Application Packaging
MSI Packaging Standards & Best Practices

Document Control

* Indicates mandatory requirement

Contents	Details
Document Name*	*MSI Packaging Standards and Best Practices*
Client	*<CLIENTNAME>*
Version Number *	*1.0*
Document Version Date*	
Author *	*Duncan Collier*
Classification*	*Internal*
Document References	
Quality Review By*	
History*	<To be populated following Client Approval>

Table of contents

1.0 Introduction

1.1 Purpose

The purpose of this document is to establish the set of standards and best practices that must be adhered to for all Microsoft Installer (MSI) application packages. These standards and best practices exist to ensure a consistent level of quality is established and maintained for all MSI packages used within the managed environment. Included within this document are:

- The Strategic Direction for application packaging.
- Standards for Package Development.

Changes to this document or the standards and best practices it details should be performed through a change control process.

1.2 Scope

The Packaging Standards and Best Practices detailed within this document should be applied to all MSIs created or transformed. Where the Standards and Best Practices differ between environments, they will be detailed in the relevant section.

2.0 Strategic Direction

2.1. Target Environment

Packages are required to work on the target platform or platforms they were designed for and are not required to support any other platforms.

2.2. MSI Version

The target platform will be required to use a minimum version of Windows Installer 3 as a prerequisite for all MSI packages.

2.3. MSI Packaging Tools

All application packages should be created or transformed using Flexera AdminStudio and/or Orca / InstEd.

2.4. Component Structure

MSIs install, repair and remove applications in parts referred to as components. Components are collections of resources that are always maintained as a unit on a user's system. A resource can be anything, including a file, registry key or shortcut. Every component is assigned a “Globally Unique Identifier” (GUID) referred to as the ComponentID or component code GUID. The following bulleted extract from Microsoft’s “Defining Installer Components” MSDN Article outlines how to organise an application into Windows Installer components. See Appendix C for the full article.

- “Define a new component for every .exe, .dll, and .ocx file. Designate these files as the key path files of their components. Assign each component a component code GUID.
- Define a new component for every .hlp or .chm help file. Designate these files as the key path files of their components. Add the .cnt or .chi files to the components holding their associated .hlp and .chm files. Assign each component a component code GUID.
- Define a new component for every file that serves as a target of a shortcut. Designate these files as the key path files of their components. Assign each component a component code GUID.
- Group all of the remaining resources into folders. All resources in each folder must ship together. If there is a possibility that a pair of resources may ship separately in the future, put these in separate components. Define a new component for every folder. Try to keep the total number of components low to improve performance. Divide the application into many components when it is necessary to have the installer check the validity of the installation thoroughly. Designate any file in the component as the key path file. Assign each component a component code GUID.
- Add registry keys to the components. Any registry key that points to a file should be included in that file's component. Other registry keys should be logically grouped with the files that require them.

2.5. Feature Structure (Non vendor MSIs)

All Current User information should be moved to the parent feature of the main application feature to facilitate more efficient repair operations. Components containing advertised shortcuts should be part of a hidden sub-feature of the main application feature.

2.6. ICE Validation

Internal Consistency Evaluators, also called ICEs, evaluate the MSI package for entries in the database records that are valid when examined individually, but that may cause incorrect behaviour in the context of the whole database. For example, the Component table may list several components that are all valid when tested individually. However, it is possible that two components could use the same GUID as their component code, violating Microsoft Best Practices. The severity of each resolvable ICE falls into one of two categories, Errors or Warnings:

- **Error** messages report database authoring that cause incorrect behaviour.
- **Warning** messages report database authoring that either causes incorrect behaviour in certain cases or results in unexpected side-effects in others.

All ICE Errors and Warnings will be resolved where possible for non vendor MSIs. Any ICE Errors and/or Warnings not resolved will be documented as an exception and an acceptable explanation offered.

ICE Errors and Warnings will not be resolved in vendor MSIs as to do so may compromise the integrity of the MSI and the intentions of the vendor.

2.7. Unattended Install

Packages are required to install silently and unattended. There should be no post-install configuration steps required; the package should perform all necessary tasks to deploy the configured application.

2.8. Unattended Uninstall

Packages are required to uninstall silently.

The MSI will only remove files and registry keys installed by the MSI itself. It is acceptable to leave files and registry keys created at runtime and/or user specific files or registry behind following uninstall.

