



Filesystem Layout for Unbundled Software Policies and Best Practices

Version 1.0
June 14, 2007

Table of contents

1 INTRODUCTION.....	3
1.1 DEFINITIONS.....	3
1.1.1 Conventions used in this document.....	3
1.2 KEY SET OF DESIGN REQUIREMENTS.....	3
1.3 GENERAL APPROACH.....	4
1.3.1 Unix Centric Layout.....	4
1.3.2 Separation of Install Image from Data.....	4
1.3.3 Autonomy of Individual Components.....	4
2 IMPLEMENTATION DETAILS.....	6
2.1 \$INSTALL_HOME/.....	6
2.2 FILE SYSTEM LAYOUT.....	6
2.2.1 \$INSTALL_HOME/bin/.....	7
2.2.2 \$INSTALL_HOME/etc/.....	8
2.2.3 \$INSTALL_HOME/etc/\$COMPONENT_ID/.....	8
2.2.4 \$INSTALL_HOME/var/.....	8
2.2.5 \$INSTALL_HOME/var/\$COMPONENT_ID/.....	8
2.2.6 \$INSTALL_HOME/include/.....	9
2.2.7 \$INSTALL_HOME/lib/.....	9
2.2.8 \$INSTALL_HOME/lib64.....	10
2.2.9 \$INSTALL_HOME/doc/.....	10
2.2.10 \$INSTALL_HOME/examples/.....	10
2.2.11 \$INSTALL_HOME/javadoc/.....	10
2.2.12 \$INSTALL_HOME/man/.....	10
2.2.13 \$INSTALL_HOME/\$COMPONENT_ID/.....	10
2.2.14 \$INSTALL_HOME/\$COMPONENT_ID/bin/.....	11
2.2.15 \$INSTALL_HOME/\$COMPONENT_ID/include/.....	11
2.2.16 \$INSTALL_HOME/\$COMPONENT_ID/lib.....	11
2.2.17 \$INSTALL_HOME/\$COMPONENT_ID/lib64.....	11
2.2.18 \$INSTALL_HOME/\$COMPONENT_ID/doc/.....	11
2.2.19 \$INSTALL_HOME/\$COMPONENT_ID/examples/.....	11
2.2.20 \$INSTALL_HOME/\$COMPONENT_ID/javadoc/.....	11
2.2.21 \$INSTALL_HOME/\$COMPONENT_ID/man/.....	12
2.3 INTERFACE CLASSIFICATION.....	12
2.4 DEFAULT FILESYSTEM PERMISSIONS.....	12
2.5 APPLYING TO A STAND ALONE PRODUCT.....	12
2.6 SHRINK TO FIT.....	12
2.7 \$COMPONENT_ID NAMING STANDARD AND GUIDELINES.....	13
2.7.1 \$COMPONENT_ID Naming Standard.....	13
2.7.2 \$COMPONENT_ID Best Practices.....	14
3 REFERENCE.....	14

1 Introduction

This document details exact requirements and best practices of the filesystem layout in support of the Multi-Platform, Multi-Install and User-Based Install features for unbundled software.

1.1 Definitions

“**Bundled Software**” is defined as a software or a consolidation that are bundled as part of operating system distribution. “**Unbundled Software**” is a software or a software consolidation that is not bundled as part of operating system distribution.

“**Multi-Install**” feature is defined as the ability to install multiple times of the same or different versions of layered software products on the same operating system (OS) instance or virtual OS instance (Solaris Zones).

“**User Based Install**” feature is defined as the ability for any user of the system to install, manage and operate an installation of the layered software products without “root” user equivalent privileges on UNIX platforms, or without “administrator” user equivalent privileges on Windows platform.

Component is a commonly used neutral term as a reference to the software components that makeup a software consolidation. Use of this term does not distinguish between component product or shared component, it refers to either component product or shared component or both. This term is used prominently throughout this document. Other terms, such as Shared Component, Component Product, Point Product etc, shall only be used as historical reference or as comparison.

Platform refers to a machines Instruction Set Architecture or processor type, such as is returned by “uname -i” on Solaris.

Platform-dependent refers to a file or a collection of files that are installed on all platforms and whose contents vary depending on the platform. An example of a platform-dependent file is compiled, executable program for a platform.

