Frequently-asked questions about upgrading from WiX v3 to WiX v4

> **INFO**
>
> TODO: WiX v4 documentation is under development.

Converting packages

Converting the `Component/@Win64` attribute

In WiX v3 authoring out in the wild, it's common to find code similar to the following:

```
<?if $(var.Platform) = "x64"?>
  <?define IsWin64 = yes ?>
  <?define ProgramFilesFolder = ProgramFiles64Folder ?>
<?elseif $(var.Platform) = "x86" ?>
  <?define IsWin64 = no ?>
  <?define ProgramFilesFolder = ProgramFilesFolder ?>
<?elseif $(var.Platform) ~= "Arm64" ?>
  <?define IsWin64 = yes ?>
  <?define ProgramFilesFolder = ProgramFiles64Folder ?>
```

```
<?endif?>
```

and then repeated use of the `IsWin64` preprocessor variable:

```
<Component ... Win64="$(var.IsWin64)">
```

WiX v3 didn't require repeated use of the `Win64` attribute. First noted in the historical record back in 2010, WiX automatically marks components according to the architecture specified when compiling them. WiX v4 continues that trend and replaces the `Win64` attribute with the clarifying `Bitness` attribute to make it possible to override the default.

WiX v3 didn't have a solution for the root Program Files folder id. WiX v4 introduces the new "standard" directory `ProgramFiles6432Folder` to solve that problem. `ProgramFiles6432Folder` automatically resolves to `ProgramFilesFolder` for an x86 package and `ProgramFiles64Folder` for an x64 or Arm64 package.

For example:

```
<Fragment>
  <StandardDirectory Id="ProgramFiles6432Folder">
    <Directory Id="CompanyFolder" Name="!(bind.Property.Manufacturer)">
      <Directory Id="INSTALLFOLDER" Name="!(bind.Property.ProductName)" />
    </Directory>
  </StandardDirectory>
</Fragment>
```

All this logic works off the platform you specify at build time, with with the `-arch` switch at the wix.exe command line or the equivalent MSBuild property `InstallerPlatform`.

WiX v3 extensions in WiX v4

This table maps WiX v3 extensions to their WiX v4 equivalents:

WiX v3 Extension	WiX v4 Extension	Documentation
WixBalExtension	WixToolset.Bal.wixext	Bal schema
WixComPlusExtension	WixToolset.ComPlus.wixext	Complus schema
WixDependencyExtension	WixToolset.Dependency.wixext	Dependency schema
WixDifxAppExtension	WixToolset.DifxApp.wixext	Difxapp schema
WixDirectXExtension	WixToolset.DirectX.wixext	Directx schema
WixFirewallExtension	WixToolset.Firewall.wixext	Firewall schema
WixGamingExtension	n/a	Removed because the Windows feature is obsolete.
WixHttpExtension	WixToolset.Http.wixext	Http schema
WixIIsExtension	WixToolset.Iis.wixext	Iis schema
WixLuxExtension	n/a	Lux was not brought forward to WiX v4.
WixMsmqExtension	WixToolset.Msmq.wixext	Msmq schema

WiX v3 Extension	WiX v4 Extension	Documentation
WixNetFxExtension	WixToolset.Netfx.wixext	Netfx schema
WixPSExtension	WixToolset.PowerShell.wixext	Powershell schema
WixSqlExtension	WixToolset.Sql.wixext	Sql schema
WixTagExtension	n/a	Software tag functionality now is built into the core toolset.
WixUIExtension	WixToolset.UI.wixext	UI schema
WixUtilExtension	WixToolset.Util.wixext	Util schema
WixVSExtension	WixToolset.VisualStudio.wixext	Vs schema

Converting custom action ids

In WiX v4's extensions, custom action ids were renamed from their WiX v3 origins for two reasons:

1. To support WiX v4's platform-specific custom actions for all three platforms that WiX v4 supports: x86, x64, and Arm64.
2. To avoid conflicts when building a package with WiX v4 that merges a merge module that was built with WiX v3 and uses WiX v3 extension custom actions. (Yes, this is an annoying edge case.)

