HP Service Activator

Solution Separation and the Deployment Manager

Edition: V51-1A

for Microsoft Windows® Server 2008 R2, HP-UX 11i v3, Solaris 10, and Red Hat Enterprise Linux 5.4 operating systems

Manufacturing Part Number: None

July 1, 2010

Legal Notices

Warranty.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices.

Trademark Notices.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Linux is a U.S. registered trademark of Linus Torvalds

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of the Open Group.

Windows® and MS Windows® are U.S. registered trademarks of Microsoft Corporation.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

Document id: HPSA p158-pd016604

Contents

Contents

Contents

Contents

Install Location Descriptors

The following names are used to define install locations throughout this guide.

Descriptor	What the Descriptor Represents
$ACTIVATOR_DB_USER	The database user name you define. Suggestion: `ovactivator`
$ACTIVATOR_OPT	The base install location of Service Activator. The UNIX® location is `/opt/OV/ServiceActivator` The Windows® location is `<install drive>:\HP\OpenView\ServiceActivator`
$ACTIVATOR_ETC	The install location of specific Service Activator files. The UNIX location is `/etc/opt/OV/ServiceActivator` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\etc`
$ACTIVATOR_VAR	The install location of specific Service Activator files. The UNIX location is `/var/opt/OV/ServiceActivator` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\var`
$ACTIVATOR_BIN	The install location of specific Service Activator files. The UNIX location is `/opt/OV/ServiceActivator/bin` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\bin`

$ACTIVATOR_THIRD_PARTY	The location for new Java™ components such as workflow nodes and modules.
	The UNIX location is
	`/opt/OV/ServiceActivator/3rd-party`
	The Windows location is
	`<install drive>:\HP\OpenView\Service Activator\3rd-party`
	Customized inventory files are stored in the following locations:
	UNIX: *$ACTIVATOR_THIRD_PARTY*`/inventory`
	Windows: *$ACTIVATOR_THIRD_PARTY*`\inventory`

$ACTIVATOR_SSH_USER	The Secure Shell user name you define. Suggestion: `ovactusr`

$JBOSS_HOME	The install location for JBoss.
	The UNIX location is
	`/opt/HP/jboss`
	The Windows location is
	`<install drive>:\HP\jboss`

$JBOSS_DEPLOY	The install location of the Service Activator J2EE components.
	The UNIX location is
	`/opt/HP/jboss/server/default/deploy`
	The Windows location is
	`<install drive>:\HP\jboss\server\default\deploy`

$PATCH_HOME	The name of the directory in which the patch or customization is located. For patches the location is: `$SOLUTION_HOME/patches/`*`PatchDir`* For customizations the location is: `$ACTIVATOR_OPT/solutions/`*`CustomizationDir`*
$SOLUTION_HOME	The name of the directory in which the solution is located, i.e.: `$ACTIVATOR_OPT/solutions/`*`SolutionName`*

Conventions

The following typographical conventions are used in this guide.

Font	What the Font Represents	Example
Italic	Book or manual titles, and manpage names	Refer to the *HP Service Activator—Workflows and the Workflow Manager* and the *Javadocs* for more information
	Provides emphasis	You *must* follow these steps.
	Specifies a variable that you must supply when entering a command	Run the command: `InventoryBuilder` <*sourceFiles*>
	Parameters to a method	The assigned_criteria parameter returns an ACSE response.
`Computer`	Text and items on the computer screen	The system replies: `Press Enter`
	Command names	Use the `InventoryBuilder` command
	Method names	The `get_all_replies()` method does the following…
	File and directory names	Edit the file `$ACTIVATOR_ETC/config/mwfm.xml`
	Window/dialog box names	In the `Test and Track` dialog…
`Computer Bold`	Text that you must type	At the prompt, type: **`ls -l`**
Keycap	Keyboard keys	Press **Return**
`[Button]`	Buttons on the user interface	Click `[Delete]`. Click the `[Apply]` button.
`Menu Items`	A menu name followed by a colon (`:`) means that you select the menu, then the item. When followed by an arrow (`->`), a cascading menu follows.	Select `Locate:Objects->by Comment`

In This Guide

This guide describes the HP Service Activator Deployment Manager tool and the Solution Separation concept.

Some uses of the Deployment Manager require a basic understanding of Service Activator while other uses require a solid understanding of all Service Activator components and how to customize them.

You should read the *User's and Administrator's Guide* and the *System Administrator's Overview* documents before you begin using the Deployment Manager. In addition, it is recommended that you read the following documents:

- Workflows and the Workflow Manager
- Inventory Subsystem
- Developing Plug-Ins and Compound Tasks

Audience

The audience for this guide is the Solutions Integrator who is developing or customizing a Service Activator solution as well as the person installing a solution.

Terminology

The following terms are used in this guide:

- **System database** – HP Service Activator stores a lot of information in the database, such as workflows, plug-ins, compound tasks, etc. The term "system database" is used to denote this database.
- **Inventory database** – This term is used to denote the database tables, sequences, contraints, etc. that are used by the Inventory subsystem.
- **Runtime system** – The term "runtime system" denotes all files and directories that are part of HP Service Activator; i.e. all files and directories located under `$ACTIVATOR_ETC`, `$ACTIVATOR_OPT`, `$ACTIVATOR_VAR`, and `$JBOSS_HOME`.
- **Deploy** – The term "deploy" merely means "install". I.e. the steps in a typical solution deployment involve copying files into the *runtime system* and adding rows (e.g. workflows) to the *system* and *inventory database*. When deploying a solution the Deployment Manager also creates a record of all deployed components (including their checksums).
- **Undeploy** – The reverse of *deploy*.
- **Patch** – In this document the term "patch" means a patch to a solution; not to the HP Service Activator product itself. Examples of patches can be a hotfix or a service pack.
- **Customization** – The term "customization" means a customization of a solution. Technically speaking, theirs is no real difference between a patch and a customization. However, the intention is that patches are developed and delivered by the owners of the solution while customizations are developed and delivered by third parties.

1 Understanding Solution Separation and the Deployment Manager

What is Solution Separation?

HP Service Activator is an open and flexible framework on which system integrators can implement various types of solutions. A typical HP Service Activator solution contains a range of different components, such as:

- **Workflows** – implementing the execution logic
- **Plug-ins** – acting as the glue between Service Activator and the activation target and executing the configuration changes
- **Inventory** –a repository for resources and services
- **Custom workflow nodes and handlers** – typically used to extend the capabilities of the Workflow Manager or to optimize workflows by creating more powerful nodes
- **Custom workflow manager modules** – adding new integrations to Service Activator; e.g. a module adding a new northbound interface protocol
- **XML files/templates** – used by workflow nodes to construct messages to be sent to activation targets or other applications
- **UI components (typically JSPs and Servlets)** – adding solution specific views to Service Activator's web-based UI
- **Northbound interfaces** – for instance, a Web Service interface through which service specific workflows can be launched

In previous versions of Service Activator managing multiple solutions running on a single Service Activator instance required careful planning. For instance, it was not possible for two solutions use identical inventory bean names even though they belonged to separate packages. Hence, if two solutions both wanted to model customers it was necessary to use different inventory bean names; e.g. "Customer" and "Subscriber".

As of version 5.0, HP Service Activator incorporates a lot of improvements allowing a much better separation between solutions. An optional solution name that can be added to workflows, inventory resource definition files, and inventory trees, along with best-practices for other components (e.g. plug-ins, template files, etc.) can ensure trouble-free coexistence of multiple solutions on a single Service Activator instance.

In addition to providing better separation between solutions, the solution separation concept also provides better separation between the components in a solution and the core components in Service Activator. This has several advantages:

- It is easier for the solutions integrator to identify which files and components belong to a particular solution and which belong to the Service Activator product itself.

- Installing (a.k.a. “deploying”) and uninstalling (a.k.a. “undeploying”) solutions becomes easier and less error-prone.

Finally, the Deployment Manager has support for versioning as well as patching and customization processes. Versioning is necessary for the Deployment Manager to find out which patches and/or customizations are compatible with the currently installed version of a solution; hence, the risk of accidentally deploying an incompatible solution patch or customization is dramatically reduced.

Understanding the Deployment Manager

The Deployment Manager is a HP Service Activator utility that guides you through the necessary steps in deploying and undeploying solutions onto Service Activator; it can operate on a single server or on multiple servers.

The capabilities of the Deployment Manager go beyond solution deployment and undeployment. The following list describes the main capabilities of the Deployment Manager:

- Deploying and undeploying Service Activator solutions on a single server or on multiple servers.
- Deploying and undeploying solution patches and customizations. This includes generation of conflict files in case a patch or customization encounters problems during deployment.

