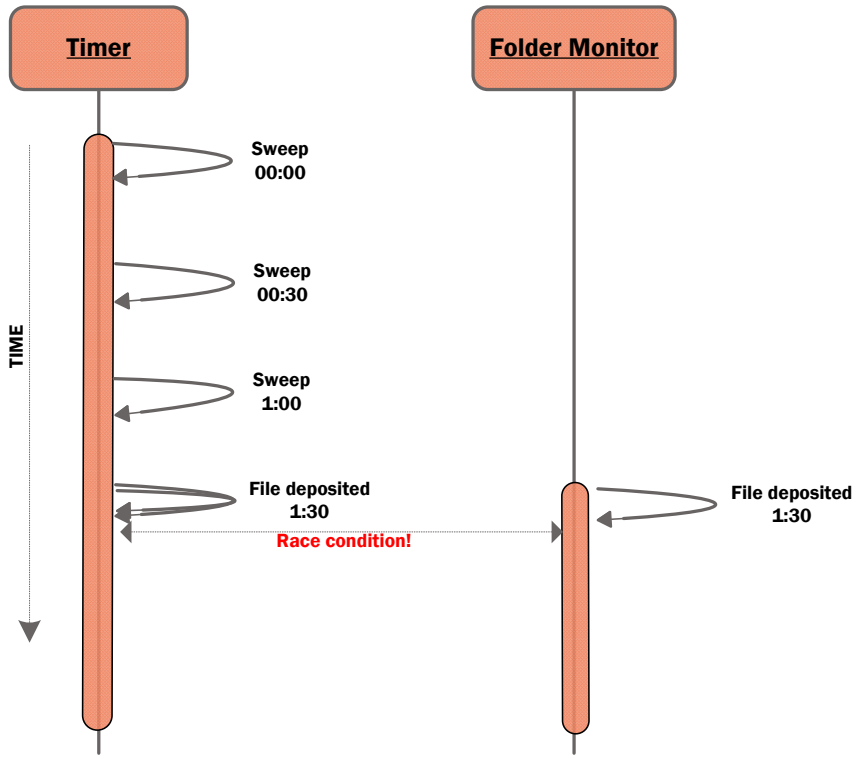


This method should not result in arbitrarily deleted files, but can adversely affect performance, because each copy operation is duplicated for matching masks across all files deposited into the folder.

For example, if you drop 1000 files into a folder, then copy *.* will occur more than 1,000 times, resulting in one million atomic file copy operations.

SEPARATE TIMER RULE FOR CLEANUP – NOT RECOMMENDED

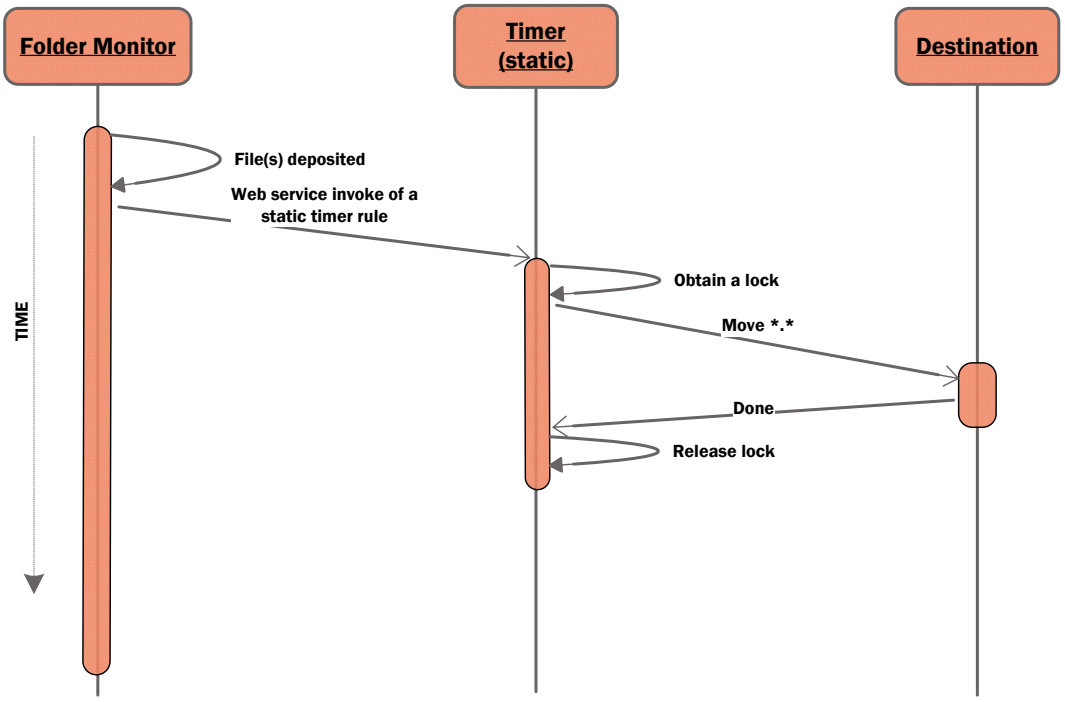
Some users deploy a secondary Event Trigger consisting of a Timer rule that runs on a recurring basis, sweeping the folder for any remaining files. This approach is also problematic in that a Timer rule could execute at virtually the same time as when files were deposited in the folder, resulting in a race condition with the folder monitor rule, which is essentially the same problem experienced in the Move Action example (using wildcards) discussed previously.



