Tech Note 1043

Wonderware Application Server Scripting Implementations Part 2: Read and Write Excel Data

Topic#: #002882
Created: May 2014

Introduction

Generally people use a database such as SQL Server, Oracle or Access to store data. However, Excel can also act as a database. Using Excel's rich file types and functionality, you can enhance your Application Server application.

This *Tech Note* introduces the approach of reading and writing Excel data from the IDE via a .NET control. It contains the following sections:

- **Read and write the Excel data using an Application Server Script**
- **Explain the SQL techniques for Excel data processing in the .NET Control**

Application Versions

- Wonderware Application Server 2012 R2 and later
- InTouch 2012 R2 and later

Read and Write the Excel Data Using an Application Server Script

This section explains reading and writing Excel data from Application Server using a custom script function library.

1. Download the **ExcelOperations.zip** file to your GR node and extract the contents to a local directory.

   In this example, we use **C:\TEMP\Excel**.
2. Create a new Galaxycalled **GalaxyExcel** and open it in the IDE.
3. Import the **ExcelOperations.dll** file and confirm the import succeeds (Figures 1,2 and 3 below).

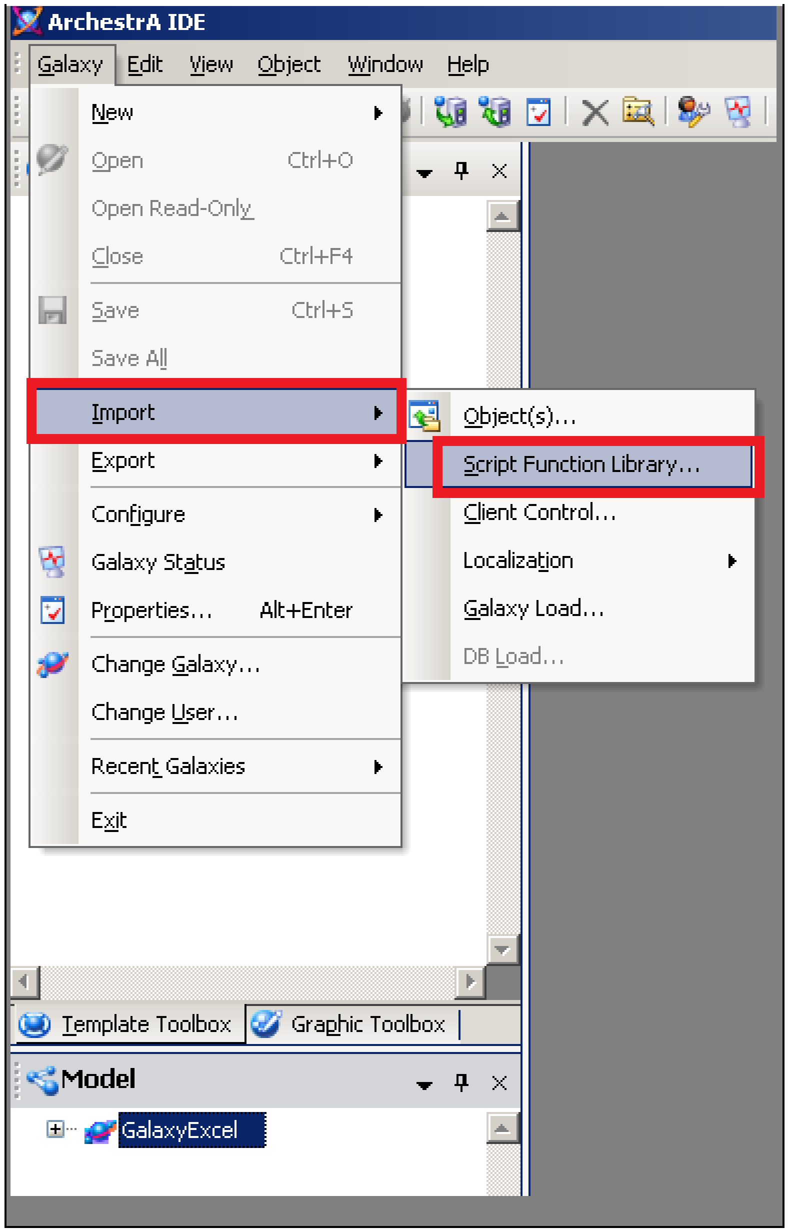
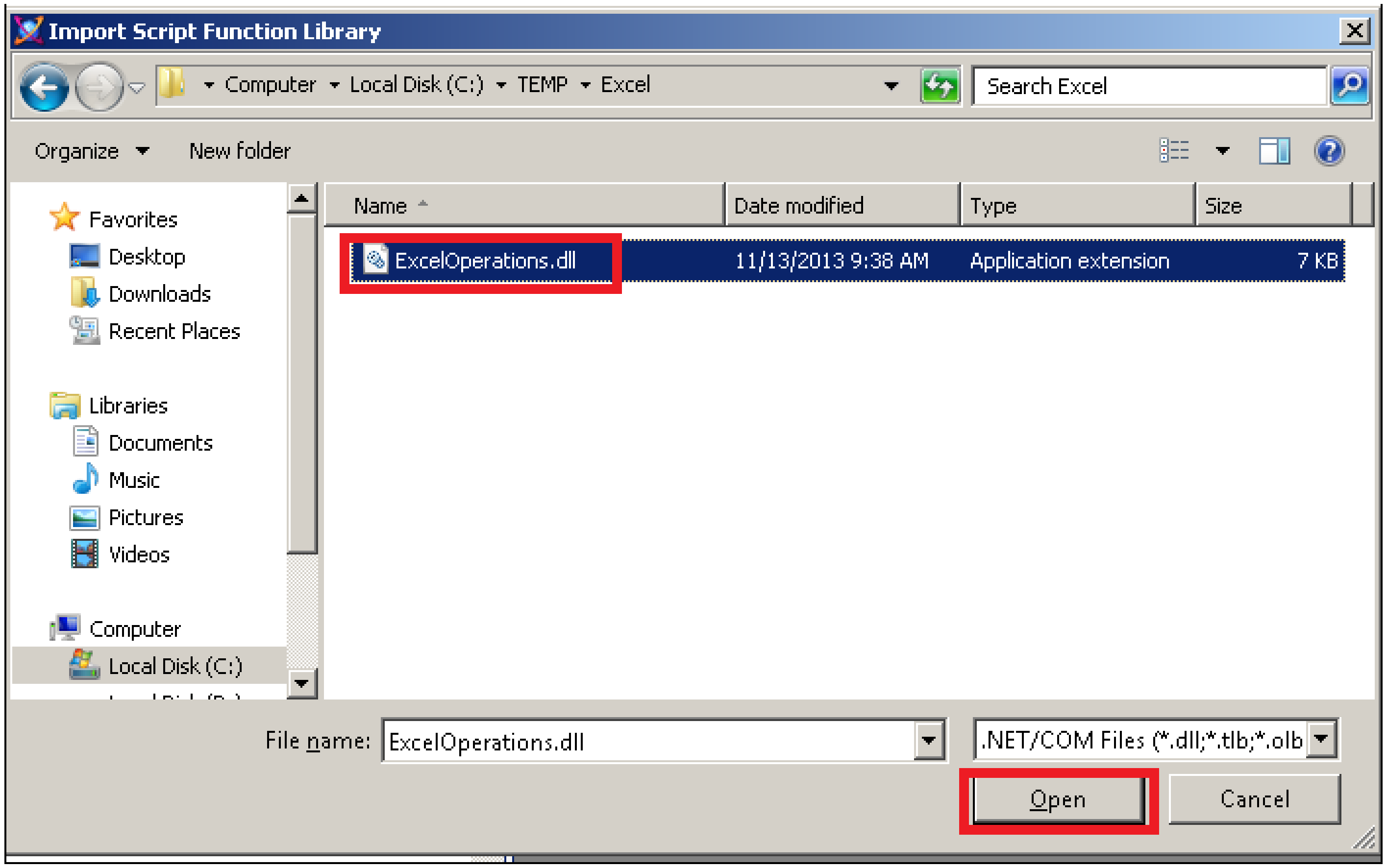


FIGURE 1: SELECT GALAXY>IMPORT>SCRIPT FUNCTION LIBRARY OPTION

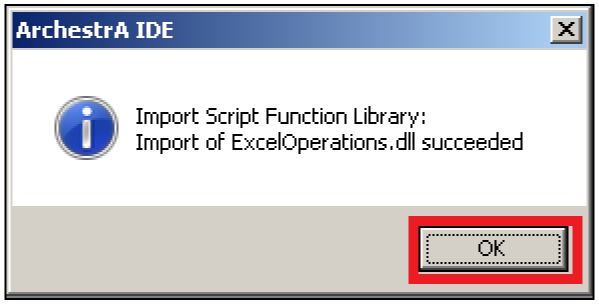


Name	Date modified	Type	Size
ExcelOperations.dll	11/13/2013 9:38 AM	Application extension	7 KB