NOTE The Deployment Manager only supports patch and customization operations on the local server; i.e. these operations are not available for remote servers.

- Verifying deployed solutions; i.e. calculate MD5 checksums of all deployed components and compare them to the MD5 checksum values that were saved when the solution (including potential patches and customizations) was deployed. This will provide the user with a list of components that have been modified after the solution was deployed.
- Checksum calculation and comparison. This operation is particularly useful if you want to compare two (or more) servers and be confident that the servers are identically configured. Color codes are used to highlight differences between servers. The checksum operation uses the MD5-Digest algorithm.
- Copying of Service Activator configuration files from one server to other servers. This feature allows you to edit configuration files on only one server and then use the Deployment Manager to push the configuration files to other servers.

In addition, the Deployment Manager provides four wizards for guiding you through some of the most common sequences of tasks. These wizards intentionally provide a limited set of options. If you want access to the full set of options you must perform the operations one by one instead of using the wizards.

Secure shell (SSH) is used for communicating between servers. This means that an SSH daemon must be running on all servers that are managed by the Deployment Manager. In addition, if SSH keys are generated on all servers and copied to the `authorized_keys` files it is possible for the Deployment Manager to log into remote servers without prompting for username and password. This is described in more detail in Chapter 3 on page 39. In addition, Chapter 3 on page 39 guides you through the Deployment Manager’s configuration options.

The Deployment Manager has a graphical user interface as well as a command-line interface, which is convenient when working over slow network connections or for calling the Deployment Manager from custom scripts. The Deployment Manager GUI and command-line interface are described in more detail in Chapter 5.

NOTE	Throughout this document the term “server” is used to describe a server with Service Activator installed. The term “cluster node” (or simply “node”) is used to describe a server that belongs to a cluster. Hence, a “cluster node” is also a “server”, but a “server” is not necessarily a “cluster node”.

Solution Deployment Scenarios

This section describes scenarios that illustrate how the Deployment Manager can be used; however, the use of the Deployment Manager is not limited to the scenarios described here.

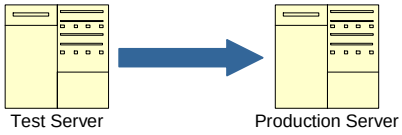
Standalone

In the simplest case, the Deployment Manager is used in a standalone mode; i.e. it is used entirely to operate against the local server. In this case the Deployment Manager can mainly be used to deploy and undeploy solutions as well as verify installed solutions.

Test/Production Server Environment

Another scenario in which using the Deployment Manager makes good sense is illustrated in Figure 1-1.

Figure 1-1 Using Deployment Manager to manage a test server and a production server



In this scenario the Deployment Manager can be used to push a solution that has already been verified to work on a test server to a production server and to deploy it on the production server. Additionally, changes in Service Activator configuration files can be tested on the test server and upon successful verification they can be copied to the production server using the Deployment Manager.

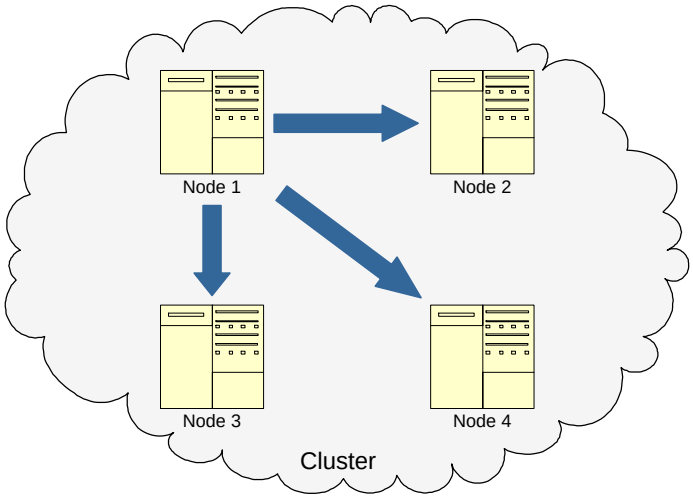
Two wizards are built into the Deployment Manager for guiding you through the following tasks:

- Copy a solution from the test server to the production server, and deploy it on the production server. Finally, compare the two servers and present the differences to the user.
- Compare the test server with the production server and present the differences to the user.

Cluster Environment

A more complex scenario where using the Deployment Manager makes good sense is shown in Figure 1-2. Four servers running Service Activator have been configured to form a cluster (whether it is a test cluster or a production cluster makes no difference).

Figure 1-2 Using Deployment Manager to manage a Service Activator cluster



In this scenario the Deployment Manager can be launched on any of the four cluster nodes. From here you can push a solution that has already been deployed on the local node to all the other nodes in the cluster. Additionally, local changes in Service Activator configuration files can easily be copied to the other nodes in the cluster ensuring that they are all identically configured.

Two wizards are built into the Deployment Manager guiding you through the following tasks:

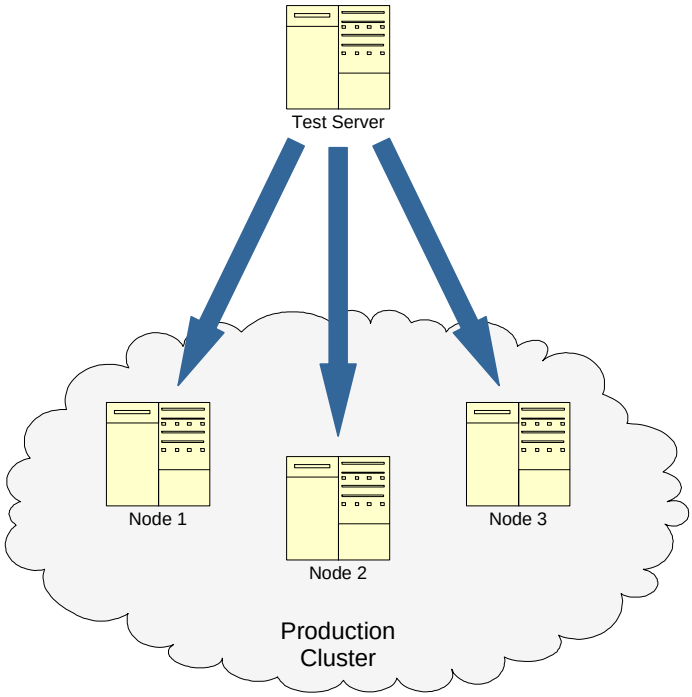
- Copy a solution from the local node to the other nodes in the cluster, and deploy it on the other nodes. Finally, compare the local node to the other nodes one by one and present the differences to the user.
- Compare the local node to the other nodes one by one and present the differences to the user.

Hybrid Environments

While the two scenarios already described are supported by the Deployment Manager through built-in wizards it may make good sense to use the tool in other scenarios.

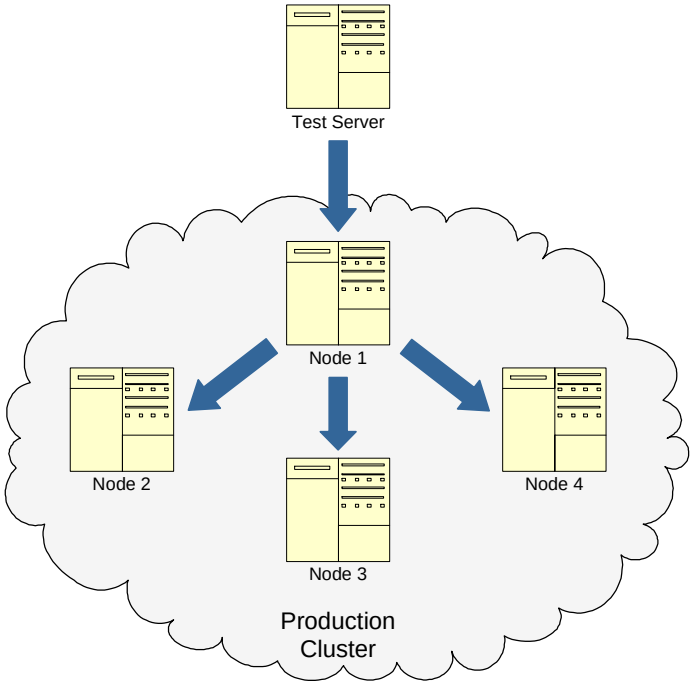
Figure 1-3 and Figure 1-4 show two scenarios where the Deployment Manager can be used to manage a hybrid environment where a test server is running standalone (from the perspective of the Service Activator runtime system) and three nodes a forming a production cluster.