2.9. Upgrades

Packages are required to upgrade all previous versions of the same application, if it was deployed using an MSI via the corporate deployment toolset and requested as part of the discovery process. All requested package upgrades will be undertaken as a major upgrade utilising a full MSI, not a Microsoft Patch (MSP).

2.10. Self Heal and Install-on-Demand (Advertising)

Self Heal and Install-on-Demand functionality is initiated when an advertised entry point of the MSI is used to access one of its resources. During this process the Windows Installer Service will verify the keypath of the relevant Components, to ascertain which need to be installed or re-installed.

Many resources can be supplied to this process from the locally cached MSI, however if files are to be installed or re-installed, these will need to be retrieved from the original source location, or a specified managed source. To enable this functionality it would be necessary to grant users file access rights to read from these locations.

2.11. Package Format (Non vendor MSIs)

Where possible, all packages will adopt a standard single MSI with internal cabinets.

Where non vendor MSI file size is in excess of 500MB, compressed external cabinet files should be utilised. In such cases, the naming convention outlined in section 3.1.3 below should be followed.

2.12. Vendor MSI Package Format

Vendor supplied MSIs will be customised with a Microsoft Transform (MST) file for the purpose of setting installation options. No direct modification to the Vendor MSI will be made as this violates Microsoft's best practices and may potentially breach the licensing or support agreements for that product. The folder and file structure will also not be altered. The following set of MSI Properties will not be altered in a transform designed for use with a Vendor supplied MSI:

- Manufacturer
- ProductID
- ProductVersion
- ProductName

2.13. Automatically Generated Transforms

Vendor supplied MSIs distributed with a proprietary tool for creating transforms, should be used in conjunction with the license agreement in place for that product in order to create an MST for the purpose of customisation.

2.14. Localisation

Packages will be re-usable across the entire business wherever possible. A generic base MSI will be created for applications which require customisation for multiple environments. All localisation settings for multiple sites will be applied through MST files.

2.15. Middleware

Middleware, such as database related software, will be packaged in a generic form so that only one instance of the package needs to be created. Examples of middleware applications are Java Runtime and Oracle.

2.16. Access and Control

Application licensing and the control of user's access to application executables will be managed outside the scope of application packaging.

2.17. Default User Profile

Packages will not make changes to the "Default User" profile.

2.18. Windows Resource Protection

Packages will not attempt to install Windows resource protected files or registry entries.

2.19. Exceptions

The standards and best practices described in this document must be adhered to in all cases unless a deviation is documented and a reason given to the exception.

3. Packaging Development

3.1. Naming Conventions

3.1.1. Application Naming Standards

The application name, version, and manufacturer should directly reflect the applications Help, About dialog box. Alternatively, the applications main executable can be used to ascertain these values.

3.1.2. MSI and MST Naming Conventions

All global or single site packages will be named as:
ApplicationName_Version_PPackageVersion_Zone_LanguageID.MSI

A package for multiple sites will be named as:
ApplicationName_Version_PPackageVersion_LanguageID.MSI
ApplicationName_Version_PPackageVersion_Zone_LanguageID.MST

The elements of the name have specific meanings as described below; all spaces must be replaced with underscores ("_"):

Naming Element	Meaning
ApplicationName	Name of the software. Where an application name is made up of multiple words, they should be separated by underscores.
Version	The fully qualified version number which follows Microsoft's Windows Installer standards i.e. aaa.bbb.ccccc.ddddd aaa and bbb are less than 255 ccccc and ddddd are less than 65535
PPackageVersion	A decimal number which is preceded by "P" and represents the package specification and build version. The initial specification for the package would be "1", and the first build would be "0" giving a package version of "1.0". If a second build is required (e.g. containing a bug fix) this would make the package version "1.1". If a new specification was requested (e.g. a feature change), this would make the package version "2.0".
Zone	Used to identify multiple sites which share applications configured in such a way that it is not possible to have a "One fits all" installation. "Global" will be used to identify a package which requires no localisation and can be used across the entire business without modification.
LanguageID	The scope of the project is to package for the English language only. However, where multiple languages are supported without additional packaging, please supply the following notation: ML = If an application automatically detects and switches to another language or is user selectable post-installation MX = Switchable during deployment such as a language transform or property value If only the English language is supported by the package without additional packaging effort, then use the following notation: EN = English only

Notes: Only alphanumeric characters, along with "." and "_" may be used within the file name. Vendor MSIs will keep their original names, but the MST will follow the above naming convention.

3.1.2.1. MSI Naming Example

If WinZip 9.0 is to be packaged for global use, the package name would be:
WinZip_9.0_P1.0_Global_EN.MSI

If the package was to be updated to make a minor modification to the MSI, but the application version was still 9.0, the name would be changed to:
WinZip_9.0_P1.1_Global_EN.MSI
If the package was to be updated to include a new feature, but the application version was still 9.0, then the name would be:
WinZip_9.0_P2.0_Global_EN.MSI

To signify a new version of an existing software package, the version would change and then the name would be:
WinZip_9.1_P1.0_Global_EN.MSI

If WinZip 9.0 is to be packaged and customised for a single site, the package name would be:
WinZip_9.0_P1.0_Croydon_Purley_Way_EN.MSI

If WinZip 9.0 is to be packaged and customised for multiple sites, the package names would be:
WinZip_9.0_P1.0_EN.MSI
WinZip_9.0_P1.0_Paul_Street_London_EN.MST
WinZip_9.0_P1.0_Hatfield_EN.MST

If WinZip 9.0 can change language during deployment, the package name would be:
WinZip_9.0_P1.0_Global_MX.MSI

If WinZip 9.0 allows the user to change the language or their regional options allow the application to switch language, the package name would be:
WinZip_9.0_P1.0_Global_ML.MSI

3.1.3. MSI and MST CAB Naming Convention

Where external CAB files are required for Vendor MSIs, a similar naming convention to the MST will be used. The CAB will have Cxxx appended to the file name, where xxx is the order in which the CAB was created. The first CAB file should be C001.

3.1.3.1. MSI and MST CAB Naming Example

If WinZip 9.0 was to be packaged for the Global Zone, the CAB files would be named:
WinZIp_9.0_P1.0_Global_C001.cab

The addition of extra files would prompt the creation of a new CAB, and the counter would be incremented as follows:
WinZip_9.0_P1.0_Global_C002.cab

3.2. Install Location

The default installation location should be "C:\Program Files\<default>", where <default> is defined as the default directory name used by the Vendor Installation.

3.3. Windows Directory Names

The official MSI Property names for standard Windows directories must be used, since their values will be automatically set by the Windows Installer engine when the package is installed. The entry for 'WindowsFolder' must exist, even if it is not referenced elsewhere in the MSI.

3.4. Shortcuts

To ensure that a consistent look and feel is presented to the user, shortcuts will follow a predefined standard regarding their placement unless otherwise specified by the client during application discovery:

- All shortcuts should be advertised excluding vendor MSIs.
- Start Menu shortcuts will be installed as per the vendor install.
- Shortcuts will not be placed on the desktop.
  Note: Where desktop shortcuts are required, they will be created on the All Users desktop.
- Quick Launch shortcuts will not be included in any package.
- Uninstall shortcuts will be removed.
- Shortcuts pointing to Readme files, URLs and URL Updates will be removed.

3.5. Audit Trail Registry Keys

Each package created must contain audit trail registry keys as follows:

Registry	Root	Key	Name	Value	Component
AudRegistry1	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Information	This application was installed with Windows Installer Technology	AudRegistry
AudRegistry2	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Information2	Packaged by [PackagingCompany] for [COMPANYNAME]	AudRegistry
AudRegistry3	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Packaging Location	[PackagingLocation]	AudRegistry
AudRegistry4	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Packager	[Author]	AudRegistry
AudRegistry5	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Package Code	[PackageCode]	AudRegistry
AudRegistry6	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Application Version	[ProductVersion]	AudRegistry
AudRegistry7	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Package Version	[PackageVersion]	AudRegistry
AudRegistry8	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Install Date	[Date]	AudRegistry
AudRegistry9	2	Software\[COMPANYNAME]\Applications\[ProductName]\[AppVerDir]	Deployment set for	[ZONE]	AudRegistry

3.6. MSI Properties

The following table lists the MSI Properties and values that must be present in the Property Table.

