COM Add-Ins In Excel 2007

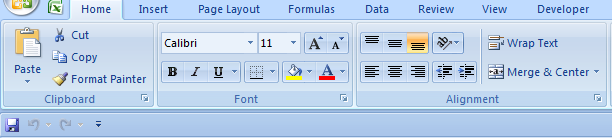
This page describes some changes for COM Add-Ins in Excel 2007.  
ShortFadeBar

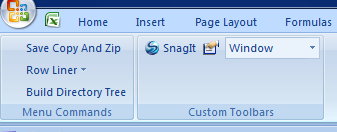
Introduction

Excel does a good job regarding backwards compatibility. A COM Add-In (CAI) that was created for Excel 2000, Excel XP (2002), or Excel 2003 should work in Excel 2007 without modifications. However, there are a few things to keep in mind when you are developing CAIs that will be used in different versions of Excel.

SectionBreak

Menus And Command Bars

With version 2007, Excel eliminated the traditional Menu and CommandBar control structure, replacing it with an interface known as the Ribbon. Depending on your "menu" selection (but it is not really a real menu), different groups of Panels will be made visible. (A Panel is a group of buttons with related functionality. In the image below, the Home "menu" is selected and three panels are shown: Clipboard, Font, and Alignment. Back in the development phase, panels were called chunks. I don't know if there is now an official name for these objects.) If you select the Home "menu", you will see one set of panels. If you click the Insert "menu", another group of panels will be made visible. A snapshot of part of the Ribbon is shown below:  


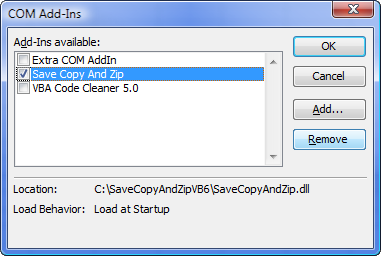
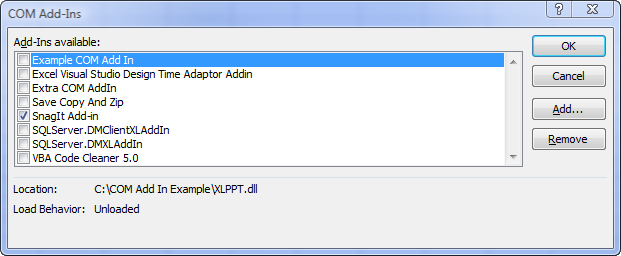
What does this mean with respect to CAI development? It means that any code in the CAI that creates commandbars, commandbar controls, or menu items will no longer work as expected. The Ribbon is customizable only via Excel files in the Excel 2007 file format (xlsx or xlsm) which can contain the necessary XML code used to modify the Ribbon. (A 2007 xlsx or xlsm Excel file is really a Zip file containing 20 or so text XML data files. One of these XML files is used to modify the Ribbon.) Neither a 2003 format (xls) file nor an XLA add-in, nor a COM Add-In can modify the Ribbon. (See Ron de Bruin's [Ribbon Customization](http://www.rondebruin.nl/ribbon.htm) page for more information.) All customizations, whether in 2003 they are new menu items, controls on existing commandbars, or new commandbars, are displayed in the same Menu Commands or Custom Toolbars panels of the Add-Ins Ribbon view. This mixes your command elements in with the command elements of other add-ins. A snapshot of the Add-Ins Ribbon view is shown below:  
  
In Excel 2003 and earlier, you would likely put some menu items on the Tools menu, others on the View menu, and perhaps still others on the Data menu. Menu items and commandbar buttons could be placed in the most logical location. In Excel 2007, all customization is put in the Add-Ins view on either the Menu Commands or Custom Toolbars panel. This lack of customization is a sore point with Excel 2007.

SectionBreak

COM Add-In Visibility

Linkage information about a COM Add-In, indicating to Excel that the CAI is available, is stored in one of two areas of the system registry:  
  
HKEY\_CURRENT\_USER\Software\Microsoft\Office\Excel\AddIns (HKCU) or   
HKEY\_LOCAL\_MACHINE\Software\Microsoft\Office\Excel\AddIns (HKLM).   
  
In Excel 2003 and earlier, only CAIs that are configured in HKCU are displayed in the COM Add-Ins dialog. This prevents one user from disabling CAIs that are intended for all users of the machine. In Excel 2003 and earlier, a user can load or unload only CAIs that are configured in HKCU. A user cannot load or unload a CAI that is configured in HKLM.

In Excel 2007, all add-ins, whether configured in HKEY\_CURRENT\_USER\Software\Microsoft\Office\Excel\AddIns or HKEY\_LOCAL\_MACHINE\Software\Microsoft\Office\Excel\AddIns are visible in the COM Add-Ins dialog. This means that individual users can disable or unload CAIs that are intended for all users of a machine.

The image below shows the COM Add-Ins dialog in Excel 2003 running on Windows Vista Ultimate.  
  
The following image shows the COM Add-Ins dialog in Excel 2007 on the same machine with the same user.  
  
As you can see, the dialog in 2007 shows more CAIs that does the 2003 dialog. This is because 2007 displays CAIs that are configured in both HKCU and HKLM while Excel 2003 displays only CAIs configured in HKCU only.

SectionBreak

Additonal Resources For COM Add-Ins

[Adding The COM Add-Ins Menu Item](http://www.cpearson.com/excel/AddingCOMAddInsMenuItem.aspx)  
[Automation Add-Ins](http://www.cpearson.com/excel/AutomationAddIns.aspx)  
[Distribution And Security Issues Of COM Add-Ins](http://www.cpearson.com/excel/COMAddInsSecurity.aspx)  
[COM Add-Ins In Excel 2007](http://www.cpearson.com/excel/COMAddIn2007.aspx)  
[COM Add-Ins, Getting The DLL Name Of](http://www.cpearson.com/excel/DLLNameOfComAddin.htm)  
[COM Add-Ins, Installer (download zip file)](http://www.cpearson.com/Zips/ComAddInInstaller.zip)

This page last updated: 23-August-2007

Adding The COM Add-Ins Menu Item

This page describes how to add the COM Add-Ins menu item to the Excel Tools menu.  
ShortFadeBar

If you are using COM Add-Ins (CAIs), either ones you wrote yourself ([described here](http://www.cpearson.com/excel/ComAddInInVB6.htm)) or ones written by others, you will want to put the COM Add-Ins menu item on your Tools menu. The default command bar settings in Excel do not include the CAI dialog, so you need to add it yourself. This pages describes how to add the menu item.

|  |  |
| --- | --- |
| Customize Dialog | **Adding The COM Add-Ins Menu Item In Excel 2000, 2002 (XP) and 2003**  Go to the View menu, choose Toolbars, and at the end of the list choose Customize. This will display the Customize dialog shown at the left. Select the Commands tab and then choose the Tools item in the Categories list. Scroll down in the Commands list to COM Add Ins.... With the Customize dialog still open, drag the COM Add-Ins item to the Tools menu. The logical place is to put it just below the existing Add-Ins... item.  Now, close the Customize dialog. Now, you can click the Com Add-ins... menu item to display the dialog, shown below. From this dialog, you can load, unload, add, or delete COM Add-ins.  COM AddIns Dialog  Note that the COM Addins dialog will display only CAIs listed in the HKEY\_CURRENT\_USER region of the System Registry. It will not display CAIs that are listed in the HKEY\_LOCAL\_MACHINE region of the System Registry. This is done to prevent a single users from modifying the environment that affects all users of machine. |