Figure 1-3 **Managing a Service Activator production cluster directly from a test server**



In the example shown in Figure 1-3 the Deployment Manager is used to push solutions directly from the test server to the three cluster nodes. There are no wizards supporting this scenario; hence, if you wish to use the Deployment Manager in such a scenario you must use the individual operations provided by the tool. In general, the use of the Deployment Manager in a scenario like this is discouraged unless you have a deep understanding of Service Activator 5.0 and solution separation.

Figure 1-4 **Managing a Service Activator production cluster indirectly from a test server**



In the scenario shown in Figure 1-4 you see the full power of the Deployment Manager. This scenario is, in essence, a combination of the two scenarios shown in Figure 1-1 and Figure 1-2.

In a scenario like this, you can first launch the Deployment Manager on the test server and use a built-in wizard to push a solution – including configuration files – to one of the nodes in the cluster (which one of the cluster nodes you choose does not make any difference). Then, in the second step, you can launch the Deployment Manager on the cluster node onto which the solution was previously deployed and use a wizard to push the solution to the other nodes in the cluster.

Solution Patching and Customization Processes

This section gives a brief overview of the patch and customization processes that are supported by the Deployment Manager.

Patch/Customization Deployment and Undeployment

The Deployment Manager supports deploying and undeploying patches/customizations on the local server. When a solution has been deployed the Deployment Manager can search for patches and customizations that are compatible with currently deployed version of the solution. The user can pick one of the available patches or customizations (typically, there is only one patch or customization to choose from) and use the Deployment Manager to deploy it.

If a solution has been patched or customized it is not possible to undeploy the solution. The user must first use the Deployment Manager to undeploy all patches and customizations. After all patches and customizations have been undeployed, the user may undeploy the solution.

Conflicts During Patch/Customizaton Deployment

When the Deployment Manager deploys solutions, patches, and customizations it creates a record of all deployed components as well as their MD5 checksums. This means that the Deployment Manager always knows which components it has deployed and what their MD5 checksums should be.

During deployment of a patch or customization, the Deployment Manager will use the recorded MD5 checksums and compare them to the checksums of those components that are about to be overwritten by the patch or customization just before they are actually overwritten.

If the MD5 checksums for a component are identical, the Deployment Manager will replace the component in the runtime system (and create a backup of the component that was overwritten).

However, if the MD5 checksums for a component are not identical, the Deployment Manager will report a so-called “conflict” and ask the user how to proceed. The user can either choose to keep the old version of the component in the runtime system or to overwrite it with the new component (and create a backup of the component that was overwritten). Regardless of the user’s choice, the Deployment Manager will create a file that lists all conflicts that were encountered when the patch or customization was deployed. This list of conflicts can in some cases be used by the solution developer to identify potential issues with the patch or customization.

Deploying Patches/Customizations in “Check Mode”

Often it will be desirable to check for conflicts before a patch or customization is actually deployed. This can be done by running the patch or customization deployment operation in so-called “check mode”. When running in “check mode” the Deployment Manager carries out all operations as it would otherwise do, except for the following:

- The components in the patch or customization are actually not deployed; i.e. the Service Activator runtime system will be left intact.
- The Deployment Manager does not record the checksums of the components in the patch or customization (because the patch/customization was actually not deployed).

Hence, after having completed a patch/customization deployment operation the Deployment Manager will have created a file listing all conflicts (if any). Ideally, there should be no conflicts when deploying a patch or customization. Conflicts typically indicate that there are issues that need to be resolved before deploying the patch or customization.

When running a patch/customization deployment operation in “check mode” it is also possible to check against an earlier version of the solution (i.e. earlier than the current version of the deployed solution). This function can be used to assist the solution developer in changing a customization that has been developed for an earlier version of a solution into a new version of the customization that can be deployed on the currently installed version of the solution. For more on this topic, read Chapter 4 on page 43.

2 Solution Separation

In previous versions of Service Activator it was possible to deploy multiple solutions on a single Service Activator instance. However, it required great care to avoid collisions in inventory bean names, workflow names, database table names, etc.

As of version 5.0 of HP Service Activator the solution integrators have been equipped with a set of capabilities allowing a much better separation of solutions. An optional solution name is now supported for workflows, inventory resource definition files, and inventory trees and along with a set of best-practices for other components it is possible to ensure a conflict free coexistence of multiple solutions.

This chapter explains the solution separation concept introduced in Service Activator 5.0 including the best-practices that should be applied to the components in a solution. In addition, it is described how to create a version file as well as a solution deployment file that specifies how the different components in a solution, patch, or customization are deployed.

NOTE The terms *deployment file* and *deployment descriptor* are used interchangeably in this document.

Overview

The main idea behind the solution separation concept is to divide the solution components from the Service Activator runtime components as well as from components belonging to other solutions.

Contrary to previous versions of HP Service Activator, the components in a solution are now located in a directory completely separate from the Service Activator runtime components. The Deployment Manager tool is then used to deploy the solution components into the Service Activator runtime system based on the specifications in a deployment file.

Deploying a solution means copying files from the solution directory to appropriate Service Activator directories, creating roles, creating SQL tables, constraints, indexes, sequences, etc. and storing workflows, inventory trees, plug-ins, compound tasks and custom SQL into the database.

The solution separation concept does not foresee an automated process for configuring Service Activator configuration files. For instance, one solution may require a special module to be configured in the Workflow Manager's configuration file (`mwfm.xml`) while another solution may need another module (or the same module with a different set of parameter values). Ensuring that Service Activator is configured appropriately is a manual task.

NOTE HP Service Activator must be shut down while deploying or undeploying a solution. This is particularly important when running on Windows systems. In this way you can be sure that no files or directories are locked by Service Activator.

Backup

When a solution, patch, or customization is deployed into the Service Activator runtime system, files or other components (such as workflows, plug-ins, etc.) may sometimes be overwritten during the deployment operation (provided that the "overwrite" attribute in the deployment descriptor is set to 'true'). If this happens the Deployment Manager will make sure to store a backup of the component that would otherwise be overwritten in a backup directory located in the solution directory.

If, at a later time, the solution, patch, or customization is undeployed, the original components will be restored from this backup directory. Backup directories will *not* be deleted after a solution, patch, or customization has been undeployed. Instead a new backup directory will be created next time the solution, patch, or customization is deployed by appending a sequence number to the directory name. Hence, if you have deployed and undeployed a solution, patch, or customization multiple times you will find multiple backup directories in the solution directory (`backup`, `backup_1`, `backup_2`, `backup_3`, …, `backup_`*`N`*). It is a manual task to clean up the backup directories if they are no longer used.

Atomicity

The Deployment Manager strives to ensure that all deploy (and undeploy) operations are conducted as an atomic operation. This means that if an error occurs during solution, patch, or customization deployment (or undeployment) the Deployment Manager will roll back all previous steps to ensure that the system is left intact.

For example, if a file to be copied into the Service Activator runtime system collides with an already existing file (or if an error occurs when writing to the database) the Deployment Manager will make sure that all changes are reverted – hence, deployed files will be deleted again, overwritten files will be restored (from the solution's backup directory), and the ongoing database transaction will be rolled back (the Deployment Manager runs all operations against the database in a single transaction).

NOTE

If the "overwrite" option is set to true the deploy operation will *not* fail in the event that a file or database component in the Service Activator runtime system is to be overwritten. See also the Section "Creating a Deployment File" on page 28.

Solution Directory Structure

In order to leverage on Service Activator's built/in mechanisms for managing solution separation, all solutions must be located in separate directories named `$ACTIVATOR_OPT/solutions/`*`SolutionName`*. The solution directory is structured as follows:

```
SolutionName/
            |-3rd-party/
            |          |-classes/
            |          |-lib/
            |-backup/
            |-bin/
            |-CT/
            |-docs/
            |-etc/
            |    |-config/
            |    |       |-inventoryTree
            |    |-dc_tasks/
            |    |-designer/
            |    |         |-handlers/
            |    |         |-nodes/
            |    |-nls/
            |    |-scripts/
            |    |-sql/
            |    |-template_files/
            |    |-tests/
            |    |      |-messages/
            |    |-umm/
            |    |-web_services/
            |    |-workflows/
            |-install/
            |-inventory/
            |-lib/
            |-plugins/
            |-UI/
            |-var/
                 |-dc/
                 |-log/
                 |-tmp/
```

You may choose to create more directories than those listed here; however, they will be unknown to the Deployment Manager.

The recommended use of the directories is described in the following subsections.