Platform-independent refers to a file or a collection of files that are installed on all platforms whose contents remains the same on all platforms. An example of a platform-independent file is a standard configuration file.

\$INSTALL_HOME is a reference to a self-contained umbrella install location.

\$COMPONENT_ID is a pre-agreed unique name for a software component. This name shall follow the guidelines as stated in Section 2.6.

Library Contents is commonly referred as a set of libraries, various databases, commands and daemons not invoked directly by a human user.

1.1.1 Conventions used in this document

The key words "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may", and "optional" in this document are to be interpreted as described in RFC-2119 [1]

1.2 Key Set of Design Requirements

The file system layout must:

1. define a consistent layout across all supported platforms – including Linux, Solaris, Windows and other Unix variants.
2. support the Multi-Install and User-Based features as defined in “Definitions”.
3. support self-containment. I.e. all interfaces needed are provided either by the native

- OS, or by components within the filesystem layout itself.
4. support rapid, low-touch adoption of quickly evolving open source projects.
 5. enable an integrated software consolidation the ability to include/exclude any unbundled software component with minimal effort.
 6. simplify administrative tasks and upgrade by maintaining the separation of component install image (read only), configurations (read-write) and runtime data (variable size).
 7. simplify dependency wiring for software components.
 8. enable the software consolidation to be viewed as an integrated software solution.
 9. provide a set of common shared locations for small components that do not want to manage their own `$(COMPONENT)_ID` location.
 10. support 32 and 64 bit binaries in the same install image on Unix. Note that supporting multiple distinct architectures (i.e. sparc and x86) in the same install image is not a requirement.
 11. support multiple versions (typically major versions) of the same component in the same install image (for example Application Server 8 and 9).

1.3 General Approach

1.3.1 Unix Centric Layout

Even though this document is specifying a file system layout that can be used across multiple platforms, it is heavily influenced by existing Unix¹ conventions. The justification for this (or possibly a rationalization) is that the majority of software that will adopt these conventions have a Unix lineage. And most of the developers that will be packaging that software are already familiar with Unix filesystem conventions.

On Windows this leads to some quirks, like the use of `etc` and `var`. But the authors consider this an acceptable trade-off, and some of these quirks have already appeared in other products on Windows (like NetBeans 5.5).

The specification is written using Unix path syntax. On Windows the path separator would be `\`

1.3.2 Separation of Install Image from Data

The layout separates a component's static install image (which does not change after installation) from its configuration and runtime data (which does change after installation). Keeping the static install image separate from the modifiable runtime data provides a number of benefits for managing the system and facilitates upgrade support since the install image can be updated with new binaries with no fear of losing configuration information.

In this layout, configuration and runtime data are kept in the `etc/` and `var/` subdirectories and all post-installation data should go into one or both of these directories. Components should consider all other installation directories to be read-only after installation, and only modifiable via patching/updating or un-installation.

1.3.3 Autonomy of Individual Components

One area of much discussion was to what degree should components be integrated into a common set of directories like is done for bundled software on Solaris. Should all the binaries share the same `bin/`? All the libraries in the same `lib/`? Or should component products have their own component specific areas, giving them more autonomy at the expense of some user convenience.

1. In this document "Unix" is meant to include Linux in addition to traditional Unix distributions.

A couple of requirements played into the final decision:

1. The layout had to support rapid, low-touch adoption of quickly evolving open source projects.
2. The layout had to support multiple versions of a given component installed concurrently. For example AppServer 8.2 and 9.1.

Given this, the decision was to error on the side of component autonomy and allow component's to have their own installation subdirectories, but not require them. This results in a layout with two levels: the top “system” level and a set of lower component specific subdirectories (named `$COMPONENT_ID` in this document).

So, when does something go into an upper system level directory versus a component specific directory? Specific guidance is given in section 2, but to summarize:

1. Any component that is non-trivial in terms of complexity should allocate a `$COMPONENT_ID`, and have its own self-contained set of component specific directories.
2. Components with a `$INSTALL_HOME/$COMPONENT_ID/bin` directory should consider placing symlinks (or wrapper scripts) to commonly used binaries in the top level `$INSTALL_HOME/bin` directory for user convenience.
3. Simple components that do not want to allocate a `$COMPONENT_ID` can go ahead and install directly into the top level directories.
4. Components that operate directly on the install image, such as package management commands, uninstallers, etc. may install into the top level directories.

2 Implementation Details

This section provides detail descriptions, best practices and implementation requirements for the filesystem layout.

2.1 \$INSTALL_HOME/

\$INSTALL_HOME is the self-contained umbrella location into which all components in a consolidation are installed. Each self-contained umbrella install location is separated from other umbrella install locations on the same OS instance in order to eliminate any filesystem level conflicts.

The layout of the filesystem contained in a given \$INSTALL_HOME is specified in the following sections.

2.2 File System Layout

An overview of the file system layout is presented in the diagram below. The diagram includes references to subsequent sections where further details can be found.

\$INSTALL_HOME/	2.1
-- bin/	2.2.1
-- etc/	2.2.2
-- \$COMPONENT_ID/	2.2.3
-- var/	2.2.4
-- \$COMPONENT_ID/	2.2.5
-- include/	2.2.6
-- lib/	2.2.7
-- lib64/ (Linux only)	2.2.8
-- doc/	2.2.9
-- examples/	2.2.10
-- javadoc/	2.2.11
-- man/	2.2.12
-- \$COMPONENT_ID/	2.2.13
-- bin/	2.2.14
-- include/	2.2.15
-- lib/	2.2.16
-- lib64/ (Linux only)	2.2.17
-- doc/	2.2.18
-- examples/	2.2.19
-- javadoc/	2.2.20
-- man/	2.2.21

2.2.1 \$INSTALL_HOME/bin/

2.2.1.1: Contains user-invoked executables/commands.

2.2.1.2: Components must not create component specific subdirectories under \$INSTALL_HOME/bin/.

2.2.1.3: Component specific executables should go in the product's \$INSTALL_HOME/\$COMPONENT_ID/bin subdirectory.

2.2.1.4: For executables that are commonly used by the end user, the component should consider creating a symbolic link (or wrapper script) in \$INSTALL_HOME/bin/ to point to the file in the component specific bin directory as a convenience to the user.

2.2.1.5: If a component is sufficiently small where it does not wish to maintain a component specific bin directory, then it may place its executables directly in \$INSTALL_HOME/bin/.

2.2.1.6: It is appropriate to place commands that operate globally on the \$INSTALL_HOME umbrella directory in \$INSTALL_HOME/bin. This could include commands for adding/removing/updating/inventorying packages, uninstallers, etc.

2.2.1.7: Solaris only: \$INSTALL_HOME/bin/ must not contain 64 bit binaries

2.2.1.8: Solaris only: The following subdirectories of bin/ should be used when 32 and 64 bit binary variants must be supported:

- `amd64/`: 64 bit binaries supporting the amd64 ISA. If a binary is located here then `bin` should contain either the 32 bit version of the binary or the 32 bit `isaexec` wrapper for the binary.
- `i86/`: 32 bit binaries supporting the i86 ISA. Typically used when a binary has both 32 and 64 bit variants and the `isaexec` wrapper is used in `bin` to select the proper executable. If it is expected that users will usually want to run the 32 bit binary even on 64 bit systems, then the 32 bit binary should be placed directly in `bin`.
- `sparcv9/`: 64 bit binaries supporting the sparcv9 ISA. If a binary is located here then `bin` must contain either the 32 bit version of the binary or the 32 bit `isaexec` wrapper for the binary.
- `sparcv7/`: 32 bit binaries supporting the sparcv7 ISA. Typically used when a binary has both 32 and 64 bit variants and the `isaexec` wrapper is used in `bin` to select the proper executable. If it is expected that users will usually want to run the 32 bit binary even on 64 bit systems, then the 32 bit binary should be placed directly in `bin`.

For details concerning pathnames for 64-bit Solaris see: PSARC/1997/220 [9] and PSARC/2004/619 [8]. For Linux and HP-UX the convention appears to be to place the 64 bit executable in `bin` and use a name to indicate it is 64 bit – possibly using the 32 bit version of the command to invoke the 64 bit versions when run on a 64 bit system. For example `adb` and `adb64` on HP-UX 11i. This specification does not address 64 bit support for the Windows platform.

Note: Many Unix file system layouts specify an `'sbin'` directory to contain commands that are targeted exclusively for system administrators. In practice this distinction generates more clutter than clarity for unbundled products. Therefore the authors have chosen not to include

an 'sbin' directory in this layout.

2.2.2 \$INSTALL_HOME/etc/

2.2.2.1: `$INSTALL_HOME/etc/` is the root of a sub-tree for the administrative and configuration files that are unique to this installation under the `$INSTALL_HOME`. Configuration files are files that control the operation of a program/component. Configuration files typically only change occasionally and do not grow to arbitrarily large size.

2.2.2.2: Components that operate globally on the `$INSTALL_HOME` umbrella directory may utilize this location (package management utilities for example).

2.2.2.3: Component specific files should not be placed directly under the root of the `$INSTALL_HOME/etc/` directory. Instead they should be placed in the component specific location as described in section 2.2.3.

2.2.3 \$INSTALL_HOME/etc/\$COMPONENT_ID/

2.2.3.1: Component specific administrative and configuration files shall be grouped under the `etc/$COMPONENT_ID/` subdirectory, in which `$COMPONENT_ID` is unique to that component.

2.2.3.2: Components have full control over the layout of the contents of this subdirectory.

2.2.3.3: Configuration information in `etc` is typically global to the component installation. If a component supports the concept of “instance specific” configuration then that data is more commonly kept in `$INSTALL_HOME/var/$COMPONENT_ID/`.

2.2.3.4: `$INSTALL_HOME/etc/$COMPONENT_ID/` is optional if a component has no administrative and configuration files.

2.2.4 \$INSTALL_HOME/var/

2.2.4.1: `$INSTALL_HOME/var/` is the root of a sub-tree for varying size data files that are unique to this installation under the `$INSTALL_HOME`. Varying size data files typically change often and may grow to an arbitrarily large size.

2.2.4.2: Components that operate globally on the `$INSTALL_HOME` umbrella directory may utilize this location (package management utilities for example).

2.2.4.3: Component specific files should not be placed directly under the root of the `$INSTALL_HOME/var/` directory. Instead they should be placed in the component specific location as described in section 2.2.5.

2.2.5 \$INSTALL_HOME/var/\$COMPONENT_ID/

2.2.5.1: Component specific variable data files shall be grouped under the `$INSTALL_HOME/var/$COMPONENT_ID/` subdirectory, in which `$COMPONENT_ID` is unique to that component. Some examples of appropriate data files are: log files, server instances, component databases.

2.2.5.2: Components have full control over the layout of the contents of this subdirectory.

2.2.5.3: `$INSTALL_HOME/var/$COMPONENT_ID/` is optional if a component has no varying size data files.

2.2.5.4: It is acceptable for components to provide a mechanism to optionally relocate their “var” data to another location. In fact such a capability is encouraged as it provides a number of benefits including the ability to:

- leverage hardware infrastructure – for example high speed storage
- better manage disk space requirements
- facilitate data migration in some upgrade scenarios (allow the sharing of a common “var” area between multiple install homes)

2.2.6 `$INSTALL_HOME/include/`

2.2.6.1: `$INSTALL_HOME/include/` is a shared install location for C/C++ header files for libraries found in `$INSTALL_HOME/lib/`

2.2.7 `$INSTALL_HOME/lib/`

2.2.7.1: `$INSTALL_HOME/lib/` is a shared install location for libraries (including jar files), various databases, commands and daemons not invoked directly by a human user.

2.2.7.2: Simple components that do not want to obtain, use and manage their own set of `$COMPONENT_ID` specific subdirectories can utilize this shared install location.

2.2.7.3: Component specific subdirectories under `$INSTALL_HOME/lib/` are not permitted. Instead components should use `$INSTALL_HOME/$COMPONENT_ID/lib`.

2.2.7.4: Components that operate globally on the `$INSTALL_HOME` umbrella directory may utilize this install location (package management utilities for example).

2.2.7.5: Solaris only: `$INSTALL_HOME/lib/` must not contain 64 bit libraries

2.2.7.6: Solaris only: The following subdirectories should be used when 32 and 64 bit library variants must be supported:

- `amd64/`: 64 bit libraries supporting the amd64 ISA. If a library is located here then `lib` should contain the 32 bit version.
- `sparcv9/`: 64 bit libraries supporting the sparcv9 ISA. If a library is located here then `lib` must contain the 32 bit version.
- `64/`: A symbolic link to the appropriate 64 bit Solaris ABI, either `amd64` or `sparcv9`.

For details concerning pathnames for 64-bit Solaris see: PSARC/1997/220 [9] and PSARC/2004/619 [8].

2.2.7.7: HP-UX only: The following subdirectory should be used when 32 and 64 bit library variants must be supported:

- `pa20_64/`: 64 bit libraries supporting the pa20_64 ISA. If a library is located here then `lib` should contain the 32 bit version.

Note: In some other file system layouts (WSARC/2004/304 [4]) the location of libraries is

influenced by their interface classification. For example there are different locations for public vs. private interfaces. While well-intentioned, this approach has not worked well in practice. It adds clutter, the granularity of interface classification is not always at the file level, and it is cumbersome that a library's location changes when its interface classification changes. There appears to be growing consensus that this approach was a mistake, and therefore it is not adopted by this specification.

2.2.8 `$INSTALL_HOME/lib64`

2.2.8.1: Linux only: location for 64 bit variants of libraries in `$INSTALL_HOME/lib`

2.2.9 `$INSTALL_HOME/doc/`

2.2.9.1: Documentation other than javadoc or man pages.

2.2.9.2: Component specific documentation should go in the product's `$INSTALL_HOME/$COMPONENT_ID/doc/` subdirectory.

2.2.10 `$INSTALL_HOME/examples/`

2.2.10.1: Examples and demos. Even though the structure of this directory is not specified, it is recommended that examples adopt the conventions described in this specification if the examples are of a non-trivial nature.

2.2.10.2: Component specific examples should go in the product's `$INSTALL_HOME/$COMPONENT_ID/examples/` subdirectory.

2.2.11 `$INSTALL_HOME/javadoc/`

2.2.11.1: Java API documentation for classes (jars) in `$INSTALL_HOME/lib/`

2.2.12 `$INSTALL_HOME/man/`

2.2.12.1: `$INSTALL_HOME/man/` is a shared install location for Unix style manual pages.

2.2.12.2: Component specific man pages should go in the product's `$INSTALL_HOME/$COMPONENT_ID/man/` subdirectory.

2.2.12.3: Components should create symbolic links in `$INSTALL_HOME/man/` to point to the man pages in the component specific man directory as a convenience to the user.

2.2.12.4: If a component is sufficiently small where it does not wish to maintain a component specific man directory, then it may place its man pages directly in `$INSTALL_HOME/man/`.

2.2.12.5: Users on Unix systems can set their `MANPATH` environment variable to `$INSTALL_HOME/man/`, so that they can access all the Unix manual pages of all the installed components under `$INSTALL_HOME` via the “man” command on UNIX.

2.2.13 `$INSTALL_HOME/$COMPONENT_ID/`

2.2.13.1: Component specific static install images shall be grouped under the `$INSTALL_HOME/$COMPONENT_ID/` subdirectory, in which `$COMPONENT_ID` is unique to that component.

2.2.13.2: Components have full control over the layout of the contents of this subdirectory. The following sections contain strong recommendations that should be followed if at all possible.

2.2.14 \$INSTALL_HOME/\$COMPONENT_ID/bin/

2.2.14.1: Component specific user-invoked executables and commands.

2.2.14.2: On Unix, for executables that are commonly used by the end user, the component should consider creating a symbolic link (or wrapper script) in `$INSTALL_HOME/bin/` to point to the file here as a convenience to the user.

2.2.14.3: Solaris only: See section 2.2.1 for details concerning managing 32 and 64 bit variants of executables.

2.2.15 \$INSTALL_HOME/\$COMPONENT_ID/include/

2.2.15.1: C/C++ include files for libraries located in `$INSTALL_HOME/$COMPONENT_ID/lib/`.

2.2.16 \$INSTALL_HOME/\$COMPONENT_ID/lib

2.2.16.1: `$INSTALL_HOME/$COMPONENT_ID/lib/` is the install location for libraries (including jar files), various databases, commands and daemons not invoked directly by a human user for the `$COMPONENT_ID` component.

2.2.16.2: Component who own its unique `$COMPONENT_ID` shall group all its library contents under this directory, and it must not use the `$INSTALL_HOME/lib/` location.

