Automating Microsoft Excel with VisualAPL

One way to incorporate the utility of Microsoft Excel in an application system is to 'automate' it with VisualAPL. Microsoft Excel is an ActiveX server, so it can be accessed from any .Net language including VisualAPL.

Because Microsoft Excel is based on Win32, the previous generation Microsoft Windows application programming interface (API), a .Net solution incorporating Microsoft Excel will not be "fully-managed".

This document provides a brief overview of the syntax that can be used to manipulate Microsoft Excel from VisualAPL. It is not meant to be a comprehensive survey of these techniques. Ultimately the programmer must study the Microsoft Excel object model in order to add significant value to an application system by automating Microsoft Excel.

For more details about the Excel object model and how it can be automated from a .Net language, see http://msdn.microsoft.com/en-us/library/syyd7czh(v=VS.80).aspx.

For a relatively simple example of an application system which combines VisualAPL, Microsoft Excel, Windows Presentation Foundation and C# see http://forum.apl2000.com/viewtopic.php?t=478.

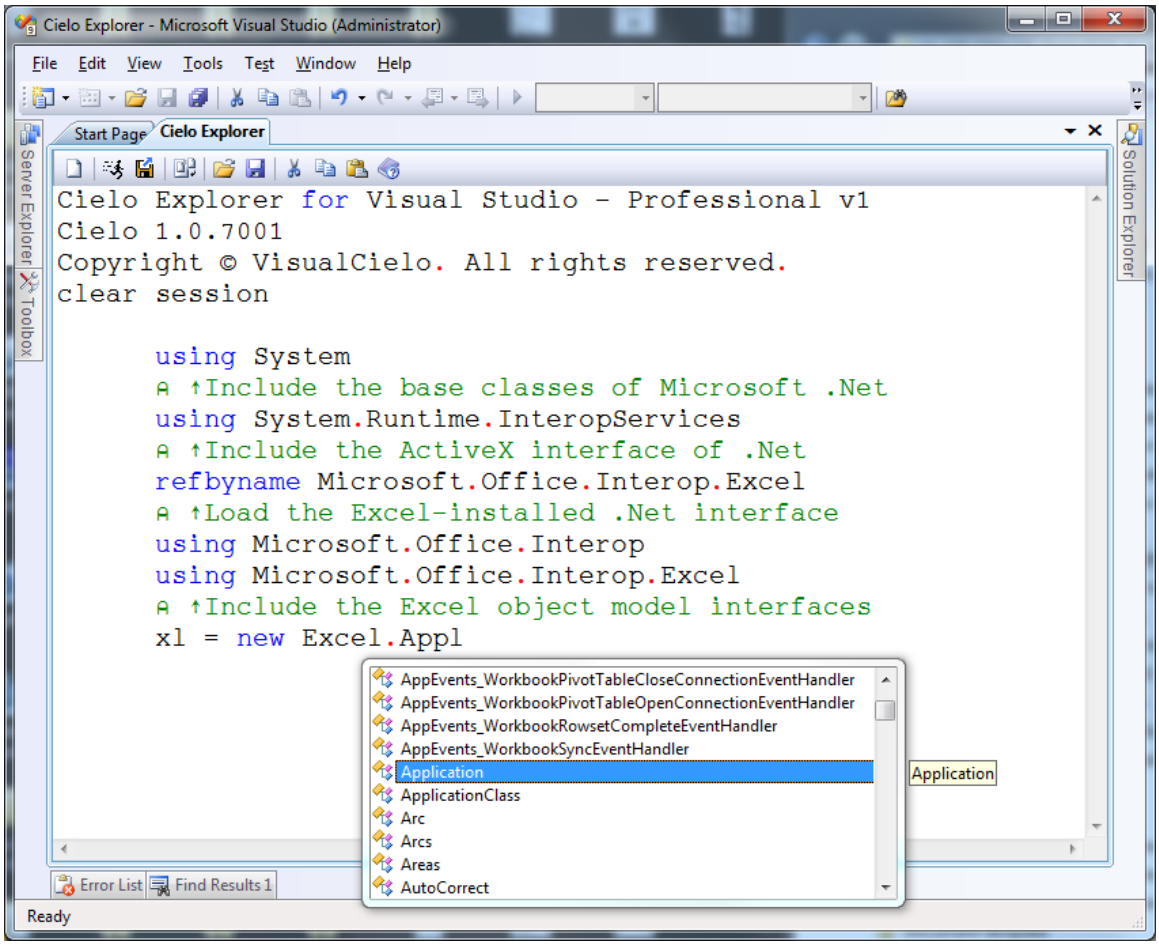
The remainder of this document uses the APLNext VisualAPL Cielo Explorer interactive session in Visual Studio 2008 to illustrate the automation of Microsoft Excel. Remember that any VisualAPL statements perfected in the Cielo Explorer may be incorporated into a compiled VisualAPL .Net assembly or a Cielo Explorer script.