3rd-party

This directory is typically used for Java classes (in the `classes` directory) and Java archives (in the `lib` directory) implementing custom workflow nodes and modules. There are, however, no restrictions to the kind of files that can be put into this directory.

backup

The `backup` directory is created by the Deployment Manager during a deploy operation. All files, plug-ins, workflows, inventory trees, etc. that would otherwise be overwritten by the Deployment Manager during a deploy operation will be backed up in this directory.

If a backup directory already exists, the Deployment Manager will append a sequence number and then create a new backup directory. This means that if a solution has been deployed followed by deploying one or more patches or customizations, multiple backup directories will be created with ascending sequence numbers – see the following example:

```
SolutionDir/
           |-backup/
           |        ...
           |-backup_1/
           |          ...
           |-backup_2/
           |          ...
           |-backup_3/
           |          ...
           | ...
```

When deploying a solution, the Deployment Manager will record all files, plug-ins, workflows, inventory trees, etc. in a file called:

SolutionDir/install/checksums_*N*.xml, where *N* is a sequence number

If, during the deploy operation, the Deployment Manager overwrites a file or database component, the Deployment Manager will save a backup of the original file or database component in the specified backup directory and record the exact location of the backup using the "backup" attribute in the `checksums_N.xml` file.

If third-parties need access to a backup file or database component, they must use the information stored in the `checksums_N.xml` file (e.g. to do automatic merging) instead of relying on a specific location in the backup directory.

During the undeploy process the Deployment Manager will first remove all solution components, patch components, or customization components from the runtime system and then restore the previous versions of all overwritten components using the files stored in the directory specified by the `<Backup>` element. Hence, the HP Service Activator runtime system will be left in same state as it was before.

bin

This directory can be used as a placeholder for executable files (binary files as well as shell scripts) that are used as part of the solution. The files in this directory will *not* be deployed into the Service Activator runtime system.

CT

This directory is used for solution specific compound tasks.

docs

This directory can be used for containing documentation for the solution. Files in this directory will *not* be deployed into the Service Activator runtime system.

etc/config

This directory should contain configuration files, DTD files, etc. that are specific for the solution.

NOTE The Deployment Manager cannot merge configuration files. If configurations changes are needed in Service Activator configuration files (e.g. "mwfm.xml") the changes must be performed manually.

etc/config/inventoryTree

This directory contains Inventory Tree definition files.

etc/dc_tasks

HP Service Activator's Data Collector utility can be extended with custom tasks. Solution specific data collection tasks can be added to this directory. Custom data collection tasks are *not* deployed to the Service Activator runtime system.

etc/designer

If the solution uses custom workflow nodes or custom error/end handlers their XML specifications can be added to this directory. Note that the Java implementation of custom nodes and handler need to be added to the 3rd-party directory. Custom handlers must be added to the `handlers`

directory whereas the custom nodes must be added to the `nodes/SolutionName` directory. Custom nodes should never be added to the `nodes/builtin` directory – this directory is reserved for workflow nodes that are shipped with the Service Activator product itself.

etc/nls

This directory can be used as a placeholder for localized version of Service Activator resource bundles.

etc/sql

This directory can be used for solution specific SQL files – for instance if a solution needs custom database tables, sequences, etc.

Files located in this directory will *not* be copied into the Service Activator runtime system. However, the SQL will be executed against the Service Activator system database if specified in the deployment descriptor.

NOTE

You should *never* have `commit` statements in your SQL files because this will prohibit the Deployment Manager from being able to successfully roll-back a deploy or undeploy operation is an error is encountered. The Deployment Manager will automatically commit all database operations as the last step of a deploy or undeploy operation.

etc/scripts

Scripts that are to be executed by the Deployment Manager during a deploy or an undeploy operation must be located in this directory.

etc/template_files

This directory typically contains Java template files (used by the Java node or the JavaRule node in the Workflow Manager), XML templates, XSL templates, etc.

etc/tests/messages

This directory can be used for test messages that can be injected into the SocketListener module from the Service Activator's web-based UI.

etc/umm

This directory can contain XML files that can be used to create roles in HP Service Activator and assign inventory trees, operation types, and branch types to roles.

etc/web_services

This directory may contain XML files that define a Web Service interface to HP Service Activator as well as WAR files (Web Archives) implementing web service interfaces.

etc/workflows

All workflows in a solution must be located in this directory.

install

This directory is used internally by the Deployment Manager to keep track of the deployment state of a solution. When a solution (or patch/customization) is deployed the Deployment Manager will create a new checksum file in this directory called `checksums_N.xml` where *N* is a sequence number starting with 1. The checksum file contains a list of all deployed components as well as

their MD5 checksums. In addition, the Deployment Manager will create a directory called *N* (i.e. using the same sequence number as for the checksum file) where is creates a copy of the deployment descriptor that was used to deploy the solution (or patch/customization). All information stored in this directory is needed when a solution (or patch/customization) is undeployed as well as for other operations (e.g. patch/customization conflict detection and verification of a deployed solution). Finally, if conflicts are encountered while deploying a patch or customization, the Deployment Manager will create a file called `conflicts.xml` in this directory listing all conflicts; if the file already exists it will be silently overwritten.

You should *not* change anything in this directory.

inventory

This directory is used for all inventory resource definition files in a solution. If you choose to let the Deployment Manager generate all inventory components for you (i.e. you do not need to customize any of the auto-generated components) a subdirectory called `temp` will be created for all output files generated by the Inventory Builder.

The reason for using a temporary work directory is that this allows you to run Inventory Builder manually in the `inventory` directory risking that Deployment Manager will overwrite your files.

lib

This directory can be used for Java classes, Java achieve files, etc. that are used locally by scripts located in the `bin` directory. Files located in this directory will *not* be deployed into the Service Activator runtime system.

plugins

All plug-ins that are needed by a solution must be located in this directory.

UI

All custom user interface components (HTML, JSPs, JavaScripts, images, etc.) must be located in this directory.

var

This directory can be used for solution specific log files, temporary files etc. Files located in this directory will *not* be deployed into the Service Activator runtime system.

Patch and Customization Directory Structure

The directory structure for patches and customizations is identical to that of a solution, except that the base directory is different.

Patches must be located in the directory:

`$ACTIVATOR_OPT/solutions/`*`SolutionName`*`/patches/`*`PatchDir`*

Customizations must be located in the directory:

`$ACTIVATOR_OPT/solutions/`*`CustomizationDir`*

`PatchDir` and *`CustomizationDir`* should consist entirely of alphanumeric characters and underscores; other than that they can be chosen freely and are completely unrelated to the name of the solution and the label or version of the patch or customization.

Solution Separation Best-Practices

Locating your solution, patch, and customization components in a separate directory does not in itself ensure solution separation. You need to set the solution name in all components supporting this and apply some best-practices on other components. This section will describe how you can ensure that your solution will be truly separated from other solutions.

Solution Names

The name of a solution is used in several places in Service Activator. Some files (such as JSPs and struts classes generated by the Inventory Builder) will be located in directories using the solution name as directory name. In addition, Java classes generated by the Inventory Builder for the UI part will use the solution name as a part of the Java package name.

This means that there are restrictions to what you may use as the name of your solution; your solution name must follow these guidelines:

- The solution name must begin with a letter (uppercase or lowercase); i.e. `a-z` or `A-Z`.
- The solution name must only contain alphanumeric characters and underscores (hence, spaces are not allowed).
- The length of the solution name must not exceed 8 characters.

In addition to choosing a good name for your solution, you should also derive an abbreviated version of your solution name. For instance, if the name of your solution is `My_Sol` the abbreviated version of your solution name could simply be `MY` or `MS`. The abbreviated solution name will be used in some of the components in the following subsections.

Workflow Names

All workflow names running on a single Service Activator instance must be globally unique. This can be ensured by prefixing all your workflow names with the solution name or an abbreviated version of your solution name.

Examples:

- `My_Sol_Upload, My_Sol_Create_User`
- `MS_Upload, MS_Create_User`

As of Service Activator 5.0, three special case-packet variables intended to be used with solution separation have been added:

- `SOLUTION_NAME` – will be set to the name of the solution by the Workflow Manager when the workflow is launched (based on the name of the solution that was specified for the workflow when it was created).
- `SOLUTION_ETC` – points to the `etc` directory in the solution catalog.
- `SOLUTION_VAR` – points to the `var` directory in the solution catalog.