File name: ExcelOperations.dll

.NET/COM Files (*.dll;*.tlb;*.olb

Figure 2: Browse And Select The DLL



Import Script Function Library:
Import of ExcelOperations.dll succeeded

Figure 3: Verify Import Succeeded

4. Import **ExcelUDA.aaPKG** and **$READnWRITE2Excel.aaPKG** into the Galaxy (Figures 4, 5, 6 and 7 below).

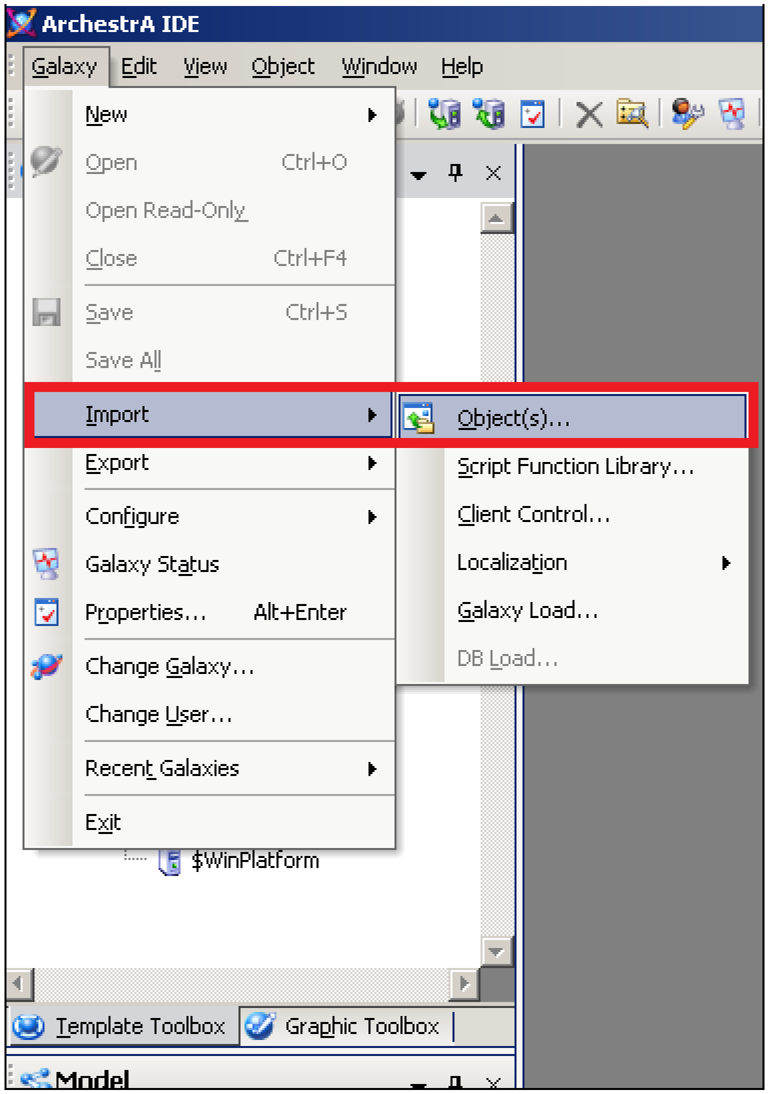
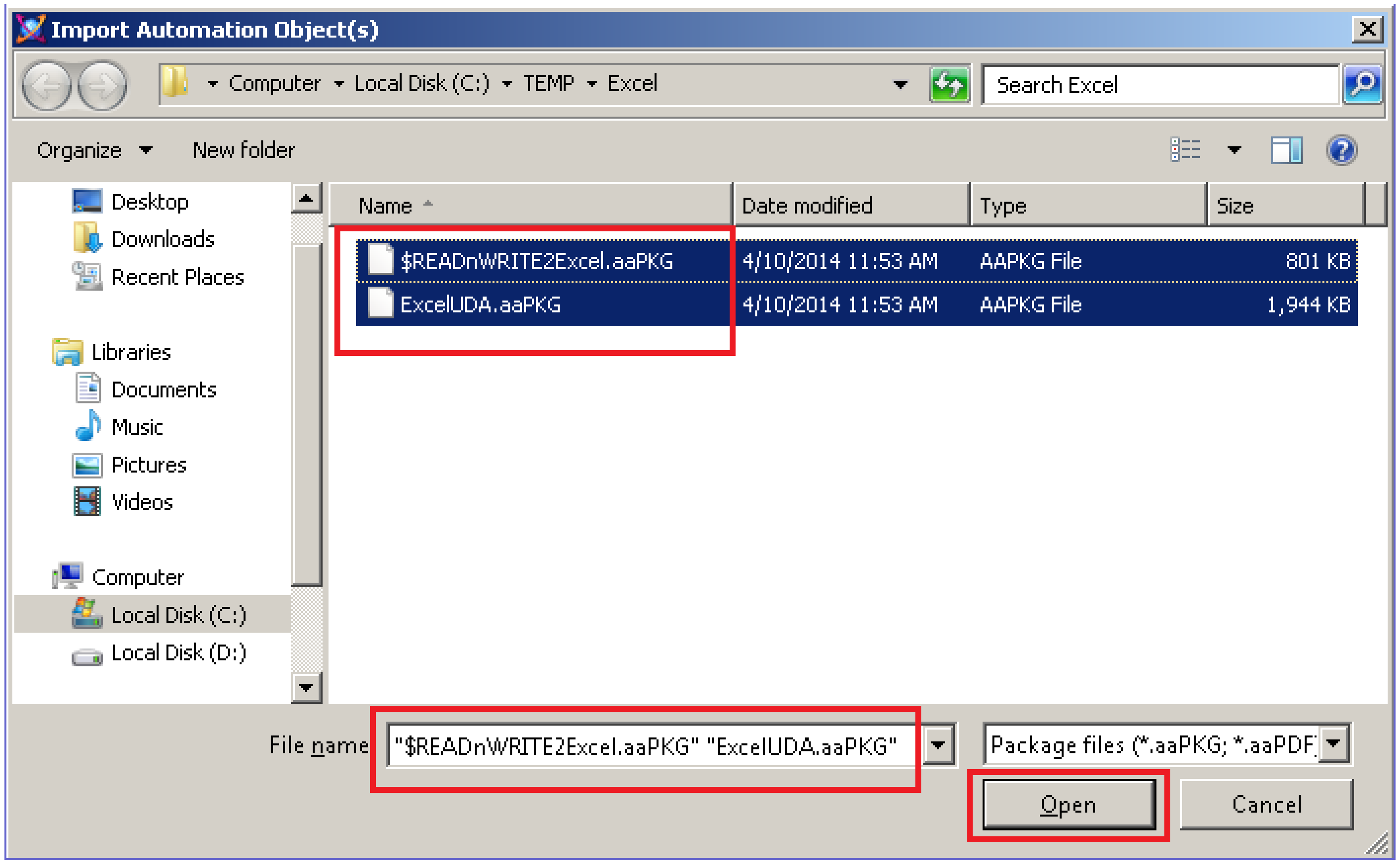


FIGURE 4: SELECT GALAXY > IMPORT > OBJECT(S) OPTION

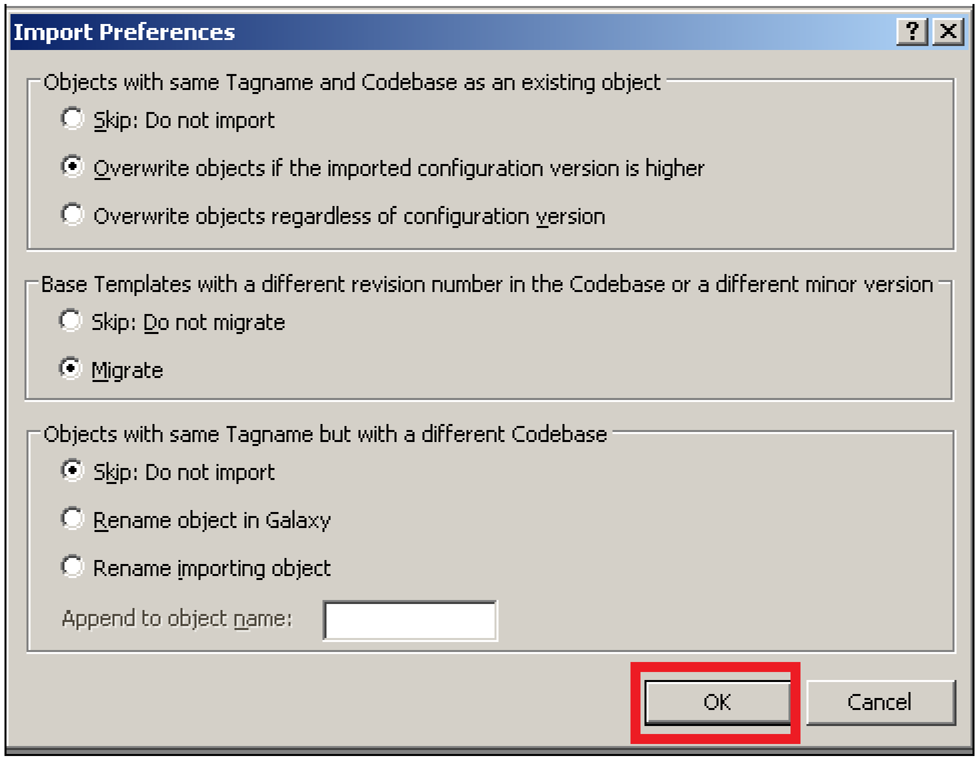


Name	Date modified	Type	Size
$READnWRITE2Excel.aaPKG	4/10/2014 11:53 AM	AAPKG File	801 KB
ExcelUDA.aaPKG	4/10/2014 11:53 AM	AAPKG File	1,944 KB

File name: "$READnWRITE2Excel.aaPKG" "ExcelUDA.aaPKG"

