Installer Class

Provides static methods for installing and configuring products and patches.

Methods

Method	
AdvertiseProduct(packagePath, perUser, transforms, locale)	Advertises a product to the local computer.
AdvertiseScript(scriptFile, flags, removeItems)	Copies an advertise script file to the local computer.
ApplyPatch(patchPackage, commandLine)	For each product listed by the patch package as eligible to receive the patch, Ap
ApplyPatch(patchPackage, installPackage, installType, commandLine)	For each product listed by the patch package as eligible to receive the patch, Ap
ConfigureFeature(productCode, feature, installState)	Configures the installed state for a product feature.
ConfigureProduct(productCode, installLevel,	Installs or uninstalls a product.

Method	
installState, commandLine)	
DetermineApplicablePatches(productPackage, patches, errorHandler)	Determines which patches apply to a specified product MSI and in what sequenc
DetermineApplicablePatches(product, patches, errorHandler, userSid, context)	Determines which patches apply to a specified product and in what sequence. If superceded patches.
EnableLog(logModes, logFile)	Enables logging of the selected message type for all subsequent install sessions i
EnableLog(logModes, logFile, append, flushEveryLine)	Enables logging of the selected message type for all subsequent install sessions i
ExtractPatchXmlData(patchPath)	Extracts information from a patch that can be used to determine whether the pat M:WixToolset.Dtf.WindowsInstaller.Installer.DetermineApplicablePatches(System.S instead of the full patch file.
GenerateAdvertiseScript(packagePath, scriptFilePath, transforms, locale)	Generates an advertise script. The method enables the installer to write to a scrip
GenerateAdvertiseScript(packagePath, scriptFilePath, transforms, locale, processor, instance)	Generates an advertise script. The method enables the installer to write to a scrip
GetErrorMessage(errorNumber)	Gets a Windows Installer error message in the system default language.

Method	
GetErrorMessage(errorNumber, culture)	Gets a Windows Installer error message in a specified language.
GetErrorMessage(errorRecord)	Gets a formatted Windows Installer error message in the system default languag
GetErrorMessage(errorRecord, culture)	Gets a formatted Windows Installer error message in a specified language.
GetFileHash(path, hash)	Gets a 128-bit hash of the specified file.
GetFileLanguage(path)	Gets the language string of the path specified using the format that the installer
GetFileVersion(path)	Gets the version string of the path specified using the format that the installer ex
GetProductInfoFromScript(scriptFile)	Gets product information for an installer script file.
GetShortcutTarget(shortcut)	Examines a shortcut and returns its product, feature name, and component if ava
InstallMissingComponent(product, component, installState)	Installs files that are unexpectedly missing.
InstallMissingFile(product, file)	Installs files that are unexpectedly missing.
InstallProduct(packagePath, commandLine)	Opens an installer package and initializes an install session.
NotifySidChange(oldSid, newSid)	[MSI 3.1] Migrates a user's application configuration data to a new SID.

Method	
OpenPackage(packagePath, ignoreMachineState)	Opens an installer package for use with functions that access the product databa
OpenPackage(database, ignoreMachineState)	Opens an installer package for use with functions that access the product databa
OpenProduct(productCode)	Opens an installer package for an installed product using the product code.
ProcessAdvertiseScript(scriptFile, iconFolder, shortcuts, removeItems)	Processes an advertise script file into the specified locations.
ProvideAssembly(assemblyName, appContext, installMode, isWin32Assembly)	Gets the full path to a Windows Installer component containing an assembly. Thi
ProvideComponent(product, feature, component, installMode)	Gets the full component path, performing any necessary installation. This metho
ProvideQualifiedComponent(component, qualifier, installMode, product)	Gets the full component path for a qualified component that is published by a pr
ReinstallFeature(product, feature, reinstallModes)	Reinstalls a feature.
ReinstallProduct(product, reinstallModes)	Reinstalls a product.

Method	
SetExternalUI(uiHandler, messageFilter)	Enables an external user-interface handler. This external UI handler is called befo to indicate that it has handled the messages.
SetExternalUI(uiHandler, messageFilter)	[MSI 3.1] Enables a record-based external user-interface handler. This external UI returning a non-zero value to indicate that it has handled the messages.
SetInternalUI(uiOptions)	Enables the installer's internal user interface. Then this user interface is used for a
UseFeature(productCode, feature, installMode)	increments the usage count for a particular feature and returns the installation st
VerifyPackage(packagePath)	Verifies that the given file is an installation package.

Properties

Property	Description
RebootInitiated	Indicates whether a system reboot has been initiated after running an installation or configuration operation.
RebootRequired	Indicates whether a system reboot is required after running an installation or configuration operation.
Version	Gets the current version of the installer.

`WixToolset.Dtf.WindowsInstaller.dll` version `4.0.2+644ed0118bae1aa885bb6cc906dc6c0b904de4d9`

AdvertiseProduct(packagePath, perUser, transforms, locale) Method

Advertises a product to the local computer.

Declaration

```
public static void AdvertiseProduct(
  string packagePath,
  bool perUser,
  string transforms,
  int locale
)
```

Parameters

Parameter	Type	Description
packagePath	string	Path to the package of the product being advertised
perUser	bool	True if the product is user-assigned; false if it is machine-assigned.

Parameter	Type	Description
transforms	string	Semi-colon delimited list of transforms to be applied. This parameter may be null.
locale	int	The language to use if the source supports multiple languages

Remarks

Win32 MSI APIs: MsiAdvertiseProduct , MsiAdvertiseProductEx

See also

- M:WixToolset.Dtf.WindowsInstaller.Installer.GenerateAdvertiseScript(System.String,System.String,System.String,System.Int32,System.Reflection.ProcessorArchitecture,System.Boolean)

Exceptions

Exception	Description
T:System.IO.FileNotFoundException	the specified package file does not exist

AdvertiseScript(scriptFile, flags, removeItems) Method

Copies an advertise script file to the local computer.

Declaration

```
public static void AdvertiseScript(
  string scriptFile,
  int flags,
  bool removeItems
)
```

Parameters

Parameter	Type	Description
scriptFile	string	Path to a script file generated by
flags	int	Flags controlling advertisement
removeItems	bool	True if specified items are to be removed instead of being created

Remarks

The process calling this function must be running under the LocalSystem account. To advertise an application for per-user installation to a targeted user, the thread that calls this function must impersonate the targeted user. If the thread calling this function is not impersonating a targeted user, the application is advertised to all users for installation with elevated privileges.

ApplyPatch(patchPackage, commandLine) Method

For each product listed by the patch package as eligible to receive the patch, ApplyPatch invokes an installation and sets the PATCH property to the path of the patch package.

Declaration

```
public static void ApplyPatch(
  string patchPackage,
  string commandLine
)
```

Parameters

Parameter	Type	Description
patchPackage	string	path to the patch package
commandLine	string	optional command line property settings

Remarks

The «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootRequired» and «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootInitiated» properties should be tested after calling this method. Win32 MSI API: MsiApplyPatch

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallerException	There was an error applying the patch

ApplyPatch(patchPackage, installPackage, installType, commandLine) Method

For each product listed by the patch package as eligible to receive the patch, ApplyPatch invokes an installation and sets the PATCH property to the path of the patch package.

Declaration

```
public static void ApplyPatch(
  string patchPackage,
  string installPackage,
  InstallType installType,
  string commandLine
)
```

Parameters

Parameter	Type	Description

Parameter	Type	Description
patchPackage	string	path to the patch package
installPackage	string	path to the product to be patched, if installType is set to
installType	InstallType	type of installation to patch
commandLine	string	optional command line property settings

Remarks

The «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootRequired» and «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootInitiated» properties should be tested after calling this method. Win32 MSI API: MsiApplyPatch

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallerException	There was an error applying the patch

ConfigureFeature(productCode, feature, installState) Method

Configures the installed state for a product feature.

Declaration

```
public static void ConfigureFeature(
  string productCode,
  string feature,
  InstallState installState
)
```

Parameters

Parameter	Type	Description
productCode	string	Product code of the product to be configured.
feature	string	Specifies the feature ID for the feature to be configured.
installState	InstallState	Specifies the installation state for the feature.

Remarks

The «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootRequired» and «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootInitiated» properties should be tested after calling this method. Win32 MSI API: MsiConfigureFeature

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallerException	There was an error configuring the feature

ConfigureProduct(productCode, installLevel, installState, commandLine) Method

Installs or uninstalls a product.

Declaration

```
public static void ConfigureProduct(
  string productCode,
  int installLevel,
  InstallState installState,
  string commandLine
)
```

Parameters

Parameter	Type	Description

Parameter	Type	Description
productCode	string	Product code of the product to be configured.
installLevel	int	Specifies the default installation configuration of the product. The parameter is ignored and all features are installed if the parameter is set to any other value than . This parameter must be either 0 (install using authored feature levels), 65535 (install all features), or a value between 0 and 65535 to install a subset of available features.
installState	InstallState	Specifies the installation state for the product.
commandLine	string	Specifies the command line property settings. This should be a list of the format Property=Setting Property=Setting.

Remarks

This method displays the user interface with the current settings and log mode. You can change user interface settings with the «see M:WixToolset.Dtf.WindowsInstaller.Installer.SetInternalUI(WixToolset.Dtf.WindowsInstaller.InstallUIOptions)» and «see M:WixToolset.Dtf.WindowsInstaller.Installer.SetExternalUI(WixToolset.Dtf.WindowsInstaller.ExternalUIHandler,WixToolset.Dtf.WindowsInstaller.InstallLogModes)» functions. You can set the log mode with the «see M:WixToolset.Dtf.WindowsInstaller.Installer.EnableLog(WixToolset.Dtf.WindowsInstaller.InstallLogModes,System.String)» function. The «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootRequired» and «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootInitiated» properties should be tested after calling this method. Win32 MSI APIs: MsiConfigureProduct , MsiConfigureProductEx

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallerException	There was an error configuring the product

DetermineApplicablePatches(productPackage, patches, errorHandler) Method

Determines which patches apply to a specified product MSI and in what sequence.

Declaration

```
public static IList<System.String> DetermineApplicablePatches(
  string productPackage,
  System.String[] patches,
  InapplicablePatchHandler errorHandler
)
```

Parameters

Parameter	Type	Description
productPackage	string	Full path to an MSI file that is the target product for the set of patches.

Parameter	Type	Description
patches	System.String[]	An array of strings specifying the patches to be checked. Each item may be the path to an MSP file, the path an XML file, or just an XML blob.
errorHandler	InapplicablePatchHandler	Callback to be invoked for each inapplicable patch, reporting the reason the patch is not applicable. This value may be left null if that information is not desired.

Return value

`IList<System.String>` An array of selected patch strings from *patches* , indicating the set of applicable patches. The items are re-ordered to be in the best sequence.

Remarks

If an item in *patches* is a file path but does not end in .MSP or .XML, it is assumed to be an MSP file. As this overload uses InstallContext.None, it does not consider the current state of the system. Win32 MSI API: MsiDetermineApplicablePatches

DetermineApplicablePatches(product, patches, errorHandler, userSid, context) Method

Determines which patches apply to a specified product and in what sequence. If the product is installed, this method accounts for patches that have already been applied to the product and accounts for obsolete and superceded patches.

Declaration

```
public static IList<System.String> DetermineApplicablePatches(
  string product,
  System.String[] patches,
  InapplicablePatchHandler errorHandler,
  string userSid,
  UserContexts context
)
```

Parameters

Parameter	Type	Description
product	string	The product that is the target for the set of patches. This may be either a ProductCode (GUID) of a product that is currently installed, or the path to a an MSI file.
patches	System.String[]	An array of strings specifying the patches to be checked. Each item may be the path to an MSP file, the path an XML file, or just an XML blob.
errorHandler	InapplicablePatchHandler	Callback to be invoked for each inapplicable patch, reporting the reason the patch is not applicable. This value may be left null if that information is not desired.
userSid	string	Specifies a security identifier (SID) of a user. This parameter restricts the context of enumeration for this user account. This parameter cannot be the special SID strings s-1-1-0 (everyone) or s-1-5-18 (local system). If is set to or , then must be

Parameter	Type	Description
		null. For the current user context, can be null and can be set to or .
context	UserContexts	Restricts the enumeration to per-user-unmanaged, per-user-managed, or per-machine context, or (if referring to an MSI) to no system context at all. This parameter can be , , , or .

Return value

`IList<System.String>` An array of selected patch strings from *patches* , indicating the set of applicable patches. The items are re-ordered to be in the best sequence.

Remarks

If an item in *patches* is a file path but does not end in .MSP or .XML, it is assumed to be an MSP file. Passing an InstallContext of None only analyzes the MSI file; it does not consider the current state of the system. You cannot use InstallContext.None with a ProductCode GUID. Win32 MSI APIs: MsiDetermineApplicablePatches MsiDeterminePatchSequence

EnableLog(logModes, logFile) Method

Enables logging of the selected message type for all subsequent install sessions in the current process space.

Declaration

```
public static void EnableLog(
  InstallLogModes logModes,
  string logFile
)
```

logModes	InstallLogModes	One or more mode flags specifying the type of messages to log
logFile	string	Full path to the log file. A null path disables logging, in which case the logModes paraneter is ignored.

Remarks

This method takes effect on any new installation processes. Calling this method from within a custom action will not start logging for that installation.

Exceptions

Exception	Description
T:System.ArgumentException	an invalid log mode was specified

EnableLog(logModes, logFile, append, flushEveryLine) Method

Enables logging of the selected message type for all subsequent install sessions in the current process space.

Declaration

```
public static void EnableLog(
  InstallLogModes logModes,
  string logFile,
  bool append,
  bool flushEveryLine
)
```

Parameters

Parameter	Type	Description
logModes	InstallLogModes	One or more mode flags specifying the type of messages to log
logFile	string	Full path to the log file. A null path disables logging, in which case the logModes paraneter is ignored.
append	bool	If true, the log lines will be appended to any existing file content. If false, the log file will be truncated if it exists. The default is false.
flushEveryLine	bool	If true, the log will be flushed after every line. If false, the log will be flushed every 20 lines. The default is true.

Remarks

This method takes effect on any new installation processes. Calling this method from within a custom action will not start logging for that installation. Win32 MSI API: MsiEnableLog

Exceptions

Exception	Description
T:System.ArgumentException	an invalid log mode was specified

ExtractPatchXmlData(patchPath) Method

Extracts information from a patch that can be used to determine whether the patch applies on a target system. The method returns an XML string that can be provided to «see M:WixToolset.Dtf.WindowsInstaller.Installer.DetermineApplicablePatches(System.String,System.String[],WixToolset.Dtf.WindowsInstaller.InapplicablePatchHandler,System.String,WixToolset.Dtf.WindowsInstaller.UserContexts)» instead of the full patch file.

Declaration

```
public static string ExtractPatchXmlData(
  string patchPath
)
```

Parameters

Parameter	Type	Description
patchPath	string	Full path to the patch being queried.

Return value

`string` XML string containing patch data.

Remarks

Win32 MSI API: MsiExtractPatchXMLData

GenerateAdvertiseScript(packagePath, scriptFilePath, transforms, locale) Method

Generates an advertise script. The method enables the installer to write to a script the registry and shortcut information used to assign or publish a product.

Declaration

```
public static void GenerateAdvertiseScript(
  string packagePath,
```

```
  string scriptFilePath,
  string transforms,
  int locale
)
```

Parameters

Parameter	Type	Description
packagePath	string	Path to the package of the product being advertised
scriptFilePath	string	path to script file to be created with the advertise information
transforms	string	Semi-colon delimited list of transforms to be applied. This parameter may be null.
locale	int	The language to use if the source supports multiple languages

Remarks

Win32 MSI APIs: MsiAdvertiseProduct , MsiAdvertiseProductEx

See also

- M:WixToolset.Dtf.WindowsInstaller.Installer.AdvertiseProduct(System.String,System.Boolean,System.String,System.Int32)

Exceptions

Exception	Description
T:System.IO.FileNotFoundException	the specified package file does not exist

GenerateAdvertiseScript(packagePath, scriptFilePath, transforms, locale, processor, instance) Method

Generates an advertise script. The method enables the installer to write to a script the registry and shortcut information used to assign or publish a product.

Declaration

```
public static void GenerateAdvertiseScript(
  string packagePath,
  string scriptFilePath,
  string transforms,
  int locale,
  System.Reflection.ProcessorArchitecture processor,
  bool instance
)
```

Parameters

Parameter	Type	Description

Parameter	Type	Description
packagePath	string	Path to the package of the product being advertised
scriptFilePath	string	path to script file to be created with the advertise information
transforms	string	Semi-colon delimited list of transforms to be applied. This parameter may be null.
locale	int	The language to use if the source supports multiple languages
processor	System.Reflection.ProcessorArchitecture	Targeted processor architecture.
instance	bool	True to install multiple instances through product code changing transform. Advertises a new instance of the product. Requires that the parameter includes the instance transform that changes the product code.

Remarks

Win32 MSI APIs: MsiAdvertiseProduct , MsiAdvertiseProductEx

See also

- M:WixToolset.Dtf.WindowsInstaller.Installer.AdvertiseProduct(System.String,System.Boolean,System.String,System.Int32)

GetErrorMessage(errorNumber) Method

Gets a Windows Installer error message in the system default language.

Declaration

```
public static string GetErrorMessage(
  int errorNumber
)
```

Parameters

Parameter	Type	Description
errorNumber	int	The error number.

Return value

`string` The message string, or null if the error message is not found.

Remarks

The returned string may have tokens such as [2] and [3] that are meant to be substituted with context-specific values. Error numbers greater than 2000 refer to MSI "internal" errors, and are always returned in English.

GetErrorMessage(errorNumber, culture) Method

Gets a Windows Installer error message in a specified language.

Declaration

```
public static string GetErrorMessage(
  int errorNumber,
  System.Globalization.CultureInfo culture
)
```

Parameters

Parameter	Type	Description
errorNumber	int	The error number.
culture	System.Globalization.CultureInfo	The locale for the message.

Return value

`string` The message string, or null if the error message or locale is not found.

Remarks

The returned string may have tokens such as [2] and [3] that are meant to be substituted with context-specific values. Error numbers greater than 2000 refer to MSI "internal" errors, and are always returned in English.

GetErrorMessage(errorRecord) Method

Gets a formatted Windows Installer error message in the system default language.

Declaration

```
public static string GetErrorMessage(
  Record errorRecord
)
```

Parameters

Parameter	Type	Description
errorRecord	Record	Error record containing the error number in the first field, and error-specific parameters in the other fields.

Return value

`string` The message string, or null if the error message is not found.

Remarks

Error numbers greater than 2000 refer to MSI "internal" errors, and are always returned in English.

GetErrorMessage(errorRecord, culture) Method

Gets a formatted Windows Installer error message in a specified language.

Declaration

```
public static string GetErrorMessage(
  Record errorRecord,
  System.Globalization.CultureInfo culture
)
```

Parameters

Parameter	Type	Description
errorRecord	Record	Error record containing the error number in the first field, and error-specific parameters in the other fields.
culture	System.Globalization.CultureInfo	The locale for the message.

Return value

`string` The message string, or null if the error message or locale is not found.

Remarks

Error numbers greater than 2000 refer to MSI "internal" errors, and are always returned in English.

GetFileHash(path, hash) Method

Gets a 128-bit hash of the specified file.

Declaration

```
public static void GetFileHash(
  string path,
  System.Int32[] hash
)
```

Parameters

Parameter	Type	Description
path	string	Path to the file

Parameter	Type	Description
hash	System.Int32[]	Integer array of length 4 which receives the four 32-bit parts of the hash value.

Remarks

Win32 MSI API: MsiGetFileHash

Exceptions

Exception	Description
T:System.IO.FileNotFoundException	the file does not exist or
could not be read	

GetFileLanguage(path) Method

Gets the language string of the path specified using the format that the installer expects to find them in in the database.

Declaration

```
public static string GetFileLanguage(
  string path
```

```
)
```

Parameters

Parameter	Type	Description
path	string	Path to the file

Return value

`string` Language string in the form of a decimal language ID, or an empty string if the file does not contain a language ID

Remarks

Win32 MSI API: MsiGetFileVersion

Exceptions

Exception	Description
T:System.IO.FileNotFoundException	the file does not exist or could not be read

GetFileVersion(path) Method

Gets the version string of the path specified using the format that the installer expects to find it in in the database.

Declaration

```
public static string GetFileVersion(
  string path
)
```

Parameters

Parameter	Type	Description
path	string	Path to the file

Return value

`string` Version string in the "#.#.#.#" format, or an empty string if the file does not contain version information

Remarks

Win32 MSI API: MsiGetFileVersion

Exceptions

Exception	Description
T:System.IO.FileNotFoundException	the file does not exist or could not be read

GetProductInfoFromScript(scriptFile) Method

Gets product information for an installer script file.

Declaration

```
public static ProductInstallation GetProductInfoFromScript(
  string scriptFile
)
```

Parameters

Parameter	Type	Description
scriptFile	string	Path to a script file generated by

Return value

`ProductInstallation` ProductInstallation stub with advertise-related properties filled in.

Remarks

Only the following properties will be filled in in the returned object:

- «see P:WixToolset.Dtf.WindowsInstaller.ProductInstallation.ProductCode»
- «see P:WixToolset.Dtf.WindowsInstaller.ProductInstallation.AdvertisedLanguage»
- «see P:WixToolset.Dtf.WindowsInstaller.ProductInstallation.AdvertisedVersion»
- «see P:WixToolset.Dtf.WindowsInstaller.ProductInstallation.AdvertisedProductName»
- «see P:WixToolset.Dtf.WindowsInstaller.ProductInstallation.AdvertisedPackageName» Other properties will be null. Win32 MSI API: MsiGetProductInfoFromScript

Exceptions

Exception	Description
T:System.ArgumentOutOfRangeException	An invalid product property was requested

GetShortcutTarget(shortcut) Method

Examines a shortcut and returns its product, feature name, and component if available.

Declaration

```
public static ShortcutTarget GetShortcutTarget(
```

```
  string shortcut
)
```

Parameters

Parameter	Type	Description
shortcut	string	Full path to a shortcut

Return value

`ShortcutTarget` ShortcutTarget structure containing target product code, feature, and component code

Remarks

Win32 MSI API: MsiGetShortcutTarget

InstallMissingComponent(product, component, installState) Method

Installs files that are unexpectedly missing.

Declaration

```
public static void InstallMissingComponent(
  string product,
  string component,
  InstallState installState
)
```

component	string	Component to be installed
installState	InstallState	Specifies the way the component should be installed.

Remarks

Win32 MSI API: MsiInstallMissingComponent

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallCanceledException	the user exited the installation

InstallMissingFile(product, file) Method

Installs files that are unexpectedly missing.

Declaration

```
public static void InstallMissingFile(
  string product,
  string file
)
```

Parameters

Parameter	Type	Description
product	string	Product code for the product that owns the file to be installed
file	string	File to be installed

Remarks

Win32 MSI API: MsiInstallMissingFile

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallCanceledException	the user exited the installation

InstallProduct(packagePath, commandLine) Method

Opens an installer package and initializes an install session.

Declaration

```
public static void InstallProduct(
  string packagePath,
  string commandLine
)
```

Parameters

Parameter	Type	Description
packagePath	string	path to the patch package
commandLine	string	command line property settings

Remarks

To completely remove a product, set REMOVE=ALL in *commandLine* . This method displays the user interface with the current settings and log mode. You can change user interface settings with the «see M:WixToolset.Dtf.WindowsInstaller.Installer.SetInternalUI(WixToolset.Dtf.WindowsInstaller.InstallUIOptions)» and «see M:WixToolset.Dtf.WindowsInstaller.Installer.SetExternalUI(WixToolset.Dtf.WindowsInstaller.ExternalUIHandler,WixToolset.Dtf.Window

sInstaller.InstallLogModes)» functions. You can set the log mode with the «see M:WixToolset.Dtf.WindowsInstaller.Installer.EnableLog(WixToolset.Dtf.WindowsInstaller.InstallLogModes,System.String)» function. The «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootRequired» and «see P:WixToolset.Dtf.WindowsInstaller.Installer.RebootInitiated» properties should be tested after calling this method. Win32 MSI API: MsiInstallProduct

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallerException	There was an error installing the product

NotifySidChange(oldSid, newSid) Method

[MSI 3.1] Migrates a user's application configuration data to a new SID.

Declaration

```
public static void NotifySidChange(
  string oldSid,
  string newSid
)
```

Parameters

Parameter	Type	Description
oldSid	string	Previous user SID that data is to be migrated from
newSid	string	New user SID that data is to be migrated to

Remarks

Win32 MSI API: MsiNotifySidChange

OpenPackage(packagePath, ignoreMachineState) Method

Opens an installer package for use with functions that access the product database and install engine, returning an Session object.

Declaration

```
public static Session OpenPackage(
  string packagePath,
  bool ignoreMachineState
)
```

Parameters

Parameter	Type	Description
packagePath	string	Path to the package
ignoreMachineState	bool	Specifies whether or not the create a Session object that ignores the computer state and that is incapable of changing the current computer state. A value of false yields the normal behavior. A value of true creates a "safe" Session object that cannot change of the current machine state.

Return value

`Session` A Session object allowing access to the product database and install engine

Remarks

Note that only one Session object can be opened by a single process. OpenPackage cannot be used in a custom action because the active installation is the only session allowed. A "safe" Session object ignores the current computer state when opening the package and prevents changes to the current computer state. The Session object should be «see M:WixToolset.Dtf.WindowsInstaller.InstallerHandle.Close» d after use. It is best that the handle be closed manually as soon as it is no longer needed, as leaving lots of unused handles open can degrade performance. Win32 MSI APIs: MsiOpenPackage , MsiOpenPackageEx

Exceptions

Exception	Description

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallerException	The product could not be opened
T:WixToolset.Dtf.WindowsInstaller.InstallerException	The installer configuration data is corrupt

OpenPackage(database, ignoreMachineState) Method

Opens an installer package for use with functions that access the product database and install engine, returning an Session object.

Declaration

```
public static Session OpenPackage(
  Database database,
  bool ignoreMachineState
)
```

Parameters

Parameter	Type	Description
database	Database	Database used to create the session
ignoreMachineState	bool	Specifies whether or not the create a Session object that ignores the computer state and

Parameter	Type	Description
		that is incapable of changing the current computer state. A value of false yields the normal behavior. A value of true creates a "safe" Session object that cannot change of the current machine state.

Return value

`Session` A Session object allowing access to the product database and install engine

Remarks

Note that only one Session object can be opened by a single process. OpenPackage cannot be used in a custom action because the active installation is the only session allowed. A "safe" Session object ignores the current computer state when opening the package and prevents changes to the current computer state. The Session object should be «see M:WixToolset.Dtf.WindowsInstaller.InstallerHandle.Close» d after use. It is best that the handle be closed manually as soon as it is no longer needed, as leaving lots of unused handles open can degrade performance. Win32 MSI APIs: MsiOpenPackage , MsiOpenPackageEx

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallerException	The product could not be opened

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallerException	The installer configuration data is corrupt

OpenProduct(productCode) Method

Opens an installer package for an installed product using the product code.

Declaration

```
public static Session OpenProduct(
  string productCode
)
```

Parameters

Parameter	Type	Description
productCode	string	Product code of the installed product

Return value

`Session` A Session object allowing access to the product database and install engine, or null if the specified product is not

installed.

Remarks

Note that only one Session object can be opened by a single process. OpenProduct cannot be used in a custom action because the active installation is the only session allowed. The Session object should be «see M:WixToolset.Dtf.WindowsInstaller.InstallerHandle.Close» d after use. It is best that the handle be closed manually as soon as it is no longer needed, as leaving lots of unused handles open can degrade performance. Win32 MSI API: MsiOpenProduct

Exceptions

Exception	Description
T:System.ArgumentException	An unknown product was requested
T:WixToolset.Dtf.WindowsInstaller.InstallerException	The product could not be opened
T:WixToolset.Dtf.WindowsInstaller.InstallerException	The installer configuration data is corrupt

ProcessAdvertiseScript(scriptFile, iconFolder, shortcuts, removeItems) Method

Processes an advertise script file into the specified locations.

Declaration

```
public static void ProcessAdvertiseScript(
  string scriptFile,
  string iconFolder,
  bool shortcuts,
  bool removeItems
)
```

Parameters

Parameter	Type	Description
scriptFile	string	Path to a script file generated by
iconFolder	string	An optional path to a folder in which advertised icon files and transform files are located. If this parameter is null, no icon or transform files are written.
shortcuts	bool	True if shortcuts should be created
removeItems	bool	True if specified items are to be removed instead of created

Remarks

The process calling this function must be running under the LocalSystem account. To advertise an application for per-user installation to a targeted user, the thread that calls this function must impersonate the targeted user. If the thread calling this

function is not impersonating a targeted user, the application is advertised to all users for installation with elevated privileges.
Win32 MSI API: MsiProcessAdvertiseScript

ProvideAssembly(assemblyName, appContext, installMode, isWin32Assembly) Method

Gets the full path to a Windows Installer component containing an assembly. This method prompts for a source and increments the usage count for the feature.

Declaration

```
public static string ProvideAssembly(
  string assemblyName,
  string appContext,
  InstallMode installMode,
  bool isWin32Assembly
)
```

Parameters

Parameter	Type	Description
assemblyName	string	Assembly name

Parameter	Type	Description
appContext	string	Set to null for global assemblies. For private assemblies, set to the full path of the application configuration file (.cfg file) or executable file (.exe) of the application to which the assembly has been made private.
installMode	InstallMode	Installation mode; this can also include bits from
isWin32Assembly	bool	True if this is a Win32 assembly, false if it is a .NET assembly

Return value

`string` Path to the assembly

Remarks

Win32 MSI API: MsiProvideAssembly

ProvideComponent(product, feature, component, installMode) Method

Gets the full component path, performing any necessary installation. This method prompts for source if necessary and increments the usage count for the feature.

Declaration

```
public static string ProvideComponent(
  string product,
  string feature,
  string component,
  InstallMode installMode
)
```

Parameters

Parameter	Type	Description
product	string	Product code for the product that contains the feature with the necessary component
feature	string	Feature ID of the feature with the necessary component
component	string	Component code of the necessary component
installMode	InstallMode	Installation mode; this can also include bits from

Return value

`string` Path to the component

Remarks

Win32 MSI API: MsiProvideComponent

ProvideQualifiedComponent(component, qualifier, installMode, product) Method

Gets the full component path for a qualified component that is published by a product and performs any necessary installation. This method prompts for source if necessary and increments the usage count for the feature.

Declaration

```
public static string ProvideQualifiedComponent(
  string component,
  string qualifier,
  InstallMode installMode,
  string product
)
```

Parameters

Parameter	Type	Description
component	string	Specifies the component ID for the requested component. This may not be the GUID for the component itself but rather a server that provides the correct functionality, as in the ComponentId column of the PublishComponent table.

Parameter	Type	Description
qualifier	string	Specifies a qualifier into a list of advertising components (from PublishComponent Table).
installMode	InstallMode	Installation mode; this can also include bits from
product	string	Optional; specifies the product to match that has published the qualified component.

Return value

`string` Path to the component

Remarks

Win32 MSI APIs: MsiProvideQualifiedComponent MsiProvideQualifiedComponentEx

ReinstallFeature(product, feature, reinstallModes) Method

Reinstalls a feature.

Declaration

```
public static void ReinstallFeature(
  string product,
  string feature,
```

```
  ReinstallModes reinstallModes
)
```

Parameters

Parameter	Type	Description
product	string	Product code for the product containing the feature to be reinstalled
feature	string	Feature to be reinstalled
reinstallModes	ReinstallModes	Reinstall modes

Remarks

Win32 MSI API: MsiReinstallFeature

Exceptions

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallCanceledException	the user exited the installation

ReinstallProduct(product, reinstallModes) Method

Reinstalls a product.

Declaration

```
public static void ReinstallProduct(
  string product,
  ReinstallModes reinstallModes
)
```

Parameters

Parameter	Type	Description
product	string	Product code for the product to be reinstalled
reinstallModes	ReinstallModes	Reinstall modes

Remarks

Win32 MSI API: MsiReinstallProduct

Exceptions

Exception	Description

Exception	Description
T:WixToolset.Dtf.WindowsInstaller.InstallCanceledException	the user exited the installation

SetExternalUI(uiHandler, messageFilter) Method

Enables an external user-interface handler. This external UI handler is called before the normal internal user-interface handler. The external UI handler has the option to suppress the internal UI by returning a non-zero value to indicate that it has handled the messages.

Declaration

```
public static ExternalUIHandler SetExternalUI(
  ExternalUIHandler uiHandler,
  InstallLogModes messageFilter
)
```

Parameters

Parameter	Type	Description
uiHandler	ExternalUIHandler	A callback delegate that handles the UI messages
messageFilter	InstallLogModes	Specifies which messages to handle using the external message handler. If the external

Parameter	Type	Description
		handler returns a non-zero result, then that message will not be sent to the UI, instead the message will be logged if logging has been enabled.

Return value

`ExternalUIHandler` The previously set external handler, or null if there was no previously set handler

Remarks

To restore the previous UI handler, a second call is made to SetExternalUI using the ExternalUIHandler returned by the first call to SetExternalUI and specifying «see F:WixToolset.Dtf.WindowsInstaller.InstallLogModes.None» as the message filter. The external user interface handler does not have full control over the external user interface unless «see M:WixToolset.Dtf.WindowsInstaller.Installer.SetInternalUI(WixToolset.Dtf.WindowsInstaller.InstallUIOptions)» is called with the uiLevel parameter set to «see F:WixToolset.Dtf.WindowsInstaller.InstallUIOptions.Silent» . If SetInternalUI is not called, the internal user interface level defaults to «see F:WixToolset.Dtf.WindowsInstaller.InstallUIOptions.Basic» . As a result, any message not handled by the external user interface handler is handled by Windows Installer. The initial "Preparing to install..." dialog always appears even if the external user interface handler handles all messages. SetExternalUI should only be called from a bootstrapping application. You cannot call it from a custom action Win32 MSI API: MsiSetExternalUI

SetExternalUI(uiHandler, messageFilter) Method

[MSI 3.1] Enables a record-based external user-interface handler. This external UI handler is called before the normal internal user-interface handler. The external UI handler has the option to suppress the internal UI by returning a non-zero value to indicate that it

has handled the messages.

Declaration

```
public static ExternalUIRecordHandler SetExternalUI(
  ExternalUIRecordHandler uiHandler,
  InstallLogModes messageFilter
)
```

Parameters

Parameter	Type	Description
uiHandler	ExternalUIRecordHandler	A callback delegate that handles the UI messages
messageFilter	InstallLogModes	Specifies which messages to handle using the external message handler. If the external handler returns a non-zero result, then that message will not be sent to the UI, instead the message will be logged if logging has been enabled.

Return value

`ExternalUIRecordHandler` The previously set external handler, or null if there was no previously set handler

Remarks

To restore the previous UI handler, a second call is made to SetExternalUI using the ExternalUIHandler returned by the first call to SetExternalUI and specifying «see F:WixToolset.Dtf.WindowsInstaller.InstallLogModes.None» as the message filter. The external user interface handler does not have full control over the external user interface unless «see M:WixToolset.Dtf.WindowsInstaller.Installer.SetInternalUI(WixToolset.Dtf.WindowsInstaller.InstallUIOptions)» is called with the uiLevel parameter set to «see F:WixToolset.Dtf.WindowsInstaller.InstallUIOptions.Silent» . If SetInternalUI is not called, the internal user interface level defaults to «see F:WixToolset.Dtf.WindowsInstaller.InstallUIOptions.Basic» . As a result, any message not handled by the external user interface handler is handled by Windows Installer. The initial "Preparing to install..." dialog always appears even if the external user interface handler handles all messages. SetExternalUI should only be called from a bootstrapping application. You cannot call it from a custom action Win32 MSI API: MsiSetExternalUIRecord

SetInternalUI(uiOptions) Method

Enables the installer's internal user interface. Then this user interface is used for all subsequent calls to user-interface-generating installer functions in this process. The owner of the user interface does not change.

Declaration

```
public static InstallUIOptions SetInternalUI(
  InstallUIOptions uiOptions
)
```

Parameters

Parameter	Type	Description

Parameter	Type	Description
uiOptions	InstallUIOptions	Specifies the level of complexity of the user interface

Return value

`InstallUIOptions` The previous user interface level

Remarks

Win32 MSI API: MsiSetInternalUI

UseFeature(productCode, feature, installMode) Method

increments the usage count for a particular feature and returns the installation state for that feature. This method should be used to indicate an application's intent to use a feature.

Declaration

```
public static InstallState UseFeature(
  string productCode,
  string feature,
  InstallMode installMode
)
```

Parameters

Parameter	Type	Description
productCode	string	The product code of the product.
feature	string	The feature to be used.
installMode	InstallMode	Must have the value .

Return value

`InstallState` The installed state of the feature.

Remarks

The UseFeature method should only be used on features known to be published. The application should determine the status of the feature by calling either the FeatureState method or Features method. Win32 MSI APIs: MsiUseFeature , MsiUseFeatureEx

VerifyPackage(packagePath) Method

Verifies that the given file is an installation package.

Declaration

```
public static bool VerifyPackage(
  string packagePath
)
```

packagePath	string	Path to the package

Return value

`bool` True if the file is an installation package; false otherwise.

Remarks

Win32 MSI API: MsiVerifyPackage

Exceptions

Exception	Description
T:System.IO.FileNotFoundException	the specified package file does not exist
T:WixToolset.Dtf.WindowsInstaller.InstallerException	the package file could not be opened

RebootInitiated Property

Indicates whether a system reboot has been initiated after running an installation or configuration operation.

Declaration

```
public static bool RebootInitiated { get; set; }
```

RebootRequired Property

Indicates whether a system reboot is required after running an installation or configuration operation.

Declaration

```
public static bool RebootRequired { get; set; }
```

Version Property

Gets the current version of the installer.

Declaration

```
public static System.Version Version { get; set; }
```