Inventory Resource Definition Files

When you create your inventory resource definition files the solution name comes into play in several places:

- In the `Solution` XML element
- In the database table name (the `DBTable` XML element)
- In the database constraint name (the `ConstraintName` XML element)

- In the Java package name (the `Package` XML element)

Since Oracle has limitation on the length of table and constraint names it is recommended that they are prefixed with the abbreviated version of your solution name. However, you should use the full solution name in the `Solution` and `Package` elements.

Example:

```
<Bean>
  <Name>Customer</Name>
  <Solution>My_Sol</Solution>
  <ConstraintName>MS_CUSTOMER</ConstraintName>
  <Package>com.acme.my_sol</Package>
  <DBTable>MS_CUSTOMER</DBTable>
  ...
</Bean>
```

Plug-Ins and Compound Tasks

Plug-ins and compound tasks support so-called name spaces. For solution specific plug-ins and compound tasks you should set the name space type to “PRIVATE” and the set value of the name space to match your solution name:

Example:

```
<NameSpace type="PRIVATE">My_Sol</NameSpace>
```

Template Files

You should place all template files (XML/XSL templates, Java templates, etc.) in your solution in a separate subdirectory with the same name as your solution. Hence, all solution specific template files should be deployed into the directory `$ACTIVATOR_ETC/template_files/`*`SolutionName`*.

Alternatively, you can choose to *keep* your templates in the solutions directory (i.e. *not* copy them to the `$ACTIVATOR_ETC/template_files` directory) and instead use the `SOLUTION_ETC` case-packet variable to refer to them from the workflow.

The latter approach would normally be preferred.

Custom Workflow Nodes and Handlers

All your custom workflow nodes and handlers should be grouped into a Java package with a name matching the name of your solution. In addition, you should always deploy the node XML descriptors in a separate directory with the same name as your solution; i.e. `$ACTIVATOR_ETC/designer/nodes/`*`SolutionName`*.

NOTE The descriptors for workflow nodes and handlers are only used by the Workflow Designer; not by the Workflow Manager. Hence, deploying these components mainly makes sense on a system that is used for development.

Custom Workflow Manager Modules

You should make sure that all your custom Workflow Manager modules are located in a Java package containing the name of your solution.

Creating a Version File

The Deployment Manager has support for a version file that is needed to resolve dependencies between a solution and compatible patches/customizations. As of HP Service Activator 5.1, the version file is mandatory for solutions, patches, and customizations.

The name of the version file is `version.xml` and it must be located in the base directory of the solution, patch, or customization. For a definition of the syntax of the `version.xml` file you should study the file `version.dtd` located in the `$ACTIVATOR_ETC/config` directory.

The version file must contain a type (“solution”, “patch”, or “customization”), a label (e.g. “Hotfix” for a solution patch), a major version, a minor version, and a revision. In addition, the version file may list one or more versions that this patch or customization can be deployed into (this is not applicable for solutions). Finally, the version file can optionally specify the version of the Service Activator product that is required in order to deploy this solution, patch, or customization.

An example of a version file for a solution is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Version SYSTEM "version.dtd">

<Version type="solution">
   <Label>VPN_Example</Label>
   <Major>5</Major>
   <Minor>1</Minor>
   <Revision>1</Revision>
</Version>
```

On the Deployment Manager’s UI the version of this solution will be displayed as “5.1.1” (i.e. the major version, the minor version, and the revision separated by dots).

NOTE The major version, the minor version, and the revision may contain alphanumerical characters and underscores.

An example of a version file for a patch is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Version SYSTEM "version.dtd">

<Version type="patch">
   <Label>Hotfix</Label>
   <Major>5</Major>
   <Minor>1</Minor>
   <Revision>2</Revision>
   <Requires>
      <Solution>VPN_Example</Solution>
      <Major>5</Major>
      <Minor>1</Minor>
      <Revision>1</Revision>
   </Requires>
   <Requires-Core>V51-1A</Required-Core>
</Version>
```

This patch can be deployed on top of the solution “VPN_Example” version 5.1.1; the Deployment Manager will refuse to deploy this patch onto any other version of the “VPN_Example” solution.

In addition, the patch will refuse to install unless the Service Activator version is “V51-1A” or later “V51-1A” versions (e.g. “V51-1A-4”).

HP Service Activator Version Dependency

As described above, it is possible in the version file to specify the version of HP Service Activator that is required in order to be able to install a solution, patch, or customization.

The syntax for the HP Service Activator version string is as follows:

```
<Letter><Version>-<Revision>[-Patch]
```

The first letter is “V” for all releases of Service Activator. The version is always comprised of two digits (i.e. “51” for HP Service Activator 5.1) and the revision is specified as a digit followed by a letter. The patch (if present) is simply a number, start with 1. If the patch is not specified, the number “0” is assumed.

If the <Requires-Core> element in the version is specified, the solution, patch, or customization is only allowed to be deployed on the specified version of Service Activator or later *compatible* versions. The Deployment Manager considers two HP Service Activator versions to be compatible if they have identical versions *and* revisions. In addition, the Deployment Manager requires that the patch number of the install version of HP Service Activator is greater than or equal to the patch number specified using the <Requires-Core> element.

Example:

If the value of the <Requires-Core> element for a solution is set to "V51-1A-3", the solution can not be deployed on HP Service Activator V51-1A, V51-1A-1, or V50-1A-2. It can only be deployed on Service Activator version V51-1A-*N*, where $N \geq 3$. If the version of Service Activator is V51-1B-*N*, the Deployment Manager will also refuse to deploy the solution.

Creating a Deployment File

A *deployment file* – or *deployment descriptor* – specifies how a solution, patch, or customization is to be deployed into the HP Service Activator runtime system and system database. The deployment descriptor is an XML file with a syntax defined by the `$ACTIVATOR_ETC/config/deploy.dtd` file. The file will typically be called `deploy.xml` and it must be located in the base directory of the solution, patch, or customization.

It is possible to have multiple deployment descriptors in the base directory of your solution, patch, or customization. Before each deploy operation you will need to specify which deployment descriptor to use.

Service Activator ships with a sample deployment descriptor file that is located in `$ACTIVATOR_ETC/config/deploy_example.xml`. This file will automatically be copied into the base directory of your solution, patch, or customization when using the Deployment Manager to create a new solution, patch, or customization skeleton.

This section explains the different parts of a deployment descriptor file element by element. You should study the `deploy.dtd` and the `deploy_example.xml` files and look at the examples while you read this section. Only a few of the examples in the `deploy_example.xml` file will be repeated in this section.

NOTE

In the remaining part of this document, *`BaseDir`* is used to refer to the base directory for solutions, patches, and customizations, respectively. I.e. the value of *`BaseDir`* is:

For solutions: `$ACTIVATOR_OPT/solutions/`*`SolutionName`*
For patches: `$ACTIVATOR_OPT/solutions/`*`SolutionName`*`/patches/`*`PatchDir`*
For customizations: `$ACTIVATOR_OPT/solutions/`*`CustomizationDir`*

In addition, *`SolutionDir`* is used to refer to the base directory of a solution.

Deployment File Solution Version Dependency

For patches and customizations it is possible to get the Deployment Manager to either process or skip certain parts of the deployment file. In order to use this functionality the "depend" attribute must be used. The following elements support the "depend" attribute:

- <Output> – part of the <Inventory> element
- <Deploy> – part of <SQL> element
- <Undeploy> – part of <SQL> element
- <Script-Name> – part of the <Scripts> element

The syntax of the "depend" attribute is simple a comma-separated list of solution versions, for instance "1.3.4, 1.3.5, 1.3.5-ACME". The Deployment Manager will evaluate the versions in the

comma-separated list in an *or*-fashion and only process the element if the dependency is met. I.e. if the value of the “depend” attribute is “1.3.4, 1.3.5, 1.3.5-ACME” the dependency is met if the deployed solution version is 1.3.4, 1.3.5, or 1.3.4-ACME.

If the “depend” attribute is absent the element will always be processed, regardless of the version of the deployed solution.

<File>

The <File> element is described first because it is used in several places in the deployment file. For convenience the <File> element supports the use of simple wildcards (*) as well as recursive wildcards (**). For example, the **/*.class pattern matches all files with the ".class" extension in all subdirectories (relative to a location which is defined by the parent deployment operation element).

A deployment operation will fail if a file (or another component, e.g. a workflow in the system database) is subject to be overwritten by the deployment operation, unless you set the <File> element’s “overwrite” attribute to “true”. In this case, the file that is subject to be overwritten will be moved to the specified backup directory before it is replaced with a file from the solution, patch, or customization. For more details on the backup process, read the section “<Backup>” on page 29.