Package files (*.aaPKG; *.aaPDF

Figure 5: Select Both Objects And Open



Import Preferences

Objects with same Tagname and Codebase as an existing object

- ( ) Skip: Do not import
- (●) Overwrite objects if the imported configuration version is higher
- ( ) Overwrite objects regardless of configuration version

Base Templates with a different revision number in the Codebase or a different minor version

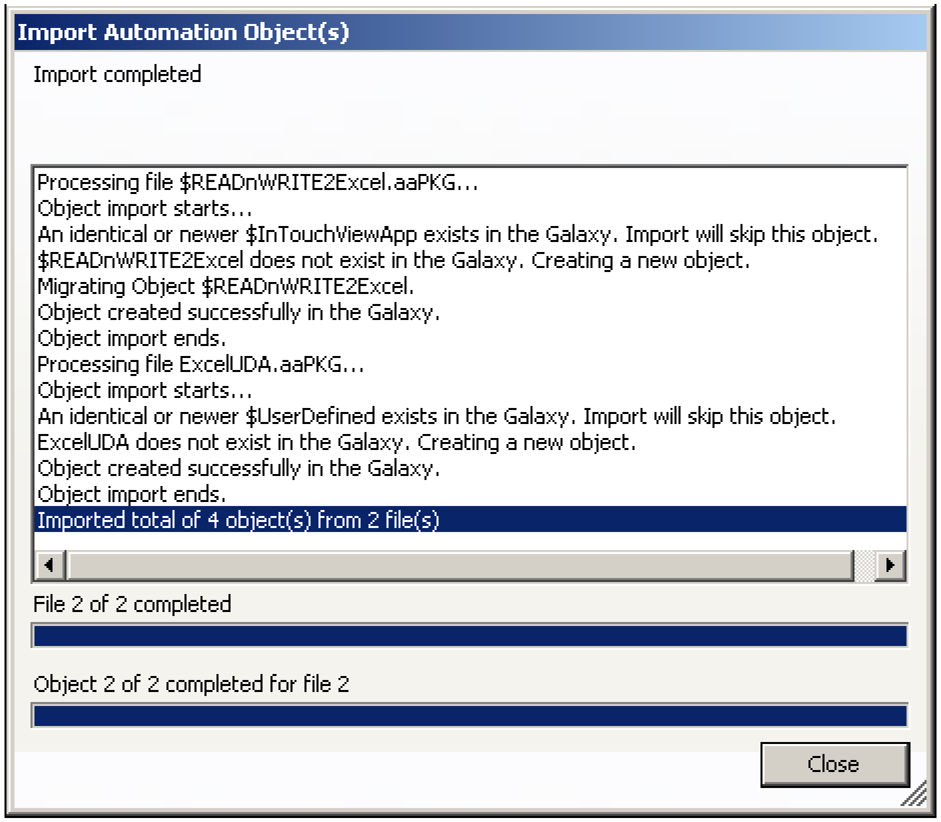
- ( ) Skip: Do not migrate
- (●) Migrate

Objects with same Tagname but with a different Codebase

- (●) Skip: Do not import
- ( ) Rename object in Galaxy
- ( ) Rename importing object

Append to object name:

Figure 6: Accept All Default Import Preferences



Import Automation Object(s)

Import completed

```
Processing file $READnWRITE2Excel.aaPKG...
Object import starts...
An identical or newer $InTouchViewApp exists in the Galaxy. Import will skip this object.
$READnWRITE2Excel does not exist in the Galaxy. Creating a new object.
Migrating Object $READnWRITE2Excel.
Object created successfully in the Galaxy.
Object import ends.
Processing file ExcelUDA.aaPKG...
Object import starts...
An identical or newer $UserDefined exists in the Galaxy. Import will skip this object.
ExcelUDA does not exist in the Galaxy. Creating a new object.
Object created successfully in the Galaxy.
Object import ends.
Imported total of 4 object(s) from 2 file(s)
```

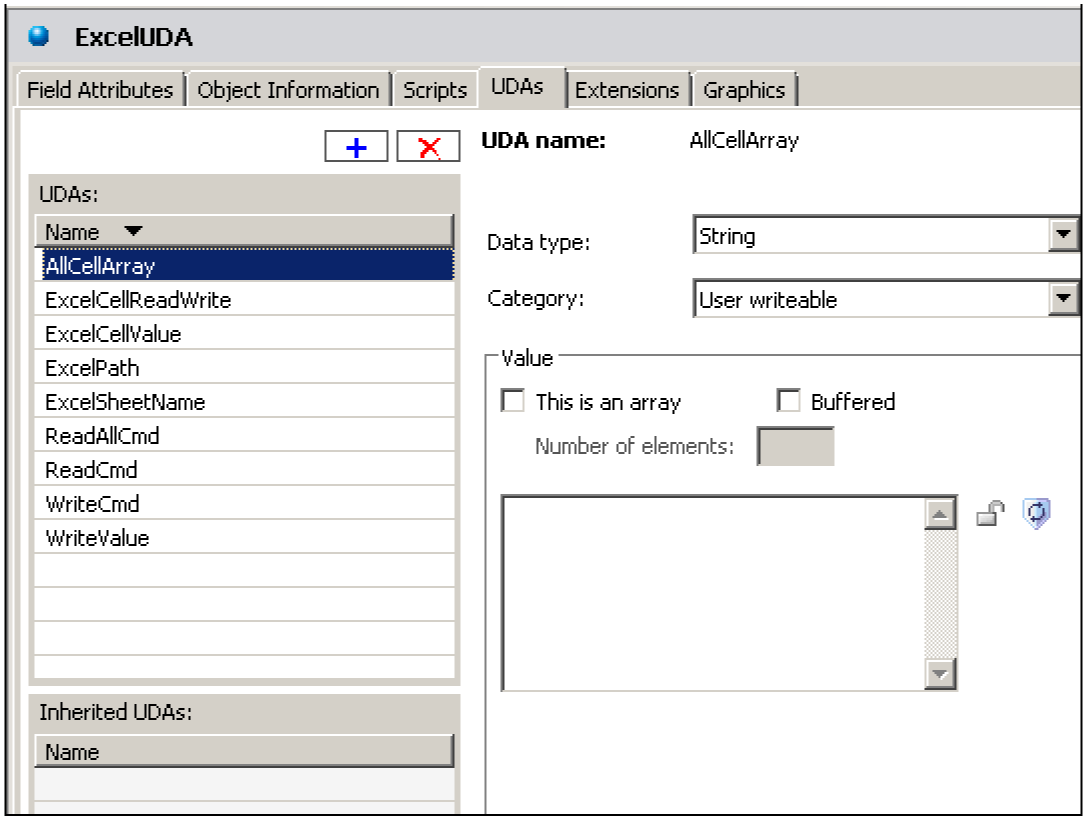
File 2 of 2 completed

Object 2 of 2 completed for file 2

Figure 7: Close the Window after Successful Import

5. Go to the Deployment View, then locate the **ExcelUDA** object and open it.

6. Click the **UDAs** tab. The UDAs listed will be used to Read and write data to Excel from InTouch (Figure 8 below).



UDA name: AllCellArray

Data type: String

Category: User writeable

Value

☐ This is an array ☐ Buffered

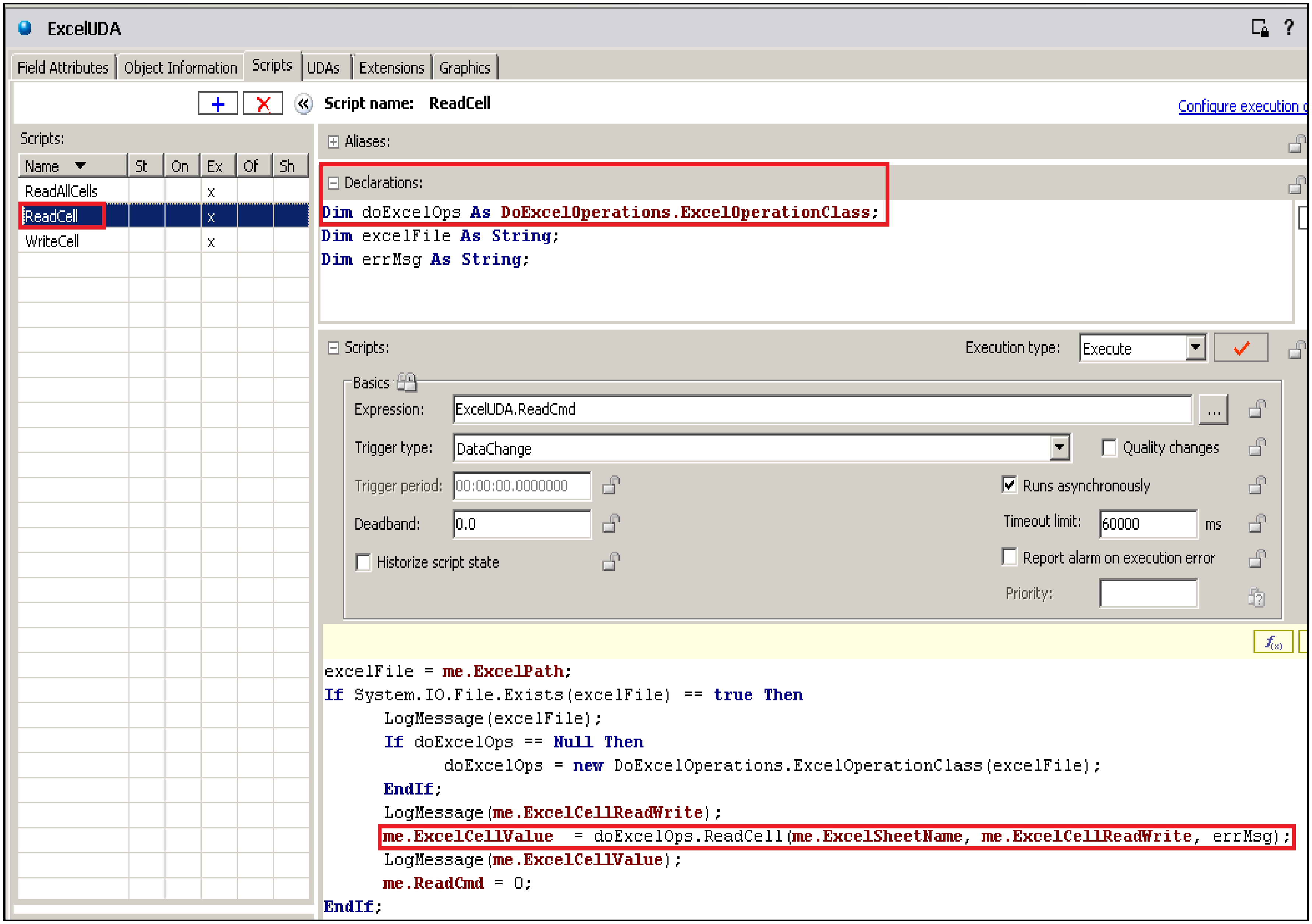
Number of elements:

FIGURE 8: LIST OF UDAS USED IN READ AND WRITE OPERATION

- **AllCellArray**: String Data Type: Holds the concatenated values from multiple Excel cells.
- **ExcelCellReadWrite**: String Data Type: This contains the Excel cell information from where the data will be read or written to. This example uses **A1**.
- **ExcelCellValue**: String Data Type: Holds the Excel cell value.
- **ExcelPath:** String Data Type: Holds the path of the Excel file. This example uses C:\TEMP\Excel\ReadWrite.xlsx. Make sure you have this path with **ReadWrite.xlsx** file in it on your local machine.
- **ExcelSheetName**: String Data Type: Holds the Excel sheet name. This example uses **Sheet1**.
- **ReadAllCmd**: Boolean Data Type: Used to trigger the data read from multiple Excel cells.
- **ReadCmd**: Boolean Data Type: Used to trigger the data read from Excel cell.
- **WriteCmd**: Boolean Data Type: Used to trigger the data write to Excel cell.
- **Write Value**: String Data Type: Holds the value to be written to the Excel cell.

7. Click the **Scripts** tab.

- Use the Script **ReadCell** to read data from an Excel cell (Figure 9 below).



```
Dim doExcelOps As DoExcelOperations.ExcelOperationClass;
Dim excelFile As String;
Dim errMsg As String;
```

Execution type: Execute

Expression: ExcelUDA.ReadCmd

Trigger type: DataChange

Quality changes [ ]

Trigger period: 00:00:00.0000000

Runs asynchronously [✓]

Deadband: 0.0

Timeout limit: 60000 ms

Historize script state [ ]

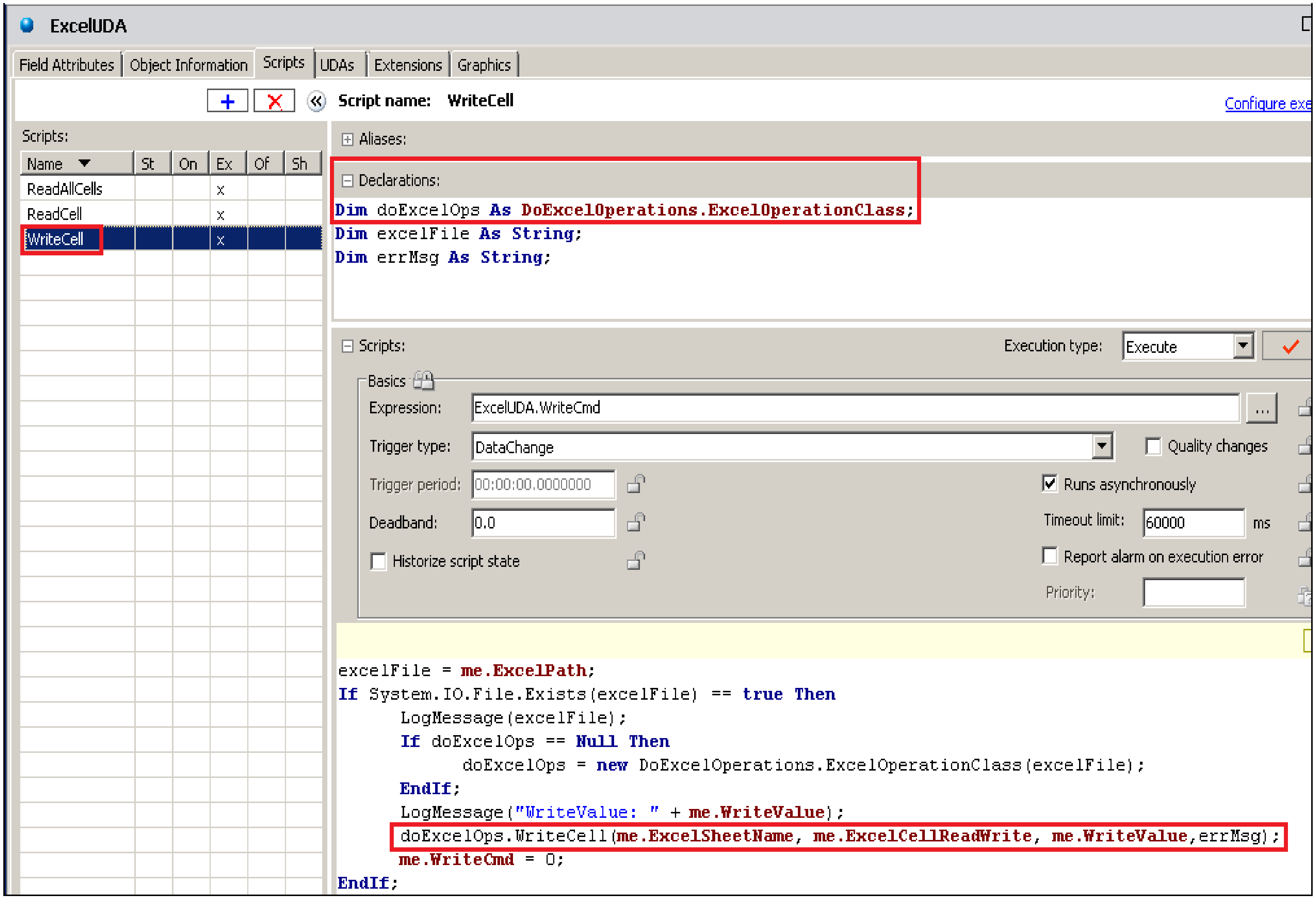
Report alarm on execution error [ ]

Priority:

```
excelFile = me.ExcelPath;
If System.IO.File.Exists(excelFile) == true Then
      LogMessage(excelFile);
      If doExcelOps == Null Then
            doExcelOps = new DoExcelOperations.ExcelOperationClass(excelFile);
      EndIf;
      LogMessage(me.ExcelCellReadWrite);
      me.ExcelCellValue  = doExcelOps.ReadCell(me.ExcelSheetName, me.ExcelCellReadWrite, errMsg);
      LogMessage(me.ExcelCellValue);
      me.ReadCmd = 0;
EndIf;
```

Figure 9: Script To Read Data From Excel Cell

- Use the Script WriteCell to write Data to an Excel cell (Figure 10 below).



Script name: WriteCell

Scripts:

Name	St	On	Ex	Of	Sh
ReadAllCells			x		
ReadCell			x		
WriteCell			x		

Aliases:

Declarations:

```
Dim doExcelOps As DoExcelOperations.ExcelOperationClass;
Dim excelFile As String;
Dim errMsg As String;
```

Scripts: Execution type: Execute

Basics

Expression: ExcelUDA.WriteCmd

Trigger type: DataChange

Quality changes

Trigger period: 00:00:00.0000000

Runs asynchronously [✓]