2.2.16.3: Unix only: See section 2.2.7 for details concerning managing 32 and 64 bit variants of libraries.

2.2.17 \$INSTALL_HOME/\$COMPONENT_ID/lib64

2.2.17.1: Linux only: location for 64 bit variants of libraries in `$INSTALL_HOME/$COMPONENT_ID/lib`

2.2.18 \$INSTALL_HOME/\$COMPONENT_ID/doc/

2.2.18.1: Documentation other than javadoc or man pages.

2.2.19 \$INSTALL_HOME/\$COMPONENT_ID/examples/

2.2.19.1: Examples or demos. Even though the structure of this directory is not specified, it is recommended that examples adopt the conventions described in this specification if the examples are of a non-trivial nature.

2.2.20 \$INSTALL_HOME/\$COMPONENT_ID/javadoc/

2.2.20.1: Java API documentation for classes (jars) in `$INSTALL_HOME/$COMPONENT_ID/lib/`.

2.2.21 \$INSTALL_HOME/\$COMPONENT_ID/man/

2.2.21.1: Component specific man pages should go in the product's \$INSTALL_HOME/\$COMPONENT_ID/man/ subdirectory.

2.2.21.2: Components should create symbolic links in \$INSTALL_HOME/man/ to point to the man pages in the component specific man directory as a convenience to the user.

2.3 Interface Classification

All names and paths stated in the table in section 2.2 are Committed.

All ISA specific directory names and paths stated in sections 2.2.1 and 2.2.7 are Committed.

Projects that adopt this file system layout should specify that the location of files that contain Committed interfaces are also Committed. Therefore any component ID's that are allocated and used in these paths are also Committed.

2.4 Default Filesystem Permissions

By default, the \$INSTALL_HOME directory and everything under it are owned by the installing user. All subdirectories specified in this document have a default permission of drwxr-xr-x.

2.5 Applying to a Stand Alone Product

This layout is designed primarily for use by software consolidations like Java ES. To apply this layout to a point product (that does not share its \$INSTALL_HOME with other point products), the point product essentially becomes the consolidation. In this case it would not have its own \$COMPONENT_ID area, instead the point product would install into the directories directly under \$INSTALL_HOME. If the point product bundled other components, then it could place those other components under \$COMPONENT_ID directories if they were of non-trivial complexity.

2.6 Shrink to Fit

If a directory described in this specification does not apply to the project implementing this specification, then that directory may be left out of the project's installation image. If a directory contains no files then the project has the option of not including the directory in its installation image.

2.7 \$COMPONENT_ID Naming Standard and Guidelines

2.7.1 \$COMPONENT_ID Naming Standard

The recommended naming standard as documented in the following table shall apply to the \$DIRECTORY_NAME portion of the \$INSTALL_HOME and the \$COMPONENT_ID.

AREA	RECOMMENDATION	RATIONALE
Character Set	Directory name shall be limited to characters A-Z, a-z, 0-9, period, hyphen, and underscore.	These are the characters specified by the POSIX standard for portable directory and file names, and are also valid for all the file systems.
First character of the directory Name	Period and hyphen shall not be the first character of the directory name.	Some operating systems have special rules for the first character of the name. For example, Windows does not permit period as the first character. Access to any directory on the UNIX platform in which its name started with hyphen is awkward.
Last character of the directory name	Period shall not be the last character of the directory name.	Some operating systems have special rules for the last character of the name. For example, Windows does not permit period as the last character in the directory name.
Length of the directory name	At the minimal, the length of the directory name must be at least 1 character, but shall be no more than 32 characters.	Limiting name length can markedly reduce the expressiveness of directory names, yet placing only very high limits on lengths inhibits widest portability. Some operating systems place limits on the total path length. For example, Windows 2000 limits paths to 260 characters total length. If the length of the umbrella install directory name is too long, it may put unnecessary limitation on the components on the depth of the subdirectories they can create, and on the total length of of the path.
Case sensitive	The directory name shall be considered case sensitive, but no two names should differ by case only.	UNIX file systems are case-sensitive, in which "Foo" and "foo" are two different directories. The Windows NTFS file system is case-insensitive in which "Foo" and "foo" both refer to the same file, and two files differing by case only may not exist in the same directory. Therefore when allocating a directory name consider the name case-insensitive and never allocate two names that differ by case only. But when referring to a directory name always assume it is case sensitive and use the true case of the directory name.
Release Version	Dotted decimal format appended to the end of the name. Optional	- Used when multiple versions of a component may be installed concurrently under the same \$INSTALL_HOME - Typically used at the major release level when component incompatibilities are introduced.