WiX v4 meets these requirements by adding a prefix that lets us version custom actions when they make changes that are incompatible with prior versions. Today, the prefix is `Wix4` (because the WiX team is full of wildly imaginative people). In WiX v5, if a particular extension made a change that was incompatible with the custom tables produced by WiX v4, that extension would

adopt a new prefix, perhaps something like `Wix5`. But fixes and changes that are backward compatible would not require changing the prefix.

The suffix distinguishes platforms:

Platform	Suffix
x86	`_X86`
x64	`_X64`
Arm64	`_A64`

For example, the `QueryNativeMachine` custom action in WiX v3 is, in WiX v4, named:

- `Wix4QueryNativeMachine_X86`
- `Wix4QueryNativeMachine_X64`
- `Wix4QueryNativeMachine_A64`

Some custom actions already have a `Wix` prefix. For those, the new prefix replaces it. So, for example, the `WixFailWhenDeferred` custom action in WiX v3 is now named:

- `Wix4FailWhenDeferred_X86`
- `Wix4FailWhenDeferred_X64`
- `Wix4FailWhenDeferred_A64`

Generally, this change is invisible because the extension handles the prefix and suffix for you. That was also true in WiX v3, but several custom actions, like `WixFailWhenDeferred` did not have a custom element in the extension. Usually, this was because there was no additional information required. In WiX v4, there's always at least one bit of additional information required: the platform the package is being built for. So WiX v4 includes custom elements like FailWhenDeferred to include custom actions in your package so you don't have to worry about prefixes and suffixes.

Other references to WiX custom actions must use the full id, including prefix and suffix.

Referencing the standard WixUI dialog sets

In WiX v3, you referenced a standard WixUI dialog set using the `UIRef` element:

```
<UIRef Id="WixUI_Mondo" />
```

WiX v4's addition of platform-specific custom actions means a compiler extension is used:

1. Add a reference to the WixToolset.UI.wixext WiX extension.
2. Add the WixToolset.UI.wixext WiX extension namespace to your WiX authoring.
3. Add the `WixUI` element to your WiX authoring.

For example:

```
<Wix xmlns="http://wixtoolset.org/schemas/v4/wxs"
  xmlns:ui="http://wixtoolset.org/schemas/v4/wxs/ui">

  <Package ...>
      ...
```

```
      <ui:WixUI
        Id="WixUI_InstallDir"
        InstallDirectory="INSTALLFOLDER"
        />
  </Package>
</Wix>
```

Converting custom WixUI dialog sets

Because of WiX v4's support for platform-specific custom actions, customizing WixUI dialog sets, especially when adding and removing dialogs, requires some care. The WixUI documentation describes what to do when creating a new custom dialog set. You'll want to make the same kind of change when converting a custom dialog set you created using WiX v3 to WiX v4. The key point is to isolate any `DoAction` control events that call custom actions to create platform-specific variants. WixUI itself does this using a preprocessor `?foreach?` processing instruction to create three fragments, one each for x86, x64, and Arm64 platforms. Each of those fragments references the platform-neutral `UI`. You can see the WixUI definitions on GitHub. Here's what a customized dialog set based on WixUI_InstallDir looks like:

```
<?foreach WIXUIARCH in X86;X64;A64 ?>
<Fragment>
    <UI Id="InstallDir_SpecialDlg_$(WIXUIARCH)">
        <Publish
          Dialog="LicenseAgreementDlg"
          Control="Print"
          Event="DoAction"
          Value="WixUIPrintEula_$(WIXUIARCH)"
          />
        <Publish
          Dialog="BrowseDlg"
          Control="OK"
```

```
            Event="DoAction"
            Value="WixUIValidatePath_$(WIXUIARCH)"
            Order="3"
            Condition="NOT WIXUI_DONTVALIDATEPATH"
            />
        <Publish
            Dialog="InstallDirDlg"
            Control="Next"
            Event="DoAction"
            Value="WixUIValidatePath_$(WIXUIARCH)"
            Order="2"
            Condition="NOT WIXUI_DONTVALIDATEPATH"
            />
    </UI>

    <UIRef Id="InstallDir_SpecialDlg" />
</Fragment>
<?endforeach?>
```

You can see the authoring and test code for this customized dialog set on GitHub.