Deadband: 0.0

Timeout limit: 60000 ms

Historize script state

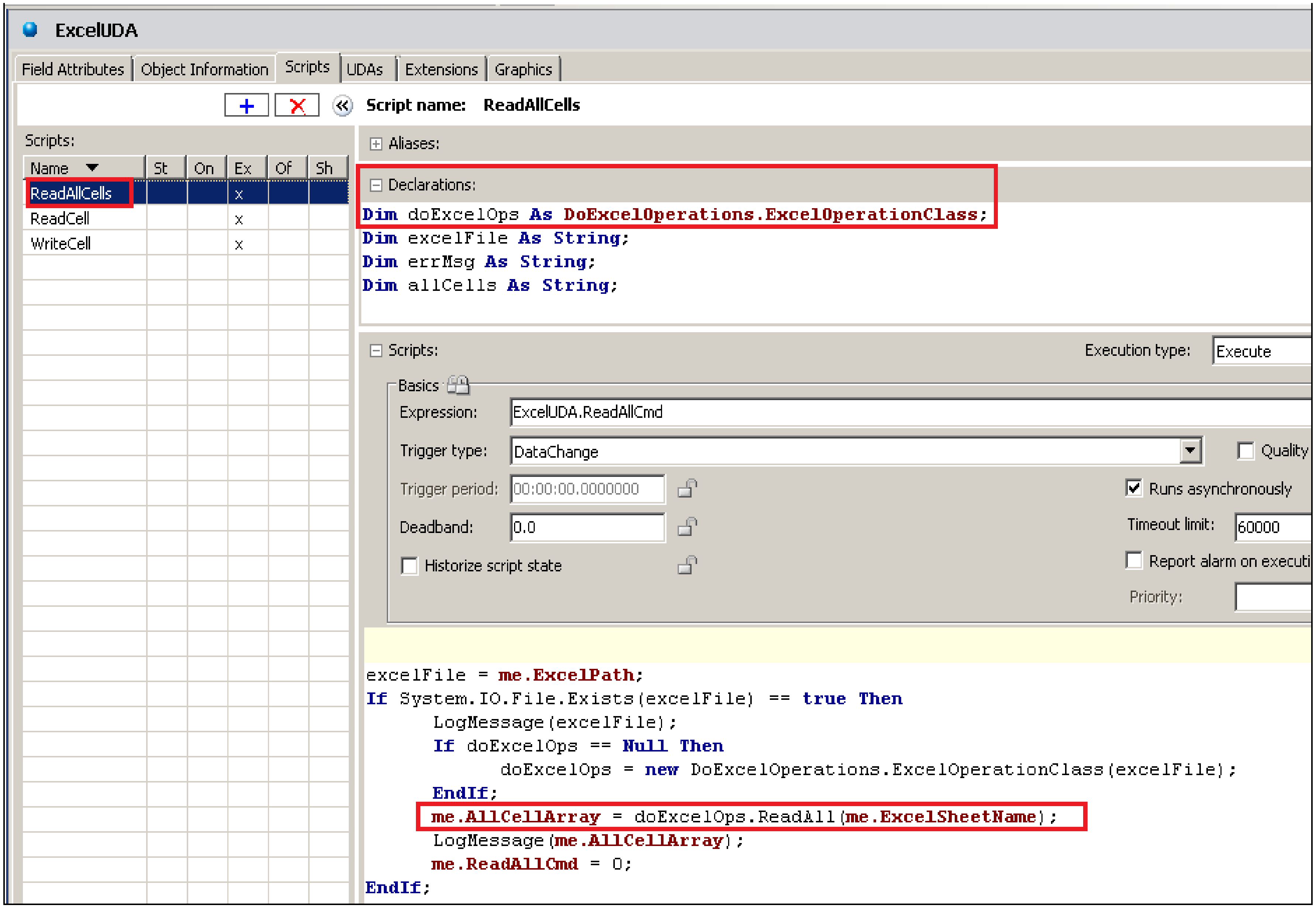
Report alarm on execution error

Priority:

```
excelFile = me.ExcelPath;
If System.IO.File.Exists(excelFile) == true Then
      LogMessage(excelFile);
      If doExcelOps == Null Then
            doExcelOps = new DoExcelOperations.ExcelOperationClass(excelFile);
      EndIf;
      LogMessage("WriteValue: " + me.WriteValue);
      doExcelOps.WriteCell(me.ExcelSheetName, me.ExcelCellReadWrite, me.WriteValue,errMsg);
      me.WriteCmd = 0;
EndIf;
```

Figure10: Script To Write Data Into Excel Cell

- Use the Script **ReadAllCells** to read from multiple cells in the Excel (Figure 11 below).



```
Dim doExcelOps As DoExcelOperations.ExcelOperationClass;
Dim excelFile As String;
Dim errMsg As String;
Dim allCells As String;
```

Execution type: Execute
Expression: ExcelUDA.ReadAllCmd
Trigger type: DataChange
Quality
Trigger period: 00:00:00.0000000
Runs asynchronously ✓
Deadband: 0.0
Timeout limit: 60000
Historize script state
Report alarm on executi
Priority:

```
excelFile = me.ExcelPath;
If System.IO.File.Exists(excelFile) == true Then
      LogMessage(excelFile);
      If doExcelOps == Null Then
            doExcelOps = new DoExcelOperations.ExcelOperationClass(excelFile);
      EndIf;
      me.AllCellArray = doExcelOps.ReadAll(me.ExcelSheetName);
      LogMessage(me.AllCellArray);
      me.ReadAllCmd = 0;
EndIf;
```

Figure 11: Script To Read All Data From Excel Cell

8. Close the **ExcelUDA** object
9. Create a new instance of the Platform, App Engine, Area, ViewEngine and a **READnWRITE2Excel** InTouchView object.
10. Assign the objects correctly under the Platform then cascade deploy the Platform (Figure 12 below).

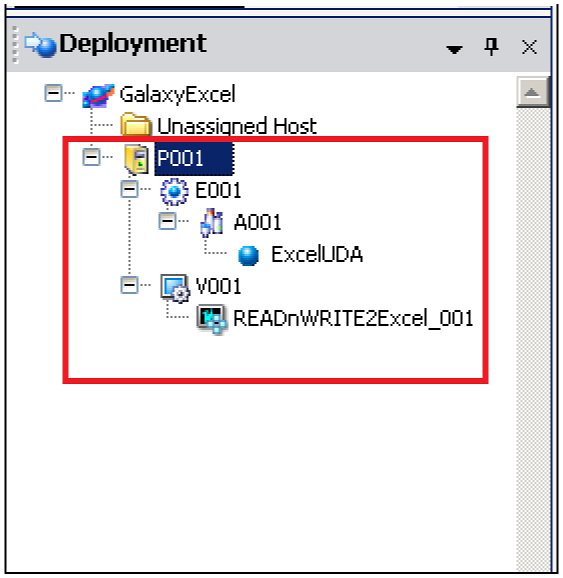


Figure 12: Create Object Instances And Deploy Platform

11. Open the **ReadWrite.xlsx** file and type the string **Hello** in cell A1 (Figure 13 below). Then save and close the file.

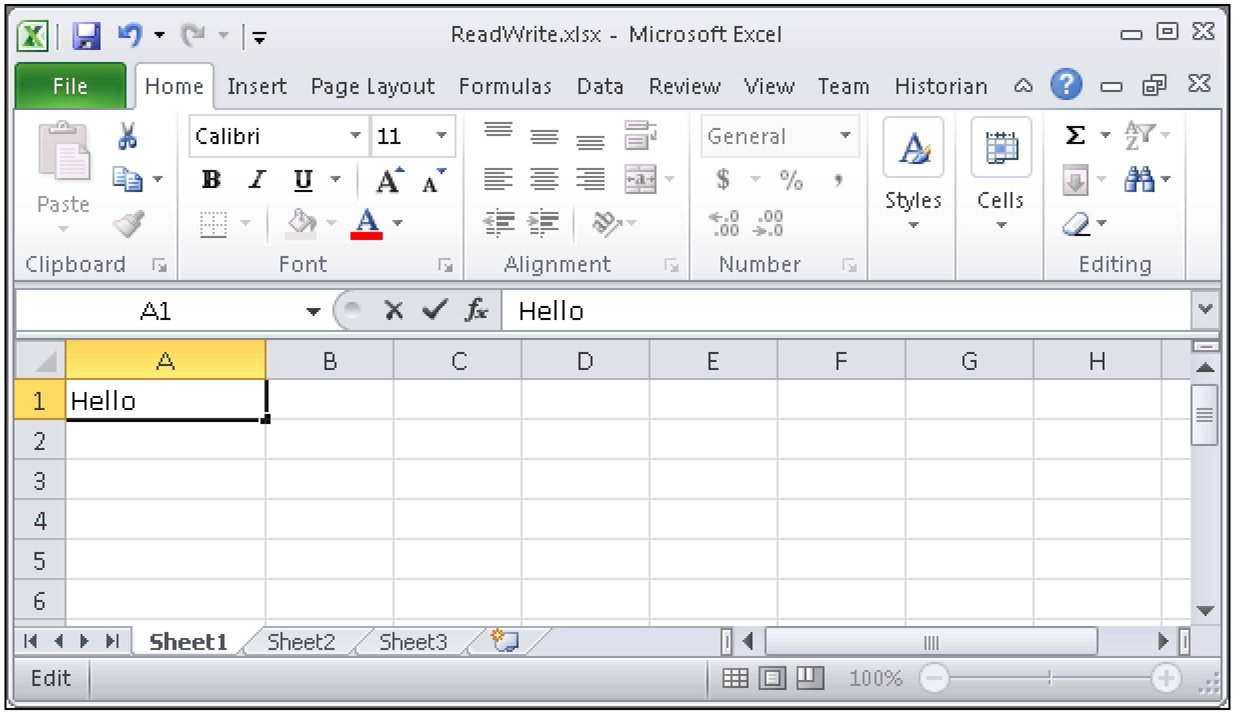
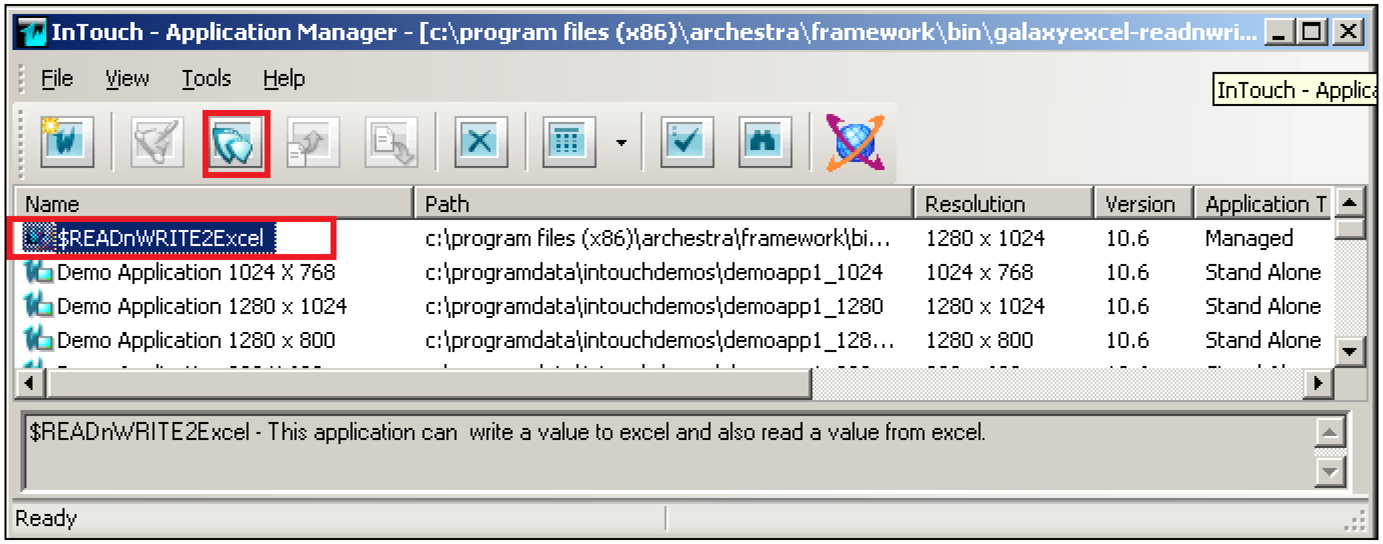


Figure13: Type a Value In Excel Cell A1

12. Open InTouch Application Manager. Find the deployed **$READnWRITE2Excel** object and open it in WindowViewer (Figure 14 and 15).


InTouch - Application Manager - [c:\program files (x86)\archestra\framework\bin\galaxyexcel-readnwri...
File
View
Tools
Help
InTouch - Applica

Name	Path	Resolution	Version	Application T
$READnWRITE2Excel	c:\program files (x86)\archestra\framework\bi...	1280 x 1024	10.6	Managed
Demo Application 1024 X 768	c:\programdata\intouchdemos\demoapp1_1024	1024 x 768	10.6	Stand Alone
Demo Application 1280 x 1024	c:\programdata\intouchdemos\demoapp1_1280	1280 x 1024	10.6	Stand Alone
Demo Application 1280 x 800	c:\programdata\intouchdemos\demoapp1_128...	1280 x 800	10.6	Stand Alone

$READnWRITE2Excel - This application can write a value to excel and also read a value from excel.

Ready

Figure 14: Select The Deployed InTouchView Application

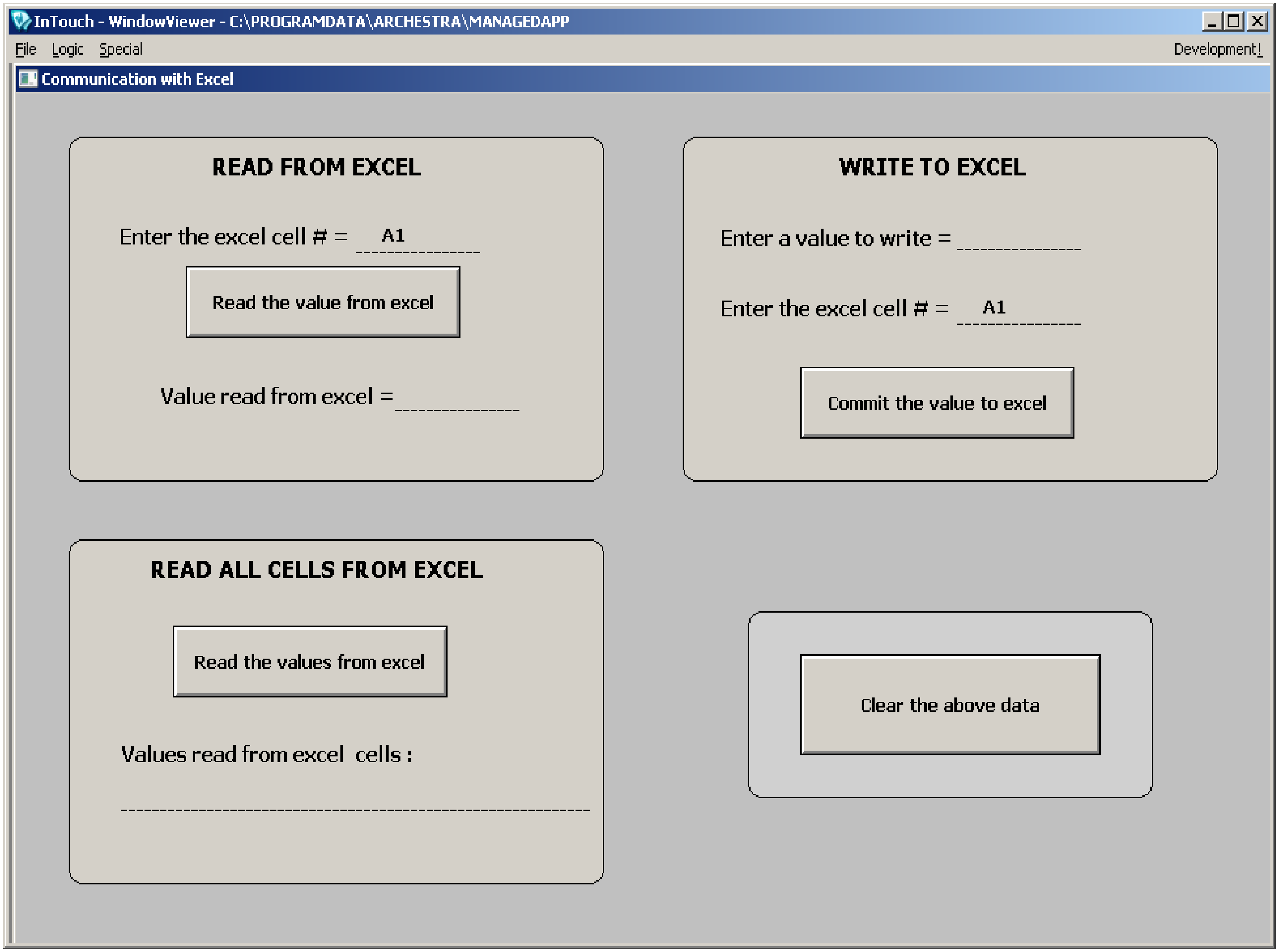

InTouch - WindowViewer - C:\PROGRAMDATA\ARCHESTRA\MANAGEDAPP
File Logic Special
Development!
Communication with Excel
READ FROM EXCEL
Enter the excel cell # = A1
Read the value from excel
Value read from excel =
WRITE TO EXCEL
Enter a value to write =
Enter the excel cell # = A1
Commit the value to excel
READ ALL CELLS FROM EXCEL
Read the values from excel
Values read from excel cells :
Clear the above data

FIGURE 15: OPEN THE DEPLOYED INTOUCHVIEW APPLICATION IN RUNTIME

13. To read the value from Excel, enter the cell # as A1 (this appears by default on WindowViewer) then click the **Read the value from excel** button. The string **Hello** is displayed in the window (Figure 16 below).

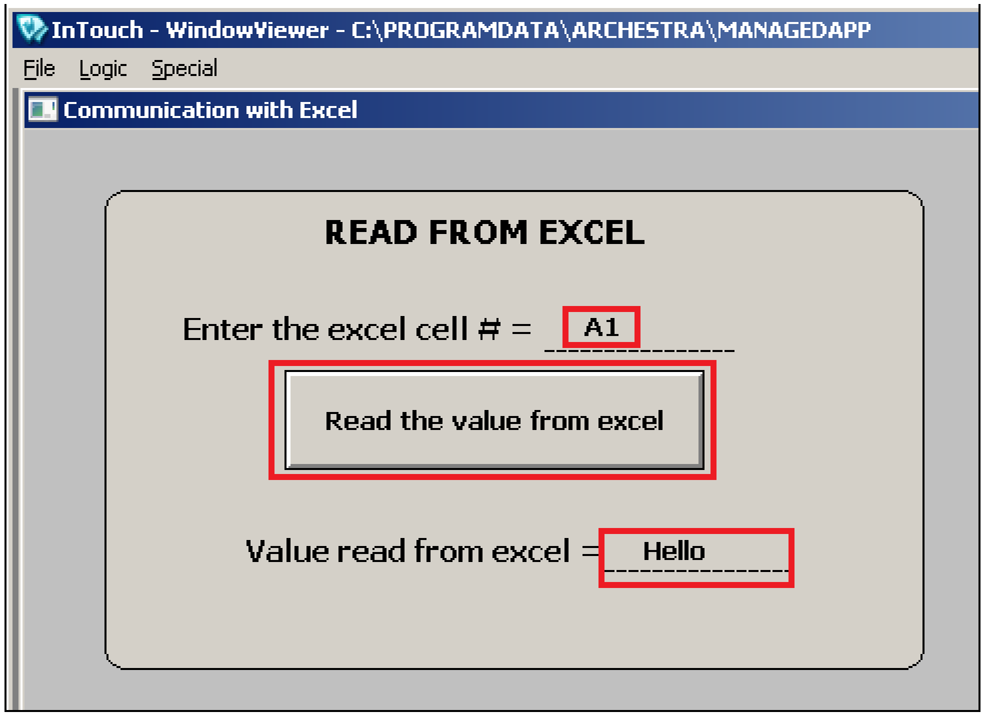
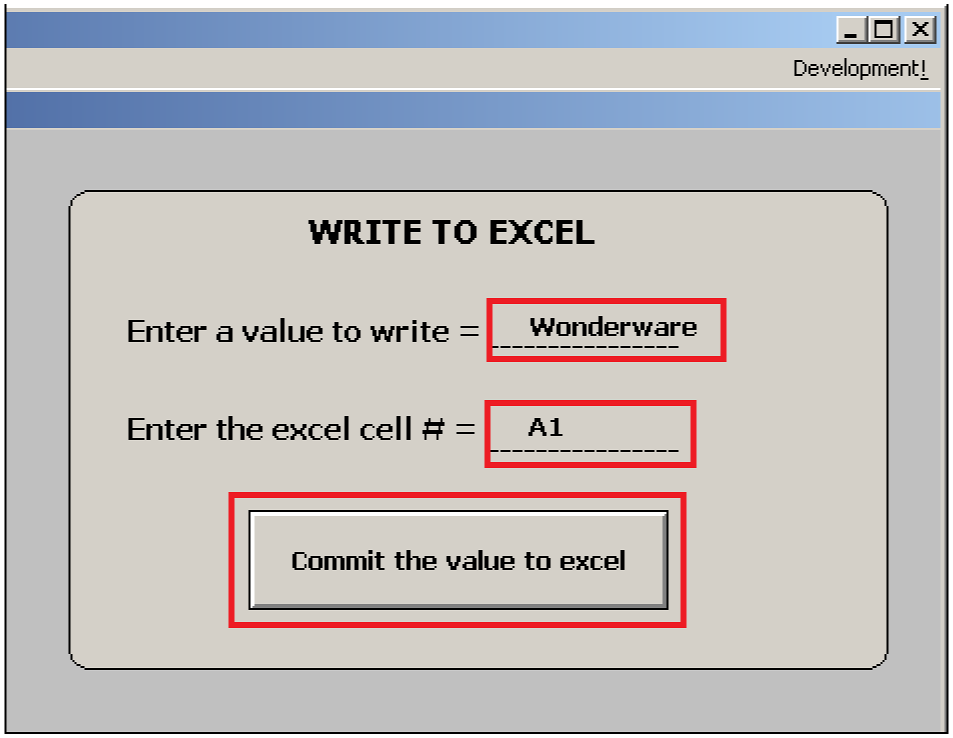

InTouch - WindowViewer - C:\PROGRAMDATA\ARCHESTRA\MANAGEDAPP
File Logic Special
Communication with Excel
READ FROM EXCEL
Enter the excel cell # = A1
Read the value from excel
Value read from excel = Hello

Figure 16: Type a Cell# And Read the Value From Excel

To Write a Value to Excel

1. Type the value to be written as **Wonderware**, type the cell # as **A1** (this appears by default in WindowViewer) then click **Commit the value to excel** button (Figure 17 below).

Note: Before writing to Excel, make sure the **ReadWrite.xlsx** file is closed.



WRITE TO EXCEL

Enter a value to write = Wonderware

Enter the excel cell # = A1

FIGURE 17: TYPE CELL # AND VALUE TO WRITE VALUE INTO EXCEL

The string **Wonderware** will be written into the Excel sheet1 cell A1 (Figure 18 below).

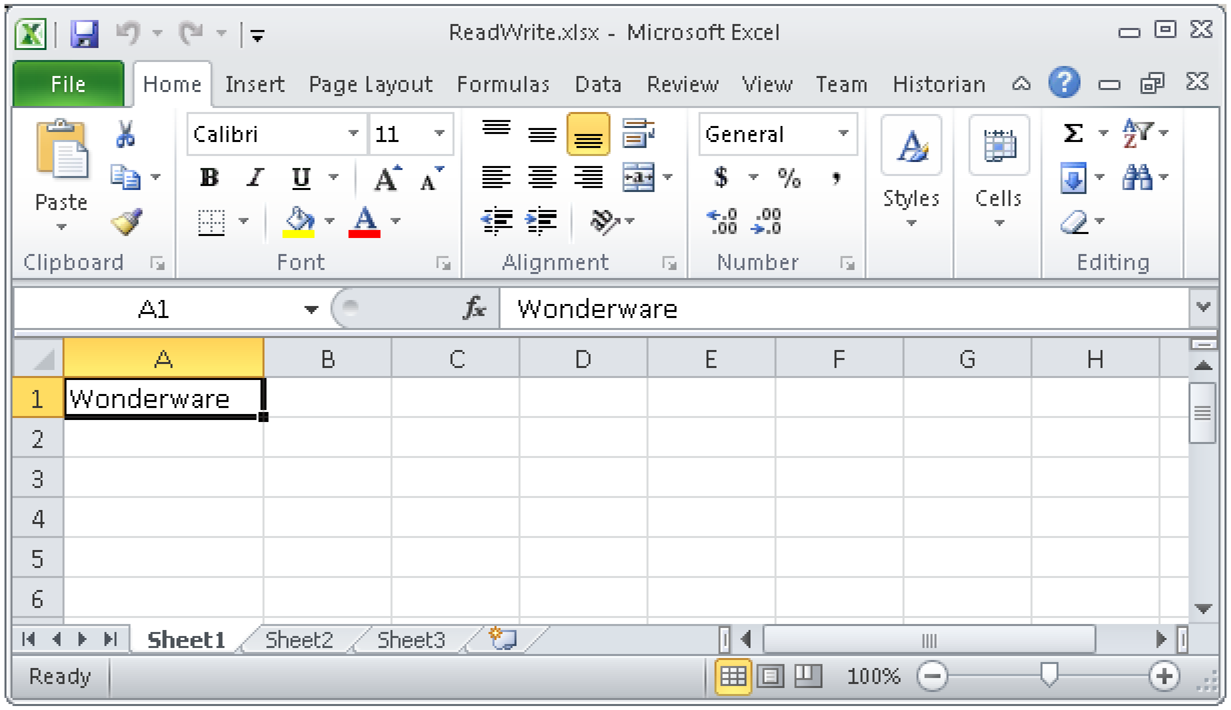
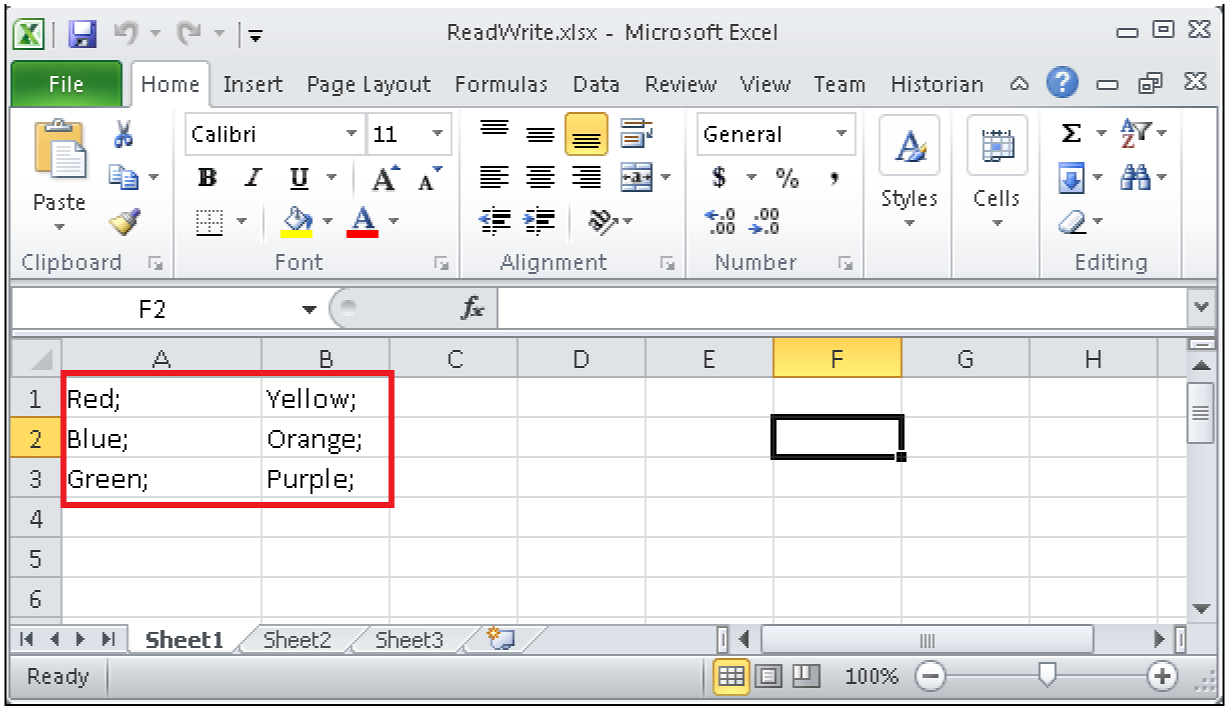


FIGURE18: VERIFY THE VALUE WRITTEN INTO EXCEL

To read values from multiple cells in Excel

1. Open the **ReadWrite.xlsx** file and type your values (Figure 19 below).

   Save and Close the **ReadWrite.xlsx** file.

