

Deploying and Using Conflict Explorer 2009

Table of Contents

1.	Introduction	3
1.1	Conflicts in software packages.....	3
1.2	Conflict management with Conflict Explorer 2009.....	4
1.3	Moderate investments and transparent modifications that significantly improve the quality of your software packages easily repay their costs!.....	4
1.4	Is there any sense in deploying Conflict Explorer 2009 if other Windows Installer developer tools with conflict features are already being used?.....	5
2.	Typography	6
3.	Terminology	6
3.1	FamilyBase.....	6
3.2	Version.....	6
3.3	Package.....	6
3.4	Family or FamilyName.....	6
4.	General	7
4.1	Features.....	7
4.2	Components and dependencies for Conflict Explorer.....	8
4.2.1	ImportMsi.EXE.....	8
4.2.2	CheckConflicts.EXE.....	10
4.2.3	ConflictExplorer.EXE.....	10
4.2.4	Settings.INI.....	10
5.	Using Conflict Explorer	14
5.1	User interface.....	14
5.2	Importing software packages.....	15
5.2.1	Importing a software package with the user interface.....	15
5.2.2	Automated import of multiple software packages into existing environments.....	16
5.3	Basic standard workflow.....	17
5.4	Conflict types.....	17
5.4.1	Registry Conflicts.....	18
5.4.2	SelfReg to SelfReg Conflicts.....	18
5.4.3	SelfReg to Registry Conflicts.....	18
5.4.4	Registry to SelfReg Conflicts.....	18
5.4.5	(Value conflicts).....	18
5.4.6	File Conflicts.....	19
5.5	Excluding conflicts.....	19
5.5.1	Filter operations.....	19
5.5.2	Structural methods.....	19
5.5.3	Recommended usage.....	21
5.6	Automated conflict resolution.....	21
5.6.1	Special rules for Registry references to files in the installation instance.....	23
5.7	Manual conflict resolution.....	24
5.8	Aligning the ComponentId.....	24

5.9	Standard rules used when identifying conflicts	27
5.10	Rules when forming the KeyPath for new components	27
5.11	Search	28
5.12	Options in the Conflict Explorer user interface	29
5.12.1	General tab	29
5.12.2	Resolve Settings tab	30
5.13	Exporting conflicts.....	30
5.14	Moving, copying and deleting packages, creating groups	31
5.14.1	Creating or renaming groups	31
5.14.2	Moving or copying package families	32
5.14.3	Deleting package families	33
5.15	Conflicts with the operating system	33
5.16	Log file.....	34
5.17	License activation process	34
5.18	Shortcut keys	36
5.19	FAQ	36
6.	Command line options	38
7.	Important functions not yet covered	39
7.1	Delete own task list.....	39
7.2	Help	39
8.	Keyword index	40

1. Introduction

Successful software conflict management is of central importance for robust software packaging within large-scale corporate environments. Ignoring the necessity of this task means that one is closing one's eyes to the consequences of failing to carry out conflict management. Some of the risks this incurs include future failures, outlay on helpdesk and support facilities, and requests for product improvements. Above and beyond this, the image and reputation of a deployment service or a migration project may suffer greatly if one forces computer users to deal with unnecessary error situations involving the tools of their trade.

1.1 Conflicts in software packages

The terms "conflict solving", "conflict validation" and "conflict resolution" are interpreted variously within the context of software packaging and by different software solutions: With some tools, for example, we find rule violations in the local design – self-contained and covering the entire software package – as the cause of conflicts (*ICE* conflicts). In other tools, "software conflicts" describe the installation of a specific file from one software package into a common environment with a file that has the same name, but a different version number and comes from a different software package. Such tools recognise this situation and recommend replacing the file with the lower version number in the relevant software package during the repackaging process or subsequent conflict management operations. And, lastly, one generally refers to software conflicts in cases where a software package uses COM components that register themselves during installation via the *DllRegisterServer()* function or via other implemented registry mechanisms (e.g. a *registry* table from the MSI file). In that case, problems can occur that are also referred to collectively as "software conflicts" and which are generally known as "DLL Hell". These occur when the above-mentioned COM components are ultimately shared between a number of different programs, and where existing registry values are overwritten with new references, existing COM components are overwritten with versions of the shared files that are not really entirely compatible, or where registry references that relate to the DLL or OCX are deleted during the uninstall process or when upgrading a software product.

Naturally, there are many other types of software conflicts present in software packages. What they all have in common, is that they manifest themselves as problems only in **subsequent operations**: they may occur as a result of the shared use of different software products or the interplay between software elements and the operating system, or before or after the installation status of software has been changed (installing, uninstalling, etc.).

The consequences are extremely variable. While in one case the conflict may mean that the software cannot even be installed successfully, in other cases it may lead to the complete failure of certain software packages or only to the failure of certain functions. In the latter case, the execution of certain functions from these software packages may itself then have deleterious effects. Within the company, such behaviour may be observed generally – across the entire infrastructure – or may well only occur under certain configuration conditions.

Since, as we have seen, there is a wealth of different conflict types, the subsequent analysis and identification of the conflicts is frequently an extremely difficult, time-consuming process, which also requires an understanding for complex interrelationships. This is also the reason why the root causes of problems often remain hidden, and one tends to not pay enough attention to the topic of "software conflicts in software packages" – since one has not correctly identified prior fault scenarios as being caused by software conflicts. This underlines the importance of professional software conflict management, which focuses on the **proactive** resolution of error scenarios – blocking the effects of software conflicts before they have a chance to spread.

1.2 Conflict management with Conflict Explorer 2009

When categorising software conflicts from software packages, we distinguish the following major groups:

1. Conflicts arising from rule violations in the design of a software package, when considering local rules that are mostly restricted to the software package itself (e.g. *ICE* conflicts, *Internal Consistency Evaluators*)
2. Conflicts arising from global design errors and design configurations within a collection of several software packages, or from the interplay between software and the operating system.

Conflict Explorer 2009 identifies and resolves conflicts that belong to the second group described above. We can now further subdivide this second group (the following list is not conclusive):

- Conflicts arising from shared resources (between software packages or between the software package and the operating system), where the resource is removed when uninstalling or upgrading a software package, thus leading to problematic situations.
- Conflicts arising from shared resources with variant content. One example is where identical keys with different values are used by INI file entries that have the same file name in different software packages. Identical registry keys with different values that are used in multiple software packages also belong to this group.
- Conflicts arising from defective component design, when considering global rules (different *KeyPath* for "identical" components with the same *ComponentId*, different resources for "the same" components, etc.)

Conflict Explorer 2009 supports conflict management of conflicts from all of these three areas – either in the version currently available or a future version. Precise details of the conflict types currently supported can be found at [5.4 Conflict types](#).

1.3 Moderate investments and transparent modifications that significantly improve the quality of your software packages easily repay their costs!

If one considers the overall costs involved in an application life cycle, one finds that software packages that are robust and of a high quality will be responsible for far less total expenditure than packages that do not meet these standards. Especially when considering the deployment of *Conflict Explorer 2009*, the investments that bring additional value are insignificant when compared to the costs that may be incurred if one does not deploy software conflict management. After all: a software deployment process without effective software conflict management may ultimately grow to represent an incalculable risk.

In discussions with software packaging experts, one is occasionally confronted with the statement that one has never actually had any software conflict problems. However, in the vast majority of cases, this merely confirms the fact that one has simply not appreciated the effects of software conflicts directly. In scenarios where corporate computer users identify a malfunction in a software package, they often try to tinker with it themselves, aiming to identify the root cause of the error. There is, of course, nothing wrong with this approach. Yet valuable minutes or even hours can pass before the user, exasperated by the problem, finally picks up the phone to ring the company's IT support. Even at this stage, the company has already incurred considerable costs. In many cases, assuming IT support is also unable to resolve the problem after a while, staff will then request or order the reinstallation of the affected software package. Occasionally, even the entire system will be reinstalled. We don't even want to mention the loss of image that the IT services suffer in the eyes of their customers. Considerably worse are the costs stemming from the efforts of all parties involved (user, company support, units responsible for software distribution and installation, etc.). Considering the company as a whole (conflict situations generally occur on multiple occasions), these costs can assume dramatic proportions. The key factor in the whole scenario is that the software

packaging team itself never hears of any of these effects – nor, of course, does it work to identify their causes. Even in individual cases where a clear localisation of the cause was possible, the unavoidable solution is to correct the affected software package. This in itself incurs new expenditure.

By deploying *Conflict Explorer 2009*, one realises the consequences of deciding not to utilise an integrated, automated software conflict correction process, featuring a highly **proactive** design model. In a mid-sized company with a triple-digit software pool, it is not unusual to find software packages containing several thousand individual conflicts with other software packages or the operating system. While working with an even larger company, we found that around 25 to 30% of the company's software packages had untreated software conflicts. This value is naturally dependent on the number of imported and deployed software packages, and increases with the size of the deployable software pool.

1.4 Is there any sense in deploying Conflict Explorer 2009 if other Windows Installer developer tools with conflict features are already being used?

Most Windows Installer developer tools used internal and local consistency checks (*Internal Consistency Evaluators, ICE*) to carry out package validation. An *ICE* validation checks the package database for entries that are individually valid, but which could cause erroneous behaviour in the context of the entire local database.

Conflict Explorer 2009 expands the checking algorithms from the **local** level to a corporate or other **global** perspective, and makes it possible to display potential conflicts within a heterogeneous desktop environment. In addition, *Conflict Explorer 2009* also enables automated conflict resolution via a robust *transform file*, which is added to the Windows Installer package. The exact nature of these modifications depends on best practice rules. The action is transparent, and can also be used for all types of Windows Installer packages without needing further modification.

In addition, many features distinguish it from similarly-conceived tools. These features aim to place the focus on software packagers and their needs, and to ensure that their successful use does not negatively impact the speed of the software deployment process.

Conflict Explorer 2009 can be used as a standalone application. However, integrating it into existing environments within the software deployment process is an equally simple matter. In addition, it can expand any existing software conflict management solutions with a range of professional functions that cannot as yet be found anywhere else.

Additional features are discussed at [4.1 Features](#)

2. Typography

Words written in *italics* include foreign and technical terms, as well as chapter references. With chapter references, you can use "Ctrl-click" to follow the link.

3. Terminology

In the terminology applied to *Conflict Explorer*, we use a few terms with unusual meanings.

In a *Conflict Explorer* context, a *FamilyName* does not refer to the product family, for example (familiar to us from the design of the *UpgradeCode*, for example), but to a *PackageFamily*, with all of its family members. Family members are represented by one or more *Packages*.

3.1 FamilyBase

This corresponds to the left portion of the package name, as far as the first "_" separator used (if any). For example: "ADOBEREADER" (from "ADOBEREADER_9.0"). It is not mandatory to use such separators in the family descriptor. However, consistent use of the separator means that software packages updated in production with a new version as part of a *major upgrade* can be left as they are in the *Conflict Explorer* database, without this resulting in any reciprocal conflicts. See also [3.7 Standard rules used when identifying conflicts](#).

For example: ADOBEREADER_8.0 will thus have no conflicts with ADOBEREADER_9.0, and vice versa.

3.2 Version

This corresponds to the version descriptor, e.g. "1.0", which may be included optionally in the family name.

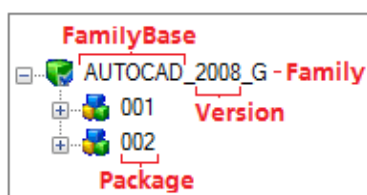
3.3 Package

A *Package* is a self-contained MSI package.

3.4 Family or FamilyName

This corresponds to the name of the software package, and is independent of the *ProductName* property. Any name may be chosen. A *Family* represents a collection of multiple *Packages*. A *Family* includes at least the *RTM* in the form of a *Package*. Patches, *small updates* and *minor upgrades* may also expand the *Family* as additional elements.

Generally, complete package families are installed onto systems (one *Package* after another). A transaction in software distribution (*INSTALL* or *REMOVE*) thus necessarily corresponds to the entire *Family*. This is the reason why conflicts between *Packages* are not taken into account as part of the identification of conflicts. (See [5.9 Standard rules used when identifying conflicts](#))



4. General

4.1 Features

Conflict Explorer is designed for use in displaying software conflicts and rectifying software packages automatically. The feature set below highlights in particular its strengths when compared with similar products:

- User-managed and automatic conflict resolution within separate or existing MST file
- Ability to selectively resolve individual conflicts, according to user settings
- Can be used without problems even on manufacturer MSI files (not only for self-repackaged packages), without the risk of negative consequences
- *Files, Registry, etc.*, data is supplemented by identification and comparison of *SelfReg* data
- Option of hiding conflicts from packages on a *PackageFamily exclude list* with a single mouse click
- Option of hiding general, individual conflicts via *Excludes*
- Real conflicts! (A file contained in one package compared to a file of the same name in another package will not lead automatically to a conflict if such files are identified as being located in different directories.)
- Consistent data. (Better data consistency than competitor products.)
- Helpful usability features. (Sorting of conflicts, column swapping, direct access to individual data items (copy to clipboard), etc.)
- Categorised and extended display of conflicts (*Value Conflicts, Standard Conflicts*)
- Powerful, robust import system! (No MSI files have yet been identified that could not be imported.)
- The majority of *Windows Installer properties* will be resolved, as long as they are not the result of *custom actions*. *AppSearch* properties are identified for each package, and also entered as clear text in the *Conflict Explorer* database.
- Support for one, multiple or variable MST files during import
- Instant visual notification (icon) when browsing packages in the tree view for conflict packages and packages that have been corrected automatically. Behaviour of this feature can be controlled precisely by user settings.
- Conflicts are not only identified between software packages, but can also be displayed between software packages and the operating system itself. This permits the creation of software packages free of any operating system conflicts.
- Parsing and display of the *action state* for a component, plus accounting for the *action state (ActionState)* during automatic conflict resolution!
- Conflict display in the *list view* is user-configurable: "*display all keys/files only once*", "*display only installed components*", etc.)
- Search through a variety of content from conflicts (F3)
- Handy display of additional information (import date, number of conflicts before/after an operation, etc.)
- A number of practical "little helpers", configurable by the user: for simple and direct opening of the underlying MSI file using a specified program (e.g. Orca), for rapid navigation through the MSI file system (Explorer), etc. Accounting for a number of different file shares (development/test/production, etc.).
- Recognition of dependencies via conflicts
- Keeping to the basic rule: "Fix as much as is needed, but as little as possible!"

