WHITE PAPER

# Managing the Life Cycle of a Suite/ Advanced UI Installation

By John Cresswell, Senior Software Engineer; Josh Stechnij, Senior Software Engineer; and Michael Urman, Manager, Software Development Engineering, Flexera Software

FLEXERA® SOFTWARE

# Managing the Life Cycle of a Suite/ Advanced UI Installation

*The Suite/Advanced UI project type is available in the Premier edition of InstallShield; this type of project includes support for packaging together multiple separate installations as a single installation while providing a unified user interface. Note that the Advanced UI project type is available in the Professional edition of InstallShield; it has support for including only one main installation; that is, it does not have support for packaging multiple separate installations.*

*To ease readability, the following white paper uses the term Suite project to refer to a Suite/Advanced UI project.*

## Introduction

The life cycle of a single package—perhaps a Windows Installer package or an InstallScript installation—consists of well-understood phases: first-time installation of a product, maintenance of the product, upgrades to the product, and product removal. In scenarios in which one package replaces another package, the level of complexity increases. A Suite installation further increases this complexity, since each package in the Suite may reach various phases at different stages during the life of the Suite. For example, a new version of a Suite installation may introduce a new package, upgrade a second package, and remove a third; it may also replace two packages with a new updated version. And on some target systems, all of those packages may behave as a first-time installation. Although the level of complexity increases with some of these Suite scenarios, the flexibility that is available makes the Suite project type a powerful choice for addressing various scenarios of modern installations. Accounting for each of the supported scenarios requires careful planning.

This white paper presents background information that explains how the Suite engine determines the state of each target system as well as the phase that should be used to run a Suite installation. This white paper also highlights how to plan and manage a Suite installation that successfully addresses each required phase in its life cycle.

## How the Suite Engine Determines the State of a Suite Installation on a Target System

At run time, the Suite engine needs to determine which state it should use to run the Suite installation:

- First-time installation, which is typically used when the product is not present on the target system
- Maintenance or removal, which is typically used when a particular product version is already present on the target system
- Upgrade, which is typically used when an earlier version of a related product is present on the target system

The following factors help determine which state is used:

- Whether the Suite is registered on the target system
- Package type (primary vs. dependency)
- Mode condition

The following sections explain these factors in more detail.

### Suite Registration

Suite projects let you specify whether you want your Suite installation to be registered on target systems; the registration is essential to ensuring that the Suite runs in the correct mode at run time. Registration involves creating the necessary registry data on the target system to add an entry

for the Suite in Add or Remove Programs. This entry lets end users perform maintenance for your Suite installation, modifying or removing if needed. The General Information view in a Suite project has a Show Add or Remove Programs Entry setting that lets you indicate the appropriate behavior, thereby indicating whether the Suite is registered on target systems. Note that to manage the life cycle of a Suite and support any of the life cycle scenarios that are described in this white paper, Yes must be selected in the Show Add or Remove Programs Entry setting.

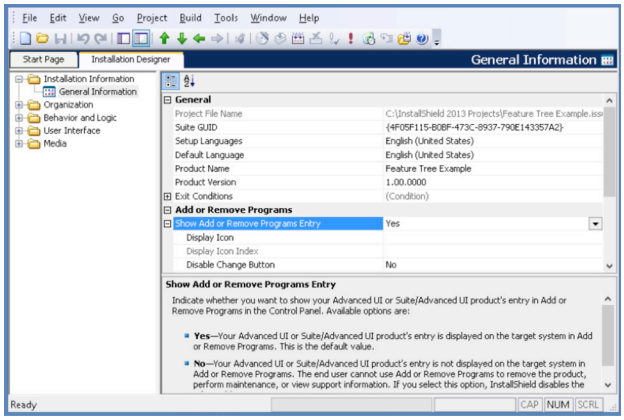


Figure 1: To manage a life cycle of a Suite, Yes must be selected in the Show Add or Remove Programs Entry setting in all versions of the Suite.

## Package Type

Each package in a Suite project needs to be identified as either a primary package or a dependency package. The package type identifies whether the presence of that package on a target system should influence whether the Suite installation runs in first-time installation mode or maintenance mode:

- Primary package—A primary package is a main part of the Advanced UI or Suite/Advanced UI installation.