When you use the `WixUI` element to reference a WixUI dialog set or a customized dialog set derived from WixUI, it adds a reference to the platform-specific `UI` for the platform of the package being built. The platform-specific `UI` then adds a reference to the platform-neutral `UI`.

Converting bundles

Bundles and the Burn engine were originally released in WiX v3.6. In general they worked reasonably well, especially for the use case that they were built for: the Visual Studio installer. Over time, various edge cases became more visible than others. A lot of real

world feedback was received on design decisions that were essentially based on a coin flip. v4 Bundles include a lot of rather small behavioral changes, along some new features, that add up to an installer that will just work for a wider audience. In a real sense, v4 includes Burn v2. This page is intended to cover all behavioral changes in the Burn engine from v3 to v4.

The v3 Variable's `string` type is now called `formatted`

In v3, the engine would almost always format a variable's value before using it. For example, if the variable had the string value `[WixBundleName] Installation` and the value of the `WixBundleName` was "My Bundle" then the formatted value was "My Bundle Installation". This caused problems when the value of a variable contained a path with square brackets, like "C:\[WixBundleName]". In this case, the formatted value needs to be "C:\[WixBundleName]" instead of "C:\My Bundle". In v4, there is a new literal string type for this use case.

In v4, the `formatted` type of the `Variable` element must be used to get the v3 behavior. To specify a literal variable, use the `string` type.

In v4, all built-in searches will save the value as a literal variable. wixstdba has functionality where it will let the user edit the value of a variable with a textbox. When populating the textbox, it formats the variable first. When saving the value from the textbox to the variable, it saves it as a literal variable.

One place that the v3 engine didn't format the variable when using it was when evaluating a condition. In v4, it does format the variable. For example in v3, if the value of `ParentVar` is "[NestedVar]" and the value of `NestedVar` is "ABC" then the condition `ParentVar = "ABC"` is false. In v4, that condition is true if `ParentVar` is a formatted string variable but false if `ParentVar` is a literal string variable.

Another place that the v3 engine didn't format the variable was in the `Log` element's `Prefix` attribute. v4 will format it, and if it is an absolute path then it will use that path (3816).

The condition operator `~<>` works differently with empty or null strings

See issue 5372.

Backwards compatibility guarantees for Bootstrapper Applications

In v3, the BA must have been built against the same library (balutil for native BA's and BootstrapperCore.dll for managed BA's) that the bundle was built with. This was build-time backwards compatibility since v3.x was never allowed to make breaking changes to the interfaces. In v4, the guarantee has been swapped for binary compatibility. If a BA is built against the v4.0 library, then it can be used for any bundle built by any version of WiX in the range [4.0, 5.0). However, if the BA is built against v4.0.1 then some changes to the source code might be required. For example, a new parameter was added to an existing event.

As a part of these changes, all events now return an HRESULT at the lowest level. If an event needs the BA to return something like a desired action, there is a separate parameter for that with its own enum.

Backwards compatibility guarantees for BAFunctions and specifying its dll at build time

v3 wixstdba supported an extensibility mechanism called BAFunctions where the user provided a native dll that was called for a handful of events. This interface works the same as the BA interface so there is binary compatibility but not source compatibility.

In v3, the dll had to be named `BAFunctions.dll`. In v4, the BAFunctions dll must be specified at build time with the `bal:BAFunctions` attribute.

x64 and ARM64 bundles

In v3, only x86 bundles were available which meant all BA's had to be x86. In v4, the bundle can be x86, x64, or ARM64. All of the

BA's dlls must be the same architecture as the bundle, or be a managed AnyCPU dll.

wixstdba behavior changes

- 5267 makes wixstdba skip checking `bal:Condition` during layout.
- 5927 makes wixstdba use `::DefDialogProc` instead of `::DefWindowProc` in the base case.
- 6537 makes wixstdba skip straight to the Success page if a newer version is installed and `bal:SuppressDowngradeError="yes"`.
- The Files in Use dialog is now shown as a Task Dialog instead of a page in the theme (PR).
- 6856 makes wixstdba (and other users of balretry) to not attempt to retry an MSI package that fails with error 1606.
- 5499 make wixstdba request elevation if the bundle requires a restart and the current process doesn't have permissions to do so.
- If not showing full UI, wixstdba will always use the action specified on the command line.

Upgrading v3 Managed Bootstrapper Applications

v4 added the ability to use .NET Core managed BA's but that is not covered here since v3 only supported .NET Framework BA's. Note that Microsoft chose a confusing naming convention for newer versions of .NET Core. .NET 5+ is a completely different runtime from .NET Framework. To use a BA written in .NET 6+, use `bal:WixDotNetCoreBootstrapperApplicationHost` instead of `bal:WixManagedBootstrapperApplicationHost`. .NET 5 and earlier versions are not supported because they were already at end of life when v4 shipped.

Even though the WixBA is not currently shipping, it is still available at https://github.com/wixtoolset/wix4/tree/develop/src/test/burn/WixToolset.WixBA. You can basically see a step-by-step example of upgrading a v3 MBA by looking at the history of that folder and `WixToolset.WixBA.csproj`. Another folder that might be useful is https://github.com/wixtoolset/wix4/tree/develop/src/test/burn/TestData/TestBA.

In v3, MBAs needed to be built against the same version of `BootstrapperCore.dll` as the bundle. They didn't need to include it as a payload because the Bal WiX extension automatically added it. However, they did need to provide a `BootstrapperCore.config` file to inform the MBA host which .NET Framework runtime it supported and which DLL contained the MBA.

In v4, the config file is named `WixToolset.Mba.Host.config`. The MBA must use the WixToolset.Mba.Core nuget package, a sample config file is included in the package. The Bal WiX extension does not add WixToolset.Mba.Core.dll or mbanative.dll so both must be added along with all the other dependencies of the MBA. The MBA project should be an SDK-style project and built for a specific RID (e.g. win-x86).

The root namespace changed from `Microsoft.Tools.WindowsInstallerXml.Bootstrapper` to `WixToolset.Mba.Core`.

In v3, the BootstrapperApplication was designated by the `BootstrapperApplication` attribute. In v4, the BA assembly must have a `BootstrapperApplicationFactory` attribute that designates the `IBootstrapperApplicationFactory`.

Specifying the Managed BA prerequisite

In v3, the prerequisite for the MBA was specified with the magic WixVariable `WixMbaPrereqPackageId`. Any further prerequisites could be specified with the `bal:PrereqSupportPackage` attribute.

In v4, the only way to specify prereqs for the MBA is through the `bal:PrereqPackage` attribute. Because this attribute can't be specified on a `PackageGroupRef`, each of the NetFx package groups have a different version that is intended to be used with an MBA. For example, use `NetFx481WebAsPrereq` for an MBA instead of `NetFx481Web`.

Prereq BA changes

When running in full UI and the command line option to prohibit restarts was provided and a restart is required, show the Success screen (3957) instead of a message box.

To implement 4718, the `bal:AlwaysInstallPrereqs` attribute was added to the MBA host elements. When "true", the host will always run the prereq BA before trying to start the .NET (Framework) runtime (the UI will not show if all prereqs are already installed). Otherwise, the host will only show the prereq BA if it couldn't load the MBA. Either way, the host will only attempt to run the prereq BA once. If the host can't load the MBA after letting the prereq BA install the prereqs, then it will show an error page with the prereq BA and then quit.

Removal of DisplayInternalUI

In v3, a bundle could choose to let an `MsiPackage` show its internal UI during install with the `DisplayInternalUI` attribute. Many people requested the ability to show the internal UI during other operations as well as the ability to choose at runtime whether to show it. So in v4, the engine gives the BA full control over the UI level of the MSI and the `DisplayInternalUI` attribute was removed.

There are two built-in BA's that support showing the MSI's internal UI. The first one is `bal:WixInternalUIBootstrapperApplication`, a brand new built-in BA in v4. (TODO: link to documentation on new BA)

The second one is wixstdba. Use the `bal:DisplayInternalUICondition` attribute to control when the MSI's UI should be shown. To get the same behavior as v3, use `bal:DisplayInternalUICondition="WixBundleAction = 6"`.