Name	Description	Value
ALLUSERS	**Determines a per user or per machine installation**	**1**
ApplicationUsers	**Defines control of dialogue**	**AllUsers**
AppVerDir	**Audit Trail registry key set to AppVersion_PPackageVersion_Zone, e.g. 6.0_P1.0_Global**	**AppVersion_PPackageVersion_Zone**
ARPCONTACT	**Provides Contact information for the application in the ARP applet**	**IT Helpdesk**
ARPNOMODIFY	**Enables or disables the Modify option for the Add or Remove Programs applet**	**1**
ARPNOREMOVE	**Enables or disables the Remove option for the Add or Remove Programs applet**	**Should not be present (i.e. enable)**
ARPNOREPAIR	**Enables or disables the Repair option for the Add or Remove Programs applet**	**1**
ARPPRODUCTICON	**Specifies the icon for the package in the Add or Remove Programs applet**	**Foreign key into the icon table pointing to the application icon**
Author	**Specifies the author of the package**	**Packager's name**
COMPANYNAME	**Company Name**	**Client's company name**
***Manufacturer**	**Name of the application's manufacturer**	**Manufacturer's name**
MSIINSTALLPERUSER	**Specifies a per machine installation**	**0**
PackagingCompany	**For audit keys**	**Computacenter**
PackagingLocation	**Packaging location**	**e.g. Hatfield**
PackageVersion	**Package Revision Version**	**e.g. 1.0**
*** ProductName**	**Application name (Spaces acceptable – No underscores)**	**e.g. Java Runtime**

* ProductVersion	String format of the product version as a numeric value in the form aaa.bbb.cccc.dddd where dddd is optional and will be ignored by the Windows Installer runtime.	Product version in the Microsoft Windows Installer Standard Format.
REBOOT	Suppresses Reboots	ReallySuppress
ROOTDRIVE	Specifies the default drive for the destination directory of the installation	C:\
ZONE	Packaging Zone the application is intended for.	e.g. GLOBAL

* These properties should not be changed within MSTs customising vendor supplied MSIs. Any other MSI Properties whose name begins with ARP*, should not be included.

The above MSI Properties should be used to variablise values throughout the MSI. Directory properties should be used to variablise paths to the deepest levels, where this is not possible Hard Coded Path references should be added as a new MSI Property that can be used to variablise values throughout the MSI.

Public Properties should be created for Server Names, IP Addresses, Network Locations and Databases etc. Any authored Properties should have their names prepended with "CC_".

3.7. Launch Conditions

Non vendor MSI packages should not contain any launch conditions.

In the case of vendor MSIs, launch conditions should remain. However, the 'AdminUser' condition should be removed.

3.8. Custom Actions

Tasks not natively supported by the Windows Installer Service are considered "out of scope". All "out of scope" tasks should be performed by approved Custom Actions. All authored Custom Actions should have their names prepended with "CC_" and should be set to run in-script (1024) and if administrative in nature have the NoImpersonate bit set (3072). Type 7, 23 and 39 Custom Actions must not be used. "Gacutil.exe" must not be called from any Custom Action.

Any scripts called via a custom action should be authored in VBScript. Where applicable, the effects of custom actions should be reversed on uninstall.

3.9. Summary Information Stream

The Summary Information Stream lists key information about the software and package creator. Note that MSTs cannot change the Summary Information Stream.

3.9.1. Title

The Title property briefly describes the type of installer package. The phrase "Installation Database" will be used for this property.

3.9.2. Subject

This property conveys to a file browser the product that can be installed using the logic and data in this installer database. This value will be set equal to the installer property ProductName.

3.9.3. Author

The Author property conveys to a file browser the manufacturer of the installation database. This value will be set equal to the installer property "Manufacturer".

3.9.4. Keywords

This property is used by file browsers to hold keywords that permit the database file to be found in a keyword search. The set of keywords typically includes "Installer" as well as product-specific keywords, and may be localised.

3.9.5. Comments

The Comments property conveys the general purpose of the installer database. This value for this summary property will be set to the following: "This installer database contains the logic and data required to install <product name>." The <product name> variable will be set equal to the installer property ProductName.

3.9.6. Template

The Template Property of an installation database indicates the target platform and language versions that are supported by the database. The “Platform” property should be set to Intel and the language id should be set to the value specified in the “ProductLanguage” Property. If the “ProductLanguage” Property does not exist, it should be set to 2057.

3.9.7. Package Code

The package code is a unique identifier of the installer package.

3.9.8. Schema

The Schema property contains the minimum installer version required. This property must be set to an integer equal or greater than 301.

3.9.9. Security

The Security property conveys whether the package should be opened as read-only. The value of this property will be set to “Read-only recommended”.

3.9.10. Image Type

The Image Type property can be used to indicate the type of source file image. This value will be set to “Compressed, source files using long file names”. All packages will be set to “Requires Elevated Privileges” unless they exclusively contain user based information.