Using the Cielo Explorer makes it very easy to try various VisualAPL statements and see the results immediately. The Visual Studio 'debugger' is fully active within the Cielo Explorer, so a source code statement with errors will be immediately identified.

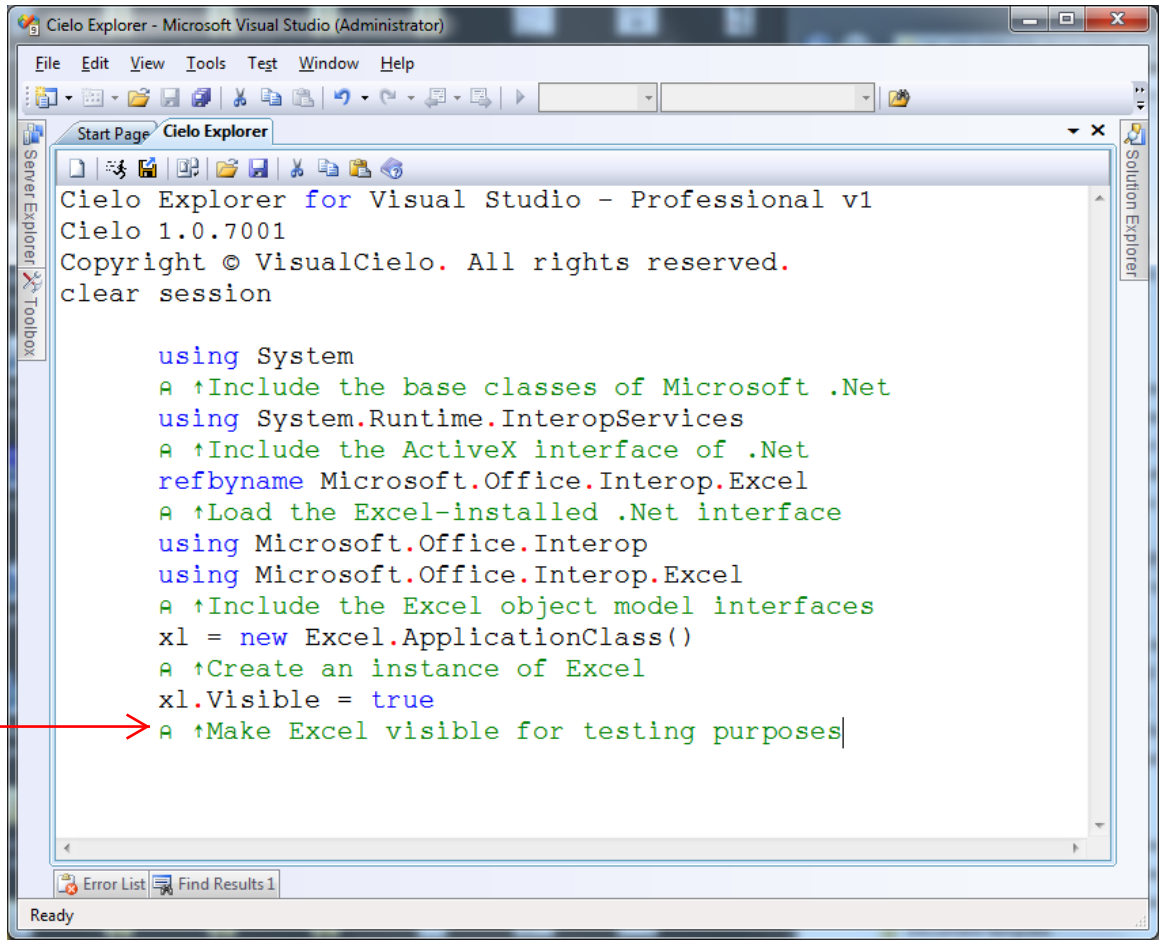
The screen captures of the Cielo Explorer in action have incorporated VisualAPL comments which explain the operations involved.

Open Visual Studio 2008 and make the Cielo Explorer window visible. Enter the VisualAPL statements indicated below to automate Microsoft Excel. These statements will be successful only if Microsoft Excel has been installed on the programmer's workstation.

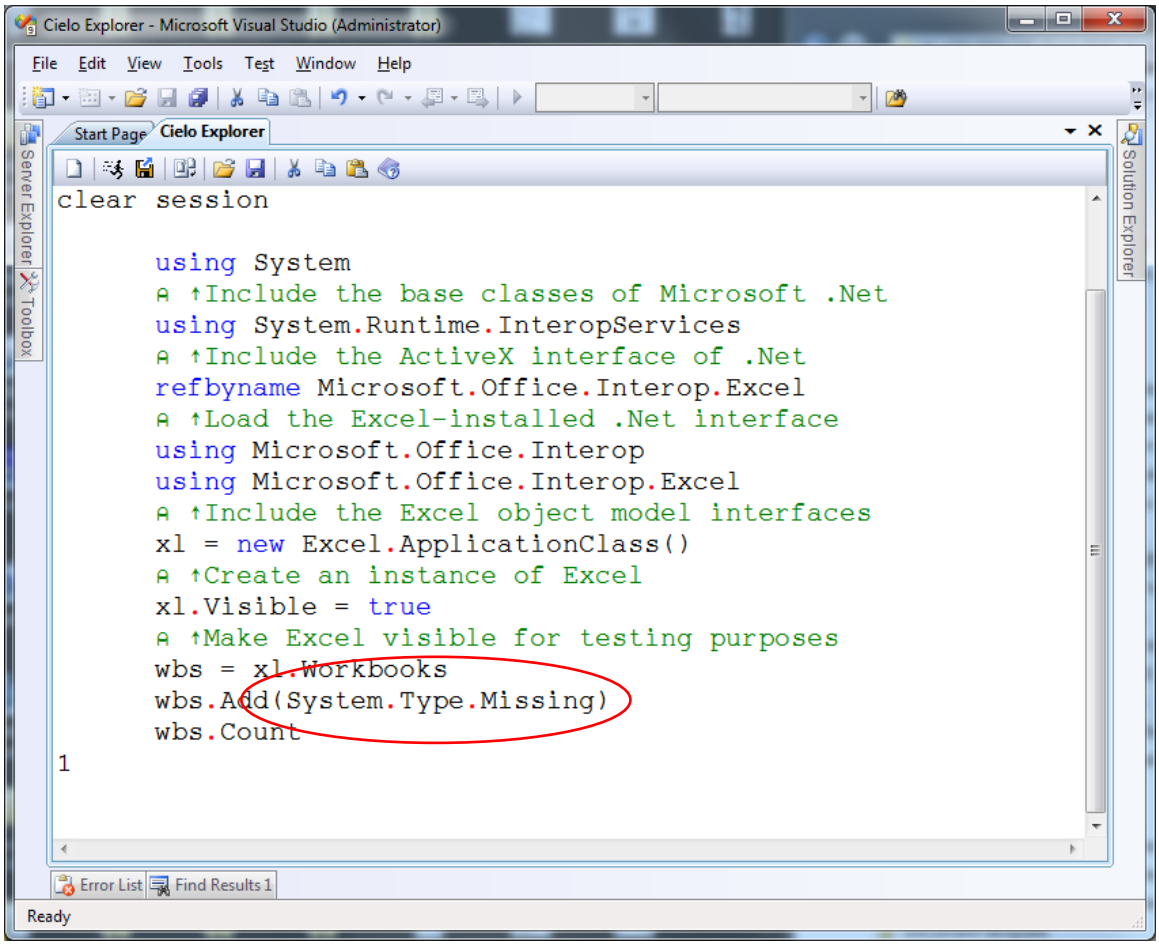
Visual Studio 'Intellisense' context-sensitive documentation is available in the Cielo Explorer, however for certain Microsoft Excel objects, Visual Studio is not able to retrieve complete Intellisense information. This is why it is important for the programmer to study the Microsoft Excel object model to know what methods, properties and events are available.