The <File> element also has an optional attribute called “solution” with the default value “false”. The “solution” attribute is only applicable for patches and customizations. If the “solution” attribute is set to “true” the Deployment Manager will copy the file (or database component) from base directory of the patch or customization to the solution directory and *not* to the Service Activator runtime system when the patch or customization is deployed. If the “solution” attribute is not specified or if it is set to “false”, the file (or database component) will be copied to the solution directory *and* to the Service Activator runtime system.

For patches and customizations it is possible to use the <File> element to delete files by setting the optional “delete” attribute to “true”. Solutions can *not* use the “delete” attribute.

Finally, the <File> element has an optional attribute called “executable” with the default value “false”. Setting this attribute to “true” will cause the Deployment Manager to set the “execute” bit after the file has been copied to the desired location. The “executable” attribute will not have any effect on Microsoft Windows operating systems.

<Name>

The <Name> element is used to uniquely identify a Service Activator solution. The solution name must be identical to the directory in which the solution files reside. For patches and customizations, the solution name must match the name of the solution onto which the patch or customization can be deployed.

The solution name should also be defined in the following components:

- Workflows
- Inventory resource definition files
- Inventory trees

You should read the section “Solution Separation Best-Practices” on page 25 for details on solution separation best-practices.

<Backup>

In Service Activator 5.0 the name of the backup directory could be specified using the <Backup> element. However, in Service Activator 5.1 the <Backup> element is no longer supported. If it is specified in the deployment descriptor the Deployment Manager will silently ignore it.

<Plugins>

The <Plugins> element specifies which plug-ins should be deployed during the deploy operation. Plug-in archives (PARs) can be referenced using the <File> element or the <Core-File> element.

Plug-ins that are referenced using the <File> element must be stored in the *BaseDir*/plugins directory. Deployment of these plug-ins will fail if a plug-in of the same name and name-space is already deployed, unless the "overwrite" attribute is set to "true".

Plug-ins that are referenced using the <Core-File> element are read from the $ACTIVATOR_OPT/SPI directory. When using the <Core-File> element, the Deployment Manager will silently overwrite identically named plug-ins during the deploy operation.

The <File> as well as the <Core-File> elements support an attribute called "delete". Setting this attribute to "true" causes the Deployment Manager to delete the plug-in (i.e. undeploy it). This functionality is only supported for patches and customizations; e.g. if a patch replaces an existing plug-in with a new plug-in with a different name.

If an already deployed plug-in is subject to be overwritten during the deploy operation, the Deployment Manager will store a backup of the previously deployed version of plug-in in the directory *SolutionDir/Backup* and add a record to the checksums_*N*.xml file.

Source:	*BaseDir*/plugins	(when using the <File> element)
	$ACTIVATOR_OPT/SPI	(when using the <Core-File> element)
Destination:	Service Activator system database	

<CT>

The <CT> element specifies which compound tasks will be deployed during the deploy operation. The compound tasks must be stored in the *BaseDir*/CT directory.

The deploy operation will fail if a compound task of the same name and namespace is already deployed, unless the "overwrite" attribute is set to "true". If an already deployed compound task is subject to be overwritten during the deploy operation, the Deployment Manager will save a backup of the previously deployed version of compound tasks in the directory *SolutionDir/Backup* and add a record to the checksums_*N*.xml file.

Source:	*BaseDir*/CT
Destination:	Service Activator system database

<Third-Party>

The <Third-Party> element is used to specify which files in the *BaseDir*/3rd-party directory are to be deployed into the $ACTIVATOR_OPT/3rd-party directory during the deploy operation.

Java classes and Java archives (JARs) in the 3rd-party directory are typically used for custom workflow nodes and modules, but there are no restrictions on their usage.

If a file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old file in the directory *SolutionDir/Backup* and add a record to the checksums_*N*.xml file.

Source:	*BaseDir*/3rd-party
Destination:	$ACTIVATOR_OPT/3rd-party

Example:

```
<Third-Party>
  <File>**/*.class</File>
  <File>lib/*.jar</File>
  <File>**/*.properties</File>
  <File>classes/com/hp/examples/*.java</File>
  <File>classes/com/hp/examples/test_image.png</File>
  <File>**/README</File>
</Third-Party>
```

<UI>

The <UI> element is used to specify which files in the *BaseDir*/UI directory are to be deployed into the $JBOSS_DEPLOY/hpovact.sar/activator.war directory during the deploy operation. UI customizations typically consist of JSPs, Java classes (servlets), image files, CSS files, and JavaScript files.

If a file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old file in the directory *SolutionDir*/Backup and add a record to the checksums_*N*.xml file.

NOTE

The Deployment Manager will set the timestamp of all UI components to the *current time* after deployment as well as after undeployment. This is necessary to ensure that the JSPs are recompiled correctly by the application server.

Source: *BaseDir*/IU

Destination: $JBOSS_DEPLOY/hpovact.sar/activator.war

Example:

```
<UI>
  <File>**/*.jsp</File>
  <File>**/*.class</File>
  <File overwrite="true">**/*.gif</File>
  <File>**/*.js</File>
</UI>
```

<Inventory>

The <Inventory> element is used to specify how to deploy a solution's inventory components.

The <Inventory> element supports an attribute called “db”. If the “db” attribute is not set, the inventory subsystem will be deployed to the system database. Otherwise, if the “db” attribute is set, the inventory subsystem will be deployed to the database identified by the value of the “db” attribute.

There are two ways to deploy inventory components; only one of them can be specified at a time.

- **Automatic** – this mode is enabled by using the <Resources> element to list all inventory resource definition files (or use *.xml to select all XML files in the inventory directory). In this mode the Deployment Manager will invoke the Inventory Builder which in turn processes the inventory resource definition files and generates Java beans, JSPs, SQL, and struts configuration and Java files. In addition, the Inventory Builder will compile all Java files, and deploy all components into the Service Activator runtime system. It is possible to disable JSP deployment by setting the “deployJSP” attribute to “false” (the default value for the “deployJSP” attribute is “true”). When running in automatic mode, the Inventory Builder uses the *SolutionDir*/inventory/temp directory as work directory.

  If you need to refer to external resource definition files, you refer to them using the xmlpath attribute. Finally, external Java classes can be included in the CLASSPATH (used when the inventory beans are compiled by the Inventory Builder) by specifying the classpath

attribute. It is possible to prefix `xmlpath` and `classpath` attributes with an environment variable; e.g. `$ACTIVATOR_OPT`, `$JBOSS_HOME`, etc. For a list supported environment variables, see section "`<Other>`" on page 37.

- **Manual** – this mode is enabled by using the `<Output>` element instead of the `<Resources>` element. When using the manual mode it is assumed that the solution developer has manually invoked Inventory Builder to generate all relevant files and (optionally) made customizations to the inventory component (for instance, by modifying JSPs, adding new JSPs, adding extra statements to the SQL files, etc.).

  The `<Output>` element supports an attribute called "depend" for patches and customizations. The value of the "depend" attribute, together with the version of the deployed solution, determines whether the `<Output>` element will be ignored or processed during a deploy operation. It is possible to specify multiple `<Output>` elements in a deployment descriptor.

  In the manual mode it is necessary to explicitly specify which inventory components will be deployed into the Service Activator runtime system; see the example below.

If a file in the runtime system is subject to be overwritten by any of the components in the inventory subsystem, the Deployment Manager will save a backup of the old file in the directory *`SolutionDir/Backup`* and add a record to the `checksums_`*`N`*`.xml` file.

Source:	*`BaseDir`*`/inventory`	
Destination:	*Classes*	`$ACTIVATOR_OPT/3rd-party/inventory/classes`
	JSQL	`$ACTIVATOR_OPT/3rd-party/inventory/classes/jsql`
	JSP	`$JBOSS_DEPLOY/hpovact.sar/activator.war/jsp/inventory/SolutionName`
	SQL	Service Activator system database
	Struts-Config	`$JBOSS_DEPLOY/hpovact.sar/activator.war/WEB-INF/struts-config`
	Struts-Classes	`$JBOSS_DEPLOY/hpovact.sar/activator.war/WEB-INF/classes`

Example (automatic mode):

```
<Inventory>
  <Resources deployJSP="false">
    <File>Site.xml</File>
    <File>Router.xml</File>
    <File>User.xml</File>
  </Resources>
</Inventory>
```

Example (manual mode):

```
<Inventory>
  <Output>
    <Classes>
      <File>**/*.class</File>
      <File>**/*.properties</File>
    </Classes>
    <JSQL>
      <File>**/*.jsql</File>
    </JSQL>
    <JSP>
      <File>**/*.jsp</File>
    </JSP>
    <SQL>
      <Deploy>createSolutionName.sql</Deploy>
```

```
      <Deploy>createSequencesSolutionName.sql</Deploy>
      <Undeploy>deleteSequencesSolutionName.sql</Undeploy>
      <Undeploy>deleteSolutionName.sql</Undeploy>
    </SQL>
    <Struts-Config>
      <File>*</File>
    </Struts-Config>
    <Struts-Classes>
      <File>*</File>
    </Struts-Classes>
  </Output>
</Inventory>
```

<Inventory-Trees>

The <Inventory-Trees> element specifies the inventory trees that belong to the solution. Unless the "overwrite" attribute is set to "true", the deploy operation will fail if you try to deploy an inventory tree with the same name and solution name as an already deployed inventory tree. Also, if an inventory tree has been assigned to a Service Activator role the Deployment Manager will fail during the *undeploy* operation, unless the "undeployForce" attribute is set to "true".

