The Toolbox Controls Installer

October 16, 2012

Abstract

This paper provides information about the Toolbox Controls Installer (TCI) for Visual Studio 2008 and higher. This paper provides guidelines for control authors to register their controls for installation into the Visual Studio Toolbox.

This information applies to all editions of Visual Studio 2008 and later that include the Toolbox.

References and resources that are discussed in this paper are listed at the end.

**Disclaimer**: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet website references, may change without notice. Some information relates to pre-released product which may be substantially modified before it’s commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2012 Microsoft. All rights reserved.



Contents

[Overview 3](#_Toc338183792)

[Should I use the TCI? 3](#_Toc338183793)

[Getting started 4](#_Toc338183794)

[Extension SDKs 4](#_Toc338183795)

[SKU-specific registration 4](#_Toc338183796)

[AssemblyFoldersEx registration 5](#_Toc338183797)

[Advanced topics 6](#_Toc338183798)

[Localizing tab names 6](#_Toc338183799)

[Localizing item names 6](#_Toc338183800)

[Specifying different tab placements for items in a single assembly 7](#_Toc338183801)

[Specifying tab order 7](#_Toc338183802)

[Troubleshooting 7](#_Toc338183803)

[The TCI isn’t installing my controls 7](#_Toc338183804)

[Updates to my assemblies aren’t being applied to the Toolbox 9](#_Toc338183805)

[The user can’t repair my installation 9](#_Toc338183806)

[Registration schema reference 9](#_Toc338183807)

[AssemblyFoldersEx registration 9](#_Toc338183808)

[SKU-specific registration 10](#_Toc338183809)

[Extension SDKs 10](#_Toc338183810)

# Overview

The Toolbox Controls Installer (TCI) is a Visual Studio feature that provides a way to install controls in the [Toolbox](http://msdn.microsoft.com/en-us/library/2381cd09.aspx). It is supported on all SKUs of Visual Studio, including the Express SKUs. The TCI finds controls by looking for registry entries, so it doesn’t require you to write code to create Toolbox items [programmatically](http://msdn.microsoft.com/en-us/library/bb166237.aspx).

When you use the TCI, your Toolbox controls are installed in two phases:

1. You create the control registration. This is done outside Visual Studio, either via a setup program you provide or via a standard deployment mechanism like VSIX. It may involve writing directly to the registry or placing a .pkgdef file on disk.
2. The TCI creates toolbox items from the registration you created in step 1. This occurs the next time the toolbox is initialized (usually when it is shown in in the Visual Studio IDE) after your registration has been created.

Throughout this document, registry paths under HKLM\SOFTWARE should have “[Wow6432Node](http://msdn.microsoft.com/en-us/library/windows/desktop/ms724072.aspx)” inserted when running on 64-bit Windows (for example, HKLM\SOFTWARE\VisualStudio becomes HKLM\SOFTWARE\Wow6432Node\VisualStudio) because Visual Studio is a 32-bit application and it can only “see” the 32-bit portion of HKLM.

In this document, the term “tab” refers to a named grouping of items in the Toolbox, such as the “Common Controls” or “All Windows Forms” groups.

# Should I use the TCI?

The TCI is a convenient way to install controls in the Toolbox. It has some advantages as well as some limitations compared to other methods. You *must* use the TCI if you need to:

* Install controls without writing code to create them programmatically (as a Visual Studio package, add-in, or external program accessing the Visual Studio object model).
* Install controls on Express SKUs.
* Install controls automatically from an [Extension SDK](http://msdn.microsoft.com/en-us/library/hh768146.aspx).

You *cannot* use the TCI if you need to:

* Install controls other than Winforms, WPF, Silverlight, or XAML. For example, Windows Phone controls and Workflow items are not supported.
* Specify the name of a WPF, Silverlight, or XAML control that differs from the type name.
* Specify the order of controls within a Toolbox tab.

If the TCI is not appropriate for your needs, please refer to the Visual Studio SDK’s [Toolbox documentation](http://msdn.microsoft.com/en-us/library/bb166364) for a description of alternatives.

# Getting started

The TCI supports three “modes” of registration, so first you need to choose which one is appropriate for your requirements.

1. Are your Toolbox controls in an extension SDK? If so, skip down to “[Extension SDKs](#_Extension_SDKs)”.
2. Do you need to do any of the following?
   1. Place controls from the same assembly on different tabs
   2. Control the order in which your tabs appear on the Toolbox
   3. Install controls only from some assemblies in a folder and ignore others
   4. Install controls on Visual Studio 2008 or earlier
   5. Install controls only for some versions of Visual Studio (after Visual Studio 2008) but not others
   6. Install controls only for some SKUs of Visual Studio but not others
   7. Use a VSIX to deploy your controls
   8. *Not* show your assemblies in Visual Studio’s “Add Reference” dialog or allow them to participate in assembly reference resolution

If so, skip down to “[SKU-specific registration](#_SKU-specific_registration)”.

1. Otherwise, skip down to “[AssemblyFoldersEx registration](#_AssemblyFoldersEx_registration)”.

## Extension SDKs

When you deploy an SDK that contains toolbox controls, the TCI will automatically “see” it and install its controls. See the [SDK documentation](http://msdn.microsoft.com/library/hh768146.aspx) for details on SDK authoring.

## SKU-specific registration

This registration mode allows you to register assemblies for a specific SKU and version of Visual Studio. It is strongly recommended that you use a VSIX for deployment (using, for example, the “Windows Forms Toolbox Control” project template from the Visual Studio SDK). The SDK’s Toolbox control templates also provide a good starting point to generate the necessary registration, although you may want to augment it depending on your needs. In particular, Silverlight and XAML controls are not supported by default; however, you can modify the ProvideToolboxControlAttribute class in the template to support them by setting a string value called “SilverlightControls” or “WindowsUIXamlControls”, respectively, to “1”. For XAML controls, you will also need to supply a string value called “TargetPlatform”; at this time, the only supported platform is “Windows,Version=v8.0”.

If you need to author your own registration or augment the default registration, please refer to the “[Registration schema reference](#_SKU-specific_registration_1)” section at the end of this document for details on the supported registration values.

Though not required, it is highly recommended that you add a “TargetFramework” string value as well. This is a target framework moniker (TFM) specifying the minimum required target framework for your control (such as “.NETFramework,Version=v4.0”), and including it will enable the TCI to install your controls more quickly because it will not have to be determined via [reflection](http://msdn.microsoft.com/en-us/library/f7ykdhsy.aspx). See [the Visual Studio Blog post on managed multi-targeting](http://blogs.msdn.com/b/visualstudio/archive/2010/05/19/visual-studio-managed-multi-targeting-part-1-concepts-target-framework-moniker-target-framework.aspx) for more information about TFMs.

If you need to support Visual Studio versions earlier than Visual Studio 2010, you will not be able to use VSIX or .pkgdef files, so you will have to provide an installer to create the necessary registry entries. You will also need to do one additional step (whenever you update your registration – on install, uninstall, or any other update) in order for Visual Studio to “see” your changes: you must increment the registry value “Default Items” under HKLM\SOFTWARE\Microsoft\<skuName>\<version>\Packages\{2c298b35-07da-45f1-96a3-be55d91c8d7a}\Toolbox.

## AssemblyFoldersEx registration

Using this registration mode, you can register assemblies in a single location to be installed into all versions (Visual Studio 2010 and later) and SKUs of Visual Studio. In your installer:

1. Deploy the assemblies you want to install into one or more folders on disk.
2. Add an entry for each of these folders under the appropriate AssemblyFoldersEx key in the registry. For example, .NET 4.0 controls should be registered under HKLM\SOFTWARE\Microsoft\.NETFramework\v4.0.30319\AssemblyFoldersEx; Silverlight 4.0 controls should be registered under HKLM\SOFTWARE\Microsoft SDKs\Silverlight\v4.0\AssemblyFoldersEx. Set the default value of the key you create to the path of the folder. The TCI doesn’t look in subfolders of the folder you register, so each subfolder must be registered separately if you want it to be searched. At runtime, the toolbox will automatically hide controls whose registration location is a higher framework version than the active project. For example, controls registered under Silverlight v4.0 will be hidden when the active project targets Silverlight v3.0; they will be visible when it targets v4.0 or higher.
3. Under each key created in step 2, add a key called “Toolbox” containing a string value called “TabName” which specifies the tab to which the controls from that folder should be added.

Here is an example (in regedit.exe export format) that registers all .NET Framework controls from assemblies in “C:\Program Files\VendorName\ControlAssemblies” to be installed in the “My Controls” tab:

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\.NETFramework\v4.0.30319\AssemblyFoldersEx\My Controls]  
@="C:\\Program Files\\VendorName\\ControlAssemblies"  
  
[HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\.NETFramework\v4.0.30319\AssemblyFoldersEx\My Controls\Toolbox]  
"TabName"="My Controls"

# Advanced topics

## Localizing tab names

If your tab names are localized, you can supply a resource ID instead of a literal string. The resource ID can take one of three forms, shown as follows (where “<id>” is either a managed resource name preceded by “@” or a native resource number preceded by “#”).

* If the resource is from an installed Visual Studio package, use the syntax <id>;<package guid>. Example: “@ToolboxTabName;{AE56C307-C10D-472E-A1D9-9A13DE10BB85}”.
* If the resource is from an assembly in the GAC, use the syntax <id>;<assembly full name>. Example: “@ToolboxTabName;MyControlAssembly, Version=1.0.0.0, Culture=neutral, PublicKeyToken=121fae78165ba3d0”.
* If the resource is in an assembly (or native resource DLL) at a known location on disk, use the syntax <id>;<path>. Example: “@ToolboxTabName;C:\Program Files\VendorName\Product\MyControlAssembly.dll”.

In all these cases, the assembly (or native DLL) you specify should be the primary one, *not* a localized satellite DLL. For example, with English and Japanese resources installed, “C:\Program Files\VendorName\Product” might contain two subfolders “en” and “ja”, each of which has MyControlAssembly.resources.dll. Your resource ID should point to “C:\Program Files\VendorName\Product\MyControlAssembly.dll,” not to a specific localized resource DLL. Visual Studio will choose the appropriate resource DLL at runtime based on its current locale.

Visual Studio’s resource loader knows how to find resources under the usual namespaces used by VS packages, but it doesn’t search every namespace in the resource assembly. If your managed resource is not being found, you may need to prepend the resource ID with a namespace (followed by a colon). Example: “@VendorName.MyControls.Resources:ToolboxTabName;C:\Program Files\VendorName\Product\MyControlAssembly.dll”.

## Localizing item names

Item names can only be customized for Winforms controls; for all other types, the display name is always derived from the type name. To customize the name of a Winforms control, create a class that derives from ToolboxItem and specify that type in the control’s ToolboxItemAttribute. In that derived class, override the Initialize method and, after calling base.Initialize, set the DisplayName property to whatever you desire (such as a localized string).

## Specifying different tab placements for items in a single assembly

By default, all items in an assembly are placed on the same tab. For AssemblyFoldersEx registration, there is no way to override this, but for SKU-specific registration, you can create a subkey of your assembly registration key called “ItemCategories” to specify a different placement for each control. Under that key, create a separate entry for each control whose placement you want to override. The name of the entry should be the namespace-qualified name of the control and the value should be the tab name. Here is an example (in .pkgdef format) which places all controls from “MyControlAssembly” in the “My Controls” tab, except “MyEditBox” and “MyTabControl”, which are placed in the “Editing” and “Layout” tabs, respectively:

[$RootKey$\ToolboxControlsInstaller\MyControlAssembly, Version=1.0.0.0, Culture=neutral, PublicKeyToken=121fae78165ba3d0]  
@="My Controls"  
  
[$RootKey$\ToolboxControlsInstaller\MyControlAssembly, Version=1.0.0.0, Culture=neutral, PublicKeyToken=121fae78165ba3d0\ItemCategories]  
"MyNamespace.Controls.MyEditBox"="Editing"  
"MyNamespace.Controls.MyTabControl"="Layout"

## Specifying tab order

For AssemblyFoldersEx registration, there is no way to override the default order in which your controls are installed. For SKU-specific registration, each assembly registration key can provide a DWORD value called “Index” which specifies the relative order of the tab created for it. Tabs are ordered with the lowest index value appearing first in the Toolbox.

# Troubleshooting

## The TCI isn’t installing my controls

First, ensure that the controls are in fact missing and not just disabled. (If disabled, they will be hidden unless you switch on “Show All” from the Toolbox context menu.) If they are only disabled, [the Visual Studio Blog post on Toolbox filtering errors](http://blogs.msdn.com/b/visualstudio/archive/2009/11/18/diagnosing-toolbox-filtering-errors.aspx) may help diagnose the problem.

Next, generate an “activity log”. To do this, first update your control registration (so Visual Studio will “see” the update and try to re-install your controls), then run Visual Studio with the /log parameter, like “devenv /log c:\users\me\vslog.xml”. In the IDE, make sure the Toolbox is visible so it will try to install your controls. (The TCI is only invoked when the Toolbox is initialized, not unconditionally when Visual Studio starts.) Then close Visual Studio and open the log file.

First, search for an entry with a “source” of “Microsoft.VisualStudio.IDE.ToolboxControlsInstaller.ToolboxInstallerPackage”. If no such tag is present, it means one of three things:

1. If this is a version of Visual Studio earlier than Visual Studio 2010, you may not have updated the TCI’s “Default Items” value.
2. There could be a corrupt installation of Visual Studio that prevents the TCI from loading.
3. The TCI itself could have been manually unregistered from the machine.

Next, search for an entry with a “description” of “Attempting to get Toolbox items from Assembly <your assembly’s full name>”. If no such tag is present, it could indicate one of the following problems:

1. The TCI failed to find or recognize your registration. Double-check that all required registration values are present, that the assembly exists on disk and that the Visual Studio process has permission to read it.
2. The TCI found two assemblies with the same full name. In Visual Studio 2010, this would result in one of them being ignored. Visual Studio 2012 can install controls from two assemblies with the same full name found in different folders.
3. The registered assembly is 64-bit. Visual Studio is a 32-bit process so it can only load 32-bit or “Any CPU” assemblies.

Next, search for an entry with a “description” of “Attempting to add Toolbox item <your item’s namespace-qualified type name>”. If no such tag is present, the TCI failed to load your assembly or failed to find your control. This could indicate one of the following problems:

1. The TCI could not resolve a dependency of your assembly. The TCI will look for dependent assemblies in AssemblyFoldersEx and in Visual Studio’s probing paths (which includes those registered under the BindingPaths registry key, as by ProvideBindingPathAttribute). For controls in extension SDKs, the TCI will also look for controls in dependent SDKs declared in your SDK’s manifest.
2. The TCI could not find any controls of the specified type (applies to SKU-specific registration only). Example: the “SilverlightControls” value was not set for a Silverlight assembly.
3. The type of assembly didn’t match the AssemblyFoldersEx location where it was registered. Example: you created a registration entry under “.NETFramework” but the assembly actually contains Silverlight controls.
4. There was an error in an attribute applied to your control class. Example: you applied ToolboxItemAttribute using the constructor that takes a type name as a string, but the name was incorrect.
5. The framework against which your assembly was built is not supported by this version of Visual Studio.

In Visual Studio 2012 and higher, the TCI will catch all exceptions that occur during control installation and add them to the log, so you should see a log entry with exception information that helps indicate the cause of the problem. In previous versions, the TCI only caught a few specific exceptions, and any that weren’t caught would result in control installation being aborted early, so there may or may not be a useful error description in the log.

## Updates to my assemblies aren’t being applied to the Toolbox

When the toolbox loads, the TCI checks only the registry, not the assemblies on disk (for performance reasons). Therefore, if you update an assembly, you must also update the registry. For SKU-specific registration, this would normally happen as a result of any assembly update, since the assembly should have a different build number, so the key name will change. For AssemblyFoldersEx registration, it’s recommended that you create a “Version” value under each “Toolbox” key and change that value with each update.

For Extension SDKs, the TCI will check the timestamp of the SDK manifest, so you will need to touch the manifest if any assemblies are updated.

## The user can’t repair my installation

If the user deletes some of your controls from the toolbox and then wants to restore them, re-running your setup in “repair” mode typically won’t accomplish that. Since the repair will just re-write the same registry values that are already in place, Visual Studio won’t be aware that any update is required.

One workaround is for the user to reset the toolbox. This should restore all the deleted controls, but it does mean that any other customizations (such as the addition of controls from the Choose Toolbox Items dialog) will be reverted as well.

Another workaround is for the user to uninstall your controls entirely, run Visual Studio (and show the toolbox to make sure it’s initialized) so that Visual Studio can “see” the controls are gone, then reinstall your controls.

# Registration schema reference

## AssemblyFoldersEx registration

<AssemblyFoldersEx root key>  
*Example: HKLM\SOFTWARE\Microsoft\.NETFramework\v4.0.30319\AssemblyFoldersEx  
Example: HKLM\SOFTWARE\Microsoft SDKs\Silverlight\v4.0\AssemblyFoldersEx*

<Product name key>  
*Arbitrary name representing a product or collection of assemblies.  
Ex: “WCF Data Services Assemblies”*

Default value  
*The path of the folder containing the assemblies.*

“Toolbox” key

“TabName” string value

Optional “Version” DWORD value  
*Increment this whenever you update the assemblies on disk so the TCI knows to re-scan them. Actually, this can have any name and type; the TCI only cares whether anything under the “Toolbox” key has changed.*

## SKU-specific registration

NOTE: for this registration mode, it is strongly recommended that you use a .pkgdef file instead of writing to the registry directly.

VS root key  
*This can be under HKLM (for all users) or under HKCU (for a specific user).  
Example: HKLM\SOFTWARE\Microsoft\VisualStudio\11.0  
Example: HKCU\SOFTWARE\Microsoft\VSWinExpress\11.0*

“ToolboxControlsInstaller” key  
*This may already be present or you may need to create it.*

<Assembly full name key>  
*Example: MyControlAssembly, Version=1.0.0.0, Culture=neutral, PublicKeyToken=121fae78165ba3d0*

Default value  
*The tab name under which the items should be added.*

Optional “Codebase” string value  
*The assembly path on disk (not necessary if the assembly is in the GAC).*

Optional “WPFControls”, “SilverlightControls”, or “WindowsUIXamlControls” string value = “1”  
*The type of controls in the assembly. If none of these values are present, Winforms controls are assumed.*

Optional (but strongly recommended) “TargetFramework” string value   
*The minimum compatible framework for the controls.  
Example: .NETFramework,Version=v4.0*

Optional “Index” DWORD value  
*The relative order of the tab.*

“TargetPlatform” string value (only for XAML controls)  
*Only “Windows,Version=v8.0” is currently supported.*

Optional “ItemCategories” key

One or more of:  
<type name> string value = <tab name>  
*Example: System.Windows.Forms.Textbox = MyControlTab*

## Extension SDKs

See the [SDK documentation](http://msdn.microsoft.com/library/hh768146.aspx) for a description of the schema.