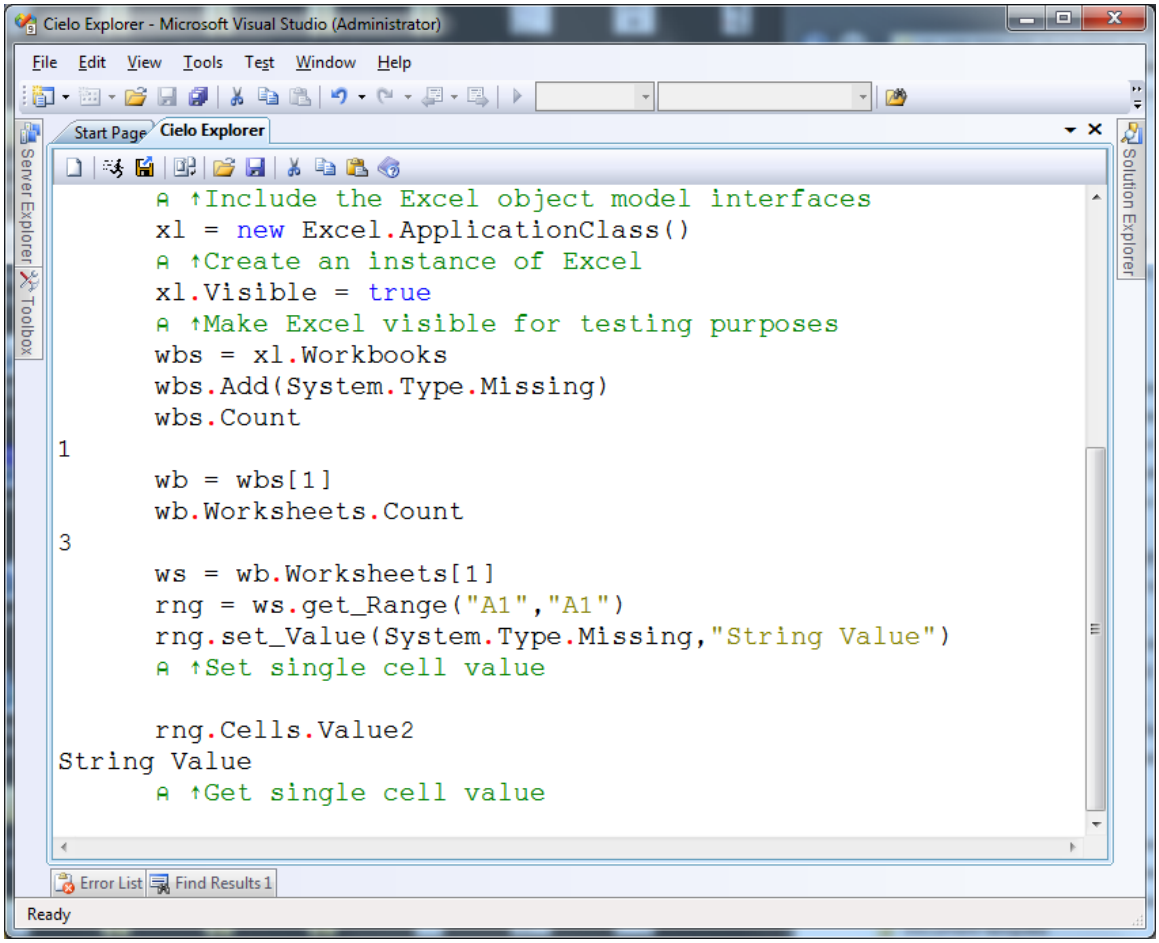
When testing an application system that is automating Excel, it is generally beneficial to make the instance of Excel visible.



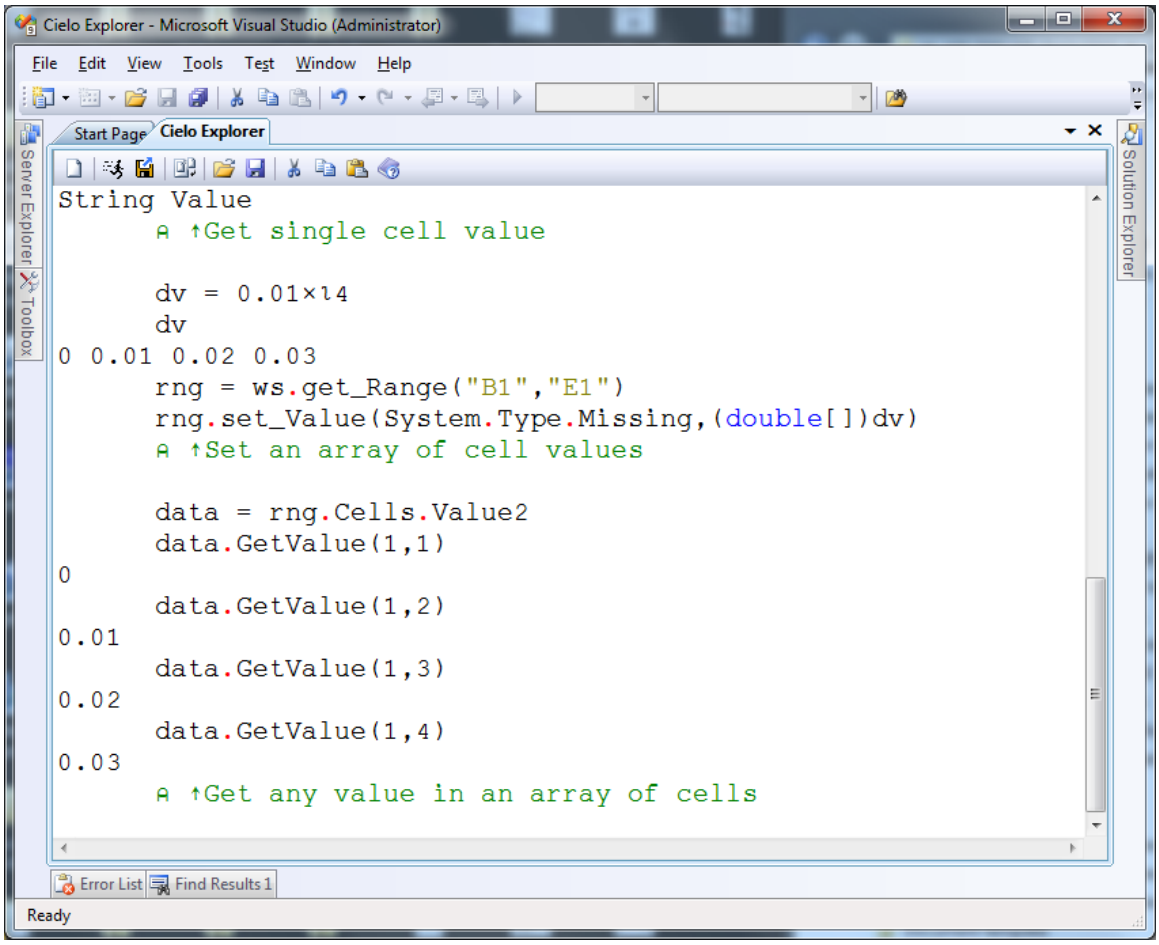
The ‘System.Type.Missing’ object is convenient when using certain Excel methods which have numerous arguments, many of which need not be specified explicitly. Before using System.Type.Missing, check the Excel object model documentation to assure that these additional arguments are actually not critical to the application system automating Excel.