4.2 Components and dependencies for Conflict Explorer

Conflict Explorer has been developed using .NET and requires .NET framework 2.0, >= SP1 on the client. In addition, .NET permissions for the directory in which it is executed must be set to *full trust*, so that *Conflict Explorer* can be started directly from a network share. The script *AddFullTrustingAndShortcuts.vbs* located in the directory *Client Setup* configures this setting appropriately, and must be run on every machine intending to use *Conflict Explorer*.

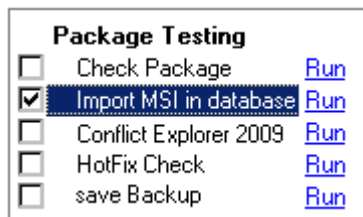
Note:

Local administrator rights are required when executing the preparatory script on a machine. On Windows Vista, the appropriate UAC screen will be displayed.

Conflict Explorer is subdivided into three separate modules: *ImportMsi.EXE*, *CheckConflicts.EXE* and *ConflictExplorer.EXE*. A configuration file named *Settings.INI* is also located in the program directory.

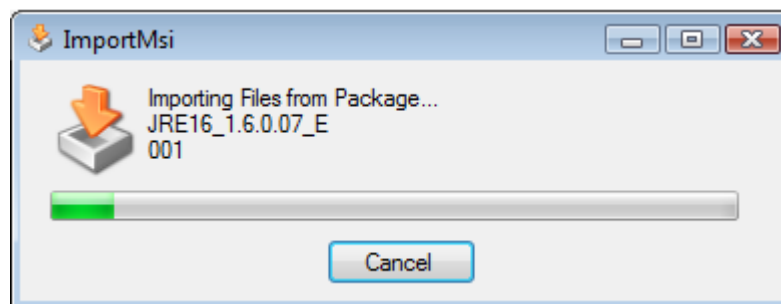
4.2.1 ImportMsi.EXE

ImportMsi imports all of the necessary information from a Windows Installer package into the *Conflict Explorer* database. *ImportMsi.EXE* is a console application and executes externally to the user interface of *Conflict Explorer*. This makes it a simple matter to incorporate *ImportMsi.EXE* into *Wise* projects. The following screenshot depicts a sample project in *Wise Package Studio*:



This could link to the following call:

```
ImportMsi.EXE "[ProjectDir]\[FileName].msi" TRANSFORMS=ALL
```

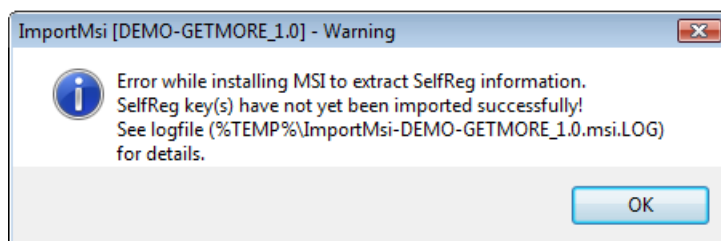


If *SelfReg* table resource links are found within the MSI file, then *ImportMsi* will perform a pseudo-installation of the Windows Installer package, thus making it possible to identify the *SelfReg* content resulting from the PE files. This is the only way of ensuring that dependencies can be located and that *SelfReg* registry keys can be identified as completely as possible.

Note:

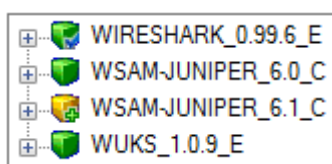
It is possible that errors may occur during pseudo-installation. Often, these can be traced back to the fact that required technical dependencies are unavailable (e.g. the InstallScript Engine). In such cases, one can simply install the technical dependency by hand and restart the import process.

A possible error message resulting from errors during the import process:

**Please note:**

Regardless of errors messages of this kind, all other information will nonetheless be written into the *Conflict Explorer* database.

Packages that have been imported but not yet scanned for conflicts (see following chapter), are indicated in *Conflict Explorer* by using a yellow icon for the *package family*.

**Note:**

During the import process, many data records in the *Conflict Explorer* database are set to read-only access. As a result, a team may not run more than one import process at a time. *Conflict Explorer* informs its users of this situation by displaying the following information:

Users: oberlind (import) ▾

Note:

Packages should **only be imported using a clean machine**, since otherwise the *action states* of the various components might show erroneous results following the import. See also the section *CleanMachinePackageLimitValue* in *Chapter 2.2.4 Settings.INI*

ImportMsi can be run from within *Conflict Explorer* by using the menu *Package/Import*, or executed directly, with or without passing it any appropriate command line options.

Note:

If – as a general naming principle – you want to read the names from the MSI file's *property* table automatically, then you may use *Settings.INI* to add the entries *PropertyPackage=PROPERTY* and *PropertyFamily=PROPERTY* to the *[Global]* section. In so doing, "PROPERTY" should be replaced by the desired property name. The corresponding *property* must of course have been set previously in the software deployment process!

For example: *PROPERTYFAMILY=ProductName*.

Alternatively, it is possible to specify the *FamilyName* and the *package* descriptor via command line options:

ImportMsi.EXE "\\MYSERVER\Package\Reader.msi" **FAMILY=ADOBEREADER_9.0 PACKAGE=001**

In addition, if the command line option **GROUPS="Name";"Name"** is used, then *ImportMsi* will assign the *package* to the specified *groups* automatically. If this command line option is not given, then the package will be assigned to the "unassigned packages" *group*. The *Conflict Explorer* GUI can be used to move it from here to the correct *group* at a later point in time.

4.2.2 CheckConflicts.EXE

CheckConflicts.EXE is the module that enables the display of conflicts. In so doing, the module compares the resources of one package with the resources of all other packages. Optionally, all of the resources of all packages can also be compared with all of the resources of all other packages. However, depending on the number of imported packages, *excludes*, groups and resources, this process may well take longer to complete. For the packaging process, where the focus is limited to a single package, a *Scan Package Conflict* is recommended in each case. This process compares a single package with all packages, delivering excellent performance in terms of data and analysis volume.



4.2.3 ConflictExplorer.EXE


This supplies the user interface for *Conflict Explorer*.

Further information can be found in Chapter [5 Using Conflict Explorer](#)

4.2.4 Settings.INI




The *Settings.INI* file is used to store global settings that are valid for all users. Note that a number of properties are also stored in the Registry under *HKCU\Software\Conflict Explorer* when altered by individual users. With some settings, entries made by the user take priority over the settings in *Settings.INI*. With other settings, entries made in *Settings.INI* cannot be overridden by the user (this is documented for each setting).

OrcaPath

Here, a program for displaying or editing MSIs can be specified. *Conflict Explorer* will access this path when the button in the toolbar is clicked. 

```
[Global]
OrcaPath=\\MYSERVER\Integration\Tools\Orca3.1.4000\Orca.EXE
```

PathReplacements

With this option, it is possible to extend the search automatically to alternative file system paths if the MSI file is not found when opening the MSI via "Edit MSI"  or via direct access  to the file path. A *re-import*  process will also use these replacement paths if it cannot locate the installation file. If your software packages are pushed one after the other to different servers or directories as part of the software deployment process (max. 3), then correct this entry to suit your mappings. Separate shares with a semicolon. The first entry must correlate to the standard directory from which the packages will be imported. If the installation file cannot be found, this directory will be replaced by the second or ultimately the last-named directory. Note that only the left-hand, variable part of the directory name must be specified if the subordinate directory structure is laid out in the same way in each case.

```
[Global]
PathReplacements=\\MYSERVER\Development\Packages;\\MYSERVER\Integration\Packages;\\MY...
```

Example:

Development share: [\\MYSERVER\Development\Packages](#)
Integration/Test: [\\MYSERVER\Integration\Packages](#)
Production: [\\MYSERVER\Production\Packages](#)

Sample software package imported from:
[\\MYSERVER\Development\Packages\German\ADOBEREADER\001](#)

Sample software package now located in:
[\\MYSERVER\Production\Packages\German\ADOBEREADER\001](#)

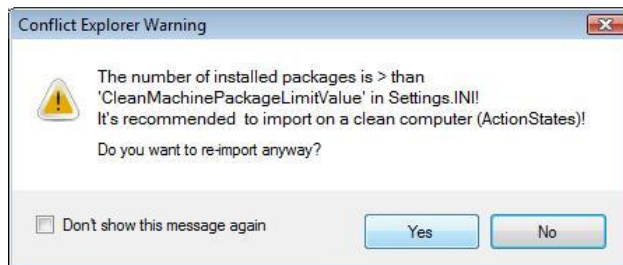
The following entry ensures that *Conflict Explorer* will always be able to access the installation file:
PathReplacements=\\MYSERVER\Development;\\MYSERVER\Integration;\\MYSERVER\Production

CleanMachinePackageLimitValue

This entry specifies the number of installed Windows Installer packages required before a system is no longer defined as a *clean machine*. When importing a package, it is important that this process takes place on a *clean machine*. If this is not the case, the likelihood increases that the *ActionState* may be wrongly identified. Where packages have a filled-in *SelfReg* table (only in such cases will a pseudo-installation take place on the *clean machine*), a prior manual installation of any technical dependencies (InstallScript Engine, for example) is of course possible.

```
[Global]
CleanMachinePackageLimitValue=6
```

If the number of Windows Installer packages installed on the client is larger than the value specified in *Settings.INI*, then a message informing users about this situation will be shown when importing software packages.



If software packages are imported on a computer loaded with many applications, then the likelihood increases that the identification of certain resources will result in an erroneous *ActionState*. If conflicts were to result from such resources, then these would not be modified via an automatic resolution process.

IsolationBeginRegistry

This entry can be used to specify a global threshold, after which component isolation should be carried out for registry conflicts (default value 15). (This value can be overridden by the user setting (Options). It is therefore valid only as an initial value, where the standard setting is not going to be used.)

```
[Global]
IsolationBeginRegistry=10
```

IsolationBeginFiles

This entry can be used to specify a global threshold, after which component isolation should be carried out for file conflicts (default value 3). See also [5.12 Options in the Conflict Explorer user interface](#). (This value can be overridden by the user setting (Options). Valid therefore only as an initial value.)

[Global]
IsolationBeginFiles=2

Extract from *Options*:

Ressources Isolation - Start to Isolate

sum of ressources minus registry conflicts to adjust > than: 15 (default 15)
sum of ressources minus file conflicts to adjust > than: 3 (default 3)

TransformsAll

Corresponds to the option shown below (see also [5.12 Options in the Conflict Explorer user interface](#)). Can be set to "True" or "False" and can be overridden by the user setting (Options). Valid therefore only as an initial value.

[Global]
TransformsDefault=True

Settings for Manual Conflict Resolving

- Apply all imported transform files (set as default)
 Apply manually-selected transform files

TransformsAllReimport

Corresponds to the option shown below (see *Options*). Can be set to "True" or "False". This causes all of the transforms found in the package directory to be used automatically when *re-import* is selected. (This value can be overridden by user settings (Options). Valid therefore only as an initial value.)

[Global]
TransformsAllReimport=True

Re-Import and Automatic Resolve Settings

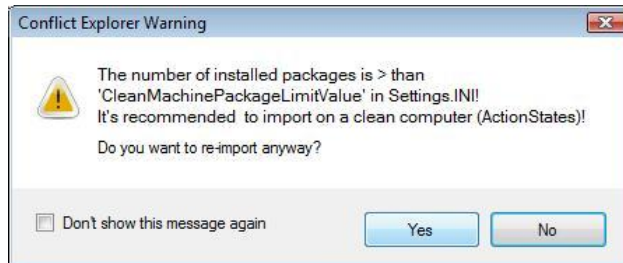
- Apply TRANSFORMS=ALL as additional option (not set as default)

ActionStateToRequestState

This setting controls whether the *ActionState* or *RequestState* for components is identified during the import process. If set to "True", a check is made to see whether a component **is scheduled for installation**. If the entry is set to "False" or omitted, a check is made to see whether the component **would have been installed** by Windows (default setting). Differences arise here

especially when the component is conditional and the *condition* yields a false value, if the component is already installed in a higher version (*KeyPath*) or if the component has been given the attribute [msidbComponentAttributesNeverOverwrite](#).

The setting can be useful when one is running the import process on computers already loaded with a wide variety of installed applications – i.e. when the import process is not being executed on a *clean machine*. If this setting is being used, then note that messages of the following type will not be shown when importing software packages.



Warning: identifying the *ActionState* (default setting, or *ActionStateToRequestState*= "False") is without a doubt the safer method of the two, when you need to clearly identify whether a component will effectively be installed or not. Accordingly, the setting *ActionStateToRequestState* should only be set to "True" in exceptional cases. On the other hand, in a worst-case scenario, the "True" setting carries the risk that *Conflict Explorer*, when resolving conflicts automatically, will resolve certain conflicts "too much" – meaning that their resolution was in fact not even necessary. *Conflict Explorer's* guiding principle of "Fix as much as is needed, but as little as possible!" would not be implemented to 100% in such cases.

Note that any user can use Options to set the setting temporarily, as long as the setting is not present in *Settings.INI*. If it is not the case generally that *RequestState* should be identified in place of *ActionState* then this is the recommended implementation! See also [5.12 Options in the Conflict Explorer user interface](#). **Note: if the entry is set to "True", then this setting – in contrast to the previously-mentioned entries – will not be overridden by the user setting (Options)!** **Warning: the setting has no effect on previously-imported software packages!**

```
[Global]
ActionStateToRequestState=True
```

Import Settings

Use component request state instead action state (not set as default)

AllowedToDelete

This entry lists users who have permissions to delete and move packages, and create *groups*. The list can simply be extended as necessary. As standard, Setup only enters the domain user who carried out the setup process. Any additions must be carried out by the administrator.

```
[Security]
AllowedToDelete=oberlind;username;username
```

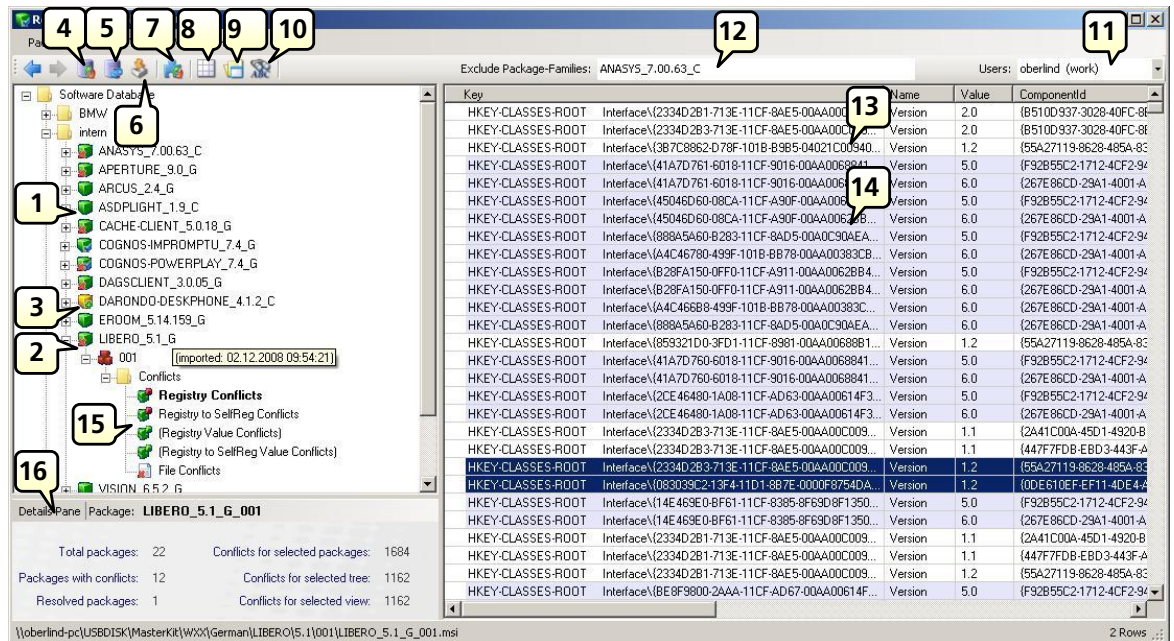
Note:

Settings.INI cannot be edited via the *Conflict Explorer* user interface and should thus only be modified by an administrator. To avoid entering erroneous settings, we recommend setting appropriate ACL rights restrictions on this file or the entire *Conflict Explorer* directory.

5. Using Conflict Explorer

The main focus of this chapter will be to describe the features and functionality provided by the user interface.

5.1 User interface



Legend:

- ① Packages without conflicts
- ② Package with conflicts
- ③ Newly-imported or re-imported package
- ④ Identifies conflicts (one to all). This should be the standard procedure where conflicts are to be identified from a single package.
- ⑤ Identifies conflicts (all to all)
- ⑥ Re-imports the software package
- ⑦ Automatic conflict resolution
- ⑧ Open the MSI with Orca or a similar MSI tool
- ⑨ Using Explorer to open a file path
- ⑩ Searching for conflict information
- ⑪ User list, showing who is working with *Conflict Explorer*
- ⑫ *List of incompatibilities*: no conflict check will be made for any of these packages (*PackageFamily exclude*)
- ⑬ Conflict display
- ⑭ Conflict display (blue: *ActionState="False"*)
- ⑮ Various conflict tree items, grouped by type
- ⑯ Detail pane: various pieces of information

5.2 Importing software packages

Before we can display or correct conflicts between software packages, we first need to import these into the *Conflict Explorer* database. To do so, *Conflict Explorer* needs the following information: where the MSI file is located, whether transforms must be used during the import process (and if so, which ones), the name to use for the software package in the *Conflict Explorer* database, which *FamilyName* the software package will use and, optionally, to which group the software package should be assigned.

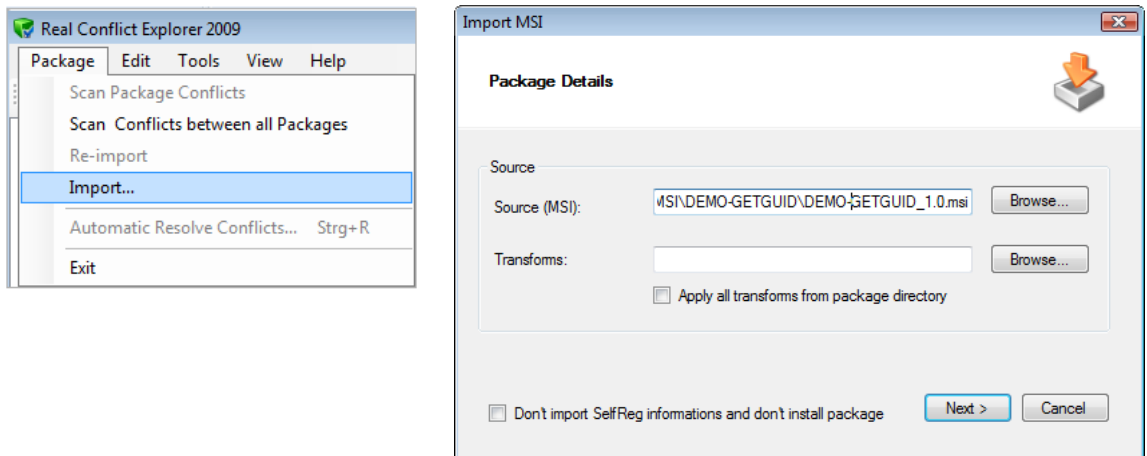
We can distinguish between three different types of import:

1. Importing a software package using the *Conflict Explorer* user interface
2. Importing a software package via *ImportMsi.EXE* and with the use of command line options (see Chapter [4.2.1 ImportMsi.EXE](#) and Chapter [6 Command line options](#)). This score can also be embedded into a *Wise project*.
3. Multiple importing of software packages using a script such as *MultiPackagelImport.vbs*

5.2.1 Importing a software package with the user interface

This is the simplest method to use to import a single software package. However, one should use an automated method for the software deployment process as a whole (see Item 2 in the grey box above).

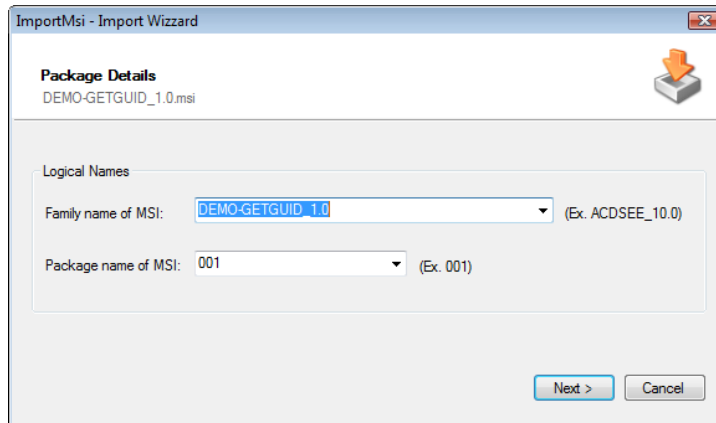
To import via the *Conflict Explorer* user interface, choose *Package/Import...*



An input screen then appears, requesting the name of the MSI file. Use the *Browse...* button to navigate to the place where you have stored the software package. Only specify a transform (second line) if this product has an associated MST file that has been designed for the software deployment process. Alternatively, in environments where the transform is stored in the same directory as the software package, one can simply select the option *Apply all transforms from package directory*. If this option is chosen, all of the MST files found in the package directory will be used for the import process in alphabetical order. This happens independently of the number and existence of transform files stored in this location.

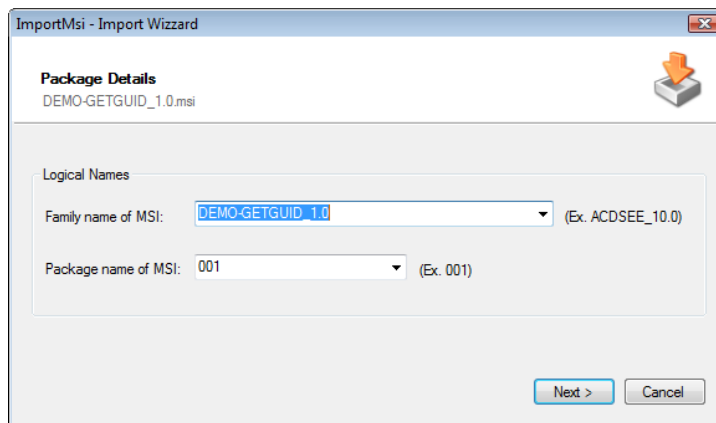
You should not select the option *Don't import SelfReg information and don't install package*. The use of this option is intended for exceptional cases, where the import process should not include *SelfReg* information and the associated pseudo-installation. This sort of scenario is possible if the pseudo-installation triggers an unrequested restart of the system, for example, or disrupts the import process in a similar way. The option will only be visible if the set of MSI file including any MST files contains *SelfReg* information.

In the next steps, a number of different descriptors are requested. If these have already been declared via the keywords *PropertyPackage* and *PropertyFamily* in *Settings.INI*, and the installation package would receive these properties in the *Property* table, then this input screen will already be completely filled out. If the screen is missing descriptors, then these must be entered. Chapter [3 Terminology](#) contains information about structure of the *Family* and a *Package*. Please also note that once a software package is being imported, it can no longer be renamed (for details, see [5.19 FAQ, third entry](#))



5.2.2 Automated import of multiple software packages into existing environments

In the directory `<Conflict Explorer>\Client Setup`, a VB script file (*MultiPackageImport.vbs*) is provided for a company environment: it enables existing software packages to be imported from a file directory without user interaction. This script is intended to be a **template for in-house modifications**. This is because certain aspects of the file structure model, the storage location of the corresponding transform files and the rules for descriptors may differ considerably from company to company. For descriptors, the template script draws on completed *Properties* in the *Property* table. Another possible scenario would be to use other mechanisms to identify the names of the *PackageFamily* and the *Package*, and then to pass these to the *ImportMsi.EXE* call by using the command line parameters `FAMILY=<XY> PACKAGE=<001>`. For all software packages where *ImportMsi* has not been given any valid names and where *Property* implementations are present in the MSI file, and input screen will appear, requesting the user to identify these names.



5.3 Basic standard workflow

1. After finalisation of the software package, the software packager imports the software package via a *Wise* project by clicking on the appropriate project entry, or by using *Package/Import...* from the *Conflict Explorer* user interface.

Note:

Import must be carried out on the operating system that is the target OS for the software package! Accordingly, Vista packages must be imported onto a **clean machine** install of Windows Vista!

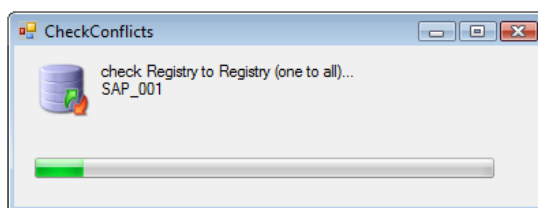
2. The package should be selected using the Explorer-like tree view in the left-hand pane of the *Conflict Explorer* user interface (a package is selected immediately by entering the first letter of its name). Following this, one then expands the tree to reach the *Conflicts* tree item.

Any packages imported but not yet scanned are indicated visually by a yellow *PackageFamily* icon.



As the tree is expanded, the status of the *Scan Package Conflicts* icon changes in the toolbar.

3. Following this, click the icon *Scan Package Conflicts* on the toolbar. This compares this package's resources with the resources of all imported packages. To identify the standard rules used during this process, consult [5.9 Standard rules used when identifying conflicts](#).



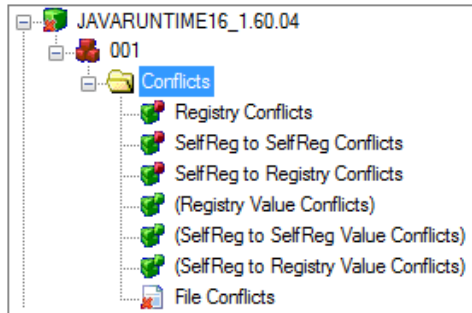
4. Conflicts can now be analysed and structural steps carried out (see [5.5 Excluding conflicts](#)) and/or you may now perform conflict resolution. (See [5.6 Automated conflict resolution](#) and [5.7 Manual conflict resolution](#))

5.4 Conflict types

Conflict Explorer is able to display a number of different types of conflicts. These are grouped both according to resource type (*Registry*, *SelfReg*, *File*) and combinations of these (e.g. *Registry to SelfReg*). They are further differentiated in the following respect: whether they are conflicts that cause problems as a result of different content types of the same resource (*value conflicts*) or conflicts that cause unintended errors when uninstalling a software product.

Pseudo-conflicts – arising from the use of different file versions of the same file within different software packages – are not analysed or displayed; this behaviour is also common to other authoring tools. If a minimum set of guidelines in terms of design and deployment of such files is observed during application development, then upgrading an identical file should never reveal any problems. Windows Installer uses its [File Versioning Rules](#) to guarantee that the latest version is always prioritised on the system.

SelfReg conflicts should be noted in particular, since no competitor application has as yet been able to identify them. Nonetheless, it is precisely this kind of conflict that often leads to the problem of an invalid installation state for another application after uninstalling a software product, or the situation where removing shared resources may result in other applications being rendered non-functional.



5.4.1 Registry Conflicts

These are conflicts where the Registry key (*Root, Key, Name*) is the same in the currently-selected package when compared with other packages and the conditions stated by the [5.9 Standard rules used when identifying conflicts](#) are applicable. One may note here the slightly inconsistent naming system. In truth, "*Registry to Registry Conflicts*" would be more appropriate. However, it has been left as *Registry Conflicts* to enable more rapid identification (shorter text) and because these tree items are the ones with the most conflicts.

5.4.2 SelfReg to SelfReg Conflicts

These are conflicts that arise when *SelfReg* registrations are compared with other *SelfReg* registrations. Best-practice rules are very clear in deprecating the use of the *SelfReg* table. Instead, one should place the equivalent information in the *Registry* table (see [Rule 19](#)). Despite this, we find filled-in *SelfReg* tables on a regular basis – a source of potential problems. Circumventing these problems is the goal of *Conflict Explorer*.

5.4.3 SelfReg to Registry Conflicts

Same principle as above, except that here, *SelfReg* information is compared to *Registry* information.

5.4.4 Registry to SelfReg Conflicts

Same principle as above, except that here, *Registry* information is compared to *SelfReg* information.

5.4.5 (Value conflicts)

There are corresponding *value conflicts* for all registry conflict types. These are

1. (*Registry Value Conflicts*),
2. (*SelfReg to SelfReg Value Conflicts*),
3. (*SelfReg to Registry Value Conflicts*) and
4. (*Registry to SelfReg Value Conflicts*).

As has already been mentioned, the conflicts here involve a situation where the exact same *Key* is used in different packages to refer to different content. Automated conflict resolution will not rectify these conflicts and they remain visible in the view. There is no urgent need to resolve such conflicts, and they must be analysed on a case-by-case basis.

Note:

It should be noted in particular that in each case, conflicts may appear in different conflict tree items. For example, this means that a conflict that has been termed a *Value Conflict* can also be displayed as a *Registry Conflict* (the former has nothing to do with the latter!)

Note:

If similar files containing *SelfReg* code are maintained in a number of software packages in different directories and contain *SelfReg* key references to the file path of the COM file, then one will come across these variant paths in the *Value Conflicts* again.

In such cases we recommend that these kinds of files are stored centrally. For example: under %SystemRoot%\System32, by setting a corresponding *Incompatibility* in *Conflict Explorer* (in so far as a shared installation of these software packages is excluded) or by carrying out a .LOCAL isolation.

5.4.6 File Conflicts

These are conflicts between files, i.e. where a file of the same name from the exact same directory is present in multiple packages and the conditions stated by the [5.9 Standard rules used when identifying conflicts](#) are applicable.

5.5 Excluding conflicts

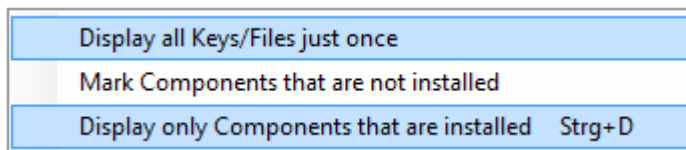
There are several ways of filtering out conflicts or ruling out their display in the first place. These options are not offered by other conflict management solutions.

First of all, one must differentiate between filter operations and the structural methods taken to exclude conflicts.

5.5.1 Filter operations

Filter operations are of a temporary nature and make the conflict display easier to read. Such filters serve to simplify the use of the program and/or increase transparency.

The following context menu entries enable these kinds of filter operations (marked in blue on the screenshot). These menu entries are also to be found in the menu *View*.



The first entry enables the display of matching conflicts so that the conflict is only shown once in the display window. Useful since it can often happen that the exact same conflict is listed multiple times because it conflicts with multiple packages.

The lower menu entry prevents conflicts from being displayed if the corresponding resources would not have been installed by the product (*ActionState*= "False" or *ActionState* of the conflicting package = "False"). Note that the default approach is simply to use a visual differentiation – namely a blue background colour.

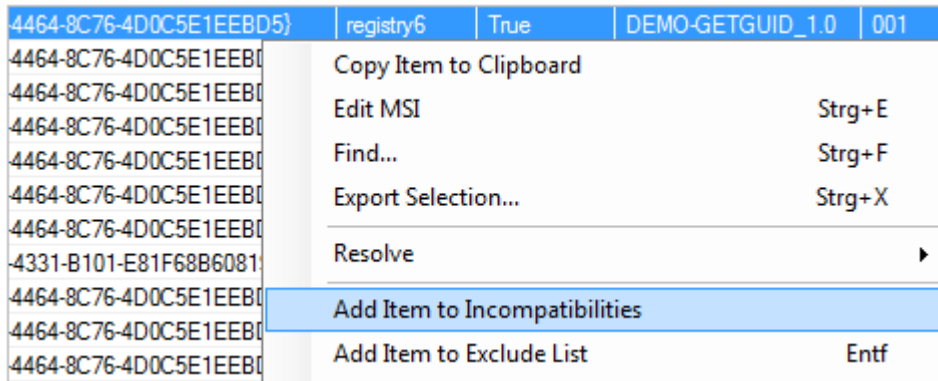
5.5.2 Structural methods

Structural methods define permanent (until subsequently modified) settings used to exclude conflicts without modifying the product. These strategies thus have no influence on the behaviour of the package during installation or uninstalling, but serve to extend the standard rules for excluding

conflicts as regards their comparison to certain software packages. See also [5.9 Standard rules used when identifying conflicts](#).

PackageFamily exclude

Using the context menu entry that appears when right-clicking on a conflict, the menu item *Add Item to Incompatibilities* can be used to extend the *PackageFamily exclude list* with the *FamilyName*. (The mouse cursor need not be pointing to the package name.)



After this operation, you will see that the *FamilyName* is now shown on the toolbar and the conflict display is reduced by all of the conflicts that were referenced by the excluded family.

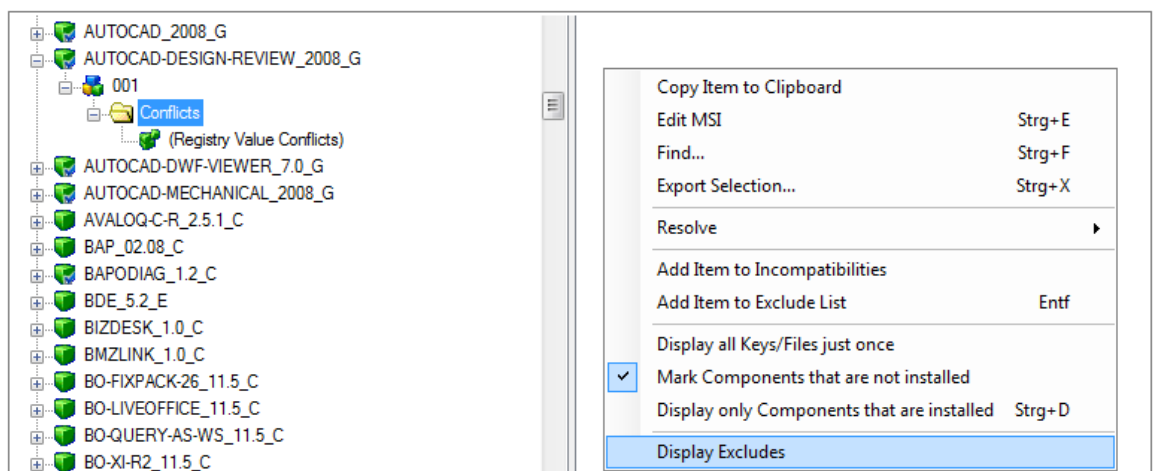
Exclude Package-Families: DWG-TRUEVIEW_17.1.65_G AUTOCAD-DWF-VIEWER_7.0_G AUTOCAD_2008_G

Deleting or amending an entry can be done directly in the toolbar text field, confirming by pressing Enter. Here, too, a *Rescan* is executed in order to update the conflict display.

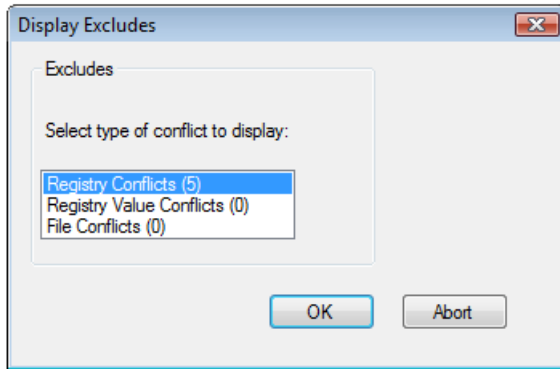
Exclude list

By adding selected conflicts to the *exclude list* (see last screenshot of the context menu – *Add Item to Excludelist*), you may selectively exclude individual conflicts from the display. This operation may also be performed by making a selection and then pressing .

If no conflict tree items are shown in the Explorer view – as a result of all items being excluded – then you can right-click on the tree item *Conflicts* in the conflict display to bring up the context menu (shown here on the right), where you can select *Display Excludes*.



The following screen will then be shown, where the type of excluded conflicts to display can be selected.



5.5.3 Recommended usage

- As a rule, one should assign software packages to *Groups*, each of which represents a set of computer types. Conflicts will then only be displayed if conflicts exist for packages in the exact same *group*. If the same software package is also present in another *group* with the same descriptor, then conflicts of this other *group* will also be taken into account when displaying conflicts!
- If other management tools can guarantee that a software package will not be installed alongside another package on a system, then these rules should also be mapped out in *Conflict Explorer* by using *Incompatibilities*. This is accomplished by extending the *PackageFamily exclude list* for the appropriate package.

Note:

Bear in mind that the *PackageFamily exclude list* must be re-created for each new *package*. Its validity does not therefore extend to the entire *PackageFamily*!

- If one can identify other scenarios where one is sure that the package selected in *Conflict Explorer* will never be installed with the conflicting package, then the *PackageFamily exclude list* should be extended to reflect this.
- If you have scenarios where the package selected in *Conflict Explorer* will be installed **and uninstalled** only together with the conflicting package, you may also extend the *PackageFamily exclude list* in such cases – assuming you are not interested in details of *Value Conflicts* (if such exist).
- For conflicts that one would like to selectively exclude from the view for other reasons, we recommend the use of the *Add Item to Excludelist* function (in the conflict entry context menu). An extended input screen is available for describing the reason in greater detail, so that this is clear to other software packagers in the course of multiple analyses.

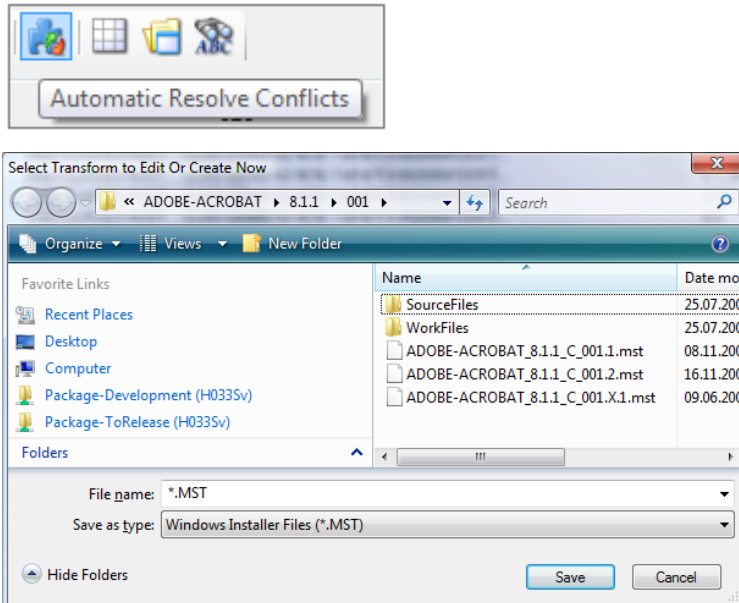
5.6 Automated conflict resolution

Automated conflict resolution is intended to be the standard procedure for rectifying conflicts. During this process, all conflicts except *Value Conflicts* are corrected, with the corrections being written to a transform (MST file).

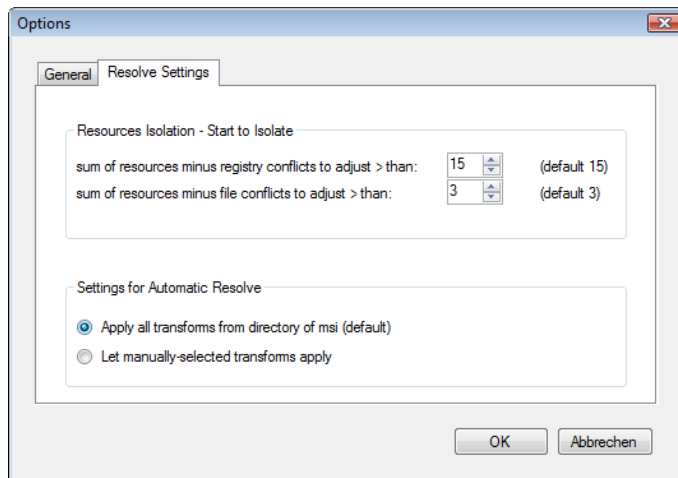
Note:

Before resolving a conflict, please ensure that you execute *Scan Package Conflicts* shortly before doing so. This is because the display of conflicts may be no longer current: group move operations may have been carried out since the last *Scan Package Conflicts*, new packages may have been imported, or existing packages may have been re-imported or removed from the database!

After you click on *Automatic Resolve Conflicts*, a screen is shown, requesting the descriptor for the transform to create.



This process uses settings from the menu *Tools/Options*. The default settings are shown below.



Note:

If an existing MST file is selected in the *Select Transform to Edit or Create Now* screen, then the existing content from this MST file will be preserved. This means that MST files may also be extended with new conflict resolutions!

With automatic conflict resolution, an attempt is first made to achieve a comparison of the *ComponentId*. If this proves unsuccessful, conflict resolution is executed using standard settings (for files, setting the *ShareDll* attribute, otherwise by setting the *Permanent* attribute). As appropriate to the settings used, should a component design be present where not all component resources give rise to conflicts, then component isolation for the conflicting resources will be carried out simultaneously..

Certain presets will also be applied. Accordingly, *Conflict Explorer* attempts resolutions only for components that are not coloured blue, i.e. ones that are marked for installation for both packages (current package and conflicting package) and whose *ActionState* is set to "True".

Note:

In certain exceptional cases (for example, where different resource types conflict within a component) it can happen that residual conflicts remain shown in the display during the first resolution operation. In such cases, you simply need to run automated conflict resolution once more, selecting the MST file you just created in the input screen.

5.6.1 Special rules for Registry references to files in the installation instance

Apart from the rules described above, *Conflict Explorer* will use additional rules in certain circumstances. *Conflict Explorer* will make these modifications during both automated and manual conflict resolution:

For all types of *Registry* and *SelfReg* conflicts (not *value conflicts*), which...

1. ... refer to files that are installed with the same MSI, and...
2. ... where the referenced files do not already belong to a modified component,

a conflict resolution with the *Permanent* attribute (*msidbComponentAttributesPermanent*) will be carried out for the file resources. This additional manipulation will also take place even when the files do not themselves appear in the *File Conflicts* node, since they may well be stored in a directory separate to the conflicting package.

Where COM files contain registry references in the *SelfReg* or *Registry* table, this additional layer of protection is intended to ensure that not only the shared registry keys but also the corresponding files are left on the system, in cases where individual software packages are uninstalled or upgraded.

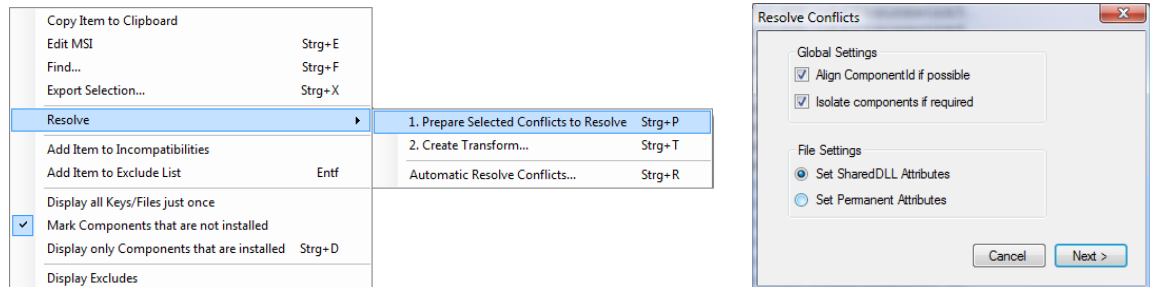
In the reverse case, where file conflicts...

1. ...are referred to in registry entries from *Registry* or *SelfReg* conflicts, and...
2. ... where the referencing registry keys do not belong to a modified component,

no additional resolution of Registry resources will take place. The reason for this stems from the practical limits to programmatic and automated techniques for identifying all of the additional registry keys that should belong to these COM components, in addition to the referencing registry key resources. The case itself is also unlikely to arise: with a file conflict with a file that uses registry keys that refer to itself, additional *Registry* or *SelfReg* conflicts would appear in the corresponding conflict nodes (assuming that the component that contains the Registry resources is not already correctly marked) and a standard conflict resolution scenario would thus be achieved. And naturally, only insofar the files from the different software packages were using the same registry keys: otherwise, an additional resolution would be necessary anyway.

5.7 Manual conflict resolution

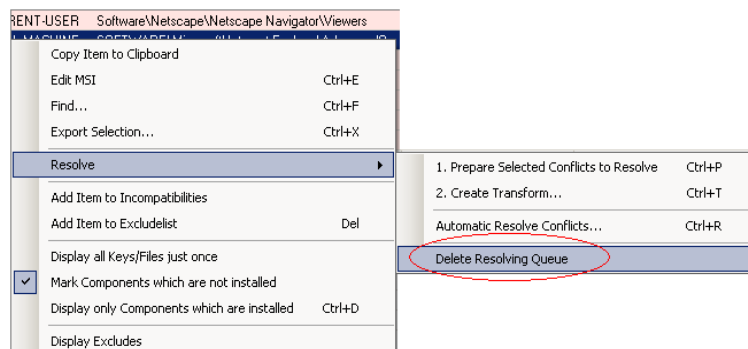
Manual conflict resolution can be used to perform user-defined conflict modifications. In this way, the settings can be modified for each individual conflict selected.



In addition, one may extend or reduce the selection for conflict resolution. To do so, the conflicts to be resolved are marked and selected via the command *1. Prepare Selected Conflicts to Resolve* in the context menu.

All conflicts destined for conflict resolution are marked in red, to aid in their identification. Note that selecting a single line for conflict resolution may result in the colouring of multiple lines in the conflict display. The underlying reason is that a single conflict may be listed multiple times if it conflicts with multiple packages.

The conflict resolution process can now be executed by using *2. Create Transform...* The user will be requested to specify the following: the base MSI, any transforms that should be applied before conflict resolution and the descriptor for the transform to be created.



Delete Resolving Queue

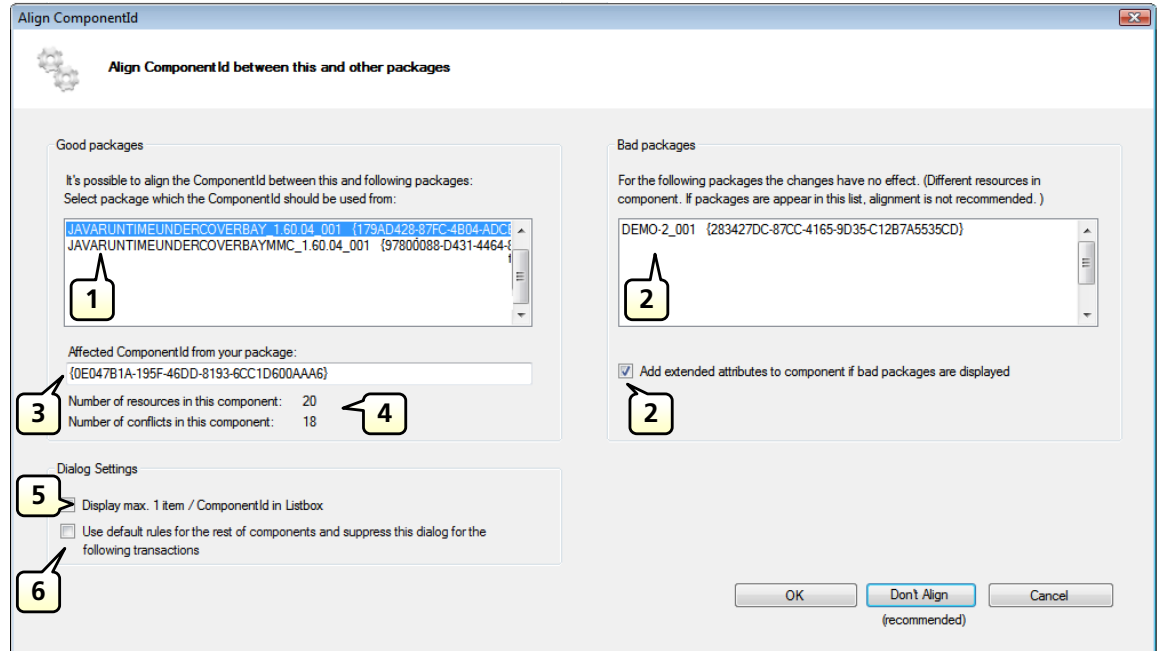
If you right-click the context menu *Delete Resolving Queue* on a red entry, then the queue destined for resolution can be deleted (see screenshot above). Warning: this action will delete the entire queue for the corresponding software package! After the operation, the lines will revert to their usual black on white display.

Warning: Even during manual conflict resolution, the rules described at [5.6.1 Special rules for Registry references to files in the installation instance](#) will still be used.

5.8 Aligning the ComponentId

During the process of aligning the *ComponentId*, a check is made to see whether the component contents causing the conflict in this case are identical: any alignment will then only take place if this is the case. During the alignment process, if *Conflict Explorer* finds exactly one package or an individual *ComponentId* from multiple software packages, which is to be adopted, then alignment takes place automatically without user interaction. However, if the conflicting resource is to be found in multiple software packages, where the conflicting component has the same structure in each case, but different *ComponentIds* are used within the software packages, then the user is

asked to name the *ComponentId* for which an alignment should be carried out. An input screen then appears, containing a variety of details that the user must confirm. Please note that *Conflict Explorer* will activate the button *OK* or *Don't Align* as standard, depending on whether or not "*bad packages*" (see below) have previously been identified.



Legend:

① Good packages

Software packages specified in this list field also use the conflicting resource, and the component content is identical with that of the software package to modify. The *ComponentId* is displayed next to the name. The user is requested to make a choice here (click the desired software package with the corresponding *ComponentId*). Please note that similar *ComponentIds* from different software packages may be displayed. By marking the checkbox *Display max. 1 item / ComponentId in List box* (5), the display is reduced in size, which simplifies the selection process.

② Bad packages

This displays information about software packages that contain the conflicting resource, but whose structure is not identical. Alignment with a software package listed in the *Good packages* list field will therefore not guarantee that the software package is free of conflicts after the operation is concluded. If the *ComponentId* selected from *Good packages* should actually be identical with a software package listed under *Bad packages*, then it is strongly recommended that you do not align the *ComponentId*.

If *Conflict Explorer* finds conflicting software packages which it then displays in this list field, then the button *Don't Align* will be activated in order to recommendation this action to the user. This is made clear by the use of the word *recommended*, placed under the button.

The default setting is to activate the checkbox *Add extended attributes to component if bad packages are displayed*. If the user nonetheless wishes to align the *ComponentId*, then additional attributes are set with this option: *SharedDLL* for file conflicts or the *Permanent* attribute for registry conflicts.

③ **Affected ComponentId from your package**

Display of the *ComponentId* of the resource causing conflict in the software package to be modified.

④ Information about the components causing the conflict in the software package to be modified.

⑤ **max. 1 item / ComponentId in List box**

By selecting this option, the input screen will show all entries in *Good packages* only once per *ComponentId*. It is possible to select and then deselect the checkbox in order to see what difference this makes to the display. The aim is to simplify the selection procedure for the *ComponentIds* to be aligned. (See also *Good packages*)

⑥ **Use default rules for the rest of components and suppress dialog for the following transactions**

By selecting this option, you can prevent the repeat appearance of this input screen during the whole modification transaction. Instead, standard settings are used to align the *ComponentId*, and the input screen is confirmed with **OK**:

1. If only one target *ComponentId* is found in later stages, and no *Bad package* identified, then the *ComponentId* found will be aligned.
2. If several target *ComponentIds* (*Good package*) are found, then the most frequently occurring *ComponentId* will be aligned.

Please note that a selection must be made from the *Good packages* list field for the component that is currently being focused on.

After selecting the option *Use default rules...* and confirming the input screen with **Don't Align**, then no adjustment to the *ComponentId* will take place for the current component, and all further components where multiple target *ComponentIds* (*Good package*) are found. However, if a **single** target *ComponentId* is found later in the transaction for a component to be aligned, and no *Bad package* identified for this, then alignment will be carried out as normal for the *ComponentId* identified.

Should you require information from the conflict display of the software package while the input screen is being shown, then you can activate *Conflict Explorer* – running in the background – and navigate as required in the conflict view or execute other functions. Please note that you cannot start a second resolution process during this phase. In addition, before concluding your activities in the input screen, you should switch back to the software package that had the original focus in the conflict display, if you have selected a different package in the tree view in the meantime.

5.9 Standard rules used when identifying conflicts

Clicking on *Scan Package Conflicts* will trigger the use of the following rules. The conflict display on the right-hand side of the user interface is the result of these check rules:

1. Conflicts between two products from the same *FamilyBase* (identical name portion before the "_") will be hidden. For example: Conflicts between MOZILLA-FIREFOX_2.00.9, MOZILLA-FIREFOX_2.00.11 and MOZILLA-FIREFOX_2.00.12 will not be shown.
2. Conflicts between Packages (e.g. 001 to 002) do not count as conflicts
3. Conflicts between resources from packages where the underlying components utilise the exact same *ComponentId* do not count as conflicts (*Value Conflicts* excepted)
4. Conflicts between resources from packages where the underlying components utilise the *msidbComponentAttributesPermanent* attribute do not count as conflicts (*Value Conflicts* excepted)
5. Conflicts between resources from packages where the underlying components utilise the *msidbComponentAttributesSharedDllRefCount* attribute do not count as conflicts (*Value Conflicts* excepted)
6. File conflicts where the file originates from %WINDIR%\System32 do not count as conflicts (Windows Installer enters these into the *SharedDLL* list automatically)
7. Conflicts from packages sourced from different *groups* are hidden, in cases where the selected package is not present in both groups
8. Conflicts that have been entered into the *exclude list* (*Conflict Explorer* function) will be hidden
9. Conflicts where the *PackageFamily* has been entered into the *PackageFamily exclude list* (*Conflict Explorer* function) will be hidden
10. Registry conflicts containing the "+" sign in the "Name" column will be hidden (equivalent to a dummy statement for Windows Installer, used to mark *keys* that are not to be deleted when uninstalling)

5.10 Rules when forming the KeyPath for new components

If component isolations have been carried out as a result of conflict resolution, then the following rules are used to create the *KeyPath* entry for a new *ComponentIsolationX* component in the transform. Rules are used in the order given: as soon as a rule is matched, then the matching rule is used (no further rules will apply):

1. If the original component from which the resources were isolated did **not possess a KeyPath**, then the new component will **not possess a KeyPath** either
2. If the resource belonging to the *KeyPath* of the old component is now found in the isolated component, then this *KeyPath* will be used in the new *ComponentIsolationX* component
3. If the new component contains *File* resources, then the first identified .EXE file is used. If no .EXE file is found, the first-identified .DLL file is used. If no .DLL file is found, the first-identified file with an unknown extension is used as the *KeyPath*.
4. If it still proves impossible to identify a *KeyPath*, then a Registry *KeyPath* is used, assuming the entry does not correspond to the following schema:
 - a. Value=0 AND
 - b. (Name= + OR Name= - OR Name= *)

See [msidbComponentAttributesRegistryKeyPath](#)

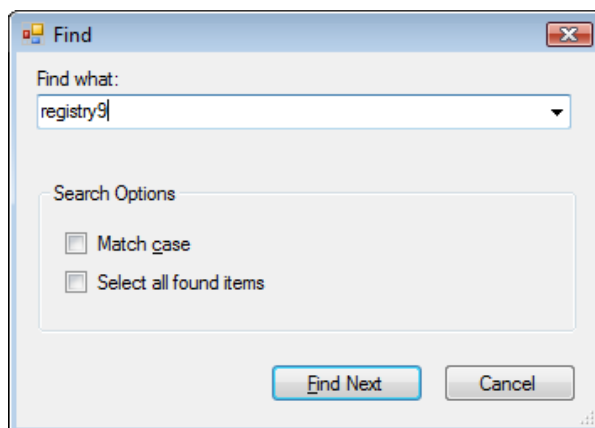
...If the Value field of the corresponding record in the Registry table is null, the Name field in that record must not contain "+", "-", or "*". For more information, see the description of the Name field in the Registry table.

The following rules apply to the old component, from which the resource(s) have been removed:

1. If the original component from which the resources were isolated did **not possess a KeyPath**, then no further operations take place
2. If the resource belonging to the *KeyPath* of the old component is still found in the old component after the isolation transaction, then no further operations take place
3. If the old component contains *File* resources, then the first identified .EXE file is used. If no .EXE file is found, the first-identified .DLL file is used. If no .DLL file is found, the first-identified file with an unknown extension is used as the *KeyPath*.
 - d. If it still proves impossible to identify a KeyPath, then a Registry KeyPath is used, assuming the entry does not correspond to the following schema:
Value=0 UND
 - e. (Name= + OR Name= - OR Name= *)

5.11 Search

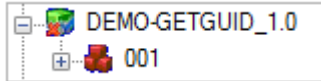
The search window may be opened via the search icon in the toolbar or by using the context menu in the conflict display. This may be used to search through all content in the conflict display for a search string. The search does not extend to multiple conflict tree items: it is restricted to the currently-selected conflict type!



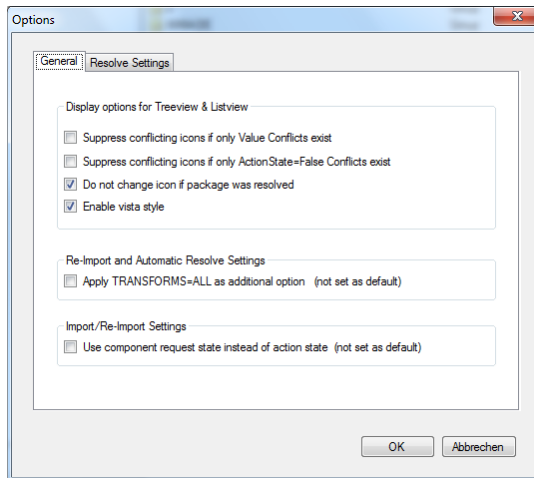
5.12 Options in the Conflict Explorer user interface

5.12.1 General tab

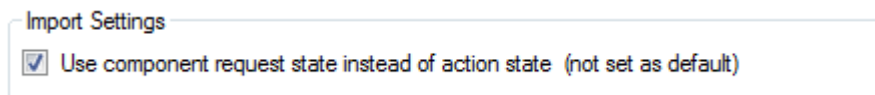
The options on the *General* tab (found under *Tools/Options*) allow users to modify the visual appearance of the *Family* and *Package* icons in the tree view, so that *Value Conflicts* or conflicts arising from components marked with *ActionState="False"* can be displayed without conflicts.



This option can be set as desired by each individual user. In addition, the same tab screen is used to specify whether all of the MST files located in the package directory should be used when *re-importing* the software package – via the icon in the toolbar or by using the menu item *Package/Reimport (Reimport Settings)*



The option *Use component request state instead of action state* controls whether the *ActionState* or *RequestState* is identified for components during the import process.



If the option is checked, a check is made to see whether a component **is assumed to be present** for installation. If not checked, a check is made to see whether the component **would have been** installed by Windows (default setting). Differences arise here especially when the component is conditional and the *condition* yields a false value. Other differences are found if the component is already installed in a higher version (*KeyPath*) or the component has been given the attribute [msidbComponentAttributesNeverOverwrite](#) and the underlying resource for the *KeyPath* is already installed on the computer.

The setting can be useful when one is running the import process on computers already loaded with a wide variety of installed applications – i.e. when the import process is not being executed on a *clean machine*. If this setting is being used, then note that messages of the following type will not be shown when importing software packages.



Warning: identifying the *ActionState* (default setting, or *ActionStateToRequestState*= "False") is without a doubt the safer method of the two, when you need to clearly identify whether a component will effectively be installed or not. Accordingly, this setting should only be set to "True" in exceptional cases. On the other hand, in a worst-case scenario, checking the option carries the risk that *Conflict Explorer*, when resolving conflicts automatically, will resolve certain conflicts "too much" – meaning that their resolution was in fact not even necessary. *Conflict Explorer's* guiding principle of "Fix as much as is needed, but as little as possible!" would not be implemented to 100% in such cases.

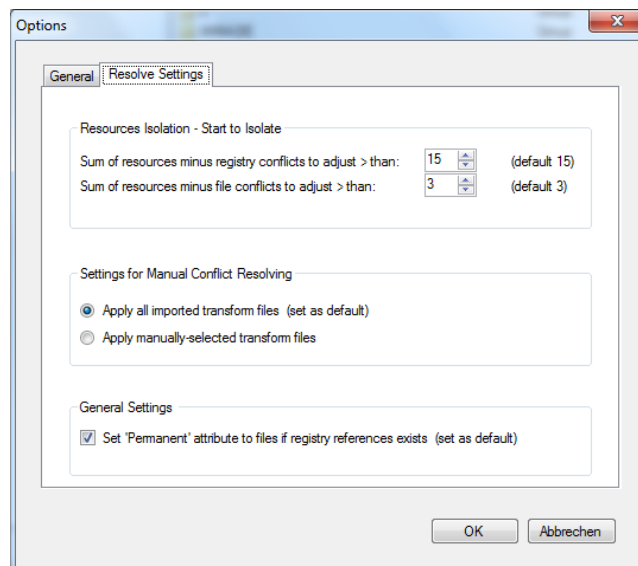
The setting can also be set permanently via *Settings.INI*. See chapter [4.2.4 Settings.INI](#).

Warning: the setting has no effect on column descriptors within *Conflict Explorer*! ActionState

5.12.2 Resolve Settings tab

The *Resolve Settings* tab is used in certain scenarios to modify the standard settings for conflict resolution. The standard settings for resource isolation reflect the following compromise: an isolation is carried out only once the underlying component contains significantly more resources than the sum of conflicting resources of this component. By increasing the value, the number of iterations needed for *Conflict Explorers* to achieve its goal will be reduced, but the likelihood that modifications will be made to resources that did not really need modifying will be increased. By lowering the threshold value, *Conflict Explorer* behaves in the opposite fashion, as would be expected.

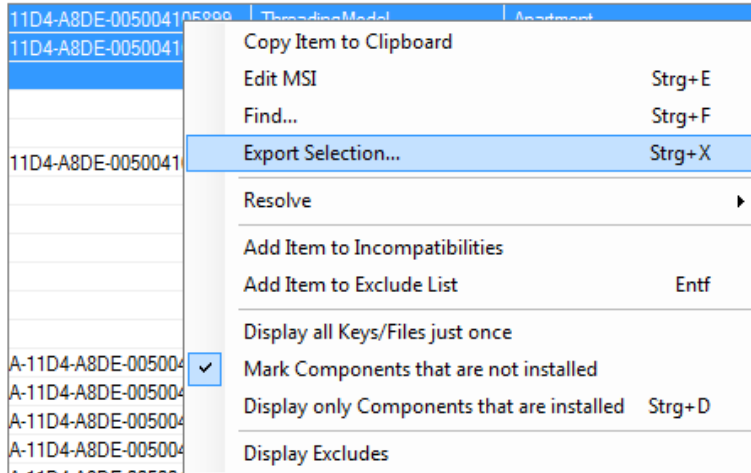
The setting under *Settings for Manual Conflict Resolving* controls the handling of MST files during automatic conflict resolution. With the option *Apply all imported transform files*, *Conflict Explorer* will apply all previous imported MST files automatically, before it resolves any conflicts. With the second option, an extra input screen will be shown for manual conflict resolution, where one can select the MST files that should be applied before conflict resolution. You should always apply the same transforms during conflict resolution as were used in the import process!



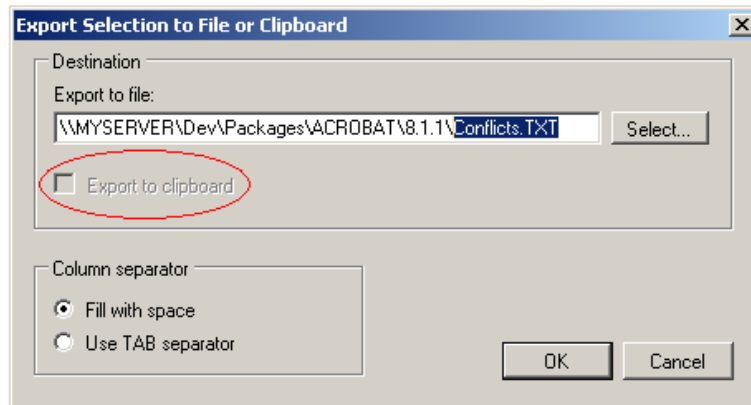
5.13 Exporting conflicts

After conflict resolution, the conflict view is usually updated and conflicts disappear from the display. However, it will occasionally be interesting to know about the conflicts present before conflict resolution. Equally, where a software release process uses the "division of responsibility" QA method and an additional person is involved, he or she might like to know about the types of conflict involved in an unchanged setup, occurring before manipulation by the software packager.

In such cases, it is possible to mark the corresponding lines in the conflict display in which one is interested. Right-click and choose the context menu item *Export Selection...*



An input screen now appears, in which you can select a file name. Alternatively, you can use the option *Export to clipboard* to redirect the output to the Windows clipboard.



5.14 Moving, copying and deleting packages, creating groups

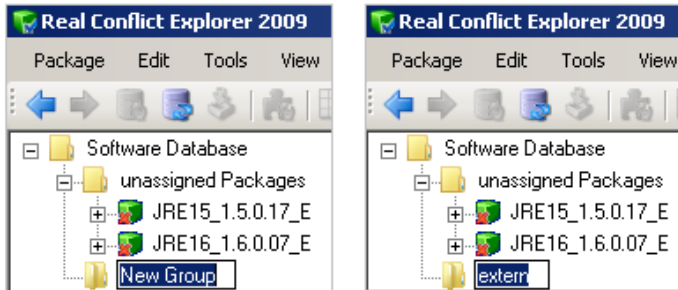
Conflict Explorer simplifies common organisational management tasks. It takes just a few steps to move software packages into other *groups* or to copy them. Delete operations are equally simple to carry out. Here, too, the user interface is similar to that found in *Windows Explorer*.

5.14.1 Creating or renaming groups

By right-clicking on the *Software Database* tree item, a context menu opens up that enables you to create a new *group*.



To complete the operation, choose a name for the group.

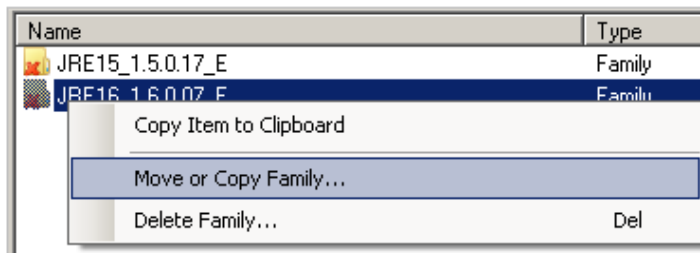


If you click twice (not double-click) on an existing group name, then you will be able to rename existing *groups*. The screen will then change as shown in the last screenshot.

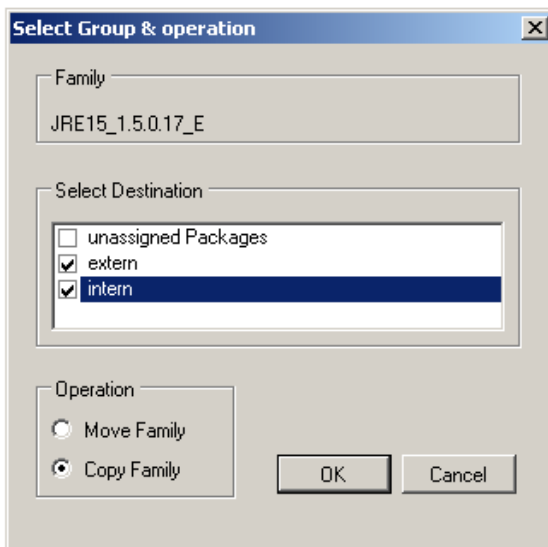
5.14.2 Moving or copying package families

You can move and copy software package families with the mouse, simply by using *drag-and-drop*. You need only click on the family name, drag it to the *group*, and then release the mouse button. If you are selecting via the list view, then you can select multiple packages simultaneously before moving or copying them. When using *drag-and-drop*, pressing and holding the *Ctrl* key means that the links will be copied. If the key is not pressed, they will be moved instead.

The same task can also be achieved via the context menu. Right-click on a family name to access the context menu ...



... and then select the desired operation. Here, copying and moving operations can be applied to multiple *groups*.



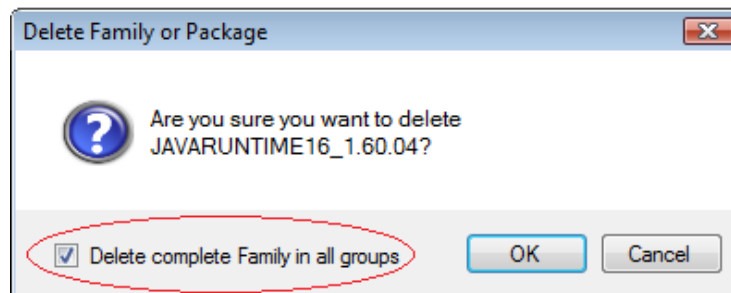
Note:

Only complete *families* may be copied and moved. Associated *packages* will also be copied or moved by *Conflict Explorer* automatically!

Please note that copying and moving operations affect conflicts. You should carry out a *rescan* of the selected packages after such operations, and also before any analyses or conflict resolutions.

5.14.3 Deleting package families

Select one or more software package families and then press the key. The following confirmation screen appears:



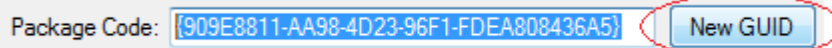
If the package family/families is/are found in multiple groups, then the checkbox *Delete complete Family in all groups* will be activated. By checking this checkbox, you can delete the package family from all referenced groups.

Note:

Deleting, copying and moving package families is a delicate task. In a team setting, it should only be carried out by an administrator. Directly after *Conflict Explorer* has been installed, only the person who performed the installation is entitled to carry out these kinds of operations. You may grant other persons these permissions if needed via the *AllowedToDelete* entry in *Settings.INI* (see also 2.2.4 *Settings.INI*).

5.15 Conflicts with the operating system

Conflict Explorer may also be used to identify potential conflicts between software packages and the operating system. Note that currently, *Conflict Explorer* cannot identify operating system resources directly, nor can it import them into the *Conflict Explorer* database. However, this is planned for a later release. Despite this, you need not do without this function. Within *Wise Package Studio*, create what's known as an "SOE Snapshot" of the operating system (*Clean Machine*), containing all current *service packs* and *patches*, but without any installed software. Rename this file as an .MSI file, and then edit this with *Orca.exe*. Create a *PackageCode* in the *Summary Information* and delete the *SelfReg table*:



If the *Directory Table* has an entry with C_ |C: then delete this too.

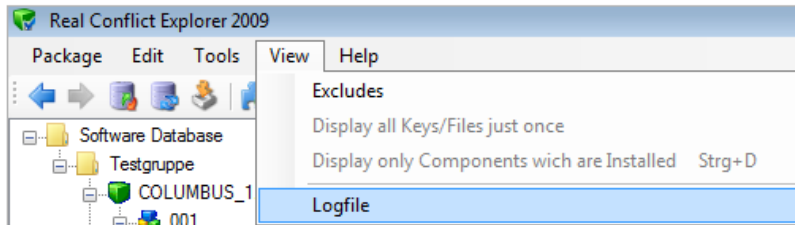


Import this .MSI file into *Conflict Explorer*. Once it is imported, proceed as normal. Identify the conflicts, and you will then be able to see the operating system conflicts under the tree item of the "operating system package" you have just imported. Consequently, in the software packages conflicting with the operating system you will find conflicts that you can resolve simply by using *Automatic Resolve Conflicts*. Note that you must not carry out any conflict resolution for the operating system package: in all such cases, correct only the software packages and not the operating system!

5.16 Log file

Particularly in cases where you are using *Conflict Explorer* in a team, you may occasionally need to find out which team member carried out which operation – and at what time. Such questions may be answered by looking at the log file.

Select *View/Logfile*

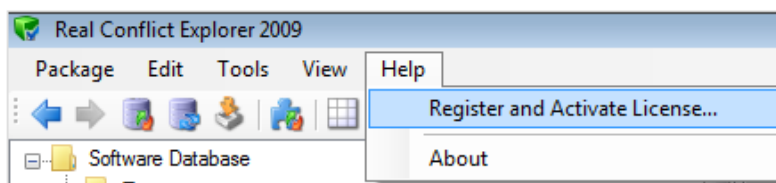


Corresponding information will be shown in the right-hand window pane.

Time	User	Task	Package	Speed
11.12.2008 14:04:27	oberlind	Delete Family	COLUMBUS_1.8.2_E	
10.12.2008 14:50:38	oberlind	Scan all Packages		2sec
10.12.2008 14:49:58	oberlind	Import Package	COLUMBUS_1.8.2_E_001	18sec
10.12.2008 14:44:04	oberlind	Delete Family	COLUMBUS_1.8.2_E	
10.12.2008 14:43:46	oberlind	Import Package	COLUMBUS_1.8.2_E_001	9sec
10.12.2008 14:43:41	oberlind	Delete Family	MSREPORTVIEW_8.0_E	
10.12.2008 14:43:06	oberlind	Delete Family	LMS-EXCELPRINT_1.0_E	
10.12.2008 14:16:57	oberlind	Import Package	EXPRESSO_3.0_E_001	11sec
10.12.2008 14:15:51	oberlind	Import Package	GUICC_1.0_E_001	17sec
10.12.2008 14:14:07	oberlind	Import Package	LMS-EXCELPRINT_1.0_E_001	59sec
10.12.2008 13:59:57	oberlind	Import Package	MSREPORTVIEW_8.0_E_001	14min 5sec
10.12.2008 13:59:25	oberlind	Import Package	NETVIZGUI_7.5_E_001	16sec
10.12.2008 13:58:18	oberlind	Import Package	PASSPORT-PC-TO-HOST_2.00.7_E_001	13sec
10.12.2008 13:56:24	oberlind	Import Package	PASSWORD-SAFE_3.13_E_001	50sec
10.12.2008 13:55:34	oberlind	Import Package	PF-HOUSEKEEPING_2.2_E_001	28sec
10.12.2008 13:53:51	oberlind	Import Package	PF-SCANNAPPLICATION_5.0.5_E_001	1min 22sec
10.12.2008 13:53:09	oberlind	Import Package	PKS-PLOG_4.3_E_001	11sec
10.12.2008 13:52:22	oberlind	Import Package	PPTVIEW2007_12.0_E_001	20sec
10.12.2008 13:51:13	oberlind	Import Package	SAFESIGN_3.0.11_E_001	18sec
10.12.2008 13:50:24	oberlind	Import Package	SCANDALL_21_E_001	18sec
10.12.2008 13:49:10	oberlind	Import Package	SENTINEL-SYSTEM-DRIVER_5.39.2_E_001	7sec
10.12.2008 13:46:13	oberlind	Import Package	TRANSLATOR_1.2_E_001	17sec

5.17 License activation process

Conflict Explorer 2009 is available only as a downloadable evaluation version. This evaluation version is subject to certain limitations. If you decide to purchase *Conflict Explorer 2009*, proceed as follows. On a computer with an installed mail client, choose *Register and Activate License...*



Next, choose the vendor whom you would like to use. By selecting *Seller Information*, you can purchase the product from one of our contractual partners or from your *Conflict Explorer* contact person. In such cases, you will need to know his or her email address. Or you can acquire the product directly from us: in this case, leave the setting at its default value.

Next, choose the product version you would like to purchase under *Order Information*. You will find an overview of prices on our website at www.realpackaging.net.

Take care not to calculate margins too tightly. If you have 90 software packages currently in use, for example, it will be more future-proof and indeed more cost-effective to acquire a more powerful product version than *BASIC*. However, if you have activated a less powerful product version, you can also upgrade to a more powerful version of the product at a later date.

You can complete your order by clicking *Order activation by mail*. This generates an email that we will reply to, including your invoice and invoiced amount, further information and our bank details. As soon as we have received your payment, we will send you an *activation pack* which you will then need to install. Following this, your product will be unlocked for the corresponding product version.

Register & Activation

Select activation options

Seller Information

I want to activate Conflict Explorer by reseller (mail to reseller)

I want to activate Conflict Explorer by RealPackaging.net (mail to us)

Order Information

EVALUATION (free for max. 30 package imports, 60 days)

BASIC (max. 100 package imports - fee required)

PROFESSIONAL (max. 200 package imports - fee required)

ENTERPRISE (unlimited package imports - fee required)

Currently activated to EVALUATION feature set.

Serial key (send print screen from this dialog if mail activation does not work)

REROL PXE01 TCILF NOC02 MC-AT SIV02 02

activation@realpackaging.net

Order activation by Mail Cancel

Note:

If you subsequently move the product to a new platform, then *Conflict Explorer* will stop working. In such cases, you can simply re-activate *Conflict Explorer*. Re-activation is free of charge and is generally completed within 24 hours. Please re-activate the product using the same email account with which you carried out the original activation – this simplifies request handling at our end.

Note:

Ensure that you only carry out activation once you have performed a successful install of the evaluation version on the platform on which you intend to use the software. Any subsequent changes on the server or to the directory structure can be made only by carrying out re-activation!

Product activation is generally a one-off operation, after which all team members in the company then have the right to use this product freely within the corporate network.

5.18 Shortcut keys

- Ctrl-A** In the conflict display pane, Ctrl-A will select all conflicts having *ActionState*="True". If Ctrl-A is then pressed again, all entries will be selected.
- Del** In the conflict display pane, pressing causes the conflict to be included in the *exclude list*. Applied to the *exclude list*, the deleted conflict will be once again available for the standard view. Applied to a *package*, it deletes a *package*, applied to a *FamilyName*, it deletes the *family* along with all of its *packages*. Applied to a *group*, it deletes the entire *group*.
- F5** If the tree view (left-hand window pane) or the list view (right-hand window pane) is active, F5 triggers a refresh and a re-reading of package data.
- F8** If a group was selected, F8 select the next imported family of this group (yellow).
- Backspace** In the tree view and list view, this selects the superordinate tree item.
- F3** In the conflict display window, F3 triggers a search for the next match for the last-entered search term.
- Ctrl** If pressed shortly before accessing *Edit MSI* and released after the application has started, this copies the path of the MSI file to the clipboard. A scenario where this can be useful is when one has started *Orca* and then wishes to apply a transform. Often, the input screen that follows has the last-selected directory active, which is in many cases not desirable. Here, one can simply press Shift-Insert in the file name input field, and then press <Enter>, giving immediate access to the path.
- Shift** Display imported MST in status bar of conflict view.

All other shortcut keys are documented in the menu or context menu.

5.19 FAQ

- Q: Can I also use *Conflict Explorer 2009* to import an MSP patch file?
- A: No. However, a patch can be applied to the base installation (MSI) via an *in-place update*, and this can then be subsequently imported. (`msiexec /p /a`)
-
- Q: How are resources handled that are stored in *custom actions*?
- A: For *Windows Installer*, the execution of *custom actions* is without a doubt one of the most complex and critical procedures during the installation process. The contents of *custom actions* remain hidden even from *Windows Installer: Conflict Explorer* is therefore also unable to identify these resources. *Custom actions* should always be implemented in a data-driven way, so as to follow best-practice rules, i.e. *custom actions* should not manipulate resources directly, but only indirectly. In such scenarios – and where the necessary resources are present in the database tables – *Conflict Explorer* can then access all resources.
- Nevertheless, we must on occasion concede to compromises with *Conflict Explorer*. If a number of resources are in fact implemented directly by *custom actions*, then these are simply not taken into account. These resources are generally few in number and thus a tiny proportion of the number of resources that are in fact accounted for by *Conflict Explorer*.
-
- Q: Why can I rename *package families*, but not *packages*?
- A: A number of subsequent tasks within *Conflict Explorer* depend on the naming schemes used when importing software packages. It's not only the case that the *PackageFamily exclude list* depends on established names and that other conflict views reference

existing names. Transactions such as a *re-import* are also dependent on naming schemes. If someone were to rename a software package whose naming scheme was originally derived from reading properties out of the MSI file's *Property* table, for example, how then could the re-import process complete successfully for the renamed package? Should it delete the existing renamed package and re-create the original descriptor (by reading it from the *Property* table)? Or should it simply ignore the *Property* table?

We are of the opinion that both options would result in confusing the user. Accordingly, we have omitted this function from *Conflict Explorer* deliberately.

If packages are to be renamed, then these should be imported as new packages with different names. The *packages* with the old descriptors can then be deleted from *Conflict Explorer*.

Q: Six months ago, a software package was imported and corrected automatically. At the time, the resulting package was free of conflicts. This conflict-free software package was passed to production and installed on the company's computers.

In the meantime, additional new software packages have been imported and the old software package now contains new conflicts in *Conflict Explorer*. Why is this so, and what should be done?

A: Within software conflict management, one generally accepts certain compromises. Since new software packages have been imported during the last six months, it is certainly possible that the old software product may now have new conflicts with the new software products. Whether a company should take action varies from case to case – as does the action to take. One possibility is as follows: if the conflicts are particularly bad, one should re-initiate conflict resolution and – for reasons of transparency – ensure the results are stored in a new transform file. For new installs, the newly-corrected software package would then be the one to use.

It's possible that corporate policy prevents such subsequent corrections. In this case, rectification would only be possible with an update or with a new release. However, the use of *Conflict Explorer* is also justified in this kind of scenario. It is greatly preferable to deploy a software package that has maybe had thousands of conflicts resolved (initial release), than not to use any conflict management at all.

Q: I have imported a software package X. This has many conflicts with software package Y. Why then does software package Y show no conflicts with software package X?

A: While conflicts arise between sets of software, this does not necessarily mean that the conflicts in such cases are reciprocal. In the case mentioned, the implication is that the underlying components have been marked correctly via attributes or that conflict resolution has already been carried out for the second software package Y – but not for software package X. *Value conflicts* are reciprocal, however. Accordingly, if a *value conflict* is indicated for one software product, then the corresponding *value conflict* must also be displayed for the conflicting software package – unless one has included this conflict in the *exclude list* or extended the *PackageFamily exclude list*.

Q: For test purposes, I have imported a software package twice – using a different name on one occasion. Why then are no conflicts displayed after a *Scan Package*, even though the packages are in the same group?

A: Since the *ComponentIds* in the component table of the underlying MSI file are also unchanged and no changes have been applied – as a result of which a *value conflict* could have arisen – no conflicts are present. *Conflict Explorer* is thus reacting appropriately.

6. Command line options

ImportMsi.EXE and *ConflictExplorer.EXE* can be controlled via command line options. The use of command line options makes it possible to implement automated process workflows. In particular, this enables elegant integration into *Projects* supplied by *Wise Package Studio*.

Command line options for *ConflictExplorer.EXE*

Command line option	Description
FAMILYNAME PACKAGENAME Example: ConflictExplorer.EXE ADOBEREADER_9.0 001	Opens <i>Conflict Explorer 2009</i> at the selected tree item. The software packager no longer needs to navigate to the package in question. The first entry found in the first possible group is selected.

Command line options for *ImportMsi.EXE*

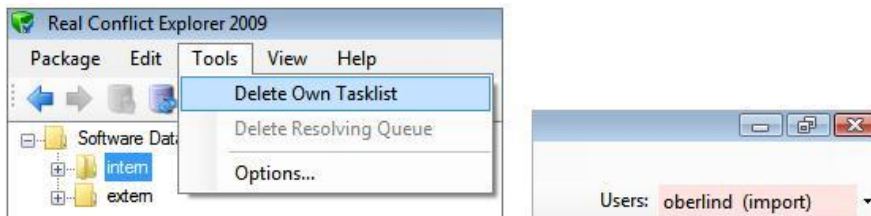
Command line option	Description
/S/Q Example: ImportMsi.exe package.msi /S	Executes the action without user interaction (as long as both family and package names can be identified). Note in particular that no error messages are displayed. This option is recommended for automated multiple package import via a script.
/M Example: ImportMsi.exe package.msi /M	Skips checking to see whether a software package is already installed on the computer before installation, or whether the computer matches the <i>clean machine</i> profile (see <i>CleanMachinePackageLimitValue</i> in 4.2.4 Settings.INI). Any corresponding error messages are not shown.
/A Example: ImportMsi.exe package.msi /A	Identifies the RequestState of components when importing, instead of the ActionState (see <i>ActionStateToRequestState</i> in 4.2.4 Settings.INI)
/?	Displays the most important command line options
/E	Option for script-based Multi-Package-Import. More details by request.
FAMILY=FAMILYNAME Example: ImportMsi.exe FAMILY=ADOBEREADER_9.0	Imports the software package with the specified family name. If <i>FamilyName</i> and <i>PackageName</i> have been specified by command line options or indirectly via the <i>Property</i> table, then no input screen will be shown for specifying the names.
/P	Import w/o selfreg detection (w/o installation)

<p>PACKAGE=PACKAGENAME</p> <p>Example: ImportMsi.exe FAMILY=ADOBEREADER_9.0 PACKAGE=001</p>	<p>Imports the software package with the specified package name. If <i>FamilyName</i> and <i>PackageName</i> have been specified by command line options or indirectly via the <i>Property</i> table, then no input screen will be shown for specifying the names.</p>
<p>GROUPS=Name[;Name[;...]]</p> <p>Example: ImportMsi.exe package.msi GROUPS=German</p>	<p>Here, you specify the group names with which the software package should be linked. Separate multiple group names with a semicolon ";". If the group names cannot be found or are not specified by this command line option, then the software package is assigned to the "<i>unassigned packages</i>" group.</p>
<p>PROPERTYFAMILY=PROPERTYNAME PROPERTYPACKAGE=PACKAGENAME</p> <p>Example: ImportMsi.exe package.msi PROPERTYFAMILY=CEFAMILY PROPERTYPACKAGE=CEPACKAGE</p>	<p>Use this option if you use <i>properties</i> in the <i>Property</i> table for assigning names. This is the recommended variant for an automated packaging process.</p>

7. Important functions not yet covered

7.1 Delete own task list

With *Delete Own*, from the menu *Tools*, you can delete all of your own tasks, so that these do not block any other processes. This is only really necessary if processes such as *ImportMsi* have not completed in their entirety.



Conflict Explorer 2009 allows you to delete only your own tasks.

7.2 Help

Help documentation is stored in the *Help* directory. If a PDF reader is installed and Regional and Language Options are set to German or English, then this document can also be called up via the *Help/Help...* menu.

8. Keyword index

%

%WINDIR%..... 27

.

.NET framework 8
.NET permissions..... 8

A

Access
 read 9
ACL
 rights limitations..... 13
ActionState..... 7, 11, 13, 19, 23, 29
ActionStateToRequestState 12, 30
Activation
 license..... 34
Activation pack 35
Add Item to Excludelist 20, 21
Add Item to Incompatibilities 20
AddFullTrustingAndShortcuts.vbs..... 8
Administrator 13, 33
Administrator rights..... 8
Aligning ComponentId 24
AllowedToDelete 13, 33
Application life cycle 4
Apply all transforms from package directory..... 15
AppSearchproperties 7
Automatic conflict resolution 14, 26

B

Bad packages 25
BASIC..... 35
Best-practice rules 18
Blue 14
Blue background colour 19

C

Check rules..... 27
CheckConflicts.EXE..... 8, 10
Clean machine 11, 13
CleanMachinePackageLimitValue 11, 38

Clear text 7
Client setup..... 8
COM components..... 3
COM file/s..... 19, 23
Command line options 9, 38, 39
 ImportMsi 9
Component..... 13
Component contents..... 24
Component design
 defective 4
Component isolation/s 11, 27
ComponentId 25
 aligning the..... 24
 comparison 22
ComponentIds 37
ComponentIsolationX 27
Components 27
 coloured blue 23
Computer type
 set 21
Condition 13
Conflict
 multiple 19
 once..... 19
 proportion 36
Conflict display..... 14
 extended 7
Conflict Explorer
 re-activating 35
 team use..... 34
Conflict management 6
Conflict resolution/s 3
 automated..... 21
 automatic 7, 14, 26
 extend 24
 limit 24
 second round 23
 standard rules 27
 user-defined 24
Conflict solving 3
Conflict types 17
Conflict validation..... 3
ConflictExplorer.EXE 8
 command line options 38
Conflict-free..... 37
Conflicting package
 together 21
Conflicts
 between packages..... 27
 between resources..... 27
 destined for conflict resolution 24
 do not count..... 27
 exclude 20

- excluding 19
- exporting 30
- filtering out 19
- identifying..... 14
- new 37
- not rectified 19
- real..... 7
- reciprocal 37
- rectifying..... 21
- red 24
- residual 23
- search 28
- selective..... 21
- tree item 20
- with operating system 33
- with the operating system..... 33
- Conflictstree item 17
- Consequences of conflicts 5
- Content 4
 - Registry key 19
- Copying
 - packages 31
- Costs 4
- Create Transform 24
- Ctrl-A 36
- Ctrlkey 32
- Custom actions..... 7, 36
 - data-driven 36

D

- Data
 - consistent 7
- Data consistency 7
- Del.....20, 33, 36
- Delete complete Family in all groups..... 33
- Delete own task list..... 39
- Delete Resolving Queue..... 27
- Deleting
 - packages 31
- Dependencies..... 7, 8
- Development 11
- Development share..... 11
- Directory structure..... 10
- Display Excludessselecting..... 20
- DLL file 27
- DllRegisterServer() 3
- Domain users 13
- Don't import SelfReg information..... 15
- Drag-and-drop 32

E

- Evaluation version..... 34

- Exclude from
 - view21
- Exclude list..... 20, 27
- EXE file27
- Export selection31
- Export to clipboard31

F

- Families
 - complete32
- Family, FAMILY6, 39
- Family names.....9
- FamilyName6
- FAMILYNAME38
- File17
- File conflict/s 19, 23, 27
- File path
 - opening14
- File resources.....27
- File system paths10
- File version3
- File Versioning Rules.....18
- File versions18
- Files.....7
- Filter operations19
- Full trust8

G

- Global design errors4
- Global rules.....4
- Good packages 25
- Group
 - same21
- Groups21
 - creating13, 31
 - different27
 - multiple32, 33
 - renaming31
- GROUPS39

H

- Help39
- Helpful usability features7
- Heterogeneous desktop environment.....5

I

- ICE conflicts4

- Icon
 - yellow 17
- Import
 - higher-performance 7
 - more robust..... 7
- Import date 7
- Import process
 - team 9
- Importing a software package 15
- ImportMsi 8
 - command line options..... 9
- ImportMsi.EXE 8, 15
 - command line options..... 38
- Incompatibilities 14, 21
 - validity 21
- In-place update 36
- Installation
 - package..... 20
- Installation file
 - not found 10
- Installation state
 - invalid 18
- Integration/Test 11
- Internal Consistency Evaluators..... 4
- IsolationBeginFiles 12
- IsolationBeginRegistry 11

K

- Key 18
- KeyPath 13, 27
 - rules 27

L

- Local design..... 3
- LOCAL isolation 19
- Local rules 4
- Log file..... 34

M

- Major groups of conflicts 4
- Manual conflict resolution 24
- Menu
 - Package/Reimport 29
 - Tools/Options 29
 - View/Logfile..... 34
- Menuview 19
- Moving
 - packages 31
- MSI file not found 10
- MSI files

- manufacturer 7
- msidbComponentAttributesNeverOverwrite 13
- msidbComponentAttributesPermanent 23, 27
- msidbComponentAttributesSharedDllRefCount 27
- MST
 - separate 7
- MST files 30
 - existing 22
- MultiPackageImport.vbs..... 15
- Multiple importing..... 15

N

- Name 18
- Naming system
 - inconsistent 18
- New conflicts 37
- Non-functioning..... 18

O

- Operating system 33
 - conflicts with 5
- Options 29
- Orca 7
- OrcaPath 10
- Order activation by mail 35
- Order Information 35

P

- Package..... 6
 - descriptor 9
 - icon..... 29
 - with conflicts 14
 - without conflicts 14
- PACKAGE..... 39
- Package directory 29
- Package families 6
 - copying 32
 - deleting 33
 - moving 32
- Package/Import 17
- PackageFamily 6, 27
- PackageFamily exclude 20
- PackageFamily exclude list..... 7
- PACKAGENAME 38
- Packages
 - copying 31
 - deleting 13, 31
 - moving 13, 31
 - unassigned 9
- Patch file..... 36

PathReplacements 10

PE files 8

Permanent attribute 22

Permission 33

Platform

- moving 35

Prepare Selected Conflicts to Resolve 24

Proactive conflict resolution 5

Processes

- blocking 39

Product activation 35

Product version 35

Production 11

Project

- Wise 8

ProjectDir

- Wise 8

Properties 7

Property

- table 9

Property table 37

PropertyFamily 9

PROPERTYFAMILY 39

PropertyPackage 9

PROPERTYPACKAGE 39

Pseudo-installation 8, 15

Q

Questions 36

Queue 27

R

Red entry 27

Registry 7, 10, 17

Registry Conflicts 18

Registry key 18

Registry keys

- identical 4

Registry references 23

Registry to Registry Conflicts 18

Registry to SelfReg 17

Registry to SelfReg Conflicts 18

Registry to SelfReg Value Conflicts 18

Registry Value Conflicts 18

RegistryKeyPath 27

Re-import 12

Reimport Settings 29

Re-importing 14

Renaming

- package families 36
- packages 36

Replacement paths 10

RequestState 13, 29

Rescan 20

Residual conflicts 23

Resolve Settings 30

Resource isolation

- standard settings 30

Resource type 17

Resources

- isolated 27
- shared 18

Root 18

S

Scan Package Conflict/s 10, 17, 27

Search 14, 28

- MSI file 10

Select Transform to Edit or Create Now 22

SelfReg 7, 17

SelfReg to Registry Conflicts 18

SelfReg to Registry Value Conflicts 18

SelfReg to SelfReg Conflicts 18

SelfReg to SelfReg Value Conflicts 18

SelfRegconflicts 18

SelfRegcontent 8

SelfRegregistrations 18

SelfRegtable/s 8, 18

Seller Information 34

Semicolon 10

Server

- multiple 10

Settings

- global 10
- permanent 20

Settings for Automatic Resolve 30

Settings.INI 8, 10

ShareDll 22

Shares 10

Shortcut keys 36

- backspace 36
- Ctrl 36
- Ctrl-A 36
- Del 36
- F3 36
- F5 36

Software conflicts 3

Software conflicts in software packages 3

Software deployment process 10

Software packages

- robust 4

Software packaging

- robust 6

Software release process 30

Software sets 37

Standard directory 10

Standard rules 27
 Standard workflow..... 17
 Structural methods 19

T

Tasks
 delete 39
 Team members 35
 Terminology 6
 Transform 21, 24
 Transform file..... 5
 Transforms
 automatic 12
 TRANSFORMS..... 38
 TransformsAll..... 12
 TransformsAllReimport..... 12

U

UAC 8
 unassigned packages 9
 Uninstalling 17
 package..... 20
 Unlocking 35
 UpgradeCode 6
 Upgrading a software package 4
 User
 permissions..... 13

User list.....14
 Users
 display9
 Uninstalling a software package.....4

V

Value conflicts 17, 18, 21
 Version 6
 latest18
 View
 excluding from21
 Vista.....8
 Visual appearance
 icon.....29

W

Windows Installer properties..... 7
 Windows Vista.....8
 Wise Package Studio
 projects38
 Wise project8

Y

Yellow icon9, 17