



Operating System Co-existence for Unbundled Software Policies and Best Practices

Version 0.1
Aug 10, 2007

DRAFT

Table of contents

1 INTRODUCTION.....	3
1.1 DEFINITIONS.....	3
2 GENERAL CONCEPTS.....	4
2.1 IDENTIFIERS.....	4
2.2 SINGLE INSTALL BEHAVIOR.....	5
3 DESKTOP INTEGRATION.....	5
3.1 START MENU.....	5
3.2 TOOLBAR NOTIFICATION.....	6
3.2.1 <i>Gnome</i>	6
3.2.2 <i>Windows</i>	6
4 SERVICE INTEGRATION.....	6
4.1 SOLARIS SMF.....	6
4.2 LINUX INIT.D.....	9
4.3 WINDOWS SERVICES.....	10
5 WINDOWS REGISTRY.....	10
6 PORT NUMBERS.....	10
7 REFERENCE.....	10

1 Introduction

This document is a companion to *File System Layout for Unbundled Software: Policies and Best Practices* (WSARC/2006/239) and *Dependency Wiring for Unbundled Software: Policies and Best Practices* (WSARC/2006/239).

This document gives best practices and requirements for integrating with Operating System services in a multi-install environment – that is an environment where multiple instances of an application may be installed into a single OS instance. Most conventions today assume that only one copy of an application is installed on a system – or occasionally multiple if they are different versions of the application.

But multi-install encourages not only multiple instances of different versions of an application, but multiple installations of the same version of an application. This increases the likely hood of clashes occurring in the namespaces of some OS services.

One such service is the OS file system – that case is covered by WSARC/2006/239. This document is meant to cover the other common OS services.

1.1 Definitions

Component: a software component that makes up part of a software consolidation. A component could be as large as an application server, or as small as a library.

\$COMPONENT_ID: a unique ID (name) that a component uses to name its subdirectories under the Install Home installation directory.

Hard Dependency: a dependency where a component will not function correctly if the dependency is not satisfied. Also known as an absolute dependency or a mandatory dependency.

Install Home: a self-contained umbrella directory that contains the installation image for a software consolidation.

\$INSTALL_HOME: a shorthand notation to refer to the Install Home umbrella directory

Soft Dependency: a dependency where a component will still function correctly even if the dependency is not satisfied, although possibly with reduced functionality. Also known as an optional dependency.

2 General Concepts

2.1 Identifiers

With the advent of Mult-Install it is possible that multiple copies of a software application may be installed on the system at the same time – either multiple copies of the same version or of different versions. Therefore we need techniques to differentiate between these multiple copies and prevent clashes in the various system namespaces.

The following are types of identifiers that will be used in subsequent portions of this specification to differentiate between multiple copies of a software application:

	Description	Pluses	Minuses	Example
Full Path (to install image)	Full path to the software's installation location in the native platform's syntax.	Guaranteed unique Conveys useful information	Can be long and cumbersome	/home/dipol/software/AppTier9
Abbreviated Path (to install image)	Shortened form of Full Path. Constructed by removing center part of Full Path and replacing with “. . .” to meet an arbitrary length requirement (exact algorithm TBD). If arbitrary length is longer than the Full Path, then this is the same as the Full Path.	Compact Conveys useful information	Not guaranteed unique	/home/di.../AppTier9
Instance Number	A unique number identifying the installation instance.	Very Compact Guaranteed unique	Conveys no useful information Requires logic to generate	2
Instance Name	A name provided by the user	Compact Likely unique Conveys useful information	Not guaranteed unique Requires user intervention	Staging

In addition to these there are existing identifiers currently in use:

	Description	Examples
Product Name	The application's marketing product name, or some subset of it.	Sun Java System Application Server Application Server
Component ID	The Component ID chosen to uniquely identify this component. See File System Layout Specification	appserver
Marketing Version	The product's marketing version string. May be abbreviated to just major (or major.minor) version depending on use case.	9, 9.1, 3.7 Update 1
Engineering Version Number	The product's engineering version number. May be abbreviated to just major (or major.minor) version depending on use case.	9, 9.1, 3.7.0.1

2.2 Single Install Behavior

It is important that the common, single install case of a product look like a traditional single product installation. The mechanisms used to differentiate multi-install instances are not needed in this case and add clutter to the user experience. Therefore the use of such mechanisms should be minimized in the single instance scenarios.

3 Desktop Integration

3.1 Start Menu

Start menu entries should be arranged in hierarchical order with the consolidation at the top, followed by individual products followed by the product's specific menu entries. For example:

Sun Application Tier 9 -> Message Queue 4 -> Admin Console

Specific guidance:

1. Menu entries for the consolidation and products should contain version numbers to differentiate between multiple installations of different versions of products. The version number should consist of only the major version number. This reduces clutter and eliminates the need to update menu entries when a software update is applied to the consolidation.
2. The first time a specific version of a consolidation is installed on a system there should be no Abbreviated Path appended to the consolidation name (see item #3).
3. When an entry for a consolidation added to the start menu, if there is a collision with an existing entry for that version of the consolidation then the Abbreviated Path of the install location should be appended to the menu entry enclosed in parenthesis. For example:

Sun Application Platform 9 (/home/di.../AppTier9)

The current recommendation is that the Abbreviated Path be truncated to 20 characters, although that number may be refined by consolidation specific guidelines.

4. Exact placement in the Start/Launch menu is left to the discretion of the project, but in

general consolidation menu entries should be placed at a relatively high level in the menu hierarchy.

5. The ToolTip for menu items should include the full path to the install image [XXX does Windows do this by default??]

Example [XXX Replace with screen shot]:

```
Sun Application Platform 9                -> Message Queue 4        ->
                                           Application Server 9 ->
                                           . . .
Sun Application Platform 9 (C:\Staging\AppTier 9) ->
Sun Application Platform 10                ->
```

3.2 Toolbar Notification

3.2.1 Gnome

3.2.2 Windows

4 Service Integration

The following sections apply to projects that deliver “services”. That is long running applications that provide services to other software.

4.1 Solaris SMF

Projects should start adopting Solaris SMF in place of the legacy `/etc/init.d` initialization scripts. Due to the wide variety of project architectures it is difficult to give specific, rigid requirements for the use of SMF for multi-install. Instead we offer the following general guidance:

1. Projects must support multiple instances of their services running concurrently. Therefore the `<single_instance/>` attribute should not be used.
2. Service names (name attribute of `service`) should lean toward the generic. For network services they should reflect the network service being provided (`http`, `ldap`, etc). For application services they should reflect the general capability or function of the service.
3. When multiple copies of an application are installed on the system and multiple instances of the application service need to be registered with SMF, then those instances should be instantiated as specific service instances of the more generic service. They should not result in brand new services. For example, if I install 3 copies of Web Server on my system I would have one `network/http` service with 3 service instances. This is true even if those were 3 different versions of Web Server.
4. If a project supports multiple application instances within one install image (like most do), then those application instances would also appear as service instances of the more generic service.¹
5. Service instance names (name attribute of `instance`) must be managed to avoid namespace conflicts.
 - A. For multi-install the first install instance should use the common instance name. For example: `default` or `admin-server`.

¹This, of course, may vary by project architecture. Application Server, for example, has Node Agents that manage server instances. In this case AS may choose to have one Node Agent instance in SMF, which manages multiple J2EE server instances.

- B. Subsequent installs should differentiate either by allowing the user to provide a name, or by incorporating some unique identifier such as a product instance or domain name, the install location, the instance data location. If a variation of a path is used then the slashes should be converted to dashes. For example: `var-opt-sun-webserver7-admin-server`.
6. Service instances should be disabled by default (`enabled='false'`) to adhere to secure by default. If the user is explicitly asked whether or not they want the service to always be started at boot time, then it is acceptable to enable the service for the user.
7. If a project ships an example SMF manifest file it should be located in: `$INSTALL_HOME/var/$COMPONENT_ID/svc/manifest`
8. If a project uses method scripts to control their service then those scripts should be placed in `$INSTALL_HOME/$COMPONENT_ID/lib/svc/method`.