The <Tree-File> element supports an attribute called "delete". Setting this attribute to "true" causes the Deployment Manager to delete the inventory tree (i.e. undeploy it). This functionality is only supported for patches or customization; e.g. if a patch replaces an existing inventory tree with a new tree with a different name.

If an inventory tree in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old inventory tree in the directory *SolutionDir/Backup* and add a record to the checksums_*N*.xml file.

Source: *BaseDir*/etc/config/inventoryTree

Destination: Service Activator system database

Example:

```
<Inventory-Trees>
  <Tree-File overwrite="true">tree1.xml</Tree-File>
  <Tree-File>tree2.xml</Tree-File>
  <Tree-File overwrite="true" undeployForce="true">tree3.xml</Tree-File>
</Inventory-Trees>
```

<UMM>

The <UMM> element specifies the XML files containing roles and their associations to inventory tree, operation types, and branch types that will be imported during a deploy operation. The Deployment Manager will not delete UMM data during an undeploy operation.

Source: *BaseDir*/etc/umm

Destination: Service Activator system database

<Workflows>

The <Workflows> element is used to specify which workflows in the *BaseDir*/etc/workflows directory are to be deployed into the Service Activator runtime system during the deploy operation.

All workflows belonging to a solution should set their solution names to the same value as the solution name defined in the deployment descriptor. This can be done using the Workflow Designer.

NOTE	Workflow names need to be globally unique. It is recommended that the names of workflows in a solution are prefixed with the solution name or an abbreviated version of the solution name. See the section “Solution Separation Best-Practices” on page 25.

If a workflow in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old workflow in the *SolutionDir*/Backup and add a record to the checksums_*N*.xml file.

Source: *BaseDir*/etc/workflows

Destination: Service Activator system database

<Web-Services>

The <Web-Services> element is used instruct the Deployment Manager to generate and deploy a Web Service interface (based on an XML file that can be created using the Web Service Designer) as well as to deploy WAR files that contain Web Service interfaces.

The <WSD> element can be used to identify a file containing a Web Service description (created using the Web Service Designer) as well as optional JAR files that are needed in order to compile the auto-generated Java sources. The XML file that describes the Web Service must be located in the *BaseDir*/etc/web_services directory.

The <war> element is used to identify WAR files containing Web Service interfaces that are to be deployed. The Deployment Manager assumes that WAR files are located in the *BaseDir*/etc/web_services directory (or in a sub-directory).

The <WSD> and the <war> elements support an optional attribute called “overwrite” (default value is “false”).

If a WAR file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old WAR file in the directory *SolutionDir*/Backup and add a record to the checksums_*N*.xml file.

Source: *BaseDir*/etc/web_services

Destination: $JBOSS_DEPLOY/hpovact.sar

<Config>

The <Config> element is used to specify which configuration files (as well as custom document type definition files, DTDs) that are to be deployed into the Service Activator runtime system during the deploy operation.

If a configuration file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old configuration file in the directory *SolutionDir*/Backup and add a record to the checksums_*N*.xml file.

Source: *BaseDir*/etc/config

Destination: $ACTIVATOR_ETC/config

<Templates>

The <Templates> element contains the template files that will be deployed into the Service Activator runtime system during the "deploy" operation. Typically, Java templates (used by the Java and the JavaRule nodes), XML templates, and XSL templates are located in this directory.

If a template file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old template file in the directory *SolutionDir*/Backup and add a record to the checksums_*N*.xml file.

Source: *BaseDir*/etc/template_files

Destination: $ACTIVATOR_ETC/template_files

<Designer>

The <Designer> element lists the XML descriptions for all custom workflow nodes and handlers that are used in a solution. The Java implementation for custom nodes and handlers are typically deployed using the <Third-Party> element (described earlier).

If a resource bundle file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old resource bundle file in the directory *SolutionDir/Backup* and add a record to the checksums_*N*.xml file.

Source: *BaseDir*/etc/designer

Destination: $ACTIVATOR_ETC/designer

<NLS>

The <NLS> element is used to list all localized resource bundle files that are to be deployed into the Service Activator runtime system.

If a resource bundle file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old resource bundle file in the directory *SolutionDir/Backup* and add a record to the checksums_*N*.xml file.

Source: *BaseDir*/etc/nls

Destination: $ACTIVATOR_ETC/nls

<Messages>

The <Messages> element specifies test message files that are deployed into the Service Activator runtime system during the deploy operation. Test messages are typically injected into the SocketListener module using the Service Activator web-based user interface.

If a message file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old message file in the directory *SolutionDir/Backup* and add a record to the checksums_*N*.xml file.

Source: *BaseDir*/etc/tests/messages

Destination: $ACTIVATOR_ETC/tests/messages

<SQL>

The <SQL> element specifies a list of custom SQL files that will be executed against the database during the deploy and undeploy operations (using the <Deploy> and <Undeploy> elements, respectively).

Common uses of this feature can be to create extra indexes, SQL sequences, or to populate the inventory database with an initial set of data.

The <Deploy> and <Undeploy> elements support an optional attribute called “db”. If the “db” attribute is not set, the SQL will be executed against the system database. Otherwise, if the “db” attribute is set, the SQL will be executed against the database identified by the value of the “db” attribute.

Finally, the <Deploy> and <Undeploy> elements support an attribute called “depend”. This attribute is only applicable for patches and customizations. The value of the “depend” attribute,

together with the version of the deployed solution, will determine whether the <Deploy> or <Undeploy> elements will be ignored or processed during a deploy or an undeploy operation.

Since SQL files are not copied to anywhere, the Deployment Manager will not back up anything while executing SQL files.

Source: *BaseDir*/etc/sql

Destination: Service Activator system database

<Scripts>

The <Scripts> element can be used to specify one or more scripts that are executed before and after the deploy and undeploy operations. It is only possible to specify one script to be executed for each of the four scenarios (i.e. before and after deployment and undeployment).

The <Script-Name> element supports an attribute called “depend”. This attribute is only applicable for patches and customizations. The value of the “depend” attribute, together with the version of the deployed solution, will determine whether the script will be executed or not during a deploy or an undeploy operation.

If a script terminates with an exit value that is different from zero, the Deployment Manager will interpret this as an error and initiate a roll-back.

The Deployment Manager can pass environment variables to the external scripts. See the Section “<Other>” on page 37 for the list of supported environment variables. In addition, the Deployment Manager can pass extra parameters, such as database username, database password etc., to the scripts.

If a deploy or undeploy operation fails, the Deployment Manager will attempt to roll back all changes. As the last step of the roll-back operation the Deployment Manager will call the <Before-Deploy> script again (or the <Before-Undeploy> script if an undeploy operation was executed). A special parameter – #ROLLBACK – is supported by the Deployment Manager so that the script knows whether it is taking part in a roll-forward or roll-back operation.

The following list of extra parameters is supported:

- #DB_HOST – The name of the host running the system database.
- #DB_PORT – Port number used by the system database.
- #DB_INSTANCE – The name of the database instance used for the system database.
- #DB_USER – The database user name.
- #DB_URL – The database connect URL that can be used to connect to the database (database username and password are included in the URL).
- #DB_PASSWORD – The database password in clear text. You should never write the value of this parameter to any files (including log files) from your scripts.
- #FORCE – This parameter is set to “true” if the Deployment Manager runs in force mode; otherwise it is set to “false”.
- #SOLUTION_NAME – The name of the solution as defined in the deployment descriptor.
- #SOLUTION_HOME – The name of the directory in which the solution is located; this is not necessarily coinciding with the solution name.
- #ROLLBACK – This parameter can only be used in the <Before-Deploy> and <Before-Undeploy> elements. During roll-forward this parameter will have the value “false” and during roll-back is will have value “true”.