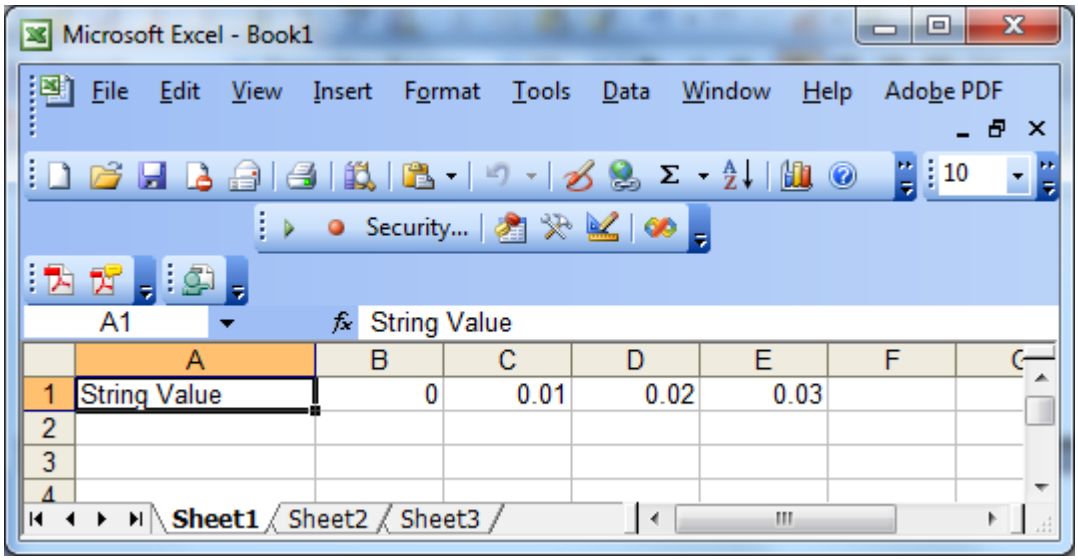
The Excel object model incorporates many subordinate objects such as indexable collections for workbooks, worksheets and cells as well as properties like Count and methods such as Add(), set_Value() and get_Value().



