

Tech Note 6

Automatic Test Procedure to Generate Report File and Email Notification

Terravic Research, Inc.

Introduction

Setting up a fully automatic test procedure is easily achieved using the scripting capabilities of Visual OPCTest Client. What will be accomplished includes: executing test cases of interest, determining the status of each executed test case, collecting all the failed test case data, storing information in a file, and finally notifying a developer by email of the failed results. Collecting test case data may be accomplished in several ways: external text file, csv file, xml file, or ADODB database connection (any other automation objects may be used for data storage as well). This Tech Note shows how to generate a test report file and email it to a developer in charge of bug fixing.

Platform Requirements

Visual OPCTest® Client 8.0.0000 or later.

Examples used in this Tech Note are based on the OPC Foundation sample servers. In order to successfully follow all the procedures outlined in this Tech Note, please download the OPC Foundation's Data Access sample server and the latest Visual OPCTest Client application. If the OPC Foundation sample server is not available, please substitute with another Data Access server and its tag definition. The substitution of server and tag definition in the sample code should be self-explanatory.

Basic Steps

To setup a script that will automatically perform testing, data collection, and notification, we need to perform these three steps:

- (1) Connect to the OPC Data Access Server
- (2) Run Validate Script Wizard
- (3) Modify the Wizard generated script based on our requirements

Step 1 - Connect to the OPC Data Access Server

Run the Client and select *Server | Connect to Server* menu item. The *OPC Server List* dialog will appear. For simplicity, select "OPC Data Access Servers Version 3.0" under LOCAL/ComputerName branch, then select "OPCSample.OpcDaServer.1" server. Click on the *Connect Server* button, then on the *Close* button. Of course, you may select a different server on either local or remote computer. Once we successfully connect to this server, there should be a server listed in the Workspace View with an entry such as "DA1\\LOCAL\\OPCSample.OpcDaServer.1" (this entry will be different based on your server location and name).

Step 2 – Run Validate Script Wizard

Better than writing the whole script from scratch, let's use the Wizard to get started with the base script. Once the Wizard completes, we will have a script code that connects the server, creates all the necessary groups adds items to each group, sets up the test case tree, and disables/enables interfaces of interest.

Right-click on “DA1\\LOCAL\OPCSample.OpcDaServer.1” in the Workspace View, and select *Validate Script Wizard* from the popup menu.

The “Test Type Selection” wizard screen appears; keep default settings and click on the *Next* button.

The “Interface Test Selection” screen appears next. As we are executing specific test cases from a script, these settings are irrelevant to us. These options are useful when performing tests manually, therefore, click on the *Next* button.

The “Test Group Selection” wizard screen appears next. This allows us to specify the different groups that the script will create for us, such as private or public, active or inactive, connected or not connected. All these groups are needed for Client to obtain predefined test case data for the standard test cases. Since this OPC Server does not support Advise Sink, deselect all options with Advise Sink (4, 5, and 6 from top), click on the *Next* button.

The “Test Item Selection” wizard screen appears. All the selected tags in this screen will be added to the groups defined in the previous wizard screen. Click on the *Browse* button to see the server’s address space. Double click on the following tags to select them:

- "Analog Types/Double"
- "Analog Types/Int"
- "Enumerated Types/Fellowship"
- "Limited Access/Read Only 1"
- "Limited Access/Write Only 1"
- "Simple Types/Float"
- "Simple Types/Int"

Click the *OK* button, and then click the *Next* button.

The “Finish – Generate Script” wizard screen appears last. Accept the defaults and click on the *Finish* button. Our script appears in the Script view.

Select from the main menu *File | Save As Script*, type ”IOPCCCommonTest” as the file name, and click on the *Save* button. Let’s test what our script creates. Select *Script | Run Script* from the main menu. Server is connected, all groups created and items are added, and finally, the test case tee is created as well. We are ready to modify the script.

Step 3 – Modify Script

Simple Test Requirements for Generating a Report File

Let’s setup a common ground of what we want to accomplish. First, we will be testing a Data Access server and we want to concentrate on an IOPCCCommon interface. As Visual OPCTest Client has over 380 test cases just for testing IOPCCCommon interface, we will base our tests on these standard test cases. Since we are testing a specific interface, we are only interested in collecting data of all the failed/warning test cases (when performing full integration test, all the test data may be collected, regardless of test case status). Once we execute all the test cases of interest, we will determine if any tests failed so a developer may be notified with the appropriate test case data. Script will collect data to the ScriptOut view, then save that data to an external text file and email it to a developer in charge of the IOPCCCommon interface development.

