InstallShield®

WHITE PAPER

Types of MSI Custom Actions

by Robert Dickau
Principal Technical Training Writer, Acresso Software

Acresso™ SOFTWARE
Powering the business of software

Types of MSI Custom Actions

Introduction

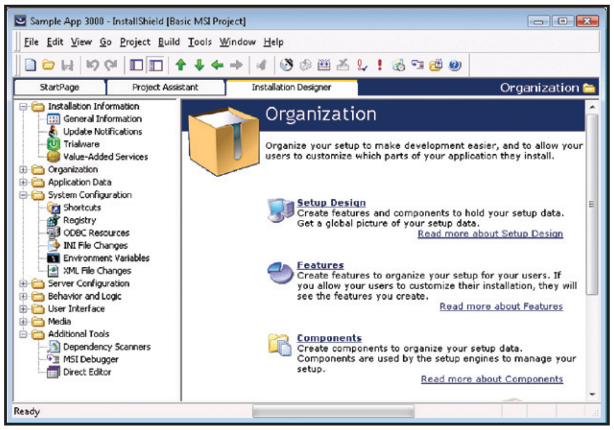
This white paper describes some of the types of custom actions supported by Windows Installer (MSI). These types include:

- Launching executables
- Calling DLL or script functions
- Setting Property-table properties and Directory-table properties
- Error actions

It also highlights how InstallShield® from Acresso Software assists you in working with custom actions.

Using the InstallShield Environment

This white paper frequently refers to the InstallShield development environment. It is assumed you are familiar with the general layout of the InstallShield interface, which contains a list of views with which you can modify different portions of your installation project.



Organization

Organize your setup to make development easier, and to allow your users to customize which parts of your application they install.

Setup Design
Create features and components to hold your setup data. Get a global picture of your setup data.
Read more about Setup Design

Features
Create features to organize your setup for your users. If you allow your users to customize their installation, they will see the features you create.
Read more about Features

Components
Create components to organize your setup data. Components are used by the setup engines to manage your setup.
Read more about Components

For example, the General Information view is where you set general product and project properties; the Setup Design view enables you to edit the features, components, and component data used by your project; the Registry view enables you to modify the registry data installed by your installation program; and the Direct Editor view gives you access to the raw MSI database tables.

It is also assumed you are familiar with some of the wizards available with InstallShield, such as the Release Wizard and Component Wizard.

- The Release Wizard, available under the Build menu and also from the Releases view, lets you describe the properties—media type, compression settings, and so forth—of a release, and then builds the specified release image.
- The Component Wizard, available by right-clicking a feature in the Setup Design view, lets you create special types of components, such as components for COM servers, fonts, and Windows services.

The InstallShield Help Library contains information about using every view and wizard in the InstallShield environment. The InstallShield Help Library is available when you press F1 with any view selected; you can also select Contents from the Help menu to view the help library.

In addition to the graphical environment, InstallShield provides several tools for modifying and building projects from the command line or an external script. For example, to build a project from the command line, batch file, or other automated process, you can use the executable IsCmdBld.exe. The IsCmdBld executable is located in the System subdirectory of the InstallShield distribution directory.

To rebuild a project, you pass IsCmdBld the project file path, the product configuration name, and the release name that you want to rebuild. A sample command appears as follows:

```
iscmdbld -p C:\ProductName.ism -a BuildConfig -r
ReleaseName
```

In addition, InstallShield provides an Automation interface, with which you can modify the contents of a project file without using the graphical environment.

Learn More about InstallShield
If you wish to learn more about the capabilities of InstallShield, please visit the Acresso Web site at www.acresso.com/installshield.

Custom Action Basics

There are two steps involved for each custom action you want to use:

1. **Define the action:** Specify what the action does (launch an executable, call a DLL function, set a property, and so forth) and its other behavior (whether to test the return value, and so forth).
2. **Schedule the action:** Specify where the action runs relative to other actions, which installation phase (immediate execution, deferred execution, and so forth) the action uses, and under what conditions the action runs.

A general principle is that you should not use a custom action when a standard action performs the desired task. One reason is that the effects of custom actions are not automatically removed when your application is uninstalled or rolled back.

