What's Not in the Book

For very specialized actions that the Windows Installer provides no solution for (eg. checking the validity and integrity of the registration key entered by the user), we can use yet another type of custom action: a DLL we write. For our example, we use a simplistic approach: we will accept the user key if its first digit is '1'.

The following source can be directly compiled with Visual C++ but minimal modifications, if any, will be required to compile it with a different compiler. The `msi.h` and `msiquery.h` header files can be acquired from the MSI SDK. You also have to link against `msi.lib`.

```
#include <windows.h>
#include <msi.h>
#include <msiquery.h>

#pragma comment(linker, "/EXPORT:CheckPID=_CheckPID@4")

extern "C" UINT __stdcall CheckPID (MSIHANDLE hInstall)
{
    char Pid[MAX_PATH];
    DWORD PidLen = MAX_PATH;

    MsiGetProperty(hInstall, "PIDKEY", Pid, &PidLen);
    MsiSetProperty(hInstall, "PIDACCEPTED", Pid[0] == '1' ? "1" : "0");
    return ERROR_SUCCESS;
}
```

To use this DLL, add the following lines to the appropriate places (now, nearing the end of the third lesson, you might be able to do this yourself but if you want to cheat, download SampleCA. (/system/files/samples/SampleCA.zip)

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/)

```
<Condition Message='This installation can only run in full UI mode.'>
    <![CDATA[UILevel = 5]]>
</Condition>

<CustomAction Id='CheckingPID' BinaryKey='CheckPID' DllEntry='CheckPID' />
<CustomAction Id='RefusePID' Error='Invalid key. Installation aborted.' />

<InstallExecuteSequence>
    <Custom Action='CheckingPID' After='CostFinalize' />
    <Custom Action='RefusePID' After='CheckingPID'>PIDACCEPTED = "0" AND NOT Installed</Custom>
</InstallExecuteSequence>

<Binary Id='CheckPID' SourceFile='CheckPID.dll' />
```

To summarize: first, we won't allow the installer to run with reduced or no UI because the user can't enter a registration key in those cases. The reason for the ugly CDATA wrapper is that XML attributes special meaning to some characters, most notably `<` and `>`. Wherever they appear in a different context, meaning *less-than* or *greater-than*, we have to escape them by wrapping the whole expression into a CDATA. Although this actual case could get away without it because it only checks for equality, it is a good habit to learn to wrap all similar conditional values just in case we need to modify them later, introducing such XML conflicts.

Then, we have a custom action named *CheckingPID* running after *CostFinalize*, when we instruct the installer to start the actual installation after having specified which features we need and where we want to install. This action will call the *CheckPID* function in our `CheckPID.dll`, bundled with the installer. The DLL sets the `PIDACCEPTED` property to either 1 or 0, according to its decision on the validity of the user key entered and stored into the `PIDKEY` property by the control involved. Note that using properties (with all uppercase names, otherwise Windows Installer will not treat them as public properties) is the only way to pass arguments to and from the custom action.

We have a second custom action named *RefusePID*, scheduled to run after the previous action. This is a conditional custom action, only run if the returned `PIDACCEPTED` property is found to be zero. In this case, the custom action will display an error message and abort the installation. But we will only be interested in this value during the installation, not while we're uninstalling the product.

To get an understanding of how these actions are called and how they relate to each other, we might run the installer with verbose logging. As it will be verbose, using a text editor and searching for our property and custom action names ("PID" will do just fine) might help locate what's really happening.

```
msiexec /i SampleCA.msi /l*v SampleCA.log
```

If the DLL we need to call has been installed rather than just included in the package, we can use:

```
<CustomAction Id='CheckingPID' FileKey='HelperDLL' DllEntry='CheckPID' />
```

‹ Extra Actions (/docs/wix/v3/tutorial/events-and-actions/extra-actions/)
up (/docs/wix/v3/tutorial/events-and-actions/)
Control Your Controls › (/docs/wix/v3/tutorial/events-and-actions/control-your-controls/)

CONTACT US (/contact-us/) | BLOG (/blog/)

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/)