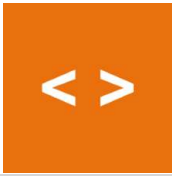


[Explore](#)[Getting Started](#)[Code Samples](#)[Resources](#)[Documentation](#)[Office Developer Center](#) > [Samples](#) > C# app automates Word (CSAutomateWord)[Download Visual Studio](#)

Quick access

[My samples](#)[Upload a sample](#)[Browse sample requests](#)



2,484,830
Points
Top 0.1%

OneCode Team
Joined Dec 2010

[View samples](#)
[Show activity](#)

More from OneCode Team

[How to convert excel file to xml format](#)
(18)[Import/export Excel worksheet in ASP.NET \(CSASPNETExcelImportExport\)](#)
(22)[C++ app automates Excel \(CppAutomateExcel\)](#)
(8)[VB app automates Excel \(VBAutomateExcel\)](#)
(4)[How to Export DataGridView Data to Excel by using Excel Object](#)
(10)[See all](#)

C# app automates Word (CSAutomateWord)

The CSAutomateWord example demonstrates how to write VC# codes to create and automate a Microsoft Word instance


Download

C# (314.4 KB)

Ratings (0)

Updated 3/2/2012

Downloaded 16,295 times

License [MS-LPL](#)Favorites [Add to favorites](#)Share     Requires [Visual Studio 2010](#)Technologies [Office](#)Topics [Automation, Word](#)[Report abuse to Microsoft](#)[Description](#)[Browse code](#)

CONSOLE APPLICATION (CSAutomateWord)

Introduction

The CSAutomateWord example demonstrates the use of C# codes to create a Microsoft Word instance, create a new document, insert a paragraph and a table, save the document, close the Microsoft Word application and then clean up unmanaged COM resources.

Office automation is based on Component Object Model (COM). When you call a COM object of Office from managed code, a Runtime Callable Wrapper (RCW) is automatically created. The RCW marshals calls between the .NET application and the COM object. The RCW keeps a reference count on the COM object. If all references have not been released on the RCW, the COM object of Office

does not quit and may cause the Office application not to quit after your automation. In order to make sure that the Office application quits cleanly, the sample demonstrates two solutions.

Solution1.AutomateWord demonstrates automating Microsoft Word application by using Microsoft Word Primary Interop Assembly (PIA) and explicitly assigning each COM accessor object to a new variable that you would explicitly call Marshal.FinalReleaseComObject to release it at the end.

Solution2.AutomateWord demonstrates automating Microsoft Word application by using Microsoft Word PIA and forcing a garbage collection as soon as the automation function is off the stack (at which point the RCW objects are no longer rooted) to clean up RCWs and release COM objects.

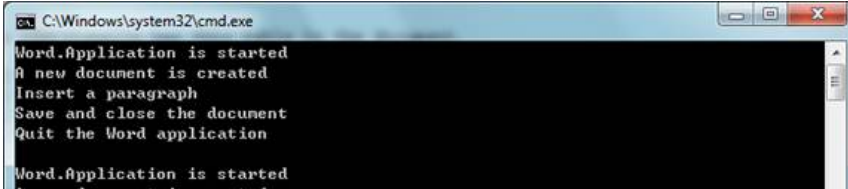
Running the Sample

The following steps walk through a demonstration of the Word automation sample that starts a Microsoft Word instance, creates a new document, inserts a paragraph and a table, saves the document, and quits the Microsoft Word application cleanly.

Step1. After you successfully build the sample project in Visual Studio 2010, you will get the application: CSAutomateWord.exe.

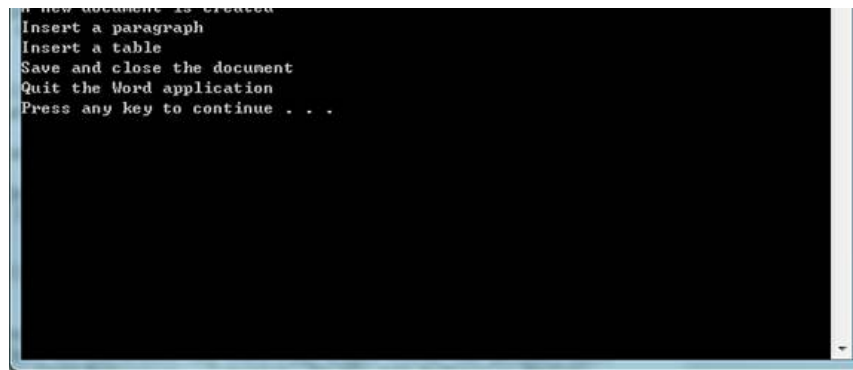
Step2. Open Windows Task Manager (Ctrl+Shift+Esc) to confirm that no winword.exe is running.

