Home (/) / WiX Toolset v3 Tutorial (/docs/wix/v3/tutorial/) / Events and Actions (/docs/wix/v3/tutorial/events-and-actions/) / Extra Actions

Extra Actions

There are many other standard actions (http://msdn.microsoft.com/en-us/library/windows/desktop/aa372023(v=vs.85).aspx) available but not scheduled by default. `ScheduleReboot,` for instance, will instruct the user to reboot after the installation:

```
<InstallExecuteSequence>
    <ScheduleReboot After='InstallFinalize' />
</InstallExecuteSequence>
```

If the need to reboot depends on a condition (for instance, the operating system the installer is running on), use a condition:

```
<InstallExecuteSequence>
    <ScheduleReboot After='InstallFinalize'>Vers
ion9X</ScheduleReboot>
</InstallExecuteSequence>
```

It's not only these so-called standard actions that you can schedule and re-schedule. There are a couple of custom actions as well (*custom* here means that they don't appear in the standard course of events but you can use them wherever and whenever you like). A very common need is to launch the application you've just installed.

Custom actions need to be mentioned in two places in the source file. First as a child of the `Product` tag (for instance, between the closing `Feature` and the `UI` tag). This `CustomAction` tag will specify what to do. To launch an executable we've just installed, refer to it using the `Id` identifier of the `File` tag specifying the file. You also have to provide a command line, although it can be left empty if not needed:

HeatWave (/docs/heatwave/)

HeatWave Build Tools (/docs/heatwave-build-tools/)

FG-WiX v3 Additional Features (/docs/fg-wix/)

WiX Toolset v4 Tutorial (/docs/wix/tutorial/)

WiX Toolset v3 Tutorial (/docs/wix/v3/tutorial/)

- Getting Started (/docs/wix/v3/tutorial/getting-started/)
- User Interface (/docs/wix/v3/tutorial/user-interface/)

```
<CustomAction Id='LaunchFile' FileKey='FoobarEXE' ExeCommand='' Return='asyncNoWait' />
```

Second, we have to schedule the action the usual way. The link between the action and the scheduling entry is provided by the matching `Id` / `Action` attribute pair. If the execution of the custom action is conditional, we can specify the condition inside the `Custom` tag. We need the condition here so that we only launch the executable when we make an installation but not when we remove the product:

```
<InstallExecuteSequence>
    ...
    <Custom Action='LaunchFile' After='InstallFinalize'>NOT Installed</Custom>
</InstallExecuteSequence>
```

In some cases, we want to start a helper utility we carry along in the installation package but we don't install on the user's machine (for instance, a readme file viewer or a special configuration utility). Instead of the `File,` we refer to the identifier in a `Binary` tag. The scheduling is the same:

```
<CustomAction Id='LaunchFile' BinaryKey='FoobarEXE' ExeCommand='' Return='asyncNoWait' />
```

We can also launch any other executable on the user's machine if we provide its name in a property:

```
<Property Id='NOTEPAD'>Notepad.exe</Property>
<CustomAction Id='LaunchFile' Property='NOTEPAD'
ExeCommand='[SourceDir]Readme.txt' Return='async
NoWait' />
```

Custom actions can also specify how their return will be handled, using a `Return` attribute. Possible values are: *check* will wait for the custom action to finish and check its return value, *ignore* will wait for the action but the return value will be ignored, *asyncWait* will run asynchronously but the installer will wait for the return value at the end of the scheduling sequence and *asyncNoWait* will simply launch the action and then leave it alone, the action might still be running after the installer finishes. This last value is the one we use when we want to launch an application or a readme file after the installation.

If we encounter an error the normal machinery can't report, we can display an error message and terminate the installation. The `Error` attribute can contain either the actual message text or the `Id` identifier of an `Error` tag:

```
<CustomAction Id='AbortError' Error='Cannot solv
e this riddle. Giving up.' />
```

There is no direct way to assign the value of a property to another one but a custom action can bridge this gap. The `Value` attribute can be a formatted string, thus we can perform some string manipulation, too (note that path references always have their trailing backslash automatically, there is no need to add an extra one):

```
<CustomAction Id='PropertyAssign' Property='PathProperty' Value='[INSTALLDIR][FilenameProperty].[ExtensionProperty]' />
```

A directory can also be set to a similarly formatted string representing a path:

```
<CustomAction Id='PropertyAssign' Directory='INSTALLDIR' Value='[TARGETDIR]\Program Files\Acme\Foobar 1.0\bin' />
```

‹ Queueing Up (/docs/wix/v3/tutorial/events-and-actions/queueing-up/)

up (/docs/wix/v3/tutorial/events-and-actions/)

What's Not in the Book › (/docs/wix/v3/tutorial/events-and-actions/whats-not-in-the-book/)

PRODUCTS (/products/)

- HeatWave (/docs/heatwave/)
- HeatWave Build Tools (/products/heatwave-build-tools/)
- FireGiant WiX v3 (/products/fg-wix/)

SERVICES (/services/)

- Support Programs (/services/support-programs/)
- Code Review (/services/code-review/)
- Phone Consultation (/services/phone-consultation/)
- Custom Development (/services/custom-development/)

DOCUMENTATION (/docs/)

- HeatWave (/docs/heatwave/)
- WiX v4 Tutorial (/docs/wix/tutorial/)
- WiX v3 Tutorial (/docs/wix/v3/tutorial/)
- FireGiant WiX v3 (/docs/fg-wix/)

- About Us (/about-firegiant/)
- Why WiX (/why-wix/)
- Leadership Team (/about-firegiant/leadership-team/)
- Privacy Policy (/about-firegiant/privacy-policy/)