

## Tech Note 11

### Automatically Store Test Results in a Custom XML Report File

Terravic Research, Inc.

#### Introduction

Results of an automatic test procedure executed by Visual OPCTest Client may be stored in a custom XML report file. The XML DOM Document object will be used to create an external XML file with all the results of executed test cases. Visual OPCTest Client will be setup to run selected standard test cases, collect all the test case data, and store the data in an external XML document. The XML document will be defined programmatically at execution time, allowing anyone to modify its structure. Only test cases with failed or warning status will be stored in the report file. Once the custom XML report is generated, it will be emailed to the developer responsible for defect tracking and reconciliation.

#### 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, XML file generation, and email 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 may be different based on your server location and name.

#### Step 2 – Run Validate Script Wizard

We will use the Validate Script 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, accept the default selections and 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 “CustomXMLTestReport” as the file name, and click the *Save* button. Let’s test what our script creates. Select *Script | Run Script* from the main menu. Server is connected, all groups are 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***

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 IOPCCOMMON interface. As Visual OPCTest Client has over 380 test cases just for testing IOPCCOMMON 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). While executing all the test cases of interest, test case data of any test status that is set to FAIL or WARNING will be placed in the XML document. After all test cases execute, an email notification will be send to a programmer in charge of the IOPCCOMMON interface development.

#### ***Execute Test Case and Store Data in the XML document***

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 it is stored in the XML document. 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
    CountTotal = CountTotal + 1

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

    REM get the test case status
    Status = OPC.GetTestCaseStatus(ServerID, TCN)

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

    If TestCaseNotBad = TRUE Then
        REM test case status is not FAIL or WARNING
        REM so there is nothing for us to place in the XML
        document

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

        Exit Sub
    End If

    REM collect all the data for this FAIL/WARNING test case

    ServerName = OPC.GetTestCaseServerName(ServerID, TCN)
    FunctionName = OPC.GetTestCaseFunctionName(ServerID, TCN)
    FunctionHR = OPC.GetTestCaseFunctionHRESULT(ServerID, TCN)
    OPCVersions = OPC.GetTestCaseOPCVersions(ServerID, TCN)
    ParamIN = OPC.GetTestCaseParamIN(ServerID, TCN)
    ParamOUT = OPC.GetTestCaseParamOUT(ServerID, TCN)
    ParamOUTLOOP = OPC.GetTestCaseParamOUTLOOP(ServerID, TCN)

```

```
ParamCALLBACK = OPC.GetTestCaseParamCALLBACK(ServerID, TCN)
ExecTime = OPC.GetTestCaseExecTime(ServerID, TCN)
ExecDuration = OPC.GetTestCaseExecDuration(ServerID, TCN)
Comment = OPC.GetTestCaseComment(ServerID, TCN)
```

*REM create xml nodes and elements/attributes*

```
Set TCElem = xmlDoc.createElement("TEST-CASE")
TCElem.setAttribute "id", TCN
TCElem.setAttribute "status", Status
```

```
Set ParamElem = xmlDoc.createElement("SERVER")
ParamElem.text = ServerName
TCElem.appendChild ParamElem
Set ParamElem = Nothing
```

```
Set ParamElem = xmlDoc.createElement("FUNCTION")
ParamElem.text = FunctionName
TCElem.appendChild ParamElem
Set ParamElem = Nothing
```

```
Set ParamElem = xmlDoc.createElement("HRESULT")
ParamElem.text = FunctionHR
TCElem.appendChild ParamElem
Set ParamElem = Nothing
```

```
Set ParamElem = xmlDoc.createElement("OPC-VERSIONS")
ParamElem.text = OPCVersions
TCElem.appendChild ParamElem
Set ParamElem = Nothing
```

```
Set ParamElem = xmlDoc.createElement("PARAM-IN")
ParamElem.text = ParamIN
TCElem.appendChild ParamElem
Set ParamElem = Nothing
```

```
Set ParamElem = xmlDoc.createElement("PARAM-OUT")
ParamElem.text = ParamOUT
TCElem.appendChild ParamElem
Set ParamElem = Nothing
```