SectionBreak

Additonal Resources For COM Add-Ins

[Adding The COM Add-Ins Menu Item](http://www.cpearson.com/excel/AddingCOMAddInsMenuItem.aspx)  
[Automation Add-Ins](http://www.cpearson.com/excel/AutomationAddIns.aspx)  
[Distribution And Security Issues Of COM Add-Ins](http://www.cpearson.com/excel/COMAddInsSecurity.aspx)  
[COM Add-Ins In Excel 2007](http://www.cpearson.com/excel/COMAddIn2007.aspx)  
[COM Add-Ins, Getting The DLL Name Of](http://www.cpearson.com/excel/DLLNameOfComAddin.htm)  
[COM Add-Ins, Installer (download zip file)](http://www.cpearson.com/Zips/ComAddInInstaller.zip)

This page last modified on 6-July-2007.

Getting the DLL Name From A COM Add-In

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | Neither Excel nor VBA provides you with a method to get the DLL name of a COM Add-In. This page contains a procedure  named *DLLOfComAddin* that will return the fully-qualified file name of the DLL of the COM Add-In you pass in as a parameter.   Since this code deals with COM Add-Ins, it will not work in Office97. This code can be used in any application that supports VBA6 (Office 2000 and later) and COM Add-Ins. There is nothing specific to Excel in the code.  The procedure gets the GUID property of the passed-in COM Add-In reference and looks up that value in the appropriate key of the  Registry and returns the value of the *InprocServer32* item.  The complete VBA module, including required constants and  Windows API declarations, is below.  This code uses the functions [TrimToNull](http://www.cpearson.com/excel/SizeString.htm#TrimToNull) [located here](http://www.cpearson.com/excel/SizeString.htm#SizeString) and  [GetSystemErrorMessageText located here](http://www.cpearson.com/excel/FormatMessage.htm).  To get the DLL name of the COM Add-In, use code like the following:  Sub AAATest()     Dim CAI As Office.COMAddIn     Dim DLLName As String     Set CAI = Application.COMAddIns(1)     DLLName = DLLOfComAddin(A\_7\_AB\_1\_ComAddIn:=CAI)     If DLLName <> vbNullString Then          MsgBox "Addin Information:" & vbCrLf & \_                "ProgID:   " & CAI.ProgID & vbCrLf & \_                "GUID:     " & CAI.GUID & vbCrLf & \_                "DLL Name: " & DLLName     End If End Sub  The Code:  All of the following code that follows should be pasted in to a new, empty,  VBA code module. You can download an example [Excel Workbook here](http://www.cpearson.com/Zips/DLLNameOfCOMAddin.zip) or just the [code bas module here](http://www.cpearson.com/Zips/modDLLOfComAddin.zip).  Option Explicit Option Compare Text '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' ' modDLLNameOfComAddin ' By Chip Pearson, chip@cpearson.com, www.cpearson.com. ' ' This module contains the DLLOfComAddin function. This function takes as its input ' parameter a reference to an existing COM AddIn and returns a string containing ' the fully-qualified DLL file name of that COM Add-In.  ' The COM Addin need not be connected. ' ' This module is entirely self-contained. It requires no additional support code. ' This code may be in any Office application that supports VBA6 (Office 2000 and later). ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''  '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' ' Constants ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''  ''''''''''''''''''''''''''''' ' Misc constants. ''''''''''''''''''''''''''''' Private Const C\_COM\_ADDIN\_CLSID\_REG\_LOCATION = "SOFTWARE\Classes\CLSID\" Private Const C\_COM\_ADDIN\_CLSID\_REG\_VALUE\_NAME = "InprocServer32" Private Const C\_PATH\_SEPARATOR = "\" Private Const ERROR\_SUCCESS = As Long 0 Private Const MAX\_PATH = As Long 260 ' Windows mandated value.  ''''''''''''''''''''''''''''' ' Registry Sections ''''''''''''''''''''''''''''' Private Const HKEY\_CLASSES\_ROOT As Long = &H80000000 Private Const HKEY\_CURRENT\_USER As Long = &H80000001 Private Const HKEY\_LOCAL\_MACHINE As Long = &H80000002 Private Const HKEY\_USERS As Long = &H80000003  ''''''''''''''''''''''''''''' ' used by FormatMessage ''''''''''''''''''''''''''''' Private Const FORMAT\_MESSAGE\_ALLOCATE\_BUFFER As Long = &H100 Private Const FORMAT\_MESSAGE\_ARGUMENT\_ARRAY As Long = &H2000 Private Const FORMAT\_MESSAGE\_FROM\_HMODULE As Long = &H800 Private Const FORMAT\_MESSAGE\_FROM\_STRING As Long = &H400 Private Const FORMAT\_MESSAGE\_FROM\_SYSTEM As Long = &H1000 Private Const FORMAT\_MESSAGE\_IGNORE\_INSERTS As Long = &H200 Private Const FORMAT\_MESSAGE\_MAX\_WIDTH\_MASK As Long = &HFF Private Const FORMAT\_MESSAGE\_TEXT\_LEN = As Long 160 ' from ERRORS.H C++ include file.   ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' ' Windows API Declares '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''  ' RegOpenKey opens an existing registry key, named in lpSubKey. It populates ' phkResult with a key value that is used by the other registry functions. Private Declare Function RegOpenKey Lib "advapi32.dll" Alias "RegOpenKeyA" ( \_     ByVal hKey As Long, \_     ByVal lpSubKey As String, \_     ByRef phkResult As Long) As Long  ' RegCloseKey closes a registry key previously opened with RegOpenKey. Private Declare Function RegCloseKey Lib "advapi32.dll" ( \_     ByVal hKey As Long) As Long  ' RegQueryValue reads the value of a registry item, opened with RegOpenKey. Private Declare Function RegQueryValue Lib "advapi32.dll" Alias "RegQueryValueA" ( \_     ByVal hKey As Long, \_     ByVal lpSubKey As String, \_     ByVal lpValue As String, \_     ByRef lpcbValue As Long) As Long  ' FormatMessage gets the descriptive error messages associated with the error ' number in dwMessageId. Private Declare Function FormatMessage Lib "kernel32" Alias "FormatMessageA" ( \_     ByVal dwFlags As Long, \_     ByRef lpSource As Any, \_     ByVal dwMessageId As Long, \_     ByVal dwLanguageId As Long, \_     ByVal lpBuffer As String, \_     ByVal nSize As Long, \_     ByRef Arguments As Long) As Long   Public Function DLLOfComAddin(A\_7\_AB\_1\_ComAddIn As Office.COMAddIn) As String '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' ' DLLOfComAddin ' By Chip Pearson, chip@cpearson.com , www.cpearson.com  ' ' This function returns the fully-qualified name of the DLL file for the  ' specified COM Add-In (CAI). A\_7\_AB\_1\_ComAddIn is a reference to an existing  ' CAI. It is not required that the CAI be connected. ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''  Dim V\_7\_AB\_1\_RegistryKeyName As String ' stores the name of the registry key we're working with Dim V\_7\_AB\_1\_RegResult As String       ' stores the name of the COM Add-In's DLL file. Dim V\_7\_AB\_1\_Res As Long               ' general puprose return code variable Dim V\_7\_AB\_1\_RegKey As Long            ' internal registry key value retrieved by RegOpenKey Dim V\_7\_AB\_1\_ErrorNumber As Long       ' stores the error number than may have occurred Dim V\_7\_AB\_1\_ErrorText As String       ' error text returned GetSystemErrorMessageText Dim V\_7\_AB\_1\_RegResultLen As Long      ' length in characters of V\_7\_AB\_1\_RegResult   ''''''''''''''''''''''''''''''''''''''''''''''''' ' Initialize the string we're going to populate with ' the DLL name. The buffer must be long enough ' to store the complete DLL file name. MAX\_PATH ' is a Windows mandated length, the maximum ' length of a fully qualified file name. ''''''''''''''''''''''''''''''''''''''''''''''''' V\_7\_AB\_1\_RegResult = String$(MAX\_PATH, vbNullChar)  ''''''''''''''''''''''''''''''''''''''''''''''''' ' ensure we didn't get NOTHING ''''''''''''''''''''''''''''''''''''''''''''''''' If A\_7\_AB\_1\_ComAddIn Is Nothing Then     MsgBox "The A\_7\_AB\_1\_ComAddIn parameter is NOTHING."     Exit Function End If  ''''''''''''''''''''''''''''''''''''''''''''''''' ' Initialzie the registry key name. It will contain ' a string similar to ' '    SOFTWARE\Classes\CLSID\{F0E54810-A875-4C54-9697-0AE40DAA7316}\InprocServer32 ' ' We will look up this key in the HKEY\_LOCAL\_MACHINE section ' of the registry. ''''''''''''''''''''''''''''''''''''''''''''''''' V\_7\_AB\_1\_RegistryKeyName = C\_COM\_ADDIN\_CLSID\_REG\_LOCATION & \_                            A\_7\_AB\_1\_ComAddIn.GUID & \_                            C\_PATH\_SEPARATOR & \_                            C\_COM\_ADDIN\_CLSID\_REG\_VALUE\_NAME  ''''''''''''''''''''''''''''''''''''''''''''''''' ' Open the registry key V\_7\_AB\_1\_RegistryKeyName. ' RegOpenKey puts in V\_7\_AB\_1\_RegKey a key value ' that is used by all other registry functions ' that access that key. ''''''''''''''''''''''''''''''''''''''''''''''''' V\_7\_AB\_1\_Res = RegOpenKey(hKey:=HKEY\_LOCAL\_MACHINE, \_                           lpSubKey:=V\_7\_AB\_1\_RegistryKeyName, \_                           phkResult:=V\_7\_AB\_1\_RegKey) If V\_7\_AB\_1\_Res <> ERROR\_SUCCESS Then     V\_7\_AB\_1\_ErrorNumber = V\_7\_AB\_1\_Res     V\_7\_AB\_1\_ErrorText = GetSystemErrorMessageText(V\_7\_AB\_1\_ErrorNumber)     MsgBox "Error opening Registry key: '" & V\_7\_AB\_1\_RegistryKeyName & "'" & vbCrLf & \_            "System Error: " & CStr(V\_7\_AB\_1\_ErrorNumber) & \_            " Hex(&H" & Hex(V\_7\_AB\_1\_ErrorNumber) & ")" & vbCrLf & \_            "Description: " & V\_7\_AB\_1\_ErrorText     Exit Function End If  ''''''''''''''''''''''''''''''''''''''''''''''''' ' Get the value from the registry. We set ' lpSubKey:=vbNullString to get the default ' value, which is the DLL file name. '''''''''''''''''''''''''''''''''''''''''''''''''  ' V\_7\_AB\_1\_RegResultLen is the length in characters ' of V\_7\_AB\_1\_RegResult, the variable that will ' receive the DLL file name. V\_7\_AB\_1\_RegResultLen = MAX\_PATH V\_7\_AB\_1\_Res = RegQueryValue(hKey:=V\_7\_AB\_1\_RegKey, \_                               lpSubKey:=vbNullString, \_                              lpValue:=V\_7\_AB\_1\_RegResult, lpcbValue:=V\_7\_AB\_1\_RegResultLen) If V\_7\_AB\_1\_Res <> ERROR\_SUCCESS Then     V\_7\_AB\_1\_ErrorNumber = V\_7\_AB\_1\_Res      V\_7\_AB\_1\_ErrorText = GetSystemErrorMessageText(V\_7\_AB\_1\_ErrorNumber)     MsgBox "Error retrieving Registry key: '" & V\_7\_AB\_1\_RegistryKeyName & "'" & vbCrLf & \_             "System Error: " & CStr(V\_7\_AB\_1\_ErrorNumber) & \_              " Hex(&H" & Hex(V\_7\_AB\_1\_ErrorNumber) & ")" & vbCrLf & \_             "Description: " & V\_7\_AB\_1\_ErrorText     RegCloseKey hKey:=V\_7\_AB\_1\_RegKey     Exit Function End If  ''''''''''''''''''''''''''''''''''''''''''''''''' ' close our registry key ''''''''''''''''''''''''''''''''''''''''''''''''' RegCloseKey V\_7\_AB\_1\_RegKey  ''''''''''''''''''''''''''''''''''''''''''''''''' ' trim V\_7\_AB\_1\_RegResult to the vbNullChar ''''''''''''''''''''''''''''''''''''''''''''''''' V\_7\_AB\_1\_RegResult = TrimToNull(V\_7\_AB\_1\_RegResult)  ''''''''''''''''''''''''''''''''''''''''''''''''' ' return the DLL name ''''''''''''''''''''''''''''''''''''''''''''''''' DLLOfComAddin = V\_7\_AB\_1\_RegResult  End Function  Private Function GetSystemErrorMessageText(ErrorNumber As Long) As String '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' ' GetSystemErrorMessageText ' ' This function gets the system error message text that corresponds to the error  ' code returned by the GetLastError API function or the Err.LastDllError property.  ' It may be used ONLY for these error codes. These are NOT the error ' numbers returned by Err.Number (for these errors, use Err.Description to get the  ' description of the message). The error number MUST be the value returned by  ' GetLastError or Err.LastDLLError. ' ' In general, you should use Err.LastDllError rather than GetLastError because under  ' some circumstances the value of GetLastError will be reset to 0 before the value is  ' returned to VB. Err.LastDllError will always reliably return the last error number  ' raised in a DLL. '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''  Dim V\_7\_AB\_1\_ErrorText As String Dim V\_7\_AB\_1\_TextLen As Long Dim V\_7\_AB\_1\_FormatMessageResult As Long Dim V\_7\_AB\_1\_LangID As Long   V\_7\_AB\_1\_LangID = 0& V\_7\_AB\_1\_ErrorText = String$(FORMAT\_MESSAGE\_TEXT\_LEN, " ") V\_7\_AB\_1\_TextLen = Len(V\_7\_AB\_1\_ErrorText) V\_7\_AB\_1\_FormatMessageResult = 0&  V\_7\_AB\_1\_FormatMessageResult = FormatMessage( \_         dwFlags:=FORMAT\_MESSAGE\_FROM\_SYSTEM Or FORMAT\_MESSAGE\_IGNORE\_INSERTS, \_         lpSource:=0&, \_         dwMessageId:=ErrorNumber, \_         dwLanguageId:=0&, \_         lpBuffer:=V\_7\_AB\_1\_ErrorText, \_         nSize:=Len(V\_7\_AB\_1\_ErrorText), \_         Arguments:=0&)  If V\_7\_AB\_1\_FormatMessageResult > 0 Then     ' FormatMessage returned some text. Take the left V\_7\_AB\_1\_FormatMessageResult      ' characters and return that text.     V\_7\_AB\_1\_ErrorText = Left$(V\_7\_AB\_1\_ErrorText, V\_7\_AB\_1\_FormatMessageResult)     GetSystemErrorMessageText = V\_7\_AB\_1\_ErrorText Else     ' Format message didn't return any text.      ' There is no text description for the specified error.     GetSystemErrorMessageText = "NO ERROR DESCRIPTION AVAILABLE" End If  End Function   Private Function TrimToNull(Text As String) As String ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' ' TrimToNull ' This function returns the portion of Text that is to the left of the vbNullChar ' character (same as Chr(0)). Typically, this function is used with strings ' populated by Windows API procedures. It is generally not used for ' native VB Strings. ' If vbNullChar is not found, the entire Text string is returned. ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''' Dim Pos As Integer Pos = InStr(1, Text, vbNullChar) If Pos > 0 Then     TrimToNull = Left(Text, Pos - 1) Else     TrimToNull = Text End If  End Function  Public Sub AAATestIt() '''''''''''''''''''''''''''''''''''''''''''''' ' Test procedure. This procedure may be ' deleted with no side effects. '''''''''''''''''''''''''''''''''''''''''''''' Dim CAI As Office.COMAddIn Dim DLLName As String  If Application.COMAddIns.Count >= 1 Then     Set CAI = Application.COMAddIns(1)     DLLName = DLLOfComAddin(A\_7\_AB\_1\_ComAddIn:=CAI)     MsgBox "Addin Information:" & vbCrLf & \_            "ProgID: " & CAI.ProgID & vbCrLf & \_            "GUID: " & CAI.GUID & vbCrLf & \_            "DLL Name: " & DLLName & vbCrLf & \_            "Connected: " & CAI.Connect Else     MsgBox "There are no COM Add-Ins. End If  End Sub |  |

Installing COM Add-Ins

This page discusses COM Add-Ins And Installation Policies  
ShortFadeBar

Introduction

Distributing COM Add-Ins to other users on other machines requires some forethought regarding access and security. Information about a COM Add-In (CAI) is stored in System Registry in either of two keys. If the CAI is for the current user only (or configured on a per-user basis for each user of a machine), the information is stored in the HKEY\_CURRENT\_USER (HKCU) region of the registry. CAIs in HKCU are displayed in the *COM Add-Ins* dialog and can be loaded and unloaded by the user. If the CAI is configured for the machine as a whole, its information is stored in the HKEY\_LOCAL\_MACHINE (HKLM) region of the registry. These add-ins can be loaded and unloaded only by their own interface or by direct manipulation of the System Registry. They do not appear in the COM Add-Ins dialog. This restriction is to prevent one user from modifying a setting that affects all users of the machine.

This page assumes that you are familiar with the System Registry and the RegEdit Registry Editor program. Changing System Registry Keys and Values can be dangerous. If you change or delete the wrong key, applications may behave incorrectly or not start. In the worst case, Windows itself may not start. It is strongly recommended that you make a backup copy of the Registry before changing it. Windows itself also keeps backup copies of the last known good Registries, so you can restore to a previous version of the Registry during the Windows startup procedure. Typically, this is done by pressing the F2 or F12 key during the BIOS load screen when you computer first starts.

SectionBreak

Typical Installation Process

Unless you use a dedicated installation program such as InstallShield or Setup Factory, the typical method of installation of a CAI on another machine is to copy the DLL to the target machine, run the RegSvr32 program to register the CAI with Windows, and then add the CAI to Excel via the COM Add-Ins dialog box. On most systems, you must be logged on to an account with adminstrator privileges in order to run the RegSvr32 program. If you are using a dedicated installer like InstallShield or Setup Factory, all of the steps outlined on this page can be automated with the setup program. Consult your program's documentation.

REGSVR32 - The RegSvr32 program is a utility provided by Windows to register ActiveX DLL and OCX component files with Windows. In the case of components written with Visual Basic 6, it calls a procedure that Visual Basic writes for you that creates and updates the appropriate System Registry keys for your component. You run RegSvr32 from the *Run* item on the Windows Start menu. The syntax, in summary, is  
RegSvr32 "C:\YourPath\YourFile.dll"  
To unregister a component and remove its registry keys, use the syntax:  
RegSvr32 "C:\YourPath\YourFile.dll" /u  
The /u switch tells RegSvr32 to unregister the component. Unregistering a component removes it from the System Registry but does not actually delete any files.

SectionBreak

Configuring A COM Add-In Per User Without Administrator Privileges

To configure the CAI for an individual user without administrative privileges, you must directly modify the System Registry. Note that modifying the System Registry can be dangerous. If you delete or modify the wrong Key or Value, applications may behave incorrectly or not start. In the worst case, Windows itself may not start. Take caution when modifying the System Registry. First, log on to the system with an account with administrator privileges and RegSvr32 the DLL file. If you attempt to RegSvr32 the DLL and you don't have administrator privileges, RegSvr32 may fail with an error 0x80004005, which is an "unspecified error", providing no meaningful information. At its core, this error indicates a security violation. Once you have run RegSvr32 with administrative privileges, follow the steps below.

* Log on to the machine as the appropriate user.
* Go to the Windows Start menu, choose *Run* and enter RegEdit to start the Registry Editor program.
* In the Registry Editor, navigate to the key HKEY\_CURRENT\_USER\Software\Microsoft\Office. If necessary, create a Key named Excel under the Office key if one does not already exist.
* Under the Excel Key, create a Key named AddIns if it does not already exist.
* In the AddIns key, create a Key named YourAddIn.ProgID where YourAddIn is the name of the DLL file (without the path and without the '.dll' extension) and ProgID is the name of the class in the add-in that contains the OnConnection and related events, or the class that implements the IDTExtensibility2 interface. This is the name of the Add In Designer object in your add in.
* In this key, create a String Value named FriendlyName and enter a name for your add-in. Then create a String Value named Description and enter a short description of your add in. Finally, create a DWORD value named LoadBehavior and give it a value of 3.
* Close the Registry Editor and open Excel. Since you specified a LoadBehavior of 3, the add in will automatically load when Excel starts. You can load and unload the CAI via the *COM Add-Ins* dialog box. If you do not have a *COM Add-Ins* item on your *Tools* menu, you need to add it. [Click here for instructions](http://www.cpearson.com/excel/AddingCOMAddInsMenuItem.aspx).
* You will need to repeat these steps for each user on the machine.

SectionBreak

Installing The Add In At The Machine Level

Rather than installing the COM Add-In for each user, you can install the CAI for all users of the machine. All users of the machine will have the Add In loaded. However, you will not be able to load or unload the CAI via the *COM Add-Ins* dialog. This restriction is designed to prevent one user from making a change that affects all users of the machine. A well designed CAI will have a user interface element such as a menu item or command bar button to unload the add in.

Note that you must be logged on to an account with administrative privileges in order to install the CAI at the machine level since you are modifying a protected region of the System Registry. If you attempt to modify the HKEY\_LOCAL\_MACHINE region of the registry without being logged on as an administrator, you will likely receive security errors.

First, register the DLL with Windows using the RegSvr32 program.

To configure the CAI for all users of the machine, follow the steps below.

* Go to the Windows Start menu, choose *Run* and enter RegEdit to start the Registry Editor program.
* In the Registry Editor, navigate to the Key HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Office.
* If necessary, create a Key named Excel under the Office Key.
* Under the Excel Key, create a Key named AddIns if it does not exist.
* Under the AddIns Key, create a Key named YourAddIn.ProgID where YourAddIn is the name of the DLL (without any path information and without the '.dll' extension) and ProgID is the name of the class in the DLL that contains the OnConnection and related events or the class that implements the IDTExtensibility2 interface.
* Within this key, create a String Value named FriendlyName and assign a name to the Add In.
* Create a String Value named Description and enter a short description of the Add In.
* Create a DWORD Value named LoadBehavior and give it a value of 3.
* Close the Registry Editor and start Excel. Since you set the LoadBehavior setting to 3, the Add In will load when Excel starts. You will not see this CAI in the *COM Add Ins* dialog because it is configured for the entire machine, not just the individual user. Individual users are not allowed to load or unload COM Add Ins that are configured in HKEY\_LOCAL\_MACHINE to prevent one user from making a change that affects all users of the machine.

SectionBreak

Automation AddIns (Excel 2002 And Later)

When you use RegSvr32 to register your COM Add In with Windows, the componenet is automatically added to the HKEY\_CLASSES\_ROOT region of the registry, and functions defined in classes of that CAI are accessible directly from worksheet cells. To enable functions in an Automation/COM Add In, open the standard *Add Ins* dialog (not the *COM Add-Ins* dialog) and click the Automation button. This will present you with a very long list of available components. Scroll down to your Automation Add In, which will be listed as the name of the DLL file (without any path information and without the '.dll' extension) followed by the class containing the functions. If you have more than one class that expose functions, you will see one entry in the list for each class. Check your item(s) and they will appear in the list of standard Add Ins. Once you have done this, you can use those functions directly in worksheet cells without qualifying their names. Components and functions in the Automation Add Ins are available to all users, regardless of the user that installed the add in.

Automation Add Ins were introduced in Excel 2002 and are not available in earlier versions.

SectionBreak

Additonal Resources For COM Add-Ins

[Adding The COM Add-Ins Menu Item](http://www.cpearson.com/excel/AddingCOMAddInsMenuItem.aspx)  
[Automation Add-Ins](http://www.cpearson.com/excel/AutomationAddIns.aspx)  
[Distribution And Security Issues Of COM Add-Ins](http://www.cpearson.com/excel/COMAddInsSecurity.aspx)  
[COM Add-Ins In Excel 2007](http://www.cpearson.com/excel/COMAddIn2007.aspx)  
[COM Add-Ins, Getting The DLL Name Of](http://www.cpearson.com/excel/DLLNameOfComAddin.htm)  
[COM Add-Ins, Installer (download zip file)](http://www.cpearson.com/Zips/ComAddInInstaller.zip)

Creating A COM Add-In In Visual Basic 6

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
|  | Introduction To Writing COM Add-Ins And Automation Add-Ins In Visual Basic 6  Beginning in Office 2000, Office programs such as Excel or PowerPoint began support for a new type of add-in called a COM Add-In (or CAI). A CAI is completely different from and independent of standard application add-ins such as an XLA addin. There are several advantages to using a COM Add-In rather than an application add-in such as an XLA file.  **Peformace:** A COM Add-In is a special type of DLL file and is compiled to native machine code, rather than the much slower interpreted code like an XLA written in VBA.  **Security:** Since a CAI is a compiled DLL file, you distribute only the DLL file to the end users. Unlike an XLA add-in, no source code is distributed. This allows you to write proprietary code without worrying about end users or competitors gaining access to your code. The password protection of an XLA project is notoriously weak and with the right tool (such as VBAKey, $40 from Passware), a user can crack the VBA password in a matter of seconds.  With a CAI, no source code is ever distributed to the end user, so your code remains safely stored on your own machine -- it is never distributed to the users.  **Multiple Application Support:** Perhaps the biggest advantages of a CAI is the ability to interface with more than one Office program.  You can write one add-in, one project, with one code base, and use that add-in in Excel, Word, and PowerPoint or any other application that supports COM Add-Ins.  Of course, you have to write code to support multiple applications, but you will still distribute only a single file that will work in any supported Office application. This greatly simplifies the development process because you have only one set of source code files to manage, promotes code reuse since the same functions can be used by all the application's code objects, and makes installation on the end user's machine simpler since there is only a single file to distribute.  **Use Of Additional Components:** You can write a CAI in any language you like as long as that language is capable of supporting COM. This includes Visual Basic and Visual C++. This article will deal only with VB6.  (Add-ins, VB.NET, and the Visual Studio Tools For Office (VSTO) will be described on an upcoming web page.) Since your CAI is written in VB6, your forms will be VB6 Forms, not VBA UserForms. This means you have the enhanced functionality of VB6 Forms such as command bars and menu bars and a Status Bar on the forms. All of these are unavailable in VBA UserForms  Working in the VB6 environment, you will have access to all controls supported by VB6, including many that are not supported in VBA UserForms.  **Callable Functions In A COM Add-In** With 2002 (XP), Excel introduced Automation Add-Ins. An automation add-in is a CAI that exposes one or more public functions in a public class module. If you load an Automation Add-In, you can call the public functions in the CAI directly from worksheet cells. This gives you the ease of use of VB and speed nearly comparable to an XLL add-in (which have their own drawbacks).  This page will take you step-by-step through the creation of a COM Add-In that supports Excel and PowerPoint 2003. It is assumed that you are familiar with VB6, using event procedures, and creating menu items and command bars with code.  Starting A New COM Add-In Project In VB6  The first thing to do to is create the basic framework for the COM Add-In.  In this section, we will not use the Add-In project template because not all versions of Office and VB provide that template and its objects. Instead, we'll take a slightly longer method using the IDTExtensibility2 interface. Using the IDTExtensibility2 Interface rather than the project template also allows us to write a single connection class that handles all supported applications. This simplifies the installation of the CAI on the end user's machine.  Before you write your first line of code, you need to create the environment for development of the CAI.  In VB6, go to the File Menu and choose "New Project" and choose "ActiveX DLL".  Remove the Class module that VB creates for you -- we won't be using this. We'll add our own class modules later. In the Properties window, change the name of the project from "Project1" to something meaningful. In this example, we will use "ExampleCAIProject" for the Project name.  From the Project menu, choose References, and add the following references to the existing reference list.   * Microsoft Add-In Designer ( C:\Program Files\Common Files\DESIGNER\MSADDNDR.DLL) * Microsoft Office Object Library (C:\Program Files\Common Files\Microsoft Shared\OFFICE11\MSO.DLL) * Microsoft PowerPoint Object Library (C:\Program Files\Office 2003\OFFICE11\msppt.olb) * Microsoft Excel Object Library (C:\Program Files\Microsoft Office\Office11\Excel.exe)   Of course, your actual path names may be different than those above.  This sets up the references to the type libraries for the common Office objects such as CommandBars and for application-specific objects. In the example project, we have references set to the Office, Excel, and PowerPoint object  libraries in addition to the standard references used by VB itself.  You can download the [complete VB6 Project, including the Installer workbook, here](http://www.cpearson.com/Zips/ExampleCAIProject.zip).  Creating The Connection Class  In this example, we will use a single class module to provide the connectivity to all the host applications, such as Excel and PowerPoint. Using a single connection class requires slightly more complicated code, but makes the installation and management of the CAI simpler. All supported applications use the same connection class.  Insert a class module to your project, name it ExampleConnect, and include the following code after the Option Explicit declaration.  Implements AddInDesignerObjects.IDTExtensibility2  The Instancing property of this class should be 5 - Multiuse.  When you implement an interface, you must include all the methods, properties, and events of that interface in your code even if you don't use those them. They must be included in the code. In the VB editor, change the dropdown box at the upper left of the code pane from "(General)" to "IDTExtensibility2". Then select each item in the dropdown box at the upper right of the code pane. Selecting an item in the dropdown list will add its procedure definition to the code pane. All methods must be included even if they are not to be used. There are five methods of IDTExtensibiliy2 that need to be defined. We will be using only two of these methods: IDTExtensibility2\_OnConnection and IDTExtensibility2\_OnDisconnection. For all the other methods, insert a comment indicating that the event is not being used and to prevent the compiler from stripping out empty procedures.  If your CAI is going to be responding to events in from the host application, you'll need to create class modules to handle these events. Create two new class modules and name them CExcelEvents and CPowerPointEvents.  These classes should have their Instancing property set to 1 - Private. In the CExcelEvents module, use the following code:  Option Explicit  Option Compare Text  Private WithEvents pExcelApp As Excel.Application  Friend Property Get ExcelApp() As Excel.Application  Set ExcelApp = pExcelApp  End Property  Friend Property Set ExcelApp(XLApp As Excel.Application)  Set pExcelApp = XLApp  End Property  Private Sub Class\_Terminate()  Set pExcelApp = Nothing  End Sub  In this module, define and code for any Excel application events that you want your CAI to handle. The event procedure code you will write in this module is identical to event procedure code that you would write in a VBA project. It is assumed you know how to write event procedures.  Use similar code in CPowerPointEvents, making the obvious changes. If your CAI is not going to respond to application events, you may omit these modules.  Add a new module to the project named modProjectGlobals. We will store project-wide global variables and constants in this module. In this module, put the following code.  If you are not using application Events in your CAI, you may omit the lines of code that refer to the event classes.  Public ThisCAI As Office.COMAddIn  Public ExcelApp As Excel.Application  Public PowerPointApp As PowerPoint.Application  Public ExcelEvents As CExcelEvents  Public PowerPointEvents As CPowerPointEvents  Public ExcelControls As CExcelControls  Public PowerPointControls As CPowerPointControls  The ThisCAI variable will hold a reference to the CAI object itself, which is passed in as the AddInInst parameter of the OnConnection event. OnConnection is automatically called when the host application (e.g., Excel) loads the add-in.  The ExcelApp and PowerPointApp variables are used to store the reference to the host application. We'll create all of the classes soon enough.  Now you are ready to add code to the OnConnection event procedure of the ExampleConnect class. In that class, write your OnConnection event as shown below. If you are not using application Events in your CAI, you may omit the lines of code that refer to the event classes.  Private Sub IDTExtensibility2\_OnConnection(ByVal Application As Object, \_  ByVal ConnectMode As AddInDesignerObjects.ext\_ConnectMode, \_  ByVal AddInInst As Object, custom() As Variant)    '''''''''''''''''''''''''''''''''''''''''''''''''  ' Set ThisCAI to the instance of this COM Add-In.  '''''''''''''''''''''''''''''''''''''''''''''''''  Set ThisCAI = AddInInst  '''''''''''''''''''''''''''''''''''''''''''''''''  ' See if we're connecting to Excel. If so, set up  ' the appropriate variables.  '''''''''''''''''''''''''''''''''''''''''''''''''  If TypeOf Application Is Excel.Application Then  Set ExcelApp = Application  Set ExcelEvents = New CExcelEvents  Set ExcelEvents.ExcelApp = Application  End If    '''''''''''''''''''''''''''''''''''''''''''''''''  ' See if we're connecting to PowerPoint. If so, set up  ' the appropriate variables.  '''''''''''''''''''''''''''''''''''''''''''''''''  If TypeOf Application Is PowerPoint.Application Then  Set PowerPointApp = Application  Set PowerPointEvents = New CPowerPointEvents  Set PowerPointEvents.PowerPointApp = Application  End If  End Sub  Creating Menu Items Or CommandBar Controls For Your COM Add-In  Unless your CAI is going to operate only in response to events raised by the host application, you will need to create some sort of user interface (e.g., a menu item or command bar button) to allow the user to access the functionality in your CAI. It is assumed that you know how to create menu items in Office VBA. The procedure is the same in VB6.  While the code will be shown on this page, it will not be explained in any detail.  For organizational purposes, you should put the code the declares, creates, and responds to menu items or commandbar controls in a separate class module, one class module for each host application you will be supporting. Create a class module called CExcelControls.  The Instancing property of this class should be set to 1 - Private. In the modProjectGlobals module, declare a variable as the CExcelControls type:  Public ExcelControls As CExcelControls  Now, in your CExcelControls class, first declare a variable that will be set to the Excel Application and variables for the commandbar controls and/or menu items.  Private pExcelApp As Excel.Application  Private ExcelToolsMenu As Office.CommandBarPopup  Private WithEvents MenuItem1 As Office.CommandBarButton  Friend Property Get ExcelApp() As Excel.Application  Set ExcelApp = pExcelApp  End Property  Friend Property Set ExcelApp(XLApp As Excel.Application)  Set pExcelApp = XLApp  End Property  To enable each control to have its own event handler code, rather than sharing a common event handler for all controls, we will not be using the Tag property of the CommandBarControl.  Since the control has no tag value, we can't later use FindControls(Tag:=SomeTag) to get references to our controls for deletion. Therefore, to enable easy deletion of the controls when the CAI is disconnected from the host application, we will store a reference to each control we create in a Collection object. In CExcelControls, declare a Collection to store the control references. Use the Class\_Initialize event to create a new Collection and the Class\_Terminate event to destroy the Collection.  Private ControlsCollection As Collection  Private Sub Class\_Initialize()  Set ControlsCollection = New Collection  End Sub  Private Sub Class\_Terminate()  Set ControlsCollection = Nothing  End Sub  Next you need to create functions that will create and delete the controls. In CExcelControls, use  Friend Sub CreateControls()  Set ExcelToolsMenu = pExcelApp.CommandBars.FindControl(Id:=C\_EXCEL\_TOOLS\_MENU\_ID)  Set MenuItem1 = ExcelToolsMenu.Controls.Add(Type:=msoControlButton, temporary:=True)  With MenuItem1  .Caption = "Click Me #1"  End With  ControlsCollection.Add Item:=MenuItem1  ' Repeat the code above for each menu item or commandbar control you need to add.  ' Be sure to add each control to the ControlsCollection object.  End Sub  Friend Sub DeleteControls()  Dim Ctrl As Office.CommandBarControl  For Each Ctrl In ControlsCollection  Ctrl.Delete  Next Ctrl  End Sub  where C\_EXCEL\_TOOLS\_MENU\_ID  is a constant declared in modProjectGlobals with a value of 30007 (also create a constant named C\_POWERPOINT\_TOOLS\_MENU\_ID with the same value).   In modProjectGlobals, declare the constants:  Public Const C\_EXCEL\_TOOLS\_MENU\_ID As Long = 30007  Public Const C\_EXCEL\_POWERPOINT\_MENU\_ID As Long = 30007  Finally, add the event handler code in the class CExcelControls for the control(s) created in CreateControls. The following is the procedure for the Click event of the menu item we added in CreateControls. Add the appropriate event procedures for all the controls you created.  Private Sub MenuItem1\_Click(ByVal Ctrl As Office.CommandBarButton, \_  CancelDefault As Boolean)  MsgBox Ctrl.Caption  End Sub  The CExcelControls class is now complete. For PowerPoint, you can use the identical code in a class name CPowerPointControls, just changing any reference from Excel to PowerPoint. Change the variable names appropriately. Of course, you do not need to have the same menu structure in PowerPoint as you do in Excel (or any other host application you are supporting). The CExcelControls class and the CPowerPointControls class are completely independent of one another, and because of the way the OnConnection code is structured (see below), either, but not both, CExcelControls or CPowerPointControls will be instantiated at any one time (per host application -- you can certainly have the CAI open simultaneously in both PowerPoint and Excel, and you don't have to worry about variables overwriting one another).  Now we have to add code in the OnConnection event of the ExampleConnect class to create an instance of CExcelControls and create the controls. Your OnConnection event in ExampleConnect should now look like the following:  Private Sub IDTExtensibility2\_OnConnection(ByVal Application As Object, \_  ByVal ConnectMode As AddInDesignerObjects.ext\_ConnectMode, \_  ByVal AddInInst As Object, custom() As Variant)  '''''''''''''''''''''''''''''''''''''''''''''''''  ' Set ThisCAI to the instance of this COM Add-In.  '''''''''''''''''''''''''''''''''''''''''''''''''  Set ThisCAI = AddInInst  '''''''''''''''''''''''''''''''''''''''''''''''''  ' See if we're connecting to Excel. If so, set up  ' the appropriate variables.  '''''''''''''''''''''''''''''''''''''''''''''''''  If TypeOf Application Is Excel.Application Then  Set ExcelApp = Application  Set ExcelEvents = New CExcelEvents  Set ExcelEvents.ExcelApp = Application  Set ExcelControls = New CExcelControls  Set ExcelControls.ExcelApp = Application  ExcelControls.CreateControls  End If    '''''''''''''''''''''''''''''''''''''''''''''''''  ' See if we're connecting to PowerPoint. If so, set up  ' the appropriate variables.  '''''''''''''''''''''''''''''''''''''''''''''''''  If TypeOf Application Is PowerPoint.Application Then  Set PowerPointApp = Application  Set PowerPointEvents = New CPowerPointEvents  Set PowerPointEvents.PowerPointApp = Application  Set PowerPointControls = New CPowerPointControls  Set PowerPointControls.PowerPointApp = Application  PowerPointControls.CreateControls  End If    End Sub  Finally, we need to delete our controls when the CAI is disconnected from its host application. Write your OnDisconnection event as shown below:  Private Sub IDTExtensibility2\_OnDisconnection(ByVal RemoveMode As AddInDesignerObjects.ext\_DisconnectMode, \_  custom() As Variant)  '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''  ' IDTExtensibility2\_OnDisconnection  ' This executes when the COM Add-In is unloaded by the host application.  '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''    ''''''''''''''''''''''''''''''''  ' Clean up Excel  ''''''''''''''''''''''''''''''''  If Not ExcelApp Is Nothing Then  '''''''''''''''''''''''''''''''''  ' Set ALL Excel object to Nothing.  ' Otherwise they can cause the  ' Excel process to continue to  ' run in the background even  ' after it appears to be closed.  '''''''''''''''''''''''''''''''''  Set ExcelEvents.ExcelApp = Nothing  Set ExcelEvents = Nothing  ExcelControls.DeleteControls  Set ExcelControls = Nothing  Set ExcelApp = Nothing  End If    ''''''''''''''''''''''''''''''''  ' Clean up PowerPoint  ''''''''''''''''''''''''''''''''  If Not PowerPointApp Is Nothing Then  '''''''''''''''''''''''''''''''''  ' Set ALL PowerPoint object to Nothing.  ' Otherwise they can cause the  ' Excel process to continue to  ' run in the background even  ' after it appears to be closed.  '''''''''''''''''''''''''''''''''  Set PowerPointEvents.PowerPointApp = Nothing  Set PowerPointEvents = Nothing  PowerPointControls.DeleteControls  Set PowerPointControls = Nothing  Set PowerPointApp = Nothing  End If    Set ThisCAI = Nothing    End Sub  Creating Callable Functions For Excel COM Add-Ins (Automation Add-Ins)  Excel 2002 (XP) and later supports Automation Add-Ins. An Automation Add-In is simply a COM Add-In containing a Public Class module containing functions that can be called directly from worksheet cells. To take advantage of this, simply create a class module called ExcelFunctions (or whatever you want).  This Instancing property of this class should be 5 - Multiuse.  In that class module, create the functions that you want to be able to call from Excel worksheet cells. Be SURE to use appropriate and proper error handling. If an untrapped error occurs, the add-in will be unloaded and you'll have to restart Excel and reload the automation add-in.  For example, in the class ExcelFunctions, you could have the simple function:  Function DoubleIt(D As Double) As Double  '''''''''''''''''''''''''''''''''''''''''''''  ' DoubleIt  ' This is an example function that simply  ' doubles the input and returns the result.  '''''''''''''''''''''''''''''''''''''''''''''  DoubleIt = D \* 2  End Function  Then, in Excel, go to the Tools menu, choose Add-Ins (*not* COM Add-Ins), and click the Automation button. In the list that is displayed, find your VB6 project name followed by "ExcelFunctions" (or whatever class name you used) and click it. It will appear in the Add-Ins list. Once that is loaded, you  can call functions in the ExcelFunctions class as if they were native Excel functions. In this example project, you would select ExampleCAIProject.ExampleConnect from the Automation Add-Ins list.  Then you can call DoubleIt with the the following formula:  **=DOUBLEIT(1234)**  Remember that a function, in VBA, an XLA add-in, or in an Automation Add-In, can **not** change any aspect of the Excel environment, including changing the value of a cell. Functions in Automation Add-Ins are no different than functions written in VBA. They can only return a value to the cell from which they are called.  Calling CAI Functions From VBA Code  To call the functions in your COM Add-In from other VBA procedure, rather than directly from the worksheet cell, see [Calling Automation Add-In Functions In VBA](http://www.cpearson.com/excel/AutomationAddins.htm#CallingInVBA) on the Automation Add-Ins page.  Installing Your COM Add-In  Once you have created your CAI, you need to install it. Installation and startup and shutdown code requires modifying the  System Registry. I have written a code module that works in both VBA and VB6 to handle the details of adding, modifying, and deleting registry keys and values. I strongly encourage you see the [Functions For Working With The System Registry page](http://www.cpearson.com/excel/Registry.htm), and that you download the modRegistry.bas module file and add it to your VB Project. We will be using the procedures in that file on this page to create, read, and write registry keys and values.  Automating The Installation With The COM Add-In Installer Workbook Because of the tedious work and potential for error when manually adding and editing the System Registry keys and values, I have written an Excel workbook that will automate the complete installation procedure of a COM Add-In.  In the workbook, you are presented with a screen like the one shown below. You simply fill out the required fields an the installer does all the work for you. It will register your DLL file with Windows and write the keys to the HKEY\_CURRENT\_USER section of the Registry, and optionally to the HKEY\_LOCAL\_MACHINE section of the Registry (see below for caveats about writing the keys to HKEY\_LOCAL\_MACHINE). http://www.cpearson.com/images/ComAddInInstaller.png  You can download the [COM Add-In Installer here](http://www.cpearson.com/Zips/ComAddInInstaller.zip). The COM Add-In Installer requires the TypeLib Information DLL file, which is included in the zip file. If you already have this component installed on your machine, the workbook will function normally. If you do not have this component installed on your machine, you will receive compiler errors, "User-defined type not defined.". In this case, copy the file TLBINF32.DLL to your "C:\Windows\System32" folder, close Excel, go to the Windows Start menu, choose Run, and enter the following and click OK:      RegSvr32 "C:\Windows\System32\TBLINF32.DLL"  Then, open this workbook, go into VBA, go to the Tools menu, choose References, and scroll down to and check "TypeLib Information". Once this reference is established, the workbook should work fine. This library is used to retrieve the available ProgIDs of available objects in the DLL file. See the *TLBINF32 Read Me.txt* file in the downloadable zip project for more information about installing the TLBINF32.DLL file.    Registering The DLL With Windows Before you can use your CAI in the host application, you need to register it with Windows. You do this with the RegSvr32 program. With all Office applications closed, go to the Windows Start menu, choose Run, and enter the following. Of course, change the file name to the complete file name of your CAI DLL file:        RegSvr32 "C:\YourFolder\ExampleAddIn.dll"  If you use the automated [COM Add-In Installer Workbook](http://www.cpearson.com/Zips/ComAddInInstaller.zip) to install your CAI, you may omit this step. The Installer does this for you.  Registry Keys You need to add a key with some values to the System Registry in order for Excel (and/or other host applications) to recognize the add-in. You can use RegEdit to edit the registry directly, or use the automated [COM Add-In Installer Workbook](http://www.cpearson.com/Zips/ComAddInInstaller.zip) to install your CAI.  If you want to use RegEdit, you will need to create a key in the HKEY\_CURRENT\_USER section of the Registry named       Software\Microsoft\Office\*AppName*\AddIns\*YourComAddInProgID*  where  *AppName* is the name of the host application which will use your CAI (e.g, "Excel" or "PowerPoint"), and  *YourComAddInProgID* is the ProgID of your COM Add-In. The ProgID is the VB6 Project Name of your CAI, followed by a period, followed by the name of the class in which you implemented the IDTExtensibility2 interface for the host application.  For example, the ProgID of the example COM Add-In describe on this page is ExampleCAIProject.ExampleConnect. Because we used a single connection class for both Excel and PowerPoint, the ProgID is the same when adding the registry entries for both Excel and PowerPoint.  Then, within this key, create the following values:   |  |  |  |  | | --- | --- | --- | --- | | **Name** | **Type** | **Description** | **Value** | | CommandLineSafe | DWORD | Indicates whether the CAI can be run from the command line. | 0 | | Description | String | A description of your CAI. | Example Description | | FriendlyName | String | The name that will be displayed in the CAI dialog box. | My Example COM Add-In | | LoadBehavior | DWORD | Indicates how the CAI will start up. | 3 |   You must now decide whether to create this same key and values in the HKEY\_LOCAL\_MACHINE section of the Registry. If you include this in the HKEY\_LOCAL\_MACHINE section of the Registry, the host application will not display this CAI in the COM Add-Ins dialog box, and the CAI cannot be loaded or unloaded by the user (without the use of VBA code).  This is done to prevent one user from changing settings used by all users.  However, it will make the CAI available to all users of the computer. If you do not include this key and its values in the HKEY\_LOCAL\_MACHINE section, the you will have to install the CAI for each user of the machine.  You can download the [complete VB6 Project, including the Installer workbook, here](http://www.cpearson.com/Zips/ExampleCAIProject.zip).  Debugging Your COM Add-In  Once you've written your CAI, you'll need to be able to debug it to find potential problems as you've added additional functionality beyond the very limited functionality presented in this example CAI.  To debug your CAI, open the project in VB6. Place breakpoints (F9) at the appropriate locations, such as in the OnConnection event procedure, or wherever in your code you need to debug. Next, press CTRL+F5 to start your DLL in debug mode with a full compile. Then open the host application. VB6 will automatically redirect the host application's reference from the compiled DLL file to the project you have in debug mode. When code execution encounters one of your breakpoints, execution will pause and you will have all the VB6 debugging tools at your disposal, such as breakpoints, watches, the Immediate Window, and so on. It is assumed you are familiar with debugging code. The procedures and tools for debugging VB6 code are the same as those in VBA. See the [Debugging VBA page](http://www.cpearson.com/excel/Debug.htm) for more details.  The COM Add-In Dialog  To manage COM Add-Ins, you need to be able to access the COM Add-Ins dialog box, shown below on the left. In the default Excel menu and commandbar configuration, there is not item to display this dialog. You should add the COM Add-Ins menu item to your Tools menu. [Click here for instructions](http://www.cpearson.com/excel/AddingCOMAddInsMenuItem.htm). | |  |
|  | http://www.cpearson.com/images/ComAddinsDialog.png | Note that only add-ins configured for the current user are displayed in the dialog (those add-in that have a key in the HKEY\_CURRENT\_USER section of the registry and do **not** have an item in the HKEY\_LOCAL\_MACHINE section of the registry). Add ins that are configured for all users (add-ins with a key in HKEY\_LOCAL\_MACHINE) are not listed in the dialog. This is done to prevent one user from changing the settings that affect all users of a machine. |  |