AREA	RECOMMENDATION	RATIONALE
Reserved Names	bin, etc, var, include, lib, lib64, man	Components must not use any of the names reserved for the filesystem layout itself. It is also recommended that teams avoid using names that could be easily confused with these reserved names.

2.7.2 \$COMPONENT_ID Best Practices

Consolidations should come up with their own set of best practices concerning component product identifiers, and maintain a registry to avoid name space collisions. A draft version of such naming standards is documented at this link:

<http://jesarch.sfbay.sun.com/Wiki.jsp?page=ComponentNaming>

In absence of consolidation specific best practices the following guidelines may be used:

Component ID's should:

1. Be all lower case
2. Start with a-z
3. Have affinity with the component's package name (e.g. base it on the primary RPM name minus the leading sun-).
4. Restrict the version number (if used) to the major release level if practical.

Some examples of well formed names: javadb, appserver9, perl5.8.4

If a component uses a version number in its component ID, it has the option of dropping that version number when the component ID is used in the `etc/` or `var/` directories if the component has some other mechanism to allow simultaneous use of different versions of the component in the same `$INSTALL_HOME` (for example if subdirectories under `var/$COMPONENT_ID` are already versioned).

The version number in the component ID should not take the place of proper versioning practices nor reduce the responsibility for maintaining backwards compatibility. For example a simple component that contains only a shared library must version that library correctly and likely should not version its component level directory.

3 Reference

[0]	Title: Multiple Installation and Management of Layered Products for Java Enterprise System (WSARC/2006/130) URL: http://sac.sfbay.sun.com/WSARC/2006/130/
[1]	Title: <i>RFC-2119: Key words for use in RFCs to Indicate Requirement Levels</i> URL: http://rfc.net/rfc2119.html
[2]	Title: <i>System Architecture Council: Interface Taxonomy</i> URL: http://sac.eng/cgi-bin/bp.cgi?NAME=interface_taxonomy.bp
[3]	Title: <i>Recommended Installation Locations for Solaris-compatible Software Components</i> URL: http://sac.sfbay/cgi-bin/bp.cgi?NAME=install_locations.bp

[4]	Title: <i>Sun Linux Installation Location Standard for Add-On Software</i> URL: http://sac.sfbay.sun.com/WSARC/2004/304/opinion.txt URL: http://sac.sfbay.sun.com/WSARC/2004/304/inception.materials/locations_1.3.txt
[5]	Title: Java Enterprise System Internationalization Architecture URL: http://i18n.red.ipplanet.com/orion/orioni18narchdoc.html
[6]	Title: Requirements for Sun projects delivering on the Windows platform URL: http://sac.sfbay.sun.com/arc/WSARC/2002/494/opinion.txt
[7]	Title: Interface Taxonomy URL: http://sac.sfbay.sun.com/cgi-bin/bp.cgi?NAME=interface_taxonomy.bp
[8]	Title: Namespace for Solaris on amd64 URL: http://sac.sfbay.sun.com/PSARC/2004/619
[9]	Title: Pathnames for 64-bit Solaris URL: http://sac.sfbay.sun.com/PSARC/1997/220
[10]	Title: Solaris Manual Page: filesystem(5) URL: http://docs.sun.com/app/docs/doc/816-5175/6mbba7f0e?a=view
[11]	Title: Filesystem Hierarchy Standard (FHS) URL: http://www.pathname.com/fhs/
[12]	Title: Design Specifications and Guidelines - Integrating with the System URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwue/html/ch11b.asp
[13]	Title: Sun RPM Packaging Standards URL: http://sac.sfbay.sun.com/WSARC/2004/009/materials/current
[14]	Title: Solaris 2.x Compatible Software Packaging Requirements and Guidelines URL: http://solaris.eng/benet/Packaging/SW_Pkg_Rqrmnts,v4_0.pdf
[15]	Title: Java ES Multi-Platform Support (Windows and HP-UX) URL: http://sac.sfbay/WSARC/2004/604/commitment.final/