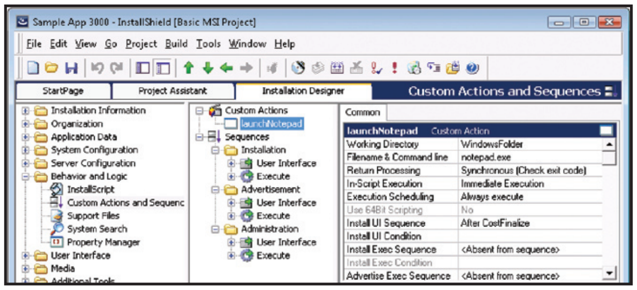
For each custom action that performs system changes, you should create corresponding uninstall and rollback actions.

Executable Custom Actions

One of the most commonly used types of custom actions is an action that launches an executable. This type of action is commonly used to open documents installed by the current installation, or to launch system executables to perform system changes that Windows Installer does not directly support. The executable that you launch with this type of custom action can be installed by the current installation, already located on the target system, or streamed into the Binary table of the MSI database.

For example, suppose you want to launch the copy of Notepad from the target system's Windows directory. You begin by opening the Custom Actions and Sequences view, right-clicking the Custom Actions icon, and selecting **New EXE > Path referencing a directory.**

In the **Working Directory** setting, enter WindowsFolder, the Directory property representing the location of the executable. In the **Filename & Command line** setting, enter the executable name **notepad.exe**.



launchNotepad	Custom Action
Working Directory	WindowsFolder
Filename & Command line	notepad.exe
Return Processing	Synchronous (Check exit code)
In-Script Execution	Immediate Execution
Execution Scheduling	Always execute
Use 64Bit Scripting	No
Install UI Sequence	After CostFinalize
Install UI Condition	
Install Exec Sequence	<Absent from sequence>
Install Exec Condition	
Advertise Exec Sequence	<Absent from sequence>

If the executable being launched is in a directory on the target system, the custom action must be placed after the standard CostFinalize action. The CostFinalize action sets the values of Directory properties, and an attempt to reference a Directory property (such as WindowsFolder or SystemFolder) will cause run-time error 2732, which is described in the Windows Installer Help Library as "Directory Manager not initialized".

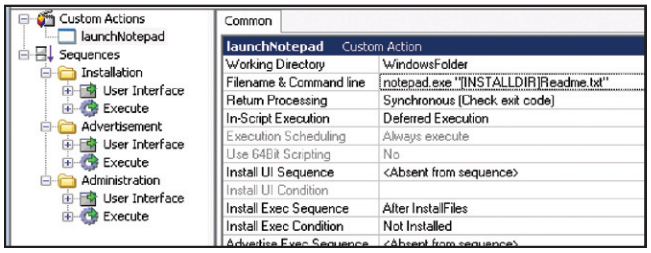
In this case, because Notepad.exe is present on the target system, this custom action can be placed in either the User Interface sequence or Execute sequence (or both), after the CostFinalize action. It is not necessary to specify deferred execution, or for the InstallFiles action to have run first.

An executable custom action does not have access to installation properties, other than those passed as command-line arguments. A typical use of a command-line argument is to pass the path to a document to the executable being launched. For example, suppose you want to launch a Readme file with Notepad.exe after data transfer takes place. In this case, the **Filename & Command line** setting for the custom action might read:

notepad.exe "[INSTALLDIR]Readme.txt"

The quotation marks around the argument are required by most executables in case the file path contains any spaces.

An action that launches an executable being installed, or one that opens a document being installed, must be scheduled for deferred execution after the standard InstallFiles action. During immediate mode, data transfer has not yet begun; and in deferred mode before InstallFiles, documents and executables will not have been placed on the target system.



launchNotepad	Custom Action
Working Directory	WindowsFolder
Filename & Command line	notepad.exe "[INSTALLDIR]Readme.txt"
Return Processing	Synchronous (Check exit code)
In-Script Execution	Deferred Execution
Execution Scheduling	Always execute
Use 64Bit Scripting	No
Install UI Sequence	<Absent from sequence>
Install UI Condition	
Install Exec Sequence	After InstallFiles
Install Exec Condition	Not Installed
Advertise Exec Sequence	<Absent from sequence>

The condition **Not Installed** ensures the action runs—that is, the Readme file is displayed—only during a first-time installation, and not during maintenance mode or uninstallation.

To ensure the action runs only during a first-time, full-UI installation, you could use the condition **(Not Installed) and (UILevel=5)**. A further possible refinement is to associate the action with the component containing the Readme file, using a component-action condition **($ComponentName=3)**.

DLL and Script Custom Actions

In addition to running executables, Windows Installer enables you to extend your installation by calling code in DLLs and scripts, where the DLL or script can be embedded in the installation or installed with the other component data. This section introduces the concepts related to calling DLL and script code.

The Windows Installer engine can directly call functions from a "Windows Installer DLL", or "MSI DLL". An MSI DLL providing custom actions exports at least one function with the following signature:

```
UINT __stdcall FunctionName(MSIHANDLE hInstall) { ... }
```

A function you intend to call from an MSI DLL custom action must use this signature, with the only variation allowed being the function name. The return value is used to determine if the function succeeds or fails, and therefore whether the installation should exit as a result of the custom action; and the MSIHANDLE argument passed to the function is a handle to the running installation, used as an argument to MSI API functions such as MsiGetProperty that query the running installation.

Windows Installer requires that a DLL used in this type of custom action must be a callable DLL, and not a Visual Basic DLL or .NET DLL. The DLL function cannot accept arguments, but instead must pass information back and forth using Windows Installer properties. The MsiSetProperty function sets the value of a property, and MsiGetProperty reads the value of a property.

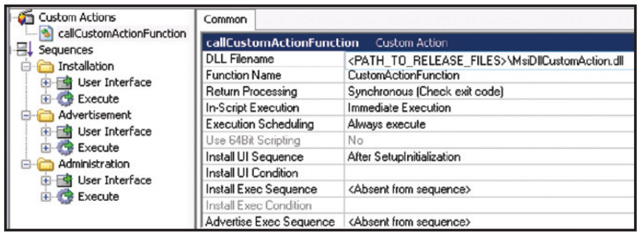
NOTE: The InstallShield environment supports a special type of DLL custom action called a "Standard DLL" action, which enables you to call DLL functions with signatures different from the MSI DLL function presented earlier. When you create a Standard DLL custom action, you will be prompted for the DLL name, the function name, and for constants or properties used to provide the function arguments and return values. Recent InstallShield versions also support "managed code" custom actions, such as C# and VB.NET DLL methods.

Code for a simple MSI DLL custom action might appear as follows:

```
#pragma comment(lib, "msi.lib")
#include <windows.h>
#include <msi.h>
#include <msiquery.h>
// an MSI DLL custom action function must use this signature
UINT __stdcall CustomActionFunction(MSIHANDLE hInstall)
{
   MessageBox(
      GetForegroundWindow( ),
      TEXT("Running MSI DLL action..."),
      TEXT("MSI DLL"),
      MB_OK | MB_ICONINFORMATION);

   return ERROR_SUCCESS; // return success to MSI
}
```

Assuming this code has been compiled into a DLL called MsiDllCustomAction.dll, and that the function name CustomActionFunction is correctly exported from the DLL (using a .def file, for example), the properties of a custom action that calls the DLL might appear similar to the following figure. In the **Function Name** setting, you specify only the function name because the return value and arguments are predefined for an MSI DLL.



callCustomActionFunction	Custom Action
DLL Filename	<PATH_TO_RELEASE_FILES>\MsiDllCustomAction.dll
Function Name	CustomActionFunction
Return Processing	Synchronous (Check exit code)
In-Script Execution	Immediate Execution
Execution Scheduling	Always execute
Use 64Bit Scripting	No
Install UI Sequence	After SetupInitialization
Install UI Condition	
Install Exec Sequence	<Absent from sequence>
Install Exec Condition	
Advertise Exec Sequence	<Absent from sequence>

Similarly, a VBScript custom action typically defines a function similar to the following:

```
Function FunctionName( )
' do something
End Function
```

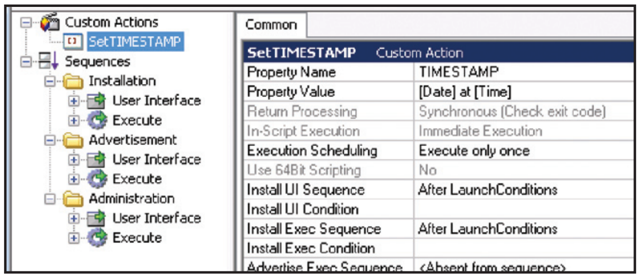
The exception is the setting **VBScript > Stored in custom action**, which can store short scripts directly in the CustomAction table, and for which you do not specify a function name.

Like MSI DLL custom action functions, VBScript custom action functions do not accept arguments, but instead must also communicate with the running installer engine using properties. In VBScript custom actions, the Session object represents the running installation, and the expression **Session.Property("PROPNAME")** enables you to get and set the value of a property.

Setting Properties

A set-a-property custom action either sets or creates a property. If the property exists in the Property table, the custom action will overwrite its value, and if the property does not exist in the Property table, the custom action will create it. As described previously, however, the values of private properties are reset to their defaults when execution switches from the User Interface sequence to the Execute sequence.

An advantage to using set-a-property custom actions over defining properties in the Property table is that the action can resolve Formatted expressions, while entries in the Property table do not resolve Formatted expressions. For example, you can create a custom action that sets TIMESTAMP to the value "[Date] at [Time]", and the values of the embedded properties Date and Time will be expanded at run time. You can then use the Formatted expression [TIMESTAMP] in, for example, a registry value, and the value will be expanded at run time.



SetTIMESTAMP	Custom Action
Property Name	TIMESTAMP
Property Value	[Date] at [Time]
Return Processing	Synchronous (Check exit code)
In-Script Execution	Immediate Execution
Execution Scheduling	Execute only once
Use 64Bit Scripting	No
Install UI Sequence	After LaunchConditions
Install UI Condition	
Install Exec Sequence	After LaunchConditions
Install Exec Condition	
Advertise Exec Sequence	<Absent from sequence>

An example of this type of custom action that exists in each new InstallShield project is called SetARPINSTALLLOCATION, which sets the predefined property ARPINSTALLLOCATION to [INSTALLDIR]. The value of ARPINSTALLLOCATION is automatically written to the target system's registry by the Windows Installer engine, enabling a custom action or external program to read the main install directory for an existing product using the MsiGetProductInfo function or Installer.ProductInfo method. Because properties cannot be set during deferred execution, the In-Script Execution setting for a set-a-property custom action cannot be changed from Immediate Execution.

Setting Directory Properties

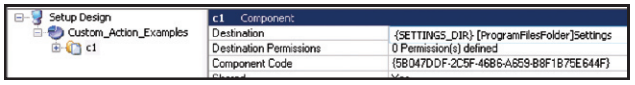
You can also use set-a-directory custom actions to set the values of Directory properties. Directory properties are those that refer to the locations of directories on the target system; built-in examples are ProgramFilesFolder, DesktopFolder, and SystemFolder. If you want to modify the location that a Directory property points to, you can use a set-a-directory action.

Directory properties are set by the disk-costing process, and therefore set-a-directory custom actions must be scheduled after the standard CostFinalize action.

NOTE: Windows Installer ensures that the value of each Directory property ends with a backslash. If your installer writes a Directory property to the registry using the format [INSTALLDIR], for example, a typical value is **C:\Program Files\Our Company\Our Application**. If the value is to be read later from within your application, your application code must account for the trailing backslash.

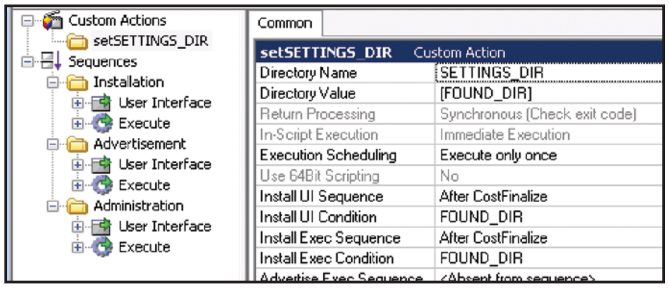
For example, suppose you have a component that you want to install to a directory located by a custom action. In this case, assume the custom action populates a property called FOUND_DIR. One option is to define the component to use a custom property with an arbitrary initial value, and then use a set-a-directory custom action to set the placeholder property value to [FOUND_DIR].

In the following figure, a component has been given a destination directory represented by the Directory property name SETTINGS_DIR, with a default value of [ProgramFilesFolder]Settings.



c1	Component
Destination	{SETTINGS_DIR} [ProgramFilesFolder]Settings
Destination Permissions	0 Permission(s) defined
Component Code	{5B047DDF-2C5F-46B6-A659-B8F1B75E644F}

To install this component to the directory located by the custom action, create a set-a-directory custom action with source SETTINGS_DIR and target [FOUND_DIR], scheduling the action after the CostFinalize action in both the User Interface and Execute sequences.



setSETTINGS_DIR	Custom Action
Directory Name	SETTINGS_DIR
Directory Value	[FOUND_DIR]
Return Processing	Synchronous (Check exit code)
In-Script Execution	Immediate Execution
Execution Scheduling	Execute only once
Use 64Bit Scripting	No
Install UI Sequence	After CostFinalize
Install UI Condition	FOUND_DIR
Install Exec Sequence	After CostFinalize
Install Exec Condition	FOUND_DIR
Advertise Exec Sequence	<Absent from sequence>

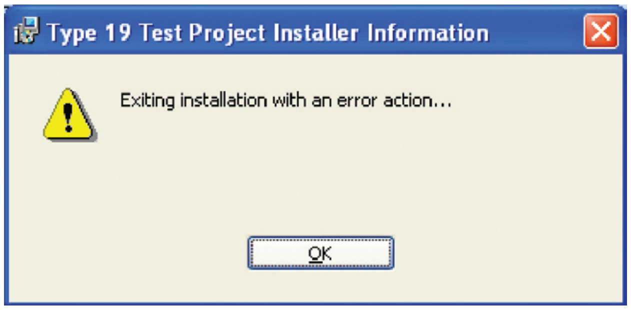
(Note that the action uses the condition FOUND_DIR, which ensures SETTINGS_DIR is changed from its default value only if custom action found the desired directory.)

At installation time, the component's files will be installed to the directory discovered by the custom action.

Error Custom Actions

Error custom actions display an error message to the end user, and then exit the installation when the user dismisses the error dialog. Because an error custom action always exits the installation, you should attach a condition to an error action to ensure it runs only when appropriate.

When you create an error action in the Custom Actions and Sequences view, you can use the Error Message setting to specify the error message that should be displayed when the action's conditions are met at run time. Suppose that you schedule the action immediately after the LaunchConditions action in the Execute sequence. When you rebuild and run the project, the effect of the custom action is to display an error dialog box similar to the following:



Exiting installation with an error action...

After the end user dismisses the error dialog box, the installation exits. (For a silent installation, the error message is by default written to an MSI log file, and also to the system's application event log.)

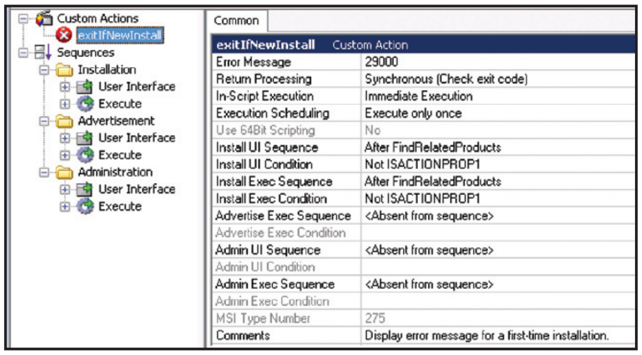
The Error Message setting for an error custom action uses the Formatted data type, so you can expand the values of properties using the syntax [PropertyName].

In addition, the Error Message setting for an error custom action can contain the number of a record in the Error table. The Error table is also exposed in the Direct Editor view. As described in the Windows Installer Help Library topic "Error Table", the error codes numbered from 25000 to 30000 are reserved for custom actions, and therefore you can use numbers in this range for your custom messages. The advantage to using an entry in the Error table, instead of using a hard-coded message in the Error Message setting, is that Error-table messages are automatically added to the InstallShield string table, and therefore you can provide localized error messages for multi-language installations.

In the Direct Editor view, you create a record in the Error table the same way you do for any other table. The fields used in the Error table are the Error field, which must be an integer error number, and the Message field, containing the message to display. When you add a record to the Error table, the InstallShield environment automatically adds an entry to the string table; for each language your project supports, you edit the string value in the String Tables view, available inside your project's General Information view.

For an example of using an error custom action, suppose you have created a major upgrade of an existing project, and want to allow the installation to continue only if an earlier product version is detected. You can detect if a major upgrade is taking place by testing if the "action property" defined in an Upgrade-table record contains any value.

First, you create the error action in the Custom Actions and Sequences view. You can schedule the custom action in both the User Interface and Execute sequences after the FindRelatedProducts action with condition Not ISACTIONPROP1. The action's properties might be as follows:

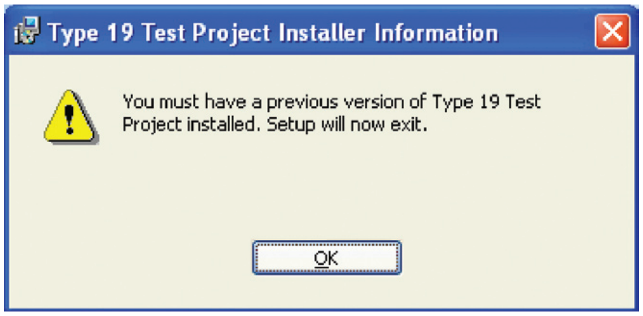


exitIfNewInstall	Custom Action
Error Message	29000
Return Processing	Synchronous (Check exit code)
In-Script Execution	Immediate Execution
Execution Scheduling	Execute only once
Use 64Bit Scripting	No
Install UI Sequence	After FindRelatedProducts
Install UI Condition	Not ISACTIONPROP1
Install Exec Sequence	After FindRelatedProducts
Install Exec Condition	Not ISACTIONPROP1
Advertise Exec Sequence	<Absent from sequence>
Advertise Exec Condition	
Admin UI Sequence	<Absent from sequence>
Admin UI Condition	
Admin Exec Sequence	<Absent from sequence>
Admin Exec Condition	
MSI Type Number	275
Comments	Display error message for a first-time installation.

Next, use the Direct Editor view to create the message in the Error table:

Error: 29000
Message: You must have a previous version of [ProductName] installed. Setup will now exit.

When you run the installation, because no earlier version of the product is detected, the following error message is displayed.



You must have a previous version of Type 19 Test Project installed. Setup will now exit.

Summary

This white paper discusses some of the types of custom actions supported by Windows Installer (MSI). It also highlights how InstallShield from Acresso Software assists you in working with custom actions.

Begin a Free Evaluation of InstallShield
You can download a free trial version of InstallShield from the Acresso Software Web site at:
www.acresso.com/installshield/eval

Want to learn more best practices for building quality installations? Join an InstallShield training class – visit www.acresso.com/training for available classes.

Powering the business of software

Acresso Software Inc.
900 National Parkway, Suite 125
Schaumburg, IL 60173
USA

Schaumburg (Global Headquarters),
Santa Clara, CA:
+1 800-809-5659

United Kingdom (Europe,
Middle East Headquarters):
+44 870-871-1111
+44 870-873-6300

Japan (Asia, Pacific Headquarters):
+81 3-5774-6253

www.acresso.com

© 2008 Acresso Software Inc. and/or InstallShield Co. Inc. All other brand and product names mentioned herein may be the trademarks and registered trademarks of their respective owners.
IS_WP_MSIActionTypes_Oct08