3.10. SelfReg Table (Non Vendor MSIs)

COM based dlls should not use the SelfReg Table to register their Classes, TypeLibs, ProgIDs and Interfaces. All registry entries describing COM interfaces should be made through the appropriate advertising tables.

3.11. IniFile Table

The IniFile Table should be used to write all ini file entries. Flat versions of ini files should not be included in the MSI unless the content of an ini file is non standard.

3.12. LockPermissions Table

The LockPermissions table will be used to alter access control lists where required. Where this is not suitable, Setacl.exe will be used.

3.13. Tool Specific Tables

Packaging tool specific entries inserted in to the MSI should be removed, e.g. Wise Tables, Wise Properties, Wise validation entries.

3.14. Environment Variables

Environment variables will be deployed for “All Users”, i.e. per machine. Any change to existing Environment variable values, should be appended or prepended. The path statement must never be replaced.

Any Environment Variables added to the target machine must be removed cleanly on uninstall.

3.15. Services

Services should be created and controlled via the appropriate MSI tables.

Any Services added to the target machine during install must be stopped and removed cleanly on uninstall.

3.16. ODBC DSNs, Drivers

ODBC DSNs should be created and controlled via the appropriate MSI tables. These should be delivered on a per machine basis.

Any ODBC entries added to the target machine must be removed cleanly on uninstall.

3.17. Terminal Services / Citrix Platform

MSI packages (Vendor’s and those created from legacy setups) prepared for deployment to a Terminal Services / Citrix platform should correspond to the following requirements which override elements of the MSI requirements specified above.

3.17.1. Current User Components

Key Path of current user components must be **EMPTY**.

3.17.2 Current User Registry

Current User Registry entries can remain in the MSI. They will be mirrored automatically and without MSI self-repair into the user's profile.

No dummy per-user registry keys should be added or kept in the MSI package. Per-user registry keys should not be used as a component key path.

3.17.3. Active Setup

All Active Setup entries should be **REMOVED** from the MSI created for Terminal Services / Citrix

3.17.4. Current User Files

Per User files should be **REMOVED** from MSI package.

In case the application doesn’t work properly without of per-user file(s) - install per-user files into the same subfolder of “All Users” profile and verify how application works in this instance. If it doesn’t help there are two options that can be applied:

a) **Use an extra tool for per-user files deployment.** Ask Terminal Services / Citrix engineer at customer site which tool they use, for deployment necessary per-user files to a user profile. It can be done by a special tool providing per-user files copying feature or providing “Run Once PerUser” feature or can be implemented through Group Policy scripts. Packaging Engineer creates <PackageName>-CopyPerUserFiles.cmd file which perform copying application per user files from “All Users” profile to user profile and puts cmd file in the package folder. Terminal Services / Citrix engineer is responsible for deployment necessary per-user files to a user profile using content of <PackageName>-CopyPerUserFiles.cmd file.

b) **Copy per user files at application shortcut launching.** Develop a VBScript for copying per-user files to the user profile (in case required per-user files are missing) before the application is launched and then launches the application. Script copies per-user files before main application launch and then launches main application. Shortcuts in the MSI package should be updated in a way to launch this script instead of the main application.

As there is no standard best practices approach for the deployment of per-user files, both options should be discussed with the customer and approved by the Terminal Services / Citrix support engineer at the customer site.

3.17.5. Shortcuts

Advertised shortcuts are **NOT** allowed for Terminal Services / Citrix.

Add ***DISABLEADVTSHORTCUTS=1*** to the Property table.

3.17.6. Multi-Platform Packages

If a package is supposed to be used on several platforms (Desktop and TS/Citrix) the Citrix/TS specific configuration should be provided by an extra MST on top of the 'desktop' package. If 'desktop' package is a vendor MSI the installation on Terminal Services / Citrix will consist of the MSI, plus 2 MST files as a minimum.