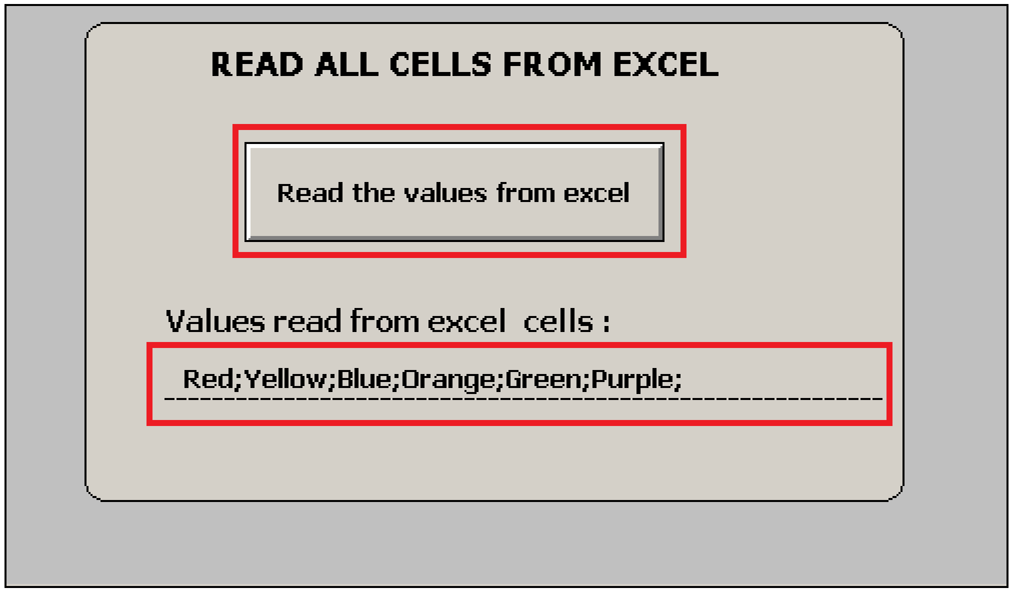


	A	B
1	Red;	Yellow;
2	Blue;	Orange;
3	Green;	Purple;

Figure 19: Multiple Values In Excel

2. In WindowViewer, click the **Read the values from excel** button.

   The values are displayed as a concatenated string (Figure 20 below).


READ ALL CELLS FROM EXCEL
Read the values from excel
Values read from excel cells :

Red;Yellow;Blue;Orange;Green;Purple;

Figure 20: Read Multiple Values From Excel In InTouch Window Viewer

Explain the SQL Techniques for Excel Data Processing in the .NET Control

Excel has many unique features, summarized in the following list.

- Accessing wide range data types and Updating large amounts of data quickly and easily.
- Charting and graphing data sets.
- Perform 'drill-down' analysis on large data sets.
- Storing, analyzing, collecting and sharing data amount the Microsoft Office World and much more...

In this *Tech Note*, our focus is to introduce a .NET component approach that allows Wonderware Application Server (WAS) to read and write data into Excel.

Similar to working with SQL Server or Microsoft Access, you need to follow the same concept of Database connection, dataset and SQL query for the Excel database operations.

- Database connection – In our example in this Tech Note, we have the following connection string in C# format:

  • **connectionString** = string.Format("Provider=Microsoft.ACE.OLEDB.12.0; **Data source**={0}; **Extended Properties**=\"Excel 12.0;**HDR**=NO\";", excelFilePath);
  conOleDB = new System.Data.OleDb.OleDbConnection(**connectionString**);

  • **Provider:** Microsoft.ACE.OLEDB.12.0
  It is the main OLEDB provider used to open the Excel sheet. So far it has been tested with Excel 2010 version. The Provider is a Microsoft Shared DLL is located at:

  -64-bit OS: **C:\Program Files (x86)\Common Files\microsoft shared\OFFICE14\ACEOLEDB.DLL**
  -32-bit OS: **C:\Program Files\Common Files\microsoft shared\OFFICE14\ACEOLEDB.DLL**

  • **Data source:** The full path of the Excel workbook which can have following extension types.

Excel Workbook Type	Description
XLS	97-2003 Excel Workbook
XLSX	2007 or later Excel Workbook
XLSB	2007 or later Office Open XML format saved in a binary format
XLSM	2007 or later Office Open XML format with macros enabled

Note: Refer to the System Platform Readme file (on your System Platform installation disc) under **Third-Party Application Prerequisites** for list of supported Excel versions.

• **Extended Properties:** Extended properties can be applied to Excel workbooks which may change the overall activity of the excel workbook from your program. The available properties are:

Property	Description
HDR	It represents Header of the fields in the excel table. Default is YES. If you don't have fieldnames in the header of your worksheet, you can specify **HDR=NO** which will take the columns of the tables that it finds as f1,f2 etc.
ReadOnly	You can also open excel workbook in read-only mode by specifying ReadOnly=true; By Default Readonly attribute is false, so you can modify data within your workbook.
MaxScanRows	Excel does not provide the detailed schema definition of the tables. Excel needs to scan the rows before deciding the data types of the fields. **MaxScanRows** specifies the number of cells to be scanned before deciding the data type of the column. By default, the value of this is **8**. You can specify any value from 1 – 16 for 1 to 16 rows. You can also make the value to 0 so that it searches all existing rows before deciding the data type. You can change the default behavior of this property by changing the value of [HKLM\Software\Microsoft\Jet\4.0\Engines\Excel\TypeGuessRows] which is 8 by default. Currently, MaxScanRows is ignored, so you need only to depend on TypeGuessRows Registry value. Hopefully, Microsoft fixes this issue to its later versions
IMEX	As mentioned MaxScanRows, Excel has to guess a number of rows to select the most appropriate data type of the column, a serious problem may occur of you have mixed data in one column. Say, you have data of both integer and text on a single column, in that case excel will choose its data type based on majority of the data. Thus it selects the data for the majority data type that is selected, and returns NULL for the minority data type. If the two types are equally mixed in the column, the provider chooses numeric over text. For e.g., In your eight (8) scanned rows, if the column contains four (4) numeric values and four (4) text values, the provider returns four (4) numbers and four (4) null values, but you really want is text values. To work around this problem for data, set "IMEX=1" in the Extended Properties section of the connection string. This enforces the ImportMixedTypes=Text registry setting. You can change the enforcement of type by changing [HKLM\Software\Microsoft\Jet\4.0\Engines\Excel\ImportMixedTypes] to numeric as well.

• **Read Cell**

```
string queryLine = String.Format("select * from [{0}${1}:{2}]", sheetName, cellName, cellName);
cmdOLEDBAdpter = new System.Data.OleDb.OleDbDataAdapter(queryLine, conOleDB);
System.Data.DataTable cellData = new System.Data.DataTable();
cmdOLEDBAdpter.Fill(cellData);
foreach (System.Data.DataRow row in cellData.Rows)
      {
          string cellValue = row[0].ToString();
      }
```

In the above code snippet, we compose a SQL statement that tries to query a single Excel cell content. For example, **Sheet1$A1:A1**.
The remaining code in the snippet is the standard .NET approach to retrieve the data from the Excel Workbook.

The standard .NET approach is to use the **OleDbDataAdapter**, **DataTable** and **DataRow** .NET component. You can click on each component for detailed information.

- **Write Cell**

```
string queryLine = String.Format("UPDATE [{0}${1}:{2}] Set F1=\"{3}\"", sheetName, cellName, cellName, value2Write);
oleDBCmd = new System.Data.OleDb.OleDbCommand(queryLine, conOleDB);
oleDBCmd.ExecuteNonQuery();
```

In this code snippet, the UPDATE is a standard SQL statement to change the value of a specific Excel cell. For example, we can change the value of Sheet1$A1:A1 with the content of **value2Write.**

Note: F1 is the default name for .NET ADO to assign to the first column. For multiple columns, you can use the following format example…

```
string queryLine = String.Format("UPDATE [{0}${1}:{2}] Set F1=\"{3}\" Set F2=\"{4}\"", sheetName, cellName1, cellName2, value2Write1, 
value2Write2);
```

- **Read all cells from an Excel sheet**

```
string queryLine = String.Format("select * from [{0}$]", sheetName);
StringBuilder allContents = new StringBuilder();
    cmdOLEDBAdpter = new System.Data.OleDb.OleDbDataAdapter(queryLine, conOleDB);
    System.Data.DataTable sheetData = new System.Data.DataTable();
    cmdOLEDBAdpter.Fill(sheetData);
    System.Data.DataSet excelDataSet = new System.Data.DataSet();
    excelDataSet.Tables.Add(sheetData);

   foreach (System.Data.DataTable thisTable in excelDataSet.Tables)
    {
        foreach (System.Data.DataRow myRow in thisTable.Rows)
       {
          foreach (System.Data.DataColumn myCol in thisTable.Columns)
          {
             string value = myRow[myCol].ToString();
             allContents.AppendLine(value);
          }
       }

    }
```

In this code snippet, the first part is the same as the **Read single cell** except for the scope of the SQL query.

However, in the data extraction section, we use three layers of foreach loop to retrieve data from Sheet:DataTableàRow:DataRowàCell:DataRow+DataColumn.

Summary

Our .NET control only implements the above three methods. In this Tech Note's Section 1, we have given the usage under Wonderware Application Server.

R. Herunde, E. Xu

Tech Notes are published occasionally by Wonderware Technical Support. Publisher: Invensys Systems, Inc., 26561 Rancho Parkway South, Lake Forest, CA 92630. There is also technical information on our software products at **Wonderware Technical Support.**

For technical support questions, send an e-mail to **wwsupport@invensys.com**.

Back to top