Upgrading custom wixstdba themes

There were so many breaking changes done to the UI library (thmutil), the XML schema, and the built-in themes that it wasn't worth

time trying to build a tool to convert v3 themes into v4. Look at the built-in themes and rebuild your theme from one of them: https://github.com/wixtoolset/wix4/tree/develop/src/ext/Bal/wixstdba/Resources.

(TODO: link to documentation about all the cool new features)

Here are some subtle changes:

- 4906 makes the Window's height and width now refer to the height and width of the window's client area instead of the window itself.
- 5843 upgraded the RichEdit control from the ancient version that was used in v3.
- 5250 fixed a bug in custom ProgressBars where the right side was drawn with the left side's pixel.

High DPI changes

v3 bundles are always DPI unaware. In v4, the default DPI awareness is Per-Monitor V2. See the `BootstrapperApplicationDll` element's `DpiAwareness` attribute for more information.

This may cause changes for how images are shown in a custom wixstdba theme, or the bundle's splash screen. This may also cause problems if a bundle claims it supports a level of DPI awareness that the BA's UI framework doesn't support.

balutil and WixToolset.Mba.Core breaking changes

- There were several enums and events declared but never used by the engine so they were deleted.
- `OnApplyNumberOfPhases` was merged into `OnApplyBegin`.

- All events that have a `Begin` event must have a corresponding `Complete` event and vice versa. At runtime, if the engine sends the BA a `Begin` event then it must send the corresponding `Complete` event and vice versa.
- 5980 requires the window handle passed to `Apply` to be a valid window handle to avoid hangs when executing Windows Installer packages.
- `OnDetectTargetMsiPackage` and `OnPlanTargetMsiPackage` were renamed to `OnDetectPatchTargetPackage` and `OnPlanPatchTargetPackage`.
- Whether restarts are allowed is no longer provided by the engine. Use `BalInfoParseCommandLine` or the managed `BootstrapperCommand.ParseCommandLine` to get this information.
- The `OnSystemShutdown` event was removed. The BA has to integrate with the OS if it wants to receive shutdown notifications because the notifications are time sensitive and the engine needs to respond promptly to avoid getting terminated.
- The source is now provided in `OnExecuteFilesInUse` and the engine passes back the return value from the BA without modifying it.

Changes to MsiPackage detection

- 3643 changes the detection of an MSI package where it included a row in the Upgrade table for a different upgrade code than itself and the row was for detection only. In v3, this would be detected as a downgrade. In v4, it is treated as a related MSI package but will not be detected as a downgrade.
- 6535 makes the engine not detect minor upgrades as present when only the base package is installed.

Changes to related bundle detection

- Related bundles that were registered but not cached were ignored by the engine. In v4, the BA is informed of all registered related bundles and whether they are cached during detect. Uncached related bundles are still never planned.

- The engine no longer provides the related operation during detect because it depended on the planned overall action which hasn't been decided yet. If the BootstrapperApplication wants to know whether the current bundle would downgrade a related bundle, it must compare the versions itself.

Changes to package caching

In v3, if a bundle's package is already installed then the engine would never try to cache it. This led to issues when the bundle later wanted to repair the package since it wasn't available in the cache (5125). In v4, the bundle will always attempt to cache a package if it is performing the Cache/Install overall action and the package was requested to be present even if it was already installed.

Between this new behavior and the confusion around why the engine was caching packages when the cache was set to "no", the cache types were renamed from "no", "yes", "always" to "remove", "keep", "force".

Some package types like `MsiPackage` don't require the source to be uninstalled. However, the source would be required for rollback. The engine will now attempt to always cache packages for uninstall, but the cache action is considered "non-vital" for packages that can uninstall without source.

Timing changes for detection

In v3 (and v4), the typical execution of a bundle goes like this: load the BA, Detect, Plan, Apply, close. However, there are times where a BA will want to restart the cycle so it's important that all of the internal state in the engine gets reset appropriately. In addition, one of the design goals was that only Detect would be querying for the state of the machine. There were multiple places where code needed to be moved to make this happen:

- 4392 makes the engine always set WixBundleInstalled during Detect.

