```
public partial class ThisAddIn
{
    private string pptxFile;
    private PowerPoint.Application ppt;
    private PowerPoint.Presentation pptx;

    private void ThisAddIn_Startup(object sender, System.EventArgs e)
    {
        Uri codeBaseUri =
            new Uri(Assembly.GetExecutingAssembly().CodeBase);
        pptxFile = Path.Combine(
            Path.GetDirectoryName(codeBaseUri.AbsolutePath), "Test.pptx");
    }

    private void ThisAddIn_Shutdown(object sender, System.EventArgs e)
    {
        // Make sure to release any references to COM objects.
        ppt = null;
        pptx = null;
        GC.Collect();
        GC.WaitForPendingFinalizers();
        GC.Collect();
        GC.WaitForPendingFinalizers();
    }

    // Use the PIAs to interop with PowerPoint.
    internal void InteropCreate()
    {
        ppt = null;
        pptx = null;

        ppt = new PowerPoint.Application();
        ppt.Visible = Office.MsoTriState.msoTrue;
        pptx = ppt.Presentations.Open(
            pptxFile,
            Office.MsoTriState.msoFalse,
            Office.MsoTriState.msoFalse,
            Office.MsoTriState.msoTrue);
    }

    // Use Process.Start to start a PowerPoint process. This does not
    // give you access to the target app's OM.
    internal void ProcessStart()
    {
        ProcessStartInfo si = new ProcessStartInfo();
        si.FileName =
          @"C:\Program Files (x86)\Microsoft Office\Office12\Powerpnt.exe";
        si.Arguments = pptxFile;
        Process.Start(si);
    }

    // Internally, Activator.CreateInstance uses reflection to find and
    // execute an appropriate constructor for the specified object.
    internal void CreateInstance()
    {
        Type t = Type.GetTypeFromProgID("PowerPoint.Application");
        object o = Activator.CreateInstance(t);
        // Note: we could cast the return from Activator.CreateInstance
        // to the PIA type that we expect, if we wanted to use the PIAs.

        t.InvokeMember(
            "Visible", BindingFlags.Public | BindingFlags.SetProperty,
            null, o, new object[] { true }, null);
        object p = t.InvokeMember(
            "Presentations",
            BindingFlags.Public | BindingFlags.GetProperty,
            null, o, null, null);
        Type t2 = p.GetType();
        t2.InvokeMember("Open",
            BindingFlags.Public | BindingFlags.InvokeMethod,
            null, p, new object[] { pptxFile }, null);
    }

```

```
74        // Internally, Marshal.BindToMoniker p/invokes to Win32 BindToMoniker.
75        internal void BindToMoniker()
76        {
77            ppt = null;
78            pptx = null;
79
80            pptx = (PowerPoint.Presentation)Marshal.BindToMoniker(pptxFile);
81            pptx.Application.Visible = Office.MsoTriState.msoTrue;
82        }
83
84        // Internally, Marshal.GetActiveObject p/invokes to Win32
85        // GetActiveObject. This will throw an exception if the object's
86        // server is not already running. Win32 GetActiveObject looks up the
87        // target ProgID in the Running Object Table.
88        internal void GetActiveObject()
89        {
90            ppt = null;
91            pptx = null;
92
93            try
94            {
95                ppt = (PowerPoint.Application)
96                    Marshal.GetActiveObject("PowerPoint.Application");
97                ppt.Visible = Office.MsoTriState.msoTrue;
98                pptx = ppt.Presentations.Open(
99                    pptxFile,
100                   Office.MsoTriState.msoFalse,
101                   Office.MsoTriState.msoFalse,
102                   Office.MsoTriState.msoTrue);
103           }
104           catch (COMException cex)
105           {
106               // If the target app is not already running, GetActiveObject
107               // will throw a COMException (0x800401E3): Operation
108               // unavailable (Exception from HRESULT: 0x800401E3
109               // (MK_E_UNAVAILABLE)).
110               Debug.WriteLine(cex.ToString());
111           }
112       }
113
114       // Internally, CreateObject calls Activator.CreateInstance - that is,
115       // it creates a new instance of the target application (if the app is
116       // single-use).
117       internal void CreateObject()
118       {
119           ppt = null;
120           pptx = null;
121
122           // GetObject requires the object class name, plus optionally a
123           // machine server name.