The following is an example manifest for the Message Queue broker service. It should be considered an illustrative example of one possible way to handle multiple product instances. It should not be considered a definitive example of how to correctly write a service manifest.

```

<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">

<service_bundle type='manifest' name='SUNWiqqu:mqbroker'>

<service name='application/jms/mqbroker' type='service' version='1'>

  <dependency
    name='local-filefilesystems'
    type='service'
    grouping='require_all'
    restart_on='none'>
    <service_fmri value='svc:/system/filesystem/local' />
  </dependency>

  <dependency
    name='network'
    grouping='require_all'
    restart_on='none'
    type='service'>
    <service_fmri value='svc:/network/service' />
  </dependency>

  <exec_method
    type='method'
    name='stop'
    exec=':kill'
    timeout_seconds='60'>
  </exec_method>

  <instance name='default' enabled='false'>
    <exec_method
      type='method'
      name='start'
      exec='/opt/AppTier9/mq/lib/svc/method/mqbroker start'
      timeout_seconds='60'>
    </exec_method>
  </instance>

  <instance name='testbroker' enabled='false'>
    <exec_method
      type='method'
      name='start'
      exec='/opt/AppTier9/mq/lib/svc/method/mqbroker start'
      timeout_seconds='60'>
    </exec_method>
    <property_group name='options' type='application'>
      <stability value='Evolving' />
      <propval name='broker_args' type='astring' value='-name testbroker -port 7777' />
    </property_group>
  </instance>

  <instance name='home-dipol-AppTier10-default' enabled='false'>
    <exec_method
      type='method'
      name='start'
      exec='/home/dipol/AppTier10/mq/lib/svc/method/mqbroker start'
      timeout_seconds='60'>
    </exec_method>
    <property_group name='options' type='application'>
      <stability value='Evolving' />
      <propval name='broker_args' type='astring' value='-port 7878' />
    </property_group>
  </instance>

  <stability value='Unstable' />

  <template>
    <common_name>
      <loctext xml:lang='C'>Message Queue Broker</loctext>
    </common_name>
  </template>
</service>
</service_bundle>

```

Some things to note about the example:

1. `<service name='application/jms/mqbroker'`: it is debatable as to what the service name should be for the Message Queue Broker. It supports JMS, but JMS is not a network protocol, it is an API. So we chose to use a fairly product oriented name in the “application” namespace.
2. `<instance name='default'`: the first instance of this service uses the default instance name. This works pretty well since we have “mqbroker” in the service name. Note that we use an explicit `<instance>` block for the default instance since multiple instances may be common with multi-install. This seems to more clearly indicate our intentions than using the `<create_default_instance>` directive.
3. `<instance name='testbroker'`: the second instance of this service is simply a second instance of mqbroker run from the same install image as the first. In this case the service instance name is the same as the instance name used for mqbroker.
4. `<instance name='home-dipol-AppTier10-default'`: the last instance of this service is run from a new version installed in a different install home. In this case it is using the default broker instance name, so we prepend it with the install location. Another option would have been to let the user pick a service instance name.

Also, since this instance depends on what is likely an NFS mounted directory, in real life it would probably want to specify a dependency on `svc:/milestone/multi-user-server`.

Using this example we see the following output from `svcs`:

```

#$ svcs -a mqbroker
svcs: -a ignored when used with arguments.
STATE      STIME      FMRI
disabled   16:05:10   svc:/application/jms/mqbroker:home-dipol-AppTier10-default
disabled   16:05:10   svc:/application/jms/mqbroker:testbroker
disabled   16:05:10   svc:/application/jms/mqbroker:default

```

4.2 Linux init.d

Linux uses the traditional Unix `init.d` mechanism for starting services. This results in two areas where we can have name space clashes with multi-install:

1. Initialization/termination scripts in `/etc/rc.d/init.d`
2. Symbolic links to those scripts that reside in the various `/etc/rc.d/rc?.d` directories.