Step3. Run the application. It should print the following content in the console window if no error is thrown.



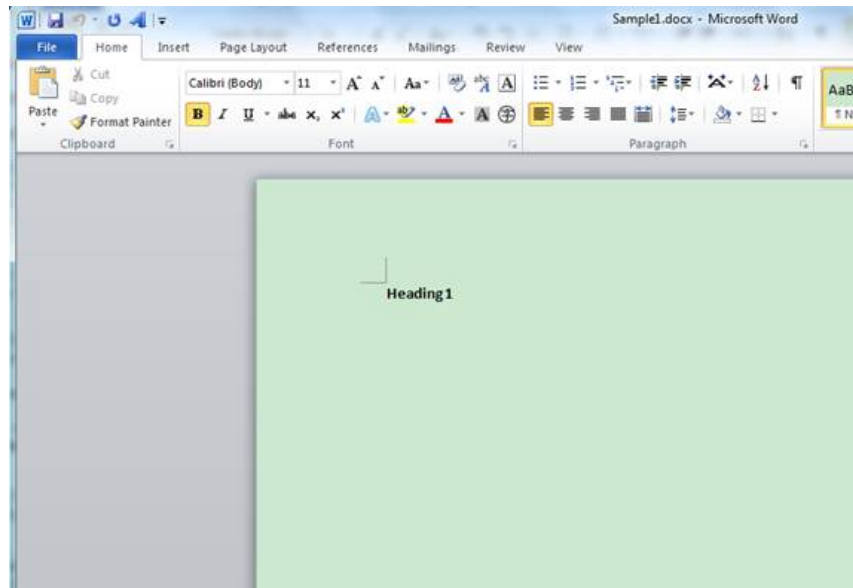
```
C:\Windows\system32\cmd.exe
Word.Application is started
A new document is created
Insert a paragraph
Save and close the document
Quit the Word application

Word.Application is started
A new document is created
```

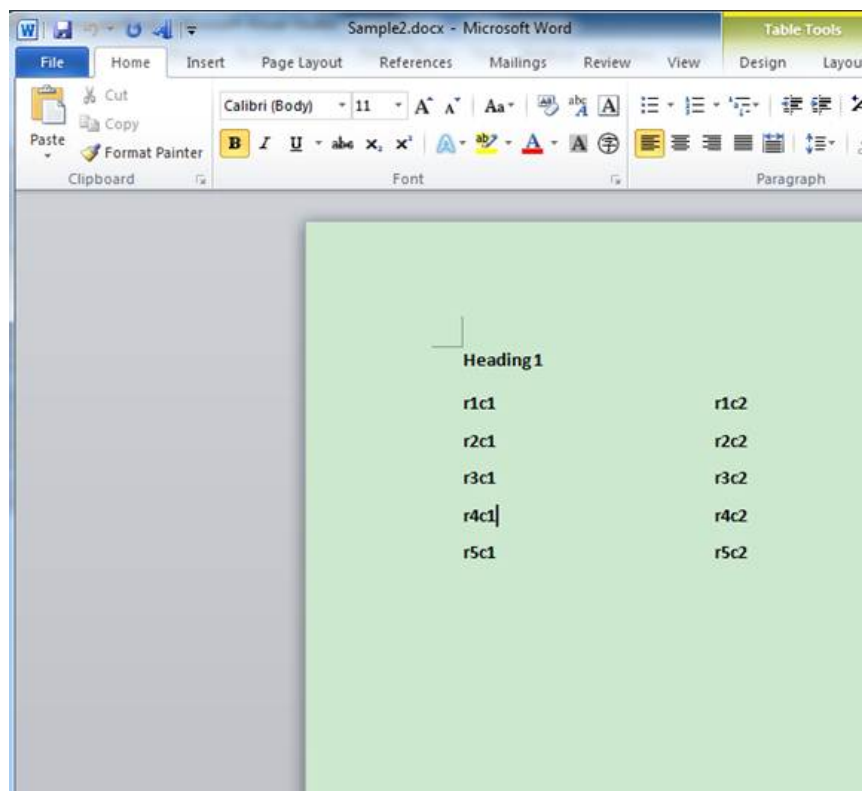


Then, you will see two new documents in the directory of the application:

Sample1.docx and Sample2.docx. Both documents have the following content.