- The information required for dependency checking is gathered from the machine during Detect instead of Plan.
- `WixBundleAction` is not set until Plan. `WixBundleCommandLineAction` can be used as a replacement though it is not recommended to rely on something that hasn't been decided yet.
- The built-in variable `RebootPending` is also reset everytime during Detect.

Plan changes

- 4539 makes sure related bundles aren't touched during the overall action of Cache.
- Plan requires a successful Detect.
- All the requested states are gathered from the BA before building the plan. This means that the planned actions for the package are given to the BA in the new `OnPlannedPackage` instead of `OnPlanPackageComplete`.
- The BA chooses whether to ignore forward compatible bundles during Plan instead of during Detect.
- In v3, the order of related bundles was determined by the order that they were enumerated from the registry. In v4, they are sorted by relation type (Detect, AddOn, Patch, Dependent, Upgrade) and then by version (older bundles first).
- For the request state `BOOTSTRAPPER_REQUEST_STATE_FORCE_ABSENT`, the engine will plan to uninstall the package even if it wasn't detected as present.
- In v3, the engine made some assumptions when planning dependency operations. In v4, all dependency operations are planned individually so that they are only executed when they are needed and can be rolled back properly. 6510 is an example of incorrect behavior in v3.
- If any related bundles are planned as downgrade, then the rest of the planning is skipped. However, downgrade related

bundles are ignored when the bundle was run from an upgrade related bundle or the bundle is being uninstalled.

- 7147 makes the engine not uninstall superseded MSI packages.

Execute changes

- 6195 sends the desired working directory to `::CreateProcess` instead of changing the current directory when executing ExePackages.
- Apply requires a successful Plan.
- After a plan is executed, the BA must perform a new Detect and Plan before performing Apply again.
- 6668 removes the bundle specific registry key that indicated that it required a reboot.
- In v3, the engine assumed that all ExePackages that used the Burn protocol were bundles. In v4, to get the same behavior as v3 you must add `Bundle="yes"` and can remove `Protocol="burn"`. It also automatically add "-norestart" to the command line.
- The bundle's dependency provider is only removed when the bundle's registration is also removed.
- 4039 changes when the install size in updated in ARP.
- The engine now blocks critical shutdown during Apply in both the elevated process and the unelevated process. When a critical shutdown request is received, it assumes that the currently executing package triggered the restart. It responds so that the OS gives it as much time as possible to let the current package complete and gracefully close so that it can pick back up where it left off after the reboot. v3 had attempted to complete the chain even though the elevated process shutdown.
- All paths passed on the command line are resolved when the parsing the command line at startup. For example, in v3 if the layout path was specified on the command line as a relative path and the BA changed the process's current directory then the

target path would have been based off of the new current directory.

- The SxS manifest for the bundle exe specifies that it handles long paths.
- The exit code for `ExePackage`s were not always correctly translated due to restart related codes.
- 6762 makes the engine aware of the failure restart codes and will return those when appropriate (`ERROR_FAIL_REBOOT_REQUIRED` and `ERROR_FAIL_REBOOT_INITIATED`).
- The engine now uses the elevated process to restart the machine if it is available.
- 3777 makes the engine compare variable names without case sensitivity when masking variable values from the command line.

Active engine changes

While the engine is running a phase like Apply, the engine is "active" and the BA is restricted from calling some functions like `IBootstrapperEngine::SetLocalSource`. The timing of when the engine starts being "active" and finishes being "active" has subtly changed but shouldn't be noticable. This restriction was added to `IBootstrapperEngine::SetUpdate`. This restriction was lifted in v3 during certain calls to `IBootstrapperApplication` and was lifted in some more in v4 like the places where a BA would call `SetUpdate`.

The engine now gets activated during `Elevate` and `Quit`.

All BA requests received after it starts to process the `Quit` are rejected.

Version handling

In v3, all versions in the engine were represented as Major.Minor.Patch.Revision in a 64-bit value, where the max of each

component was 65535. In v4, all versions are represented as strings. When the engine needs to compare strings, it uses a WiX version of SemVer 2.0.

util:ProductSearch will no longer ignore products that have versions that can't be parsed to the 64-bit representation.

As requested in 4808, Burn will now upgrade a bundle if the existing version is the same as the installing bundle instead of installing them side by side.

Hidden Variables

Hidden variables are only to keep the engine from logging its value.

Hidden Variables are no longer encrypted since they would always have to be decrypted and sent to a separate process to be used with a package. If encryption is desired, then it's the BA's responsibility for encrypting it before setting the variable and decrypting it when it needs to be used.

A Variable can no longer be both Hidden and Persisted. Persisted variable values were always stored in cleartext in v3.

util:RegistrySearch changes

5355 changes the behavior to not clear the variable when the target key or value is missing.

Registration changes

As reported in multiple issues like 4822, v3 bundles would stay registered in certain cases where everyone thinks it shouldn't. The registration state of the bundle should be more intuitive now, though the exact rules are too complex to document here.

When the bundle unregisters, it will do its best to remove all dependency registrations that it could have added.

When the bundle performs Cache, Install, Uninstall, Modify, or Repair then it will ensure the bundle is registered and cached at the beginning of Apply regardless of whether it was already registered or cached (5702).

To help users avoid using registry cleaners when a bundle's registration is "stuck", a couple of new command-line options were added.

- -burn.ignoredependencies=ALL means that the engine will skip dependency checks for packages and related bundles that it is going to uninstall.
- /unsafeuninstall means that the engine will always unregister itself at end of Apply.

Cache changes

In v3, the engine attempted to detect individual payloads so that it could plan them individually as well. This caused problems because it didn't perform full verification during Detect and didn't give enough progress to the BA during Apply. In v4, the package is considered to be cached if any of its payloads exist in the package cache. This information is needed for the rules around when the bundle should keep its registration, and never used for whether the package should be cached during Apply. The plan now only includes cache actions for packages and not payloads or containers.

The engine now does full verification for the package every time during Apply if it planned to cache it. The BA is always informed of a payload's progress for acquisition (downloading or copying to the temp cache folder), staging (moving or copying the file from the temp cache folder to the package cache folder), and verification (verifying either the Authenticode signature or the hash and file size).

For each individual payload in a package that is being cached, the engine will first attempt to verify the file in the package cache. If it fails verification then it will be deleted from there and then will attempt to acquire it. The engine will try to acquire it from the following sources in ranked order:

1. If included in a container, the temp cache folder where it was extracted from the container.
2. If included in a container, the container.
3. Local locations on the machine.
4. Download url.

`OnCacheAcquireResolving` replaces `OnResolveSource`. This allows the BA to override the preferred order above. The BA can make the engine skip its resolve algorithm altogether by providing the source in `OnCacheAcquireBegin`. The BA can also provide the source in `OnCacheAcquireResolving` or `OnCacheAcquireComplete` though it has to request a retry in that case.

Authenticode verification is only allowed when the user provided the certificate information in the bundle's source code. Gathering the certificate information during the build is no longer supported. There are some payloads that must have the source file available at build time, like the .msi for `MsiPackage`. It is impossible to use Authenticode verification for these payloads. Authenticode verification with the `Catalog` is no longer supported.

If the engine has a download URL for a payload and it can't find it locally, then it will recommend that it be downloaded. In v3, the BA always had to request the download but in v4 the BA will now have to take action if it doesn't want it downloaded.

During layout, all caching operations are performed in the user-level process and not the elevated process.

When `WixBundleOriginalSource` is set and the file at that location exists with the correct size, prefer that location for attached containers (5586).

Rollback changes

Error in non-vital rollback boundary

6309 makes it work as documented - if a package fails inside of a non-vital rollback boundary then the chain should continue

successfully at the next rollback boundary. In v3, the rest of the packages in the rollback boundary were executed.

Block rollback of packages that have dependents

With custom provider keys, it's possible for a bundle's package to have dependents before it's installed. In v4, the bundle will not rollback the package in this case.

Reinstall upgrade related bundles

3421 - when a bundle fails installing and it has upgrade related bundles, it will run those bundles with the install action during rollback.

Clean room path for elevated processes

The clean room path for elevated processes now respects the system level `TMP`/`TEMP` environment variable instead of using the hardcoded Temp folder in the Windows directory. On newer versions of Windows, if running as SYSTEM it uses C:\Windows\SystemTemp.