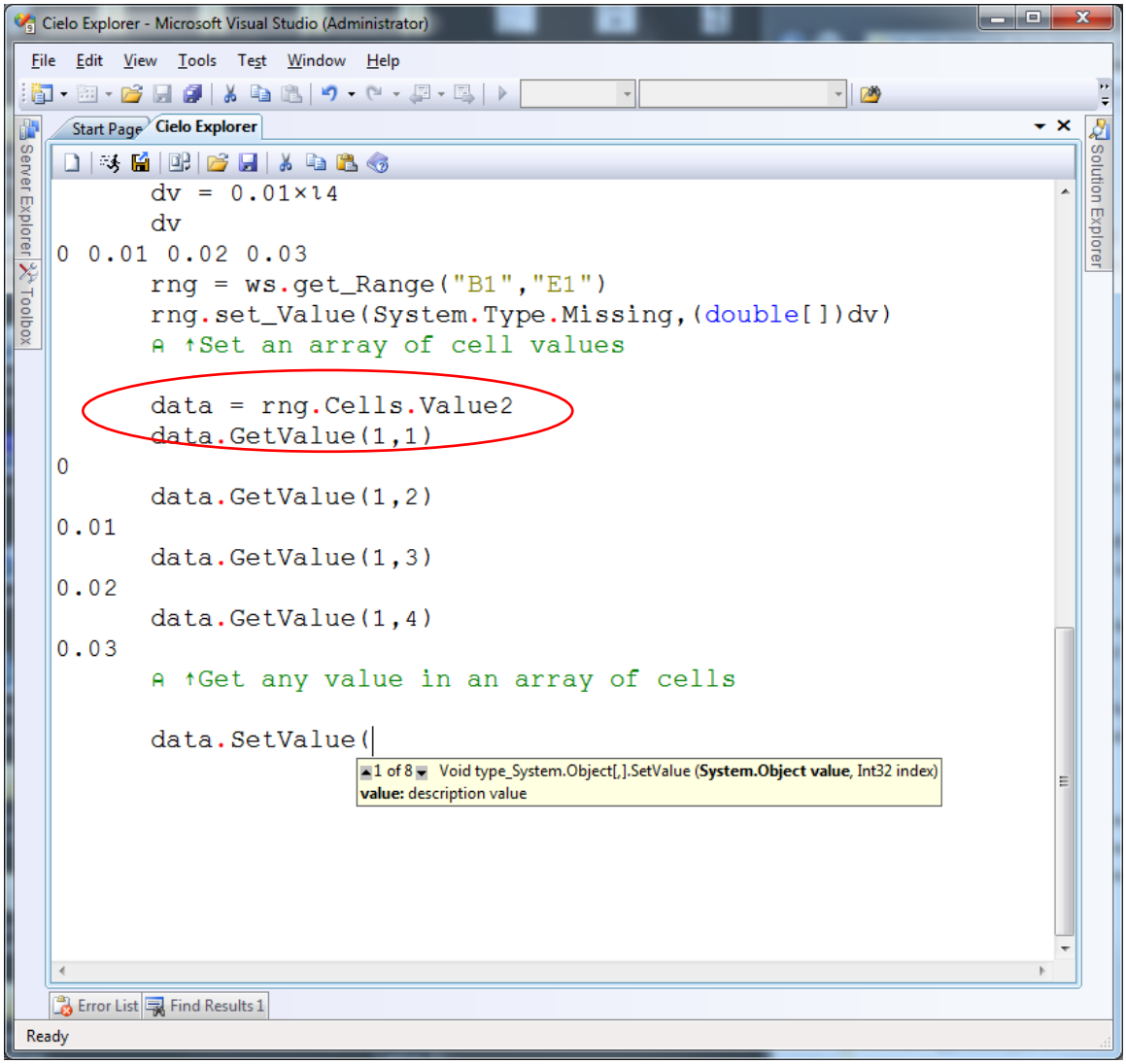
Excel ranges may reference individual cells or arrays of cells supporting some array-based operations.



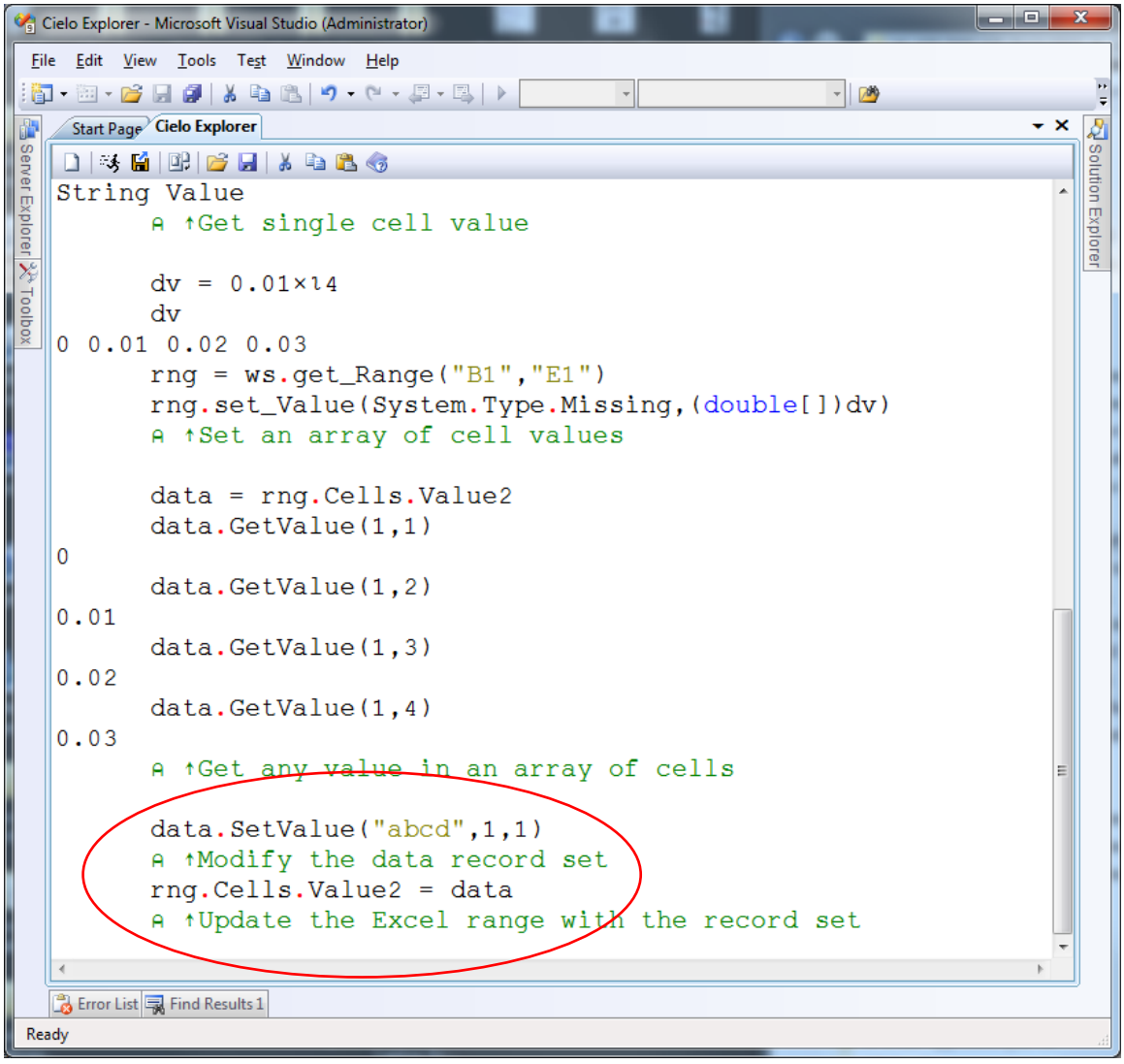
Here is the Excel application after the above Excel automation operations have been completed.