Due to the wide variety of project architectures it is difficult to give specific, rigid requirements in this area. Instead we offer the following general guidance:

1. Projects must support multiple instances of their services running concurrently.
2. Initialization scripts in `/etc/rc.d/init.d` are typically named after the daemon process they control. The first installation of a product should use an unqualified version of whatever name(s) it has chosen. For example: `mqbrokerd`
3. Subsequent installations of a particular version of a product should prefix the initialization script name with the `$INSTALL_HOME` of the product, converting slashes to dashes. For example: `opt-AppTier9-mqbrokerd`. Alternate schemes are acceptable. For example using a user provided name, or prefixing with the path to instance data (as opposed to the installation home).

4. Major releases of a product (which default to a different install location) should consider incorporating the major version number of the product into the initialization script name to avoid namespace clashes with previous major releases. For example: `mqbrokerd4`
5. The symbolic links in the `/etc/rc.d/rc?.d` directories should follow the same naming convention as the initialization script they point to, prefixing the name with `S##` or `K##` as appropriate.
6. If initialization script files are provided for the user but not installed, then they should be placed in: `$INSTALL_HOME/etc/$COMPONENT_ID/init.d`.
7. Linux distributions have various mechanisms for managing init scripts (Debian's `update-rc.d` and `invoke-rc.d` [3], Red Hat's `chkconfig` [4], etc). It is recommended that you make yourself familiar with these mechanisms and any additional requirements they impose on your initialization scripts.

4.3 Windows Services

5 Windows Registry

In order to reduce platform specific code it is recommended that applications avoid use of the Windows Registry.

6 Port Numbers

Port number conflicts are an obvious by-product of multi-install. The following recommendations attempt to minimize customer impact of managing port numbers:

1. Minimize use of static port numbers. Where possible use dynamic port numbers (49152 and above). For example you may have one static port that runs a rendezvous service that advertises the dynamic port numbers in use by other services.
2. Static port numbers in the range 1024-49151 should be registered with the IANA [5]
3. Port number conflicts should be detected during the configuration phase of installation and alternate, free ports should be recommended.

7 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: Filesystem Layout for Multi-Install Architecture (WSARC/2006/239) URL: http://sac.sfbay.sun.com/Archives/CaseLog/arc/WSARC/2006/239/
[2]	Title: Dependency Wiring for Unbundled Software (WSARC/2007/445) URL: http://sac.sfbay.sun.com/Archives/CaseLog/arc/WSARC/2007/445
[3]	Title: Debian Policy Manual URL: http://www.debian.org/doc/debian-policy/
[4]	Title: Red Hat SysV Init Runlevels URL: http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/ref-guide/s1-boot-init-shutdown-sysv.html

[5]	Title: IANA Port Numbers URL: http://www.iana.org/assignments/port-numbers
[6]	
[7]	Title: Solaris 10 Managing Services URL: http://docs.sun.com/app/docs/doc/817-1985/6mhm8o5rl
[8]	Title: Web Server 7.0 SMF Support (WSARC 2007/467) URL: http://sac.eng/WSARC/2007/467/
[9]	
[10]	Title: Requirements for Sun projects delivering on the Windows platform URL: http://sac.sfbay.sun.com/arc/WSARC/2002/494/opinion.txt
[11]	Title: RFC-2119: Key words for use in RFCs to Indicate Requirement Levels URL: http://rfc.net/rfc2119.html
[12]	
[13]	Title: Interface Taxonomy URL: http://sac.sfbay.sun.com/cgi-bin/bp.cgi?NAME=interface_taxonomy.bp