Another shortcoming to this approach is that you would need a separate Timer rule for each Folder Monitor rule.

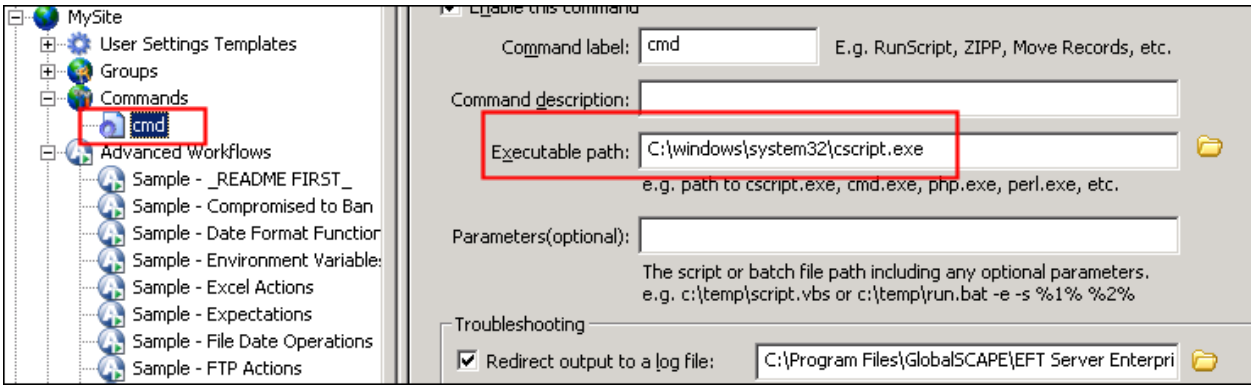
Separate and Block on Move Action – Recommended

This approach places the Move operation into a separate rule that is invoked via EFT Server's Web Services. Prior to moving the files, a lock is obtained on a temporary file so that only a single Timer rule thread can perform the Move action. The diagram below (which merges script and timer operations for clarity) demonstrates this approach.

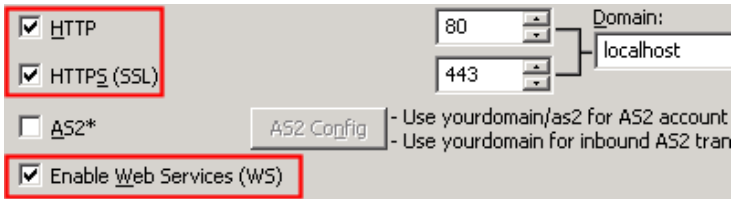


Steps:

1. Create a Custom Command that will run **cscript.exe**, which will execute a script to obtain the lock and then execute the Timer rule via web services:



2. Next, enable HTTP or HTTPS and Web Services on the Site:



3. Now copy the below script into your text editor and save it as **webservice2.vbs** to your C: drive (or choose an alternate name/destination and edit the script accordingly):

```
Dim oArgs: Set oArgs = WScript.Arguments
Dim strURL, strAdminUserName, strAdminPassword, sLckFile
Dim strResponseValue, xmlhr, strWSParams, strFileName, strRemoteFileName, strRuleName
Const ForAppending = 8

If(oArgs.count < 2) Then
        'missing input arguments
                WScript.Echo("ERROR: Incorrect number of parameters passed! [Number of arguments
passed : " & oArgs.count &"]")
                WScript.Echo("ERROR: Usage: cscript webservice2.vbs source_path
destination_folder")
                Call WScript.Echo(-1)
Else
               strFileName = trim(oArgs(0))
               strRemoteFileName = trim(oArgs(1))
End If

'Variables
sLckFile = "c:\file.lock"'change location if desired
strURL = "http://192.168.101.115/WebService/InvokeEventRule"'you MUST change the URL to match
your server host address
strAdminUserName = "admin"'you MUST change this to match your admin credentials
strAdminPassword = "test"'you MUST change this to match your admin credentials
strResponseVaule =""
strRuleName = "WS_UPLOAD" 'only change if you gave the static timer rule a name other than
"WS_UPLOAD"

Set oFSO = CreateObject("Scripting.FileSystemObject")
'Now get file lock
bUpdFinished = False
iLoops = 0
On Error Resume Next
Do
'open for appending
Set f = oFSO.OpenTextFile(sLckFile, ForAppending, True)
If Err.Number = 70 Then
'Permission denied error
' Waiting 1/2 a second before trying again
' Other threads will block here
WScript.Sleep 500
ElseIf Err.Number <> 0 then
'WScript.Echo "Unexpected Error #: " & Err.Number
bUpdFinished = True
Else
On Error GoTo 0
'Do the job here, the file will now be locked by this script and
'nobody else will be able to continue until the f.Close instruction
'WScript.Echo "Got the lock" - processing file(s)

'####################Don't change anything below unless you know what you are
doing####################
strWSParams = "EventRuleName=" & strRuleName & "&EventParams=FILENAME=" &strFileName &
";DESTFOLDER=" &strRemoteFileName
'MsgBox "strWSParams= " &strWSParams
Set xmlhr = CreateObject("MSXML2.ServerXMLHTTP.3.0") 'earlier versions don't support setOption
             If Not ( xmlhr Is Nothing ) Then
```

```
            'xmlhr.setProxy 2, "localhost:8888", "" 'only edit if need to go through proxy
            xmlhr.setOption 2, SXH_SERVER_CERT_IGNORE_ALL_SERVER_ERRORS ' ignore cert errors or
other desired options. Won't work with XMLTTHP, requires ServerXMLHTTP
            xmlhr.Open "POST", strURL, False, strAdminUserName, strAdminPassword
            xmlhr.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
            xmlhr.setRequestHeader "User-Agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; AWE-xmlhttp;)"
             'MsgBox "We are going to make a request to : " & vbCrLf & strURL & vbCrLf & "With :
username = " & strAdminUserName & vbCrLf & "Password: " & strAdminPassword & vbCrLf & "Params:" &
vbCrLf & strWSParams
            xmlhr.Send(strWSParams)
             'MsgBox "responseText = '" & xmlhr.responseText & "'" & vbCrLf & "responseXML = '"
& xmlhr.responseXML.xml & "'"
            strResponseValue = xmlhr.responseText
             'MsgBox "Response Is: " & vbCrLf & strResponseValue
             Dim iPos
             Dim iPos2 ' Parse the return code from the Web Service. I chose this over XML for
simpler readability, but XPATH could also be used.
              iPos = InStr( 1, strResponseValue, "<int" )
              If ( iPos > 0 ) Then
                ' MsgBox "Found '<int' at position " & CStr(iPos) & " ('" & Mid(
strResponseValue, iPos ) & "')..."
                iPos = InStr( iPos + 1, strResponseValue, ">" )
                If ( iPos > 0 ) Then
                  ' MsgBox "Found '>' at position " & CStr(iPos) & "('" & Mid( strResponseValue,
iPos ) & "')..."
                  iPos2 = InStr( iPos+1, strResponseValue, "<" )
                  If ( iPos2 > 0 ) Then
                  ' MsgBox "Found '<' at position " & CStr(iPos2) & "('" & Mid(
strResponseValue, iPos2 ) & "')..."
                   strResponseValue = Mid ( strResponseValue, iPos +1, iPos2 - iPos-1)
                  End If
                End If
              End If
            End If
'MsgBox "Response Code: " & strResponseValue
'release the lock
'WScript.Echo "Release the lock"

f.Close
bUpdFinished = True
End If
Err.Clear
Loop Until bUpdFinished
On Error GoTo 0
Wscript.quit
```

Please do not forget to modify the following variables in the script to match your environment:

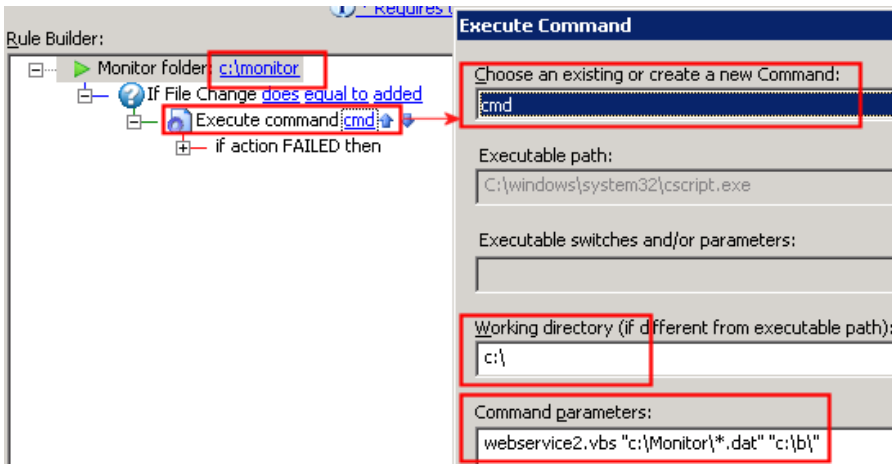
- strAdminUserName, which is the admin account name that will invoke web services
- strAdminPassword = password for the above account
- strURL = EFT Server's host address and web services path.

Optional variables include:

- strRuleName = only change if you change the name of the timer rule created in step 5 below.
- sLckFile = Only change if you want to create the lock file in another location.

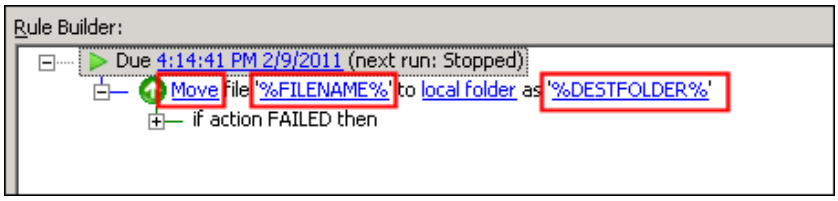
(**Note:** When copying text from one document to another, unwanted characters are often copied with it. Please review the text after you paste it, comparing it with the script above. You might also need to edit values in the script to suit your environment.)

4. Create or modify your existing folder monitor rule so that it calls the custom command and passes in the path to be processed (along with wildcard mask) and the destination folder. You can hard code these values or use static values:



5. Create a Timer rule named “WS_UPLOAD” (if you choose a different name then you must modify the **webservice2.vbs** script), and set the Run value to “Once” and the Start Date value to a date in the past.

6. Add a Copy/Move operation, select “LAN” as the offload method, type %FILENAME% in the source path, and type %DESTFOLDER% in the destination path. (Note: Any modifications to these values must also be made in the **webservice2.vbs** script):



7. Now you are ready to test. You can test by dragging a few files into the monitored folder (matching the mask provided) or you can call the **webservice2.vbs** script directly from the command line. For example:

   ```
   C:\cscript.exe webserivce2.vbs "source_path\*.* " "destination_folder"
   ```

   (Uncomment the msgbox or wscript.echo commands to troubleshoot, if necessary.)

   Now when the Folder Monitor executes due to a file being deposited, the separate, blocking static Timer rule will move (sweep) all files in the monitored folder, including those deposited during periods where the folder monitor missed files (see top), guaranteeing that all files will be processed.