Appendix A - Organizing Applications into Components (MSDN)

Windows Installer installs and removes an application or product in parts referred to as components. Components are collections of resources that are always installed or removed as a unit from a user's system. A resource can be a file, registry key, shortcut, or anything else that may be installed. Every component is assigned a unique component code GUID. Authors of installation packages should only create components, and versions of components, that can be installed and removed without damaging other components. Also, the removal of a component should not leave behind any orphaned resources on the user's computer, such as unused files, registry keys, or shortcuts. To ensure this, authors should adhere to the following general rules when organizing resources into components:

- Never create two components that install a resource under the same name and target location. If a resource must be duplicated in multiple components, change its name or target location in each component. This rule should be applied across applications, products, product versions, and companies.
- Note that the previous rule means that two components must not have the same key path file. The key path value points to a particular file or folder belonging to the component that the installer uses to detect the component. If two components had the same key path file, the installer would be unable to distinguish which component is installed. Two components however may share a key path folder.
- Do not create a version of a component that is incompatible with all previous versions of the component. This rule should be applied across applications, products, product versions, and companies.
- Do not create components containing resources that will need to be installed into more than one directory on the user's system. The installer installs all of the resources in a component into the same directory. It is not possible to install some resources into subdirectories.
- Do not include more than one COM server per component. If a component contains a COM server, this must be the key path for the component.
- Do not specify more than one file per component as a target for the Start menu or a Desktop shortcut.

When organizing an application into components, package authors may need to add, remove, or modify the resources in an existing installation. In this case, the author must decide whether to provide the resources by introducing a new component or by modifying existing components and changing them into a new version of the component. Because a unique component code must be assigned when a new component is introduced, authors must determine whether their changes require changing the component code.

Source**:** http://msdn.microsoft.com/en-us/library/Aa370561

Appendix B - Changing the Component Code (MSDN)

When specifying the components for an installation, package authors should follow the general rules for component organization described in Organizing Applications into Components. Authors may need to introduce new components or modify existing components. If the addition, removal, or modification of resources effectively creates a new component, then the component code must also be changed.

Creating a New Component
Introduce a new component and assign it a unique component code when making any of the following changes:

- Any change that has not been shown by testing to be compatible with previous versions of the component. In this case, you must also change the name or target location of every resource in the component.
- A change in the name or target location of any file, registry key, shortcut, or other resource in the component. In this case, you must also change the name or target location of every resource in the component.
- The addition or removal of any file, registry key, shortcut, or other resource from the component. In this case, you must also change the name or target location every resource in the component.
- Recompiling a 32-bit component into a 64-bit component.

When introducing a new component
The author needs to do one of the following to ensure that the component does not conflict with any existing components:

- Change the name or target location of any resource that may be installed under the same name and target location by another component.
- Otherwise guarantee that the new component is never installed into the same folder as another component which has a resource under a common name and location. This includes localized versions of files with the same file name.
- When changing the component code of an existing component, also change the name or target location of every file, registry key, shortcut, and other resource in the component.

Creating a New Version of a Component
A new version of a component is assigned the same component code as another existing component. Modifying a component without changing the component code is only optional in the following cases:

- The changes to the component have been proven by testing to be backward compatible with all previous versions of the component.
- The author can guarantee that the new version of the component will never be installed on a system where it would conflict with previous versions of the component or applications requiring a previous version.
- The component code of a new version of a component must not be changed when it would result in two components sharing resources, such as registry values, files, or shortcuts.

Source: http://msdn.microsoft.com/en-us/library/Aa367849

Appendix C - Defining Installer Components (MSDN)

The following outlines how to organize your application into Windows Installer components.

- Begin by obtaining a directory and file tree for all of the files and other resources used in your application.

- Identify any files, registry keys, shortcuts, or other resources that are shared across applications and can be provided by existing components available as merge modules. You must not include any of these resources in the components you author. Instead obtain these components by merging the merge modules into your installation package.

- The following steps describe how to organize the remaining resources of the application into components.
    - Define a new component for every .exe, .dll, and .ocx file. Designate these files as the key path files of their components. Assign each component a component code GUID.
    - Define a new component for every .hlp or .chm help file. Designate these files as the key path files of their components. Add the .cnt or .chi files to the components holding their associated .hlp and .chm files. Assign each component a component code GUID.
    - Define a new component for every file that serves as a target of a shortcut. Designate these files as the key path files of their components. Assign each component a component code GUID.
    - Group all of the remaining resources into folders. All resources in each folder must ship together. If there is a possibility that a pair of resources may ship separately in the future, put these in separate components.
    - Define a new component for every folder. Try to keep the total number of components low to improve performance. Divide the application into many components when it is necessary to have the installer check the validity of the installation thoroughly.
    - Designate any file in the component as the key path file.
    - Assign each component a component code GUID.
    - Add registry keys to the components. Any registry key that points to a file should be included in that file's component. Other registry keys should be logically grouped with the files that require them.

Source: http://msdn.microsoft.com/en-us/library/Aa368269