- #VERSION – The parameter is only applicable during deployment and undeployment of patches and customizations. During deployment of patches/customizations the value of this parameter is set to the version of the deployed solution (e.g. “1.3.5”). During undeployment of patches/customizations this parameter is set to the version of the installed solution *prior* to the installation of the latest patch or customization (i.e. the solution version that the Deployment Manager is rolling the solution back to).

Script location: *BaseDir*/etc/scripts

Example:

```
<Scripts>
  <Before-Deploy>
    <Script-Name>inst_pre.sh</Script-Name>
    <Argument>#DB_HOST</Argument>
    <Argument>#DB_PORT</Argument>
    <Argument>#DB_INSTANCE</Argument>
    <Argument>#DB_USER</Argument>
    <Argument>#DB_PASSWORD</Argument>
    <Argument>#ROLLBACK</Argument>
  </Before-Deploy>
  <After-Deploy>
    <Script-Name>inst_post.sh</Script-Name>
    <Argument>$ACTIVATOR_ETC</Argument>
    <Argument>$ACTIVATOR_OPT</Argument>
    <Argument>$JBOSS_DEPLOY</Argument>
  </After-Deploy>
  <Before-Undeploy>
    <Script-Name>uninst_pre.sh</Script-Name>
    <Argument>$ACTIVATOR_ETC</Argument>
    <Argument>$ACTIVATOR_OPT</Argument>
    <Argument>$JBOSS_DEPLOY</Argument>
    <Argument>#ROLLBACK</Argument>
  </Before-Undeploy>
  <After-Undeploy>
    <Script-Name>uninst_post.sh</Script-Name>
    <Argument>#DB_HOST</Argument>
    <Argument>#DB_PORT</Argument>
    <Argument>#DB_INSTANCE</Argument>
    <Argument>#DB_USER</Argument>
    <Argument>#DB_PASSWORD</Argument>
  </After-Undeploy>
</Scripts>
```

NOTE On Windows you may want to specify a mapping between file extensions and the interpreter to be used to run the script – see Section “File Extensions” on page 42 for more details.

<Other>

The <Other> element is used to specify additional files and directories that will be copied into the Service Activator runtime system or into any other location on the server during the deploy operation.

The Deployment Manager does not impose any restriction on the “src” and “dest” attributes. It is possible to copy the entire contents of directories by setting the “directory” attribute to “true” (default value is “false”).

The <Copy> element can be used to delete files by setting the value of the optional “delete” attribute “true”. The “delete” attribute is only supported for patches and customizations.

Finally, the <Copy> element has an optional attribute called “executable” with the default value “false”. Setting this attribute to “true” will cause the Deployment Manager to set the “execute” bit after the file has been copied to the desired location. The “executable” attribute will not have any effect on Microsoft Windows operating systems.

If a file in the runtime system is subject to be overwritten, the Deployment Manager will save a backup of the old file in the directory *SolutionDir/Backup* and add a record to the checksums_*N*.xml file.

The <Other> element must contain a list of <Copy> elements each specifying a file to be copied. The source file and destination file are specified using the “src” and “dest” attributes, respectively.

You can use the following environment variable to specify source and destination files and directories:

- $ACTIVATOR_OPT
- $ACTIVATOR_VAR
- $ACTIVATOR_ETC
- $JBOSS_HOME
- $JBOSS_DEPLOY
- $ACT_THIRD_PARTY_CLASSES
- $ACT_THIRD_PARTY_LIB
- $ACT_THIRD_PARTY_INV
- $JAVA_HOME
- $SOLUTION_HOME
- $PATCH_HOME

Example:

```
<Other>
  <Copy src="/home/hpsa_user/scripts/custom_service.sh"
        dest="/sbin/init.d/custom_service.sh" executable="true" />
  <Copy src="$ACTIVATOR_OPT/solutions/My_Solution/file.xml"
        dest="/etc/my_service/file.xml" />
  <Copy directory="true" overwrite="true" src="$SOLUTION_HOME/crmportal.sar"
        dest="$JBOSS_DEPLOY/crmportal.sar" />
</Other>
```

3 Configuring the Deployment Manager

This chapter walks you through the Deployment Manager's configuration options. The default settings that come with the shipped version of Deployment Manager's should meet the needs for most users; however, depending on your requirements you may want to change the Deployment Manager's default settings.

Preparing the Servers

If you want to use the Deployment Manager in an environment with multiple servers you need to configure all servers so that Deployment Manager can log in and run commands.

NOTE — If you only plan to use the Deployment Manager on a single server you can safely skip this section.

The requirements for running the Deployment Manager against multiple servers are:

- A Secure Shell (SSH) server must be running on all servers
- The Service Activator user (e.g. "root" on UNIX) must be able to log in via SSH
- The directory `$ACTIVATOR_OPT/bin` must be included in the default PATH on all servers

Read the section "Installing and Configuring Secure Shell" in the *Installation Guide* for details describing how to install SSH.

Secure Shell Configuration for Automatic Login (optional)

SSH supports "automatic login" in the sense that the SSH client does not prompt the user for a password when it connects to the remote server. The Deployment Manager can be configured to use this feature thereby greatly improving the user's experience.

To enable support for automatic login you need to generate a set of SSH keys on all servers in your Service Activator environment and create an `authorized_keys` file on all servers containing a list of public keys from all servers.

For more information, read the section "Installing and Configuring Secure Shell" in the *Installation Guide*. You can also study the documentation available at:

- http://www.openssh.org/

NOTE — You can still use the Deployment Manager without distributing SSH keys. In this case the Deployment Manager will prompt you for username and password when you perform operations against a remote server. Usernames and passwords will be cached in memory (i.e. they are *not* written to the file system); hence, you will only be prompted once per server.

Deployment Manager Configuration Parameters

The Deployment Manager can be configured by editing the `dm.xml` file located in `$ACTIVATOR_ETC/config`. This section will describe all the Deployment Manager's configuration options. You should study the `dm.xml` file while reading this section to get a more complete picture.

NOTE — The `dm.xml` file should not be confused with the `deploy.xml` file. The `deploy.xml` is the deployment descriptor specifying how a solution is deployed; see Chapter 2 on page 19.

Default Secure Shell Settings

You can specify the default secure shell username and identity file in the `<SSH-Config>` XML element in the `dm.xml` file. By setting these parameters you can avoid being prompted for SSH usernames and passwords the when you work on a remote machine.

If all your servers are configured with identical usernames and SSH identity files this is the only place where you need to configure SSH. Otherwise, if (some of) your servers require SSH usernames and/or identity files different from the default values, you need to specify an SSH username and identity file explicitly for those servers; see the next section.

Example:

```
<SSH-Config>
  <Username>actuser</Username>
  <Identity-File>C:/cygwin/home/actuser/.ssh/id_rsa</Identity-File>
</SSH-Config>
```

NOTE — The default secure shell settings can also be configured using the Deployment Manager's graphical user interface; see the section "Preferences" on page 52.

Servers

You can define servers that the Deployment Manager can manage by adding `<Node>` tags inside the `<Servers>` XML element. The `<Node>` tag accepts the following attributes:

- **`ip`** – specifies the IP address of the server. *Mandatory.*
- **`hostname`** – specifies the hostname of the server. *Mandatory.*
- **`cluster_member`** – specifies whether or not the server is member of the same cluster as the local server. Possible values are "true" and "false". The value of the attribute is used by Deployment Manager to determine whether workflows, plug-ins, inventory trees, compound tasks, and SQL should be deployed (or undeployed) by default when running remote operations against the server. *Mandatory.*

NOTE — If a server is a member of another cluster (i.e. not the same cluster as the local server) you should set the "cluster_member" attribute to "false". The Deployment Manager can not differentiate between remote servers that are members of another cluster and remote servers running in standalone mode.

- **`os`** – specifies the operating system installed on the server. Possible values are "unix" and "windows". If the attribute is not specified the Deployment Manager will attempt to auto-detect the server's operating system. The "os" attribute is used by some Deployment Manager operations to look for files in the correct locations. *Optional.*

- **ssh_user** – specifies the SSH username to be used when operating on this server. If the attribute is not specified the Deployment Manager will use the default SSH username specified in the <SSH-Config> element. Finally, if neither of the two are specified you will be prompted for username and password. *Optional.*
- **ssh_identity** – specifies the SSH identity file that is used to log into this server. If the attribute is not specified the Deployment Manager will use the default SSH identity file specified in the <SSH-Config> element. If no SSH identity file has been specified you will be prompