Utility custom actions

WiX includes a number of utility custom actions in the WixToolset.Util.wixext WiX extension. To use them, add a package reference to WixToolset.Util.wixext in your .wixproj or use `wix extension` and `wix build -ext` at the command line.

WixShellExec custom actions

The WixShellExec family of custom actions lets you launch documents and URL targets using the Windows shell's registered file type associations. A common use is to launch ReadMe files in the user's preferred text editor or Internet URLs in the user's preferred browser. WixShellExec avoids the need to use, for example, `RegistrySearch` to locate the user's default applications. Note that WixShellExecute can only be used as an immediate custom action as it launches an application without waiting for it to close. WixShellExec reads its target from the WixShellExecTarget property, formats it, and then calls ShellExecute with the formatted value. For example:

```
<SetProperty Id="WixShellExecTarget" Value="appwiz.cpl" Before="LaunchArpCustomAction" Sequence="execute"
/>
<CustomAction Id="LaunchArpCustomAction" BinaryRef="Wix4UtilCA_$(sys.BUILDARCHSHORT)"
DllEntry="WixShellExec" Execute="immediate" Return="check" />

<InstallExecuteSequence>
  <Custom Action="LaunchArpCustomAction" Before="AppSearch" />
</InstallExecuteSequence>
```

Launch an embedded file

The WixShellExecBinary custom action is a variant of WixShellExec that extracts a file from the MSI package's Binary table and launches it. The file is named after the Binary row's id, so give the id like a file name with extension. For example:

```
<Binary Id="readme.rtf" SourceFile="readme.rtf" />
<Property Id="WixShellExecBinaryId" Value="readme.rtf" />
<CustomAction Id="LaunchReadmeCustomAction" BinaryRef="Wix4UtilCA_$(sys.BUILDARCHSHORT)"
DllEntry="WixShellExecBinary" Execute="immediate" Return="check" />

<InstallExecuteSequence>
  <Custom Action="LaunchReadmeCustomAction" Before="AppSearch" />
</InstallExecuteSequence>
```

Launch a file without elevation

The WixUnelevatedShellExec custom action lets you launch a process with "normal" user privileges when running with elevated privileges. WixUnelevatedShellExec is useful to ensure that, for example, you don't launch a user's web browser with elevated privileges. (That's not typically a problem with modern browsers, luckily.)

```
<Property Id="WixUnelevatedShellExecTarget" Value="https://wixtoolset.org/" />
<CustomAction Id="LaunchImportantSiteCustomAction" BinaryRef="Wix4UtilCA_$(sys.BUILDARCHSHORT)"
DllEntry="WixUnelevatedShellExec" Execute="immediate" Return="check" />

<InstallExecuteSequence>
  <Custom Action="LaunchImportantSiteCustomAction" Before="AppSearch" />
</InstallExecuteSequence>
```

BroadcastEnvironmentChange and BroadcastSettingChange custom actions

The BroadcastEnvironmentChange and BroadcastSettingChange elements in the Util schema schedule custom actions that send a WM_SETTINGCHANGE message to all top-level windows indicating that settings have changed. BroadcastEnvironmentChange indicates that environment variables have changed. BroadcastSettingChange indicates that unspecified settings have changed.

Other programs can listen for WM_SETTINGCHANGE and update any internal state with the new setting.

Windows Installer itself sends the WM_SETTINGCHANGE message for settings it changes while processing an MSI package but cannot do so for changes a package makes via custom action. Also, Windows Installer does not send WM_SETTINGCHANGE for environment variable changes when a reboot is pending.

CheckRebootRequired custom actions

The CheckRebootRequired element in the Util schema schedules a custom action that schedules a reboot based on a deferred custom action that called the WixToolset.WcaUtil function `WcaDeferredActionRequiresReboot`. The approach is necessary because a deferred custom action cannot directly schedule a reboot.

ExitEarlyWithSuccess custom actions

The ExitEarlyWithSuccess element in the Util schema schedules a custom action that does nothing except return the value ERROR_NO_MORE_ITEMS. This return value causes Windows Installer to skip all remaining actions in the package and return a process exit code that indicates a successful installation.

This custom action is useful in cases where you want setup to exit without actually installing anything, but want it to return success to the calling process. A common scenario where this type of behavior is useful is in an out-of-order installation scenario for an .msi that implements major upgrades. When a user has version 2 of an .msi installed and then attempts to install version 1, this custom action can be used in conjunction with the Upgrade table to detect that version 2 is already installed to cause setup to exit without installing anything and return success. If any applications redistribute version 1 of the .msi, their installation processes will continue to work even if the user has version 2 of the .msi installed on their system.

FailWhenDeferred custom actions

When authoring deferred custom actions (which are custom actions that change the system state) in an MSI, it is necessary to also provide an equivalent set of rollback custom actions to undo the change in system state in case the MSI fails and rolls back. The rollback behavior typically needs to behave differently depending on if the MSI is currently being installed, upgraded, repaired, or uninstalled. This means that both success and failure cases need to be accounted for when coding and testing a set of deferred custom actions to make sure that they are working as expected.

The failure cases are often difficult to simulate by unit testing the custom action code directly because deferred custom action code typically depends on state information provided to it by Windows Installer during an active installation session. As a result, this type of testing usually requires fault injection in order to cause the rollback custom actions to be executed at the proper times during real installation scenarios.

WiX includes a simple deferred custom action to help make it easier to test rollback custom actions in an MSI. The FailWhenDeferred custom action always fails when it is executed, making it easy to inject a failure into your MSI to test your rollback custom actions.

The FailWhenDeferred element in the Util schema schedules the FailWhenDeferred custom action with a condition of `WIXFAILWHENDEFERRED=1`. To inject a failure during installation rollback, for example, use a command line like this:

```
msiexec.exe /i MyProduct.msi /qb /l*vx %temp%\MyProductInstallFailure.log WIXFAILWHENDEFERRED=1
```

If you have scenarios you want to test where a package or bundle takes a while to install, you can write a simple MSI package that includes the WaitForEvent custom actions to simulate this behavior. These custom actions -- available in both immediate and deferred modes -- wait for either of the globally named automatic reset events documented below and will either return ERROR_INSTALL_FAILURE or ERROR_SUCCESS depending on which event you signal.

- `Global\WixWaitForEventFail` - when signaled, the custom actions return ERROR_INSTALL_FAILURE.
- `Global\WixWaitForEventSucceed` - when signaled, the custom actions return ERROR_SUCCESS.

This is especially useful in test cases when you don't want or need to build your entire product and only want small test packages.

The WaitForEvent and WaitForEventDeferred elements in the Util schema schedule the WaitForEvent custom actions.

Edit this page