This can be overriden on the command line or through policy specified in the registry (5856).

Mixing bundle elements with non-bundle elements

A common mistake in v3 was to try to use MSI elements like the NetFx detection properties in a bundle. In v4, this will generate warnings. For example:

> warning WIX1150: The binder doesn't know how to place the following symbol into the output: SymbolName: 'CustomAction', Id: 'Wix4NetFxScheduleNativeImage_X64'

Unfortunately the changes that were necessary to make these warnings less cryptic were declined so it might be difficult to figure

out exactly what is causing this error in your code.

Recommending or Requiring DetectCondition

The engine always knows how to install and uninstall Windows Installer packages, but needs help for uninstalling ExePackages. v3 had some edge cases where it would try to uninstall an ExePackage even though the user hadn't specified any uninstall arguments or how to detect that it was installed. There are now checks at build time to make sure that an ExePackage contains a sane combination of the `DetectCondition`, `UninstallArguments`, and `Permanent` attributes.

In v3, specifying an empty string for `UninstallCommand` was the same as not specifying it. In v4, if the uninstall command line is empty then `UninstallArguements` must be specified with an empty string.

MsuPackages are no longer uninstallable

See 6749.

Dropped support for XP

Bundles are supported on Vista SP2, Server 2008 SP2, or newer.

Dropped support for SHA1

Bundles use the SHA512 hash algorithm instead of SHA1 when verifying file hashes.

Remove BITS support

The BITS protocol is no longer supported in the engine.

Change to the default for the `Visible` attribute on `MsiPackage`

For an `MsiPackage` with `Permanent="yes"`, the default for `Visible` is now "yes" instead of "no".

More accurate parsing of Bundle elements that contained file information

The v3 compiler code for parsing `Payload` elements was reused for other elements with file information, like `BootstrapperApplication`. This means that it sometimes parsed attributes that shouldn't have been allowed on those other elements.

Build time errors to prevent runtime errors

In v3, bundles could be built in such a way that the engine would work properly at runtime. There are some scenarios that are now build errors to prevent these runtime errors:

- Each package in the bundle has a cache id, which needs to be unique.
- Each payload in the bundle has a relative file path that is used when the payload gets cached or extracted. In v4, collisions are detected.
- Payloads must either be for the BootstrapperApplication or for a package/external payload.

Build time errors to catch likely mistakes

- A package can't be assigned to multiple containers because the engine only supports one.
- All packages and rollback boundaries must be included in the chain.

- All payloads must be referenced by the BootstrapperApplication, a package, or as an external payload.
- More stringent checks to prohibit reserved bundle ids.

Changes that should be handled by `wix convert`

BootstrapperApplicationDll

In order for the Bal extension to include the correct architecture, the `BootstrapperApplication` element changed to be something that can be specified multiple times. The file information for that moved to the new `BoostrapperApplicationDll` element. The optional v3 elements `bal:WixStandardBootstrapperApplication` or `bal:WixManagedBootstrapperApplicationHost` are required in v4, and the theme is specified in the `Theme` attribute instead of using a `BootstrapperApplicationRef`.

Replace CustomTable with BundleCustomData

Bundles include a file for the BA with data about the bundle in BootstrapperApplicationData.xml. In v3, arbitrary data could be added to that file with `CustomTable`. In v4, use `BundleCustomData` instead.

Rename ExePackage command line attributes

`InstallCommand`, `RepairCommand`, `UninstallCommand` renamed to `InstallArguments`, `RepairArguments`, `UninstallArguments`.

Redesign RemotePayload and each of the Package elements

Elements like `MsiPackage` actually contained two different concepts - the package's main file (e.g. `SourceFile` and `DownloadUrl`) and information about how and when to (un)install the package (e.g. `InstallCondition` and `Permanent`). In v4, information about

the package's main file should be specified in a new element, e.g. `MsiPackagePayload` for `MsiPackage`. For backwards compatibility, it was decided to continue to allow the core file information on the package element as long as the child package payload elements is not also specified. `RemotePayload` was removed and the appropriate package payload must be used instead.

Edit this page