  At run time, if all of the primary packages in the installation are absent from the target system, the installation may run as a first-time installation. If any of the primary packages are present on the target system, the installation may run in maintenance mode.

- Dependency package—The presence or absence of a dependency package on the target system does not influence which mode is used to run the installation.

  An example of a package that is typically flagged as a dependency package rather than as a primary package is the .NET Framework installation. The .NET Framework may need to be present on target systems in order for your product to function correctly; however, its presence should not be used to determine whether the Suite is already installed on a given target system. Otherwise, if your product has never been installed on a target system but the .NET Framework is present, your Suite installation would never run as a first-time installation.

The Package Type setting that is available when you select a package in the Packages view indicates whether the package is a primary package or a dependency package. You can change the value of this setting as needed for any packages in your project.

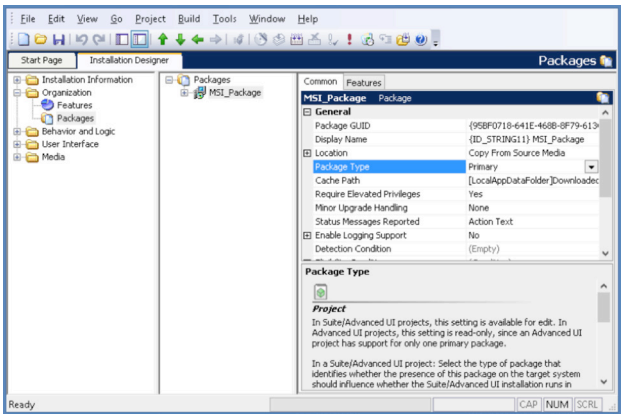


Figure 2: The Package Type setting in the Packages view lets you specify whether a package is a primary package or a dependency package. Only primary packages influence whether the Suite behaves as a first-time installation or runs in maintenance mode.

## Mode Conditions

At run time, a Suite installation runs in one of the following modes:

- Install mode, in which the installation behaves as a first-time installation
- Maintenance mode, in which the installation enables end users to change which features are installed, remove the product or features, and (if supported) repair the product.

InstallShield creates install and maintenance mode conditions in Suite projects automatically. The mode conditions are not available for edit within InstallShield; they are visible only when the project file (.issuite) is viewed in a text editor. The mode condition that InstallShield builds into the release is visible in the Setup.xml file in the Suite release's Interm folder. These mode conditions determine whether a Suite installation runs in first-time installation mode or in maintenance mode.

```
- <Mode>
  - <Install>
    - <When>
      - <Any>
        - <Not>
            <DetectPrimary Id="*"/>
          </Not>
          <SuiteInstalled SuiteId="*" Compare="NotEqual" Version="*"/>
        </Any>
      </When>
  </Install>
  - <Maintenance>
    - <When>
      - <Any>
          <DetectPrimary Id="*"/>
        </Any>
      </When>
  </Maintenance>
</Mode>
```

Figure 3: Example of a placeholder mode condition in an .issuite project file

The install mode condition is based on the following factors:

- **Detection conditions**—Part of the install mode condition consists of the detection conditions of all of the primary packages in the Suite project. If none of the primary packages' detection conditions evaluate as true (that is, if none of the primary packages are already installed), the detection condition part of the install mode condition is true. If one or more of the primary packages' detection conditions evaluate as true (that is, at least one of the primary packages is already installed), the detection condition part of the install mode condition is false.
- **Suite Installed condition**—The other part of the install mode condition consists of a Suite Installed condition, which may trigger the installation to run in first-time installation mode if the same version of the Suite installation is absent from the target system. The Suite Installed part of the install mode condition compares the Suite GUID and the version of the Suite installation with, if applicable, those of the Suite installation that is installed on the target system. If these values are different, the Suite Installed part of the install mode condition suggests that the Suite installation is absent, so a first-time installation may result. For more details about the Suite Installed type of condition check, see "Detecting Whether a Specific Version of an Advanced UI or Suite/Advanced UI Installation Is Already Installed" in the InstallShield Help Library.

  Note that the Suite engine's use of the version number to differentiate between different product versions is similar to that of the Windows Installer engine and the InstallScript engine. Like the Windows Installer and InstallScript engines, the Suite engine uses only the first three fields of the Suite version to distinguish between different product versions; it ignores the fourth field. Thus, the Suite engine sees version 1.0.0.0 and 1.0.0.1 as the same version, but 1.0.0.0 and 1.0.1.0 as different versions.

If all of the primary packages' detection conditions evaluate as false, or if the Suite Installed condition indicates that the Suite installation is not already installed, the install mode condition is true, and the installation runs as a first-time installation.

The maintenance mode condition is based on the detection conditions of all of the primary packages in the Suite project. If any of the primary packages' detection conditions evaluate as true (that is, if one or more of the primary packages are already installed), the mode condition is true, and the installation runs in maintenance mode.

## Configuring Products to Install Side by Side with Each Other

One possible scenario for product installations allows for every release of a product to be installed side by side with existing or future versions, instead of updating or replacing existing versions of a product with each new release. The side-by-side scenario is similar to what is provided by products such as Microsoft Visual Studio or InstallShield. End users can typically install each version of these products with earlier or later versions of the product on a given target machine. Suite installations, .msi packages, and InstallScript installations all support this type of product installation scenario.

For an .msi package to support such a side-by-side scenario, the product code, package code, and product version at a minimum should be changed. InstallScript installations require changing the product code and the product version. Making these changes allows for such packages to be able to install with existing versions on the machine without replacing or upgrading them.

One additional change to consider making in the case of side-by-side .msi or InstallScript installations is the target install location. Since changing the product codes of these types of installations allows them to be installed with existing versions, it is always best to ensure that the files installed by such packages do not overlap and cause shared reference count issues such as files left on the machine after products are uninstalled.

The requirements for allowing a product installed by a Suite installation to be able to install side by side with other versions includes the following:

1. Change the product codes, product versions, and (for .msi packages), package codes as relevant in each primary package that is in the Suite project.
2. Change the target installation locations of each primary package
3. Build new versions of the primary packages to incorporate the changes to product code, product version, package code, and target installation location, and include these new versions of the packages in the Suite project.
4. Change the Suite GUID and version of the Suite project. These are configurable in the General Information view. This triggers the installation to run as a first-time installation on target systems on which an earlier version of the Suite is already present, thereby allowing side-by-side instances of different versions of the product.
5. Build a release of the new version of the Suite installation.

Requirements 1, 2 and 3 allow the packages of the Suite to install side by side with other versions of the packages; requirements 4 and 5 allow the Suite itself to install side by side with other versions of the Suite.

Note that changing the Suite GUID and version is typically not sufficient to permit Suite installations and the packages that they include to install side by side. If the primary packages in the Suite are more or less the same (same product codes, package code, versions, etc.), the install mode condition still evaluates to true, allowing the Suite to

run as a first-time installation. However, the Suite engine does not install a primary package that shares product codes with existing versions since the Suite detects them as already installed on the target system. Worse still, in the event that either Suite that installed these packages is removed from a machine, the remaining Suite installation will be broken because its primary packages were removed by the unrelated Suite.

While primary packages are the main focus for installation in a Suite installation, dependency packages still merit some discussion. Since these packages do not affect what mode the Suite runs in (by default), and since they are often shared across unrelated products, these typically do not need to be changed in side-by-side installation scenarios. For example, Product A version 1.00 and Product A version 2.00 that can be installed together side by side may both depend on the Microsoft Visual C++ 10.0 runtime. Since this dependency is required by both products but only needs to be installed once, no changes to the Suite or the VC++ runtime package need to be made. Once either version of Product A is installed on a target machine, the VC++ 10.0 runtime will be installed if it is not already installed. Subsequently installing the other version of Product A skips the installation of the VC++ dependency since it is already present. Uninstalling either product leaves the dependency on the machine, allowing the remaining product installations to continue to function correctly. (Dependency packages generally do not have a remove operation enabled.)

## Managing Upgrades through Suites

The first time your Suite is run on a machine things are simple: the machine has never seen any of your packages or your Suite, and install operations are run as expected. However, consider a scenario in which you have a Suite of packages—some of which are already installed on a given machine, and some are not. You have upgrades to roll out. You may have added packages, removed packages, or merged them. The following sections explore these types of typical upgrade scenarios and provide guidance on how to configure a Suite project and its packages to create a successful installation.

### Configuring Upgrades that Replace Earlier Versions If Present

In many cases, the number of packages in the Suite project does not change from one version to the next. The goal of a new version of the Suite installation is to upgrade the application on the machine. Different versions of the application cannot coexist with each other, so the upgrade must in some way replace the files on the system. The AdminStudio installations typically fall into this category, since only one version of AdminStudio can be installed on a machine at a given time, and it is rare that the list of packages within the installation changes.

For example, when Suite 1.0 was released, it contained primary packages Pack A 1.0 and Pack B 1.0. Suite 2.0 needs to completely replace version 1.0 if it is present, leaving just 2.0 registered on the machine. As with all upgrade scenarios, the individual packages need to be configured to ensure they successfully replace their counterparts already registered on the machine. Equally important to success is the Suite running in the correct mode condition so that appropriate package operations are run.

In order for the individual packages to complete the upgrade, the Suite needs to run as a first-time installation. Incrementing the Suite version ensures that the Suite is not detected as already being registered. Then as long as the new packages are not already installed, as determined by their detection conditions, the Suite runs as a first-time installation. The general strategy is independent of the package type; however, there are slight differences in how things should be configured.

If all the packages in Suite 1.0 and Suite 2.0 are .msi packages, there are a number of ways of upgrading the packages. To perform a major upgrade, Pack A 2.0 and Pack B 2.0 need to have entries in the Upgrade table to enable the original packages to be removed before the version 2.0 files are installed. Since the product code and package code are being changed between versions in this scenario, no other changes need to be made to the Suite project. For minor upgrades, you can use the Minor Upgrade Handling setting in the Packages view of the Suite project to manage the installation. If you select Automatic in this setting for the .msi packages, at run time if an .msi package with the same product code is detected on the machine, the correct options are passed to the Windows Installer engine to allow the minor upgrade to succeed.

For InstallScript packages the new installation is responsible for performing the upgrade of the existing package. Whether you are using the built-in InstallScript engine support to perform the upgrade or you are scripting the uninstallation yourself, the configuration from a Suite project perspective is the same. Since the version number in the new InstallScript project is higher than that already on the target machine, the Suite engine's built-in support allows the upgrade to take place. The built-in detection conditions that are generated for the InstallScript project use the version. Since this condition becomes part of the mode condition, no extra configuration is required in the Suite project.

For other installation technologies, it is likely that .exe packages are being used. The Suite engine in general does not know how to determine if an executable file is installed on a machine. So once version 2.0 of the executable file has been created and configured to perform the upgrade from 1.0 to 2.0, you need to manually create the new package's detection condition in the Suite project. The new executable file must not be detected on the target machine; otherwise, the install operation is not passed to the command line at run time. For additional details, consult the documentation of the installation technology that you are using to create the executable package.

Note that if the end user chose to install only some of the features in the base Suite, the feature selections are not preserved automatically when the Suite upgrade is run. To learn how to add this support, see "Preserving End Users' Feature Selections from One Version of a Suite to the Next Version" on page 8.

**Adding a New Package to a Suite**

Another scenario to consider for Suite installations is one in which new packages are added to a new version of a Suite. These could either add new functionality delivered through new .msi or InstallScript packages, or update an existing package, such as through a Windows Installer patch (MSP). While the overall changes that are required to support this scenario are similar in each case, this section considers each of these separately.

**Updating Existing Functionality Through New Packages (Patching)**

In some cases, you may want to update an existing Suite installation, or, more specifically, some packages in the Suite installation through patches. For .msi packages, you can accomplish this through .msp packages that are patches for the existing .msi packages. Taking this approach lets you create a Suite installation that installs the very latest version of a product on a machine with or without an earlier version of the product. You can create a new version of the Suite installation that includes a new patch that updates a base .msi package in the installation. If the patch is authored correctly—either through the use of cumulative minor upgrades, or through using the Windows Installer MinorUpdateTargetRTM patch property—the patch contained in the Suite can constantly be replaced with a new patch.

As an example, consider a scenario where a Suite installation contains a number of .msi packages that comprise the main functionality of an antivirus product (though this scenario is unlikely due to self-updating behavior of such products). One of the .msi packages in this Suite contains the main antivirus engine and virus definition files. As part of the product lifetime, the engine is updated on regular intervals along with the virus definition files. The first release of this product is installed by a Suite installation. The updates to the engine and virus definitions are defined through patches (.msp files) to the main .msi package. Once the initial Suite is released, updates can be included in an updated Suite installation that contains an additional package for the .msp file.

In order to build an update to a Suite that implements this type of scenario, add the .msp package through the Packages view in the existing Suite project. The package can be configured as necessary, though no special changes should be needed once the .msp file is added. In addition to adding the package, create a new feature in the Suite project, and configure it to be not visible (thereby preventing an end user from deselecting the feature at run time). Then associate the patch package with the new hidden feature. This then ensures that the patch package is installed during any first-time installation operation of the Suite.

The Suite version in the General Information view should also be changed. Changing the suite version but keeping the Suite GUID the same allows an existing instance of the Suite on target systems to be upgraded. At run time, the install mode condition evaluates to true because of the Suite version change. The Suite engine then runs as a first-time installation. Since the original packages that were in the earlier version of the Suite may already be installed, the installation of these is skipped during the Suite upgrade. The new patch package is applied to the target system since the feature it is associated with is selected to install by default.

One issue to be aware of with this approach relates to the features in the Suite. In the earlier version of the Suite, an end user may have selected or deselected certain features if the Suite was authored to allow this. However, during the upgrade, the original feature selections are not available, since the Suite engine does not persistently track feature states across installation sessions. For information on how to add this support, see "Preserving End Users' Feature Selections from One Version of a Suite to the Next Version" on page 8.

**Adding New Functionality through New Packages**

Another scenario that you may encounter involves adding new packages in a Suite to provide additional functionality to a product (as opposed to just updating the existing packages contained in a Suite). While this may not be as likely to occur, it is still possible to deliver such changes through a Suite installation.

By way of example, consider a product to which you may eventually add a new subproduct at some time after its initial release. In an Office-like product, this could involve adding a spreadsheet application into the product after already shipping it with a word processor. Creating a new .msi package to install the spreadsheet application may be easier than adding the functionality to the existing package, and it averts some issues with .msi packages. Separating it into another .msi package also allows for easier maintenance of installing and uninstalling this particular part of the product independent of the rest of the product.

Two different approaches are available for this scenario when you add the new .msi package as a new package in the Suite project:

- Change the Suite version to ensure that the Suite runs as a first-time installation on machines that have the existing Suite installed.
- Leave the Suite version the same, which causes the Suite to run in maintenance mode if it was already installed on a target machine.

Both of these approaches are valid, and choosing which to use depends on the requirements for the installation.

If you change the Suite version and add a new .msi package to the project, consider associating the package with a new feature to allow end users to separately select and install this package independent of other parts of the Suite. This approach enables end users to install the Suite as would normally be expected for a first-time installation on machines that do not have any existing version installed. On machines that do have an existing version installed, changing the Suite version allows for running the Suite as a first-time installation again. Since a Suite in this scenario has been updated from an earlier version, this is a recommended way of delivering an update that includes new packages to provide new product functionality. Behaving as a first-time installation gives end users the expectation that the product is different from the existing version they had installed. One caveat to this approach is that the existing Suite's feature states are not maintained in the updated Suite session. However, it is possible to preserve the existing feature states into the updated Suite session. For more information, see "Preserving End Users' Feature Selections from One Version of a Suite to the Next Version" on page 8.

If you do not change the Suite version, the only required significant change to the Suite project would be adding the new .msi package and a new feature that is associated with that package. Although a new feature this is not strictly necessary, it provides for a Suite that has a reduced degree of complexity. Running this Suite on a machine that does not have an earlier version results in first-time-installation behavior. Running this Suite on a machine that has an earlier version results in the updated Suite running in maintenance mode. The new package can be installed by selecting its feature from the maintenance selection wizard page. The downside to this approach is that it is unclear that this Suite has been updated, and end users may not notice the new feature. In addition, changes to existing packages (through something like Windows Installer minor upgrade packages) should not be included. The upgrades will run, but this experience would be unexpected for an end user installing this Suite, since it looks like it would run only normal maintenance operations.