Sample2.docx additionally has this table in the document.



Step4. In Windows Task Manager, confirm that the winword.exe process does not exist, i.e. the Microsoft Word instance was

Using the code

Step1. Create a Console application and reference the Word Primary Interop Assembly (PIA). To reference the Word PIA, right-click the project file and click the "Add Reference..." button. In the Add Reference dialog, navigate to the .NET tab, find [Microsoft.Office.Interop.Word 12.0.0.0](#) and click OK.

Step2. Import and rename the Excel interop namespace:

```
C#
using Word = Microsoft.Office.Interop.Word;
```

Step3. Start up a Word application by creating a Word.Application object.

```
C#
oWord = new Word.Application();
```

Step4. Get the Documents collection from Application.Documents and call its Add function to create a new document. The Add function returns a Document object.

```
C#
// Create a new Document and add it to document collection.
oDoc = oWord.Documents.Add(ref missing, ref missing, ref missing, ref missing);
Console.WriteLine("A new document is created");
```

Step5. Insert a paragraph.

```
C#
oParas = oDoc.Paragraphs;
oPara = oParas.Add(ref missing);
oParaRng = oPara.Range;
oParaRng.Text = "Heading 1";
oFont = oParaRng.Font;
oFont.Bold = 1;
oParaRng.InsertParagraphAfter();
```

Step6. Insert a table.

The following code has the problem that it invokes accessors which will also create RCWs and reference them. For example, calling Document.Bookmarks.Item creates an RCW for the Bookmarks object. If you invoke these accessors via tunneling as this code does, the RCWs are created on the GC heap, but the references are created under the hood on the stack and are then discarded. As such, there is no way to call Marshal.FinalReleaseComObject on those RCWs. To get them to release, you would either need to force a garbage collection as soon as the calling function is off the stack (at which point these objects are no longer rooted) and then call GC.WaitForPendingFinalizers, or you would need to change the syntax to explicitly assign these accessor objects to a variable that you would then explicitly call Marshal.FinalReleaseComObject on.

```
C#
oBookmarkRng = oDoc.Bookmarks.get_Item(ref oEndOfDoc).Range;
oTable = oDoc.Tables.Add(oBookmarkRng, 5, 2, ref missing, ref missing);
oTable.Range.ParagraphFormat.SpaceAfter = 6;
for (int r = 1; r <= 5; r++)
{
    for (int c = 1; c <= 2; c++)
    {
        oTable.Cell(r, c).Range.Text = "r" + r + "c" + c;
    }
}

// Change width of columns 1 & 2
```

```
oTable.Columns[1].Width = oWord.InchesToPoints(2);
oTable.Columns[2].Width = oWord.InchesToPoints(3);
```

Step7. Save the document as a docx file and close it.

C#

```
object fileName = Path.GetDirectoryName(
    Assembly.GetExecutingAssembly().Location) + "\\Sample1.docx";
object fileFormat = Word.WdSaveFormat.wdFormatXMLDocument;
oDoc.SaveAs(ref fileName, ref fileFormat, ref missing,
    ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing, ref missing,
    ref missing);
((Word._Document)oDoc).Close(ref missing, ref missing, ref missing);
```

Step8. Quit the Word application.

C#

```
((Word._Application)oWord).Quit(ref notTrue, ref missing, ref missing);
```

Step9. Clean up the unmanaged COM resource. To get Word terminated rightly, we need to call Marshal.FinalReleaseComObject() on each COM object we used. We can either explicitly call Marshal.FinalReleaseComObject on all accessor objects:

and/or force a garbage collection as soon as the calling function is off the stack (at which point these objects are no longer rooted) and then call GC.WaitForPendingFinalizers.

C#

```
// See Solution2.AutomateWord
GC.Collect();
GC.WaitForPendingFinalizers();
// GC needs to be called twice in order to get the Finalizers called
// - the first time in, it simply makes a list of what is to be
// finalized, the second time in, it actually is finalizing. Only
// then will the object do its automatic ReleaseComObject.
GC.Collect();
GC.WaitForPendingFinalizers();
```

More Information

[MSDN: Word 2007 Developer Reference](#)

[How to automate Microsoft Word to create a new document by using Visual C#](#)

[The Designer Process That Would Not Terminate \(Part 2\)](#)

Microsoft®
All-In-One Code Framework

Community Resources

O365 Technical Network

MSDN Forums

UserVoice

Follow Us

Twitter

Facebook

Office Dev Blog