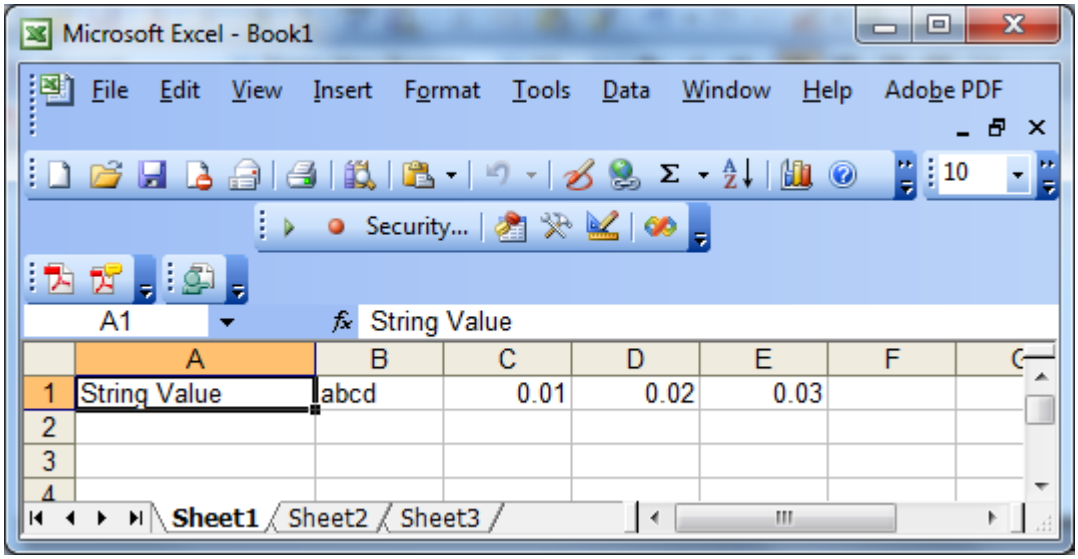
It is possible to abstract the data in an Excel range into a ‘data’ object.



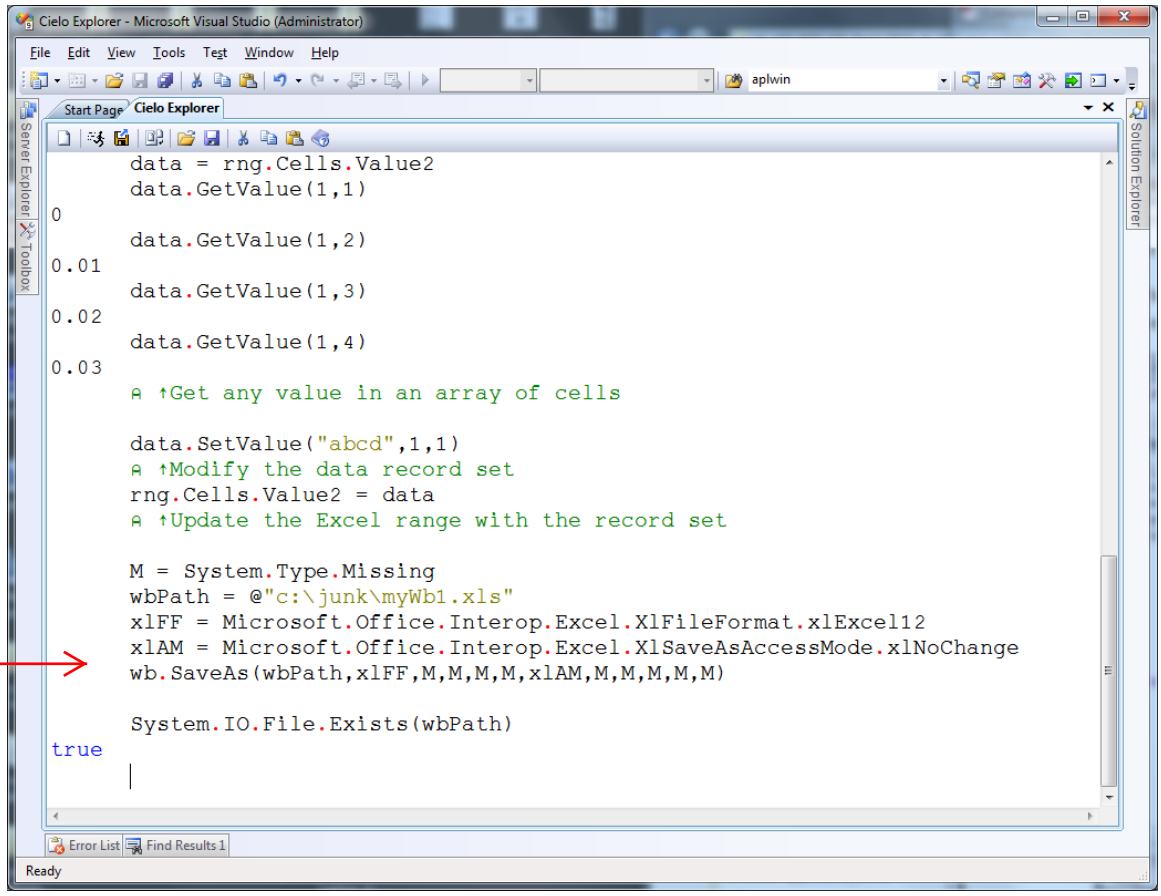
The data object associated with an Excel range object can be manipulated by getting or setting values within it. The data object can then be used to update the Excel range object.



Here is the Excel application after the data object updated the range object.



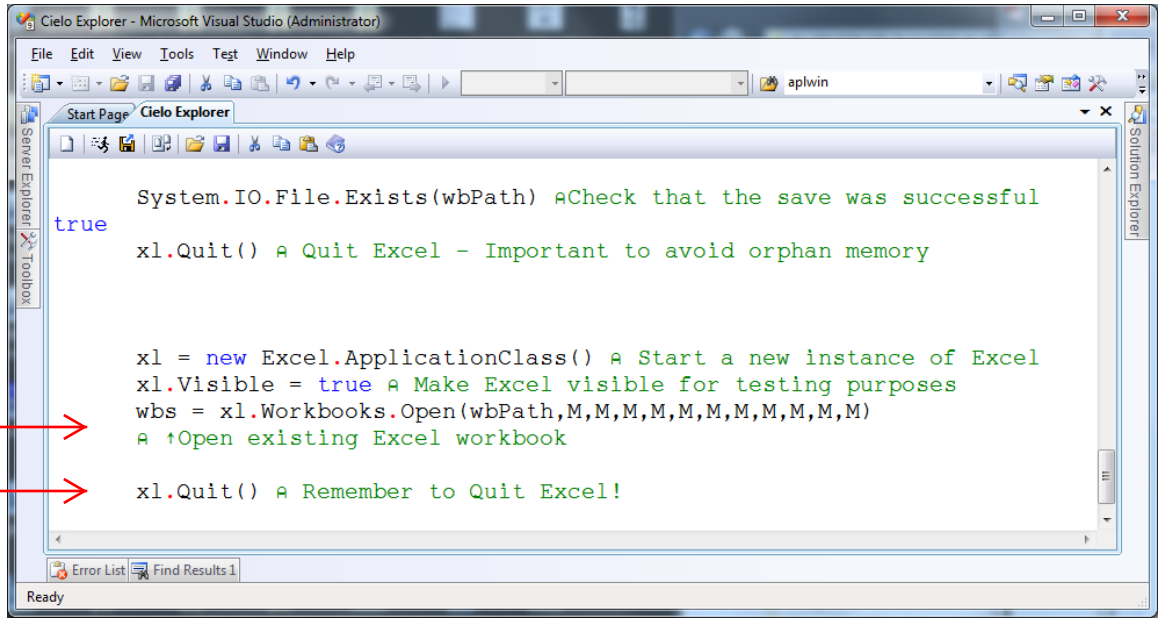
The Excel Workbook.SaveAs() method has a complex argument structure. Generally only certain of these arguments need to be explicitly specified.



When Excel is being automated by a .Net programming language, it is best to check for the successful operation of automation statements, for example using the System.IO.File.Exists() method after the Excel SaveAs() method is used.

Failure to properly quit the Excel application can result in orphan, sometimes invisible, Excel instances which consume memory until the workstation is restarted or these instances are manually ended using the Windows 'Task Manager'.

The Excel Workbooks.Open() method also has a rather complex argument structure.



The Excel object model provides alternate syntax for many statements. Here is an example of saving the current workbook using indexing on the Workbooks collection.