124           ppt = (PowerPoint.Application)
125               Microsoft.VisualBasic.Interaction.CreateObject(
126               "PowerPoint.Application", "");
127           ppt.Visible = Office.MsoTriState.msoTrue;
128           pptx = ppt.Presentations.Open(
129               pptxFile,
130               Office.MsoTriState.msoFalse,
131               Office.MsoTriState.msoFalse,
132               Office.MsoTriState.msoTrue);
133       }
134
135       // Internally, GetObject calls either Marshal.BindToMoniker or
136       // Activator.CreateInstance - that is, it either uses an existing
137       // instance of the target app, or creates a new instance.
138       internal void GetObject()
139       {
140           ppt = null;
141           pptx = null;
142
143           // GetObject requires either the executable filename or the object
144           // class name.
145           ppt = (PowerPoint.Application)
146               Microsoft.VisualBasic.Interaction.GetObject(
```

```
            "", "PowerPoint.Application");
        ppt.Visible = Office.MsoTriState.msoTrue;
        pptx = ppt.Presentations.Open(
            pptxFile,
            Office.MsoTriState.msoFalse,
            Office.MsoTriState.msoFalse,
            Office.MsoTriState.msoTrue);
    }

    // ActivateMicrosoftApp activates a Microsoft application if it is
    // running  or starts a new instance of it if it is not. Restricted
    // to these apps: Access, FoxPro, Outlook, PowerPoint, Project, Word.
    // Note that this  does not give you access to the target app's OM.
    internal void ActivateMicrosoftApp()
    {
        this.Application.ActivateMicrosoftApp(
            Excel.XlMSApplication.xlMicrosoftPowerPoint);
    }


    [DllImport("User32")]
    public static extern int GetClassName(
        int hWnd, StringBuilder lpClassName, int nMaxCount);

    // Callback passed to EnumChildWindows to find any window with the
    // registered classname "paneClassDC" - this is the class name of
    // PowerPoint's accessible document window.
    public bool EnumChildProc(int hwnd, ref int lParam)
    {
        StringBuilder windowClass = new StringBuilder(128);
        GetClassName(hwnd, windowClass, 128);
        if (windowClass.ToString() == "paneClassDC")
        {
            lParam = hwnd;
        }
        return true;
    }

    public delegate bool EnumChildCallback(int hwnd, ref int lParam);

    [DllImport("User32")]
    public static extern bool EnumChildWindows(
        int hWndParent, EnumChildCallback lpEnumFunc, ref int lParam);

    [DllImport("User32")]
    public static extern int FindWindowEx(
        int hwndParent, int hwndChildAfter, string lpszClass,
        int missing);

    // AccessibleObjectFromWindow gets the IDispatch pointer of an object
    // that supports IAccessible, which allows us to get to the native OM.
    [DllImport("Oleacc.dll")]
    private static extern int AccessibleObjectFromWindow(
        int hwnd, uint dwObjectID,
        byte[] riid,
        ref PowerPoint.DocumentWindow ptr);

    // Get the window handle for a running instance of PowerPoint.
    internal void GetAccessibleObject()
    {
        ppt = null;
        pptx = null;

        try
        {
            // Walk the children of the desktop to find PowerPoint's main
            // window.
            int hwnd = FindWindowEx(0, 0, "PP12FrameClass", 0);
            if (hwnd != 0)
            {
                // Walk the children of this window to see if any are
                // IAccessible.
                int hWndChild = 0;
```

```
                    EnumChildCallback cb =
                        new EnumChildCallback(EnumChildProc);
                    EnumChildWindows(hwnd, cb, ref hWndChild);

                    if (hWndChild != 0)
                    {
                        // OBJID_NATIVEOM gets us a pointer to the native
                        // object model.
                        uint OBJID_NATIVEOM = 0xFFFFFFF0;
                        Guid IID_IDispatch =
                            new Guid("{00020400-0000-0000-C000-000000000046}");
                        PowerPoint.DocumentWindow ptr = null;
                        int hr = AccessibleObjectFromWindow(
                            hWndChild, OBJID_NATIVEOM,
                            IID_IDispatch.ToByteArray(), ref ptr);
                        if (hr >= 0)
                        {
                            ppt = ptr.Application;
                            ppt.Visible = Office.MsoTriState.msoTrue;
                            pptx = ppt.Presentations.Open(
                                pptxFile,
                                Office.MsoTriState.msoFalse,
                                Office.MsoTriState.msoFalse,
                                Office.MsoTriState.msoTrue);
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex.ToString());
            }
        }
    }

```