```
Set ParamElem = xmlDoc.createElement("PARAM-OUTLOOP")
ParamElem.text = ParamOUTLOOP
TCElem.appendChild ParamElem
Set ParamElem = Nothing
```

```
Set ParamElem = xmlDoc.createElement("PARAM-CALLBACK")
ParamElem.text = ParamCALLBACK
TCElem.appendChild ParamElem
Set ParamElem = Nothing
```

```
Set ParamElem = xmlDoc.createElement("EXEC-TIME")
ParamElem.text = ExecTime
TCElem.appendChild ParamElem
```

```

Set ParamElem = Nothing

Set ParamElem = xmlDoc.createElement("EXEC-DURATION")
ParamElem.text = ExecDuration
ParamElem.setAttribute "unit", "milliseconds"
TCElem.appendChild ParamElem
Set ParamElem = Nothing

Set ParamElem = xmlDoc.createElement("COMMENT")
ParamElem.text = Comment
TCElem.appendChild ParamElem
Set ParamElem = Nothing

root.appendChild TCElem

End Sub

```

### ***Create XML Document Object***

The XML DOM Document object must be created first before test execution may commence. A root of the XML document is created along with a comment showing the Visual OPCTest Client application version. Once the xmlDoc variable is set, it may be used in other subroutines to store test case data and test statistics.

```

Public Sub CreateXMLdoc

    REM *** ROOT

    Set xmlDoc = CreateObject("msxml2.DOMDocument.4.0")

    xmlString = "<?xml version=""1.0"" encoding=""UTF-8""
standalone=""yes"" ?><OPCTEST></OPCTEST>"
    xmlDoc.loadXML xmlString

    Set root = xmlDoc.documentElement

    REM *** COMMENTS
    Set comment = xmlDoc.createComment("Visual OPCTest Client "
    & OPC.GetAppVersion())
    root.appendChild comment
    Set comment = Nothing

End Sub

```

### ***Send Email Notification Function***

After all the test cases are executed and the XML report file has been created, a notification may be sent to the programmer 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, XML file attachment, and message body. Once the email is sent, the email system object is released from memory.

```

Sub NotifyDeveloperByEmail(EmailAddress, XMLFilename)

    OPC.StatusPaneMsg "Generating email to " & EmailAddress

    Set golApp = CreateObject("Outlook.Application")
    Set objNewMail = golApp.CreateItem(olMailItem)
    objNewMail.Recipients.Add EmailAddress
    objNewMail.Attachments.Add XMLFilename
    objNewMail.Subject = "XML Test Results Notification"
    objNewMail.Body = "XML Test Results Attached. Please verify
results."
    objNewMail.Send

    Set objNewMail = Nothing
    Set golApp = Nothing

End Sub

```

### ***Executing Test Cases of Interest***

Before test case execution may begin, we need to create the XML DOM Document object by calling our local subroutine:

```
call CreateXMLdoc()
```

Set the counter variables to 0 before test case execution function calls. These values will be inserted into the XML report file in the STATS section:

```

CountFail = 0
CountWarning = 0
CountTotal = 0

```

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 TCN (test case number) as a 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 XML document and the next test case may be executed.

Once all the test cases of interest are executed, XML document is saved in an external file and it is then emailed to the developer responsible for bug fixing:

```

call StoreStats()

XMLFilename = "C:\MyTests\MyTestResults.xml"
EmailAddress = "Developer@TheCompanyXYZ.com"

xmlDoc.save(XMLFilename)

```

```
call NotifyDeveloperByEmail(EmailAddress, XMLFilename)
```

Just before our script completes, we should release the XML DOM Document object from memory.

```
Set root = Nothing  
Set xmlDoc = Nothing
```

Please note, only test cases with status set to FAIL or WARNING are placed in the XML document (as defined by our `ExecTestCase` subroutine). Changing this subroutine to store all the 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 a custom XML report file, we can place collected test case data in a text files or an external database.

## Sample Code File

Complete sample code file: `MyCustomXMLTestReport.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](http://www.opctest.com)  
[www.terravic.com](http://www.terravic.com)