Execute Test Case and Store Data in the ScriptOut view

First we need to write a subroutine that will execute a given test case and collect its data when status is set to either fail or warning. This will allow us to encapsulate test case execution into a

single subroutine so we can use it on different test cases. This subroutine accepts OPC Data Access server handle (Client scripting handle) and a test case number in string format, such as "2.1.1.2". Comments are self-explanatory and are preceded by the REM keyword. If the test case status after execution is anything but FAIL or WARNING, then the subroutine simply exits. Otherwise, data is collected for the FAIL/WARNING test case and stored in the ScriptOut view. Additionally, counters are updated to hold the total number of failed and warning test case results.

```
Public Sub ExecTestCase(ServerID, TCN)

    REM just a flag to determine if test case passed or not
    TestCaseNotBad = TRUE

    REM execute the given test case
    OPC.ExecuteTestCase ServerID, TCN

    REM get the test case status, i.e. "FAIL"
    TestCaseStatus = OPC.GetTestCaseStatus(ServerID, TCN)

    REM check if it failed/warning or ok
    REM keep track of fail/warning total count
    If StrComp( TestCaseStatus, "FAIL", vbTextCompare ) = 0
    Then
        CountFail = CountFail + 1
        TestCaseNotBad = FALSE
    End If
    If StrComp( TestCaseStatus, "WARNING", vbTextCompare ) = 0
    Then
        CountWarning = CountWarning + 1
        TestCaseNotBad = FALSE
    End If

    REM ok, do we need to collect this test case data?
    If TestCaseNotBad = TRUE Then
        REM not really, but still need to check if test case
        is
        REM disabled or incomplete, anyway, we are done
        REM as we only want fail/warning test cases

        IsDisabled = OPC.IsTestCaseDisabled(ServerID, TCN)
        If IsDisabled Then
            OPC.ScriptOutputEntry "Test Case " & TCN & " -
            " & TestCaseStatus
            OPC.ScriptOutputEntry " This test case is
            disabled"
            OPC.ScriptOutputEntry " "
        End If

        Exit Sub
    End If

    REM get all the test case data and results,
    REM place them in ScriptOut view
```

```

OPC.ScriptOutputEntry "Test Case " & TCN & " - " &
TestCaseStatus

InfoServerName = OPC.GetTestCaseServerName(ServerID, TCN)
OPC.ScriptOutputEntry " ServerName: " & InfoServerName

InfoFunctionName = OPC.GetTestCaseFunctionName(ServerID,
TCN)
OPC.ScriptOutputEntry " FunctionName: " & InfoFunctionName

InfoFunctionHR = OPC.GetTestCaseFunctionHRESULT(ServerID,
TCN)
OPC.ScriptOutputEntry " FunctionHR: " & InfoFunctionHR

InfoOPCVersions = OPC.GetTestCaseOPCVersions(ServerID, TCN)
OPC.ScriptOutputEntry " OPCVersions: " & InfoOPCVersions

OPC.ScriptOutputEntry " Params:"

InfoParamIN = OPC.GetTestCaseParamIN(ServerID, TCN)
If InfoParamIN <> "" Then
    OPC.ScriptOutputEntry " " & InfoParamIN
End If

InfoParamOUT = OPC.GetTestCaseParamOUT(ServerID, TCN)
If InfoParamOUT <> "" Then
    OPC.ScriptOutputEntry " " & InfoParamOUT
End If

InfoParamOUTLOOP = OPC.GetTestCaseParamOUTLOOP(ServerID,
TCN)
If InfoParamOUTLOOP <> "" Then
    OPC.ScriptOutputEntry " " & InfoParamOUTLOOP
End If

InfoParamCALLBACK = OPC.GetTestCaseParamCALLBACK(ServerID,
TCN)
If InfoParamCALLBACK <> "" Then
    OPC.ScriptOutputEntry " " & InfoParamCALLBACK
End If

InfoExecTime = OPC.GetTestCaseExecTime(ServerID, TCN)
OPC.ScriptOutputEntry " ExecTime: " & InfoExecTime

InfoExecDuration = OPC.GetTestCaseExecDuration(ServerID,
TCN)
OPC.ScriptOutputEntry " ExecDuration: " & InfoExecDuration
& " msec"

InfoComment = OPC.GetTestCaseComment(ServerID, TCN)
OPC.ScriptOutputEntry " Comment: " & InfoComment

OPC.ScriptOutputEntry " "