**Removing a Package from a Suite**

Consider the situation where you need to remove a package for a new release of your software. Perhaps you are deprecating a tool or you have merged one tool into another and removed one of the packages. As with most other upgrade scenarios, the Suite should be configured to run as a first-time installation if an earlier version is present, so the Suite version needs to be updated.

If the packages are not configured carefully, the package that is being removed can be orphaned on the machine. This means that the package is still installed—the contents of the package are still present, but the Suite that is registered contains no information about this package. This results in the package being permanently installed without some manual intervention from an end user.

Simply removing the unwanted package from the new Suite results in the package being orphaned at run time. The install operations of the remaining packages are the only commands that are run; in general, these commands do not remove another installation from the machine. The Suite needs to contain information on how and when to remove the package that is no longer required. How this is accomplished is dependent on the contents of the Suite and the type of package being removed.

If the package being removed is an .msi package, there are other .msi packages in the Suite, and a major upgrade can be performed for one or more of the .msi packages, the situation is simplified somewhat. In that case you can configure one of the other .msi packages in the Suite to remove the unneeded package through an entry in the Upgrade table and the RemoveExistingProducts action.

For all other situations further configuration is required to remove the package. In one approach, you can add to the Suite a new package that is responsible for removing the old package. To configure the removal, use the command line and silent command line settings under the Install setting of the Operations area in the Packages view, and enter command-line statements that uninstall the payload of the old package. This "remove" package is a pseudo package, since it does not contain any payload and it does not install anything on target machines.

Examples of more complicated cases are when the other packages in the Suite are Windows Installer minor upgrade packages or the package being removed is not an .msi package. In these cases you need a method that would remove the old package when a Suite that does not contain the package runs as a first-time installation. The solution here is to configure the new Suite to contain a package—most likely an .exe package—that as part of its install operation performs the uninstallation of the unwanted package. That is, in the Packages view of the Suite project, add a new .exe package that does not contain any payload. Use the command line and silent command line settings under the Install setting of the Operations area to specify command-line statements that uninstall the unwanted package. For an .msi package, you would want the package to use the /x uninstallation command-line parameter and probably the /qn silent parameter (for example, msiexec /x {product code of unwanted .msi} /qn). For InstallScript, you would pass the /uninst parameter. For other installation technologies, you would run the appropriate commands required to remove the package. All of these commands should be placed in the "remove" package's install operation command line; the install operation is the only one that needs to be supported.

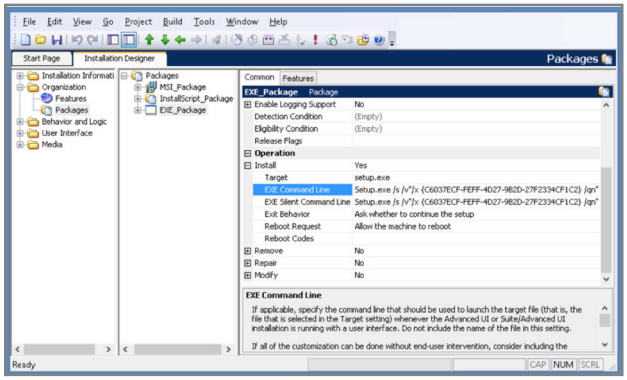

Figure 4: Command-line settings for a "remove" package that removes an .msi package

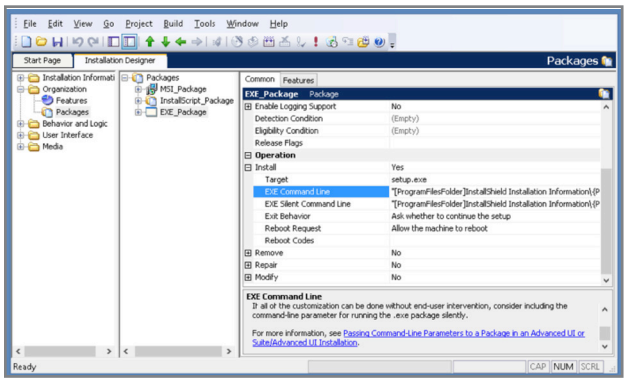

Figure 5: Command-line settings for a "remove" package that removes an InstallScript package: "[ProgramFilesFolder]InstallShield Installation Information\{PRODUCT-GUID-HERE}\Setup.exe" /uninst

The detection condition for the remove package needs to be configured so that the package runs only at the appropriate time—when the old package is detected on the machine. To do this, you can simply use the detection condition from the earlier Suite's install operation and negate it. The remove package then runs only if the package is already on the machine. The Suite engine has built-in InstallScript Package and MSI Package condition types for detecting InstallScript and .msi packages. Using these conditions within a None condition group ensures that the remove package is run only on machines on which the old package is present.

Note that if the package that you want to remove is an .msp patch, Windows Installer removes it when it removes the base .msi package, or when a different .msp file supersedes it. Therefore, it cannot be orphaned at the .msp level; only by orphaning the base .msi package will you orphan the .msp file.

For package technologies that the Suite engine cannot automatically detect, simply use the same condition that was used in the previous package's detection condition, but within a None condition group. All remove packages should be marked as dependency in the Package Type setting to prevent their detect condition being used in the mode condition of the new Suite.

Additional considerations may be required for the conditioning of a remove package. For example, you may need to use an eligibility condition to prevent the package from being considered for installation. This may be especially true if the feature tree of the Suite is large and complex. Use the aforementioned details as a baseline for implementing such package removals.

**Preserving End Users' Feature Selections from One Version of a Suite to the Next Version**

When installation authors upgrade a complex installation that offers several options, it is common to try to respect the choices that the end user made when first installing the product. Windows Installer offers this across major upgrades by means of the MigrateFeatureStates action. But for MigrateFeatureStates to work correctly, several preconditions must be met. In particular, the feature tree must not have changed greatly from that in the preceding package. Suite installations have more flexibility in offering the same upgrade experience, in that the layout of the feature tree need not be unchanged; however, there is no MigrateFeatureStates action to do the work. Instead, the installation author must configure conditions within the Suite to provide a smooth upgrade. The idea is simple, but the implementation can be intricate. Configuring this correctly can significantly enhance the end user's overall appreciation for your product's installation.

Here's how to approach end users' feature selections. Out of the box, an installation of a Suite typically installs all features. However, a condition on each feature can override this. For each feature whose state should be copied from an earlier version, specify a condition that is true if its predecessor is on the machine. Typically this will look somewhat like the detection condition of a package that was associated with this feature in an earlier release. To handle a new feature, or cases when the earlier version of the Suite is not installed, consider also whether the feature should be enabled in a first-time installation. The following table shows four common scenarios in which a feature is associated with a single .msi package, as well as conditional statements that support each of those scenarios.

Table 1: Conditions for Preserving End Users' Feature Selections

| Scenario | Sample Operators and Condition Checks for Feature Conditions in the Features View of the Suite Project |
|---|---|
| A new feature, enabled by default | Condition<br>Any |
| A new feature, disabled by default | Condition<br>None |
| An upgrading feature, enabled by default for first-time installations | Condition<br>Any<br>MSI Upgrade<br>None<br>Suite Installed |
| An upgrading feature, disabled by default for first-time installations | Condition<br>All<br>MSI Upgrade<br>Suite Installed |

Consider the MSI Upgrade detection cases represented by the feature conditions of the above table carefully. The footprint provided by an earlier version of the package may not match the footprint of the current version, and the end user may have skipped one or more intermediate versions of your product. No matter how the end user got to the current version of your product, the installation should choose reasonable features to install, and choosing the correct condition—or conditions—is central to ensuring that this happens.

## Summary

This white paper provides insight on how the Suite engine chooses between a first-time installation experience and a maintenance experience in different run-time scenarios. It also explains how to manage Suite installations at different phases during their life cycles. The following table summarizes the various approaches for addressing different run-time scenarios.

## Summary of Suite Life Cycle Changes

| Scenario | Required Changes for a Primary Package in the Suite | Required Changes to the Suite Project |
|---|---|---|
| Side by side<br><br>(The new version of the product can be installed on a machine that has the earlier version.) | 1. Change the product code.<br>2. Change the product version.<br>3. Change the package code (for an .msi package).<br>4. Change the target installation location.<br>5. Build the new version of the package. | 1. Change the Suite GUID.<br>2. Change the Suite version.<br>3. Include the new version of the package in the project.<br>4. Build a release of the new version. |
| Old version replacement<br><br>(Replace an old version with a new version if present; otherwise, install the new version.) | For a major upgrade .msi package:<br>1. Change the product code.<br>2. Change the product version.<br>3. Change the package code.<br>4. Update the files and other data as needed.<br>5. Add entries to the Upgrade table.<br>6. Build the new version of the package.<br><br>For a minor upgrade .msi package:<br>1. Change the product version.<br>2. Change the package code.<br>3. Update the files and other data as needed.<br>4. Build the new version of the package.<br><br>For an InstallScript package:<br>1. Change the product version.<br>2. Update the files and other data as needed.<br>3. Build the new version of the package.<br><br>For other package technologies (.exe packages, etc.), consult the appropriate documentation for creating a new version that replaces the earlier version. | 1. Change the Suite version.<br>2. Replace the old version of the package in the project the new version.<br>3. For .exe packages and packages other than .msi and InstallScript, update the detection conditions of the new version of the package as needed.<br>4. Build a release of the new version of the Suite. |
| Updated functionality, new package<br><br>(Update functionality by introducing a new package.) | Create an .msp package for the base .msi package. | 1. Change the Suite version.<br>2. Add the .msp package to the project.<br>3. Associate the .msp package with a new hidden feature to ensure that it is applied during any first-time install.<br>4. Build a release of the new version. |
| New functionality, new package<br><br>(Add new functionality by introducing a new package.) | Create and build the new package. | For method 1, in which the resulting Suite behaves as a first-time installation if an earlier version of the Suite is already present:<br>1. Change the Suite version.<br>2. Add the new package to the project.<br>3. Optionally associate the new package with a new feature.<br>4. Build a release of the new version.<br><br>For method 2, in which the resulting Suite runs in maintenance mode:<br>1. Add the new package to the project.<br>2. Optionally associate the new package with a new feature.<br>3. Build a release of the new Suite. |

## Summary of Suite Life Cycle Changes

| Scenario | Required Changes for a Primary Package in the Suite | Required Changes to the Suite Project |
|---|---|---|
| Remove a package | The approach varies, depending on what other kinds of packages are available in the Suite.<br><br>For a major upgrade .msi package:<br>If the new version of the Suite contains a major upgrade .msi package, configure the .msi package (through the Upgrade table and the RemoveExistingProducts action) to uninstall the package that needs to be removed.<br><br>For other kinds of packages<br>(e.g., a minor upgrade .msi package or an InstallScript package):<br>Create a "remove" package that does not install anything but only uninstalls the package that needs to be removed. | For a major upgrade .msi package:<br>1. Change the Suite version.<br>2. Replace the package that is being upgraded with the major upgrade .msi package (that uninstalls the package that needs to be removed).<br>3. Build a release of the new version.<br><br>For other kinds of packages:<br>1. Change the Suite version.<br>2. Add the remove package to the Suite project, and select Dependency for its Package Type setting in the Packages view.<br>3. Configure the remove package's detection condition so that it is run only if the package to be removed is detected on target systems.<br>4. Configure the command line and silent command line settings (Packages view > Operations area > Install subarea) to specify command-line statements that uninstall the unwanted package.<br>5. Build a release of the new version. |
| Preserve end users' feature selections from one version to the next | | In the new version of the Suite project, incorporate feature conditions that detect the packages that end users previously chose whether to install:<br>• If a package predecessor is installed because an end user selected its feature in the earlier Suite version, the feature condition in the new Suite should evaluate to true to trigger the package's feature to be selected again.<br>• If a package predecessor is not installed because an end user deselected its feature in the earlier Suite version, the feature condition in the new Suite should evaluate to false to prevent the package's feature from being selected.<br>• If a package has no predecessor, the feature condition should reflect whether the package should be installed by default. |

**Flexera Software LLC**
1000 East Woodfield Road,
Suite 400
Schaumburg, IL 60173 USA

Schaumburg
(Global Headquarters):
+1 800-809-5659

United Kingdom (Europe,
Middle East Headquarters):
+44 870-871-1111
+44 870-873-6300

Australia (Asia,
Pacific Headquarters):
+61 3-9895-2000

For more office locations visit:
www.flexerasoftware.com



IS_WP_AdvUILifecycle_Feb13