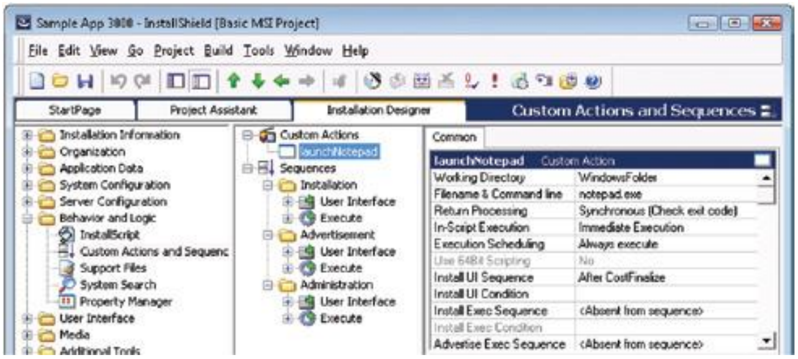
Executable Custom Actions for MSIs

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”.

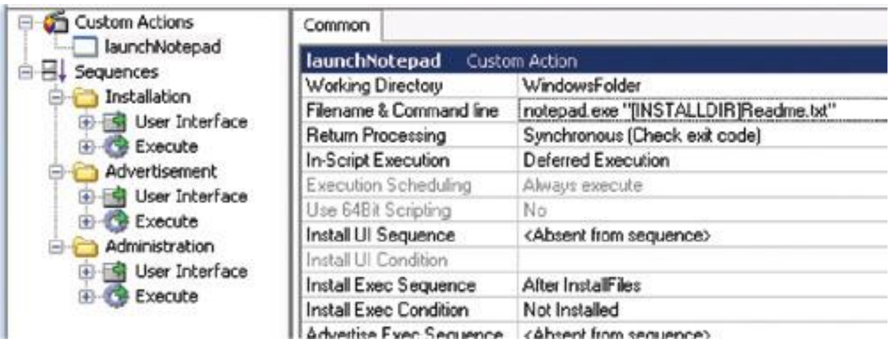
In this case, because Notepad.exe is present on the target system, this custom action can be placed in either the User Interface sequence or Execute sequence (or both), after the CostFinalize action. It is not necessary to specify deferred execution, or for the InstallFiles action to have run first.

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)**.