```

End Sub

Save Results Function

All the data stored in the ScriptOut view will be saved to an external text file. File name and location is passed in as a parameter *Filename*. There is an option to save the timestamp from the ScriptOut view, we will not save the timestamp

```
Public Sub SaveReport(Filename)

    DIM SaveTimestamp
    SaveTimestamp = FALSE

    OPC.ScriptOutputEntry " "
    OPC.ScriptOutputEntry "Report Generated on " & CStr(Now)
    OPC.ScriptOutputEntry " "

    OPC.StatusPaneMsg "Saving test report to " & Filename
    OPC.ScriptOutputSave Filename, SaveTimestamp

End Sub
```

Send Email Notification Function

Once we have a report file, it may be emailed to a developer in charge of IOPCCommon development. The email system must be installed and available on the test machine, as a default source email address will be used (From). The destination email address is passed in as a parameter *EmailAddress*. This function creates the email system object and sets all the email parameters, such as destination address and attached file. Once the email is send, the email system object is released from memory.

```
Sub NotifyDeveloperByEmail(EmailAddress,Filename)

    OPC.StatusPaneMsg "Generating email to " & EmailAddress
    Set golApp = CreateObject("Outlook.Application")
    Set objNewMail = golApp.CreateItem(olMailItem)

    objNewMail.Recipients.Add EmailAddress
    objNewMail.Attachments.Add Filename
    objNewMail.Subject = "Test Results Notification"
    objNewMail.Body = "Test Results Attached. Please verify results."
    objNewMail.Send

    Set objNewMail = Nothing
    Set golApp = Nothing

End Sub
```

Executing Test Cases of Interest

After the supporting subroutines are coded, execution of the test cases may begin. All the test cases are executed from within the `Main` subroutine. Simply issuing the following call to run a test case will do the job:

```
TCN = "2.1.1.2"  
call ExecTestCase( ServerID, TCN )
```

Here, we are defining a test case number as string that we wish to execute. Calling our local subroutine will actually execute the test case, and determine if test case status is set to `FAIL` or `WARNING`. If it is, then the test case data will be placed in the `ScriptOut` view and the next test case may be executed.

Once all the test cases of interest are executed, saving a report and notifying a developer by email is achieved with the following four lines of code:

```
ResultsFilename = "C:\MyTests\MyIOPCCCommonTestResults.txt"  
EmailAddress = "Developer@TheCompanyXYZ.com"  
  
call SaveReport(ResultsFilename)  
  
call NotifyDeveloperByEmail(EmailAddress, ResultsFilename)
```

First the filename and email address are defined. Filename may contain a directory path as shown in the sample code above. Calling our subroutines defined earlier completes the test procedure by generating a report and emailing the developer with test results. Please note, only test cases with status set to `FAIL` or `WARNING` are included in the report (as defined by our `ExecTestCase` subroutine). Changing this subroutine to include all test case data may be accomplished by removing the initial `If` statement.

Conclusion

By extending the script generated by the `Validate Script Wizard`, it is possible to truly customize the test case execution and report generation. Executing individual test cases may be replaced by executing individual functions or even entire interfaces with just one line of code. However, executing individual test cases, gives us the most flexibility in collecting test case data. Besides storing data in external text files, we can place collected test case data in a database or on the web.

Sample Code File

Complete sample code file: `MyIOPCCCommonTest.otx`
Sample code file may be found upon installation of the `Visual OPCTest Client` software.

Contact

Contact Terravic with any questions regarding `Visual OPCTest Client`. Please let us know your concerns, desired improvements, or current issues with our software. We pride ourselves in making changes quickly based on your input. Please contact us at `support @ terravic.com`.

www.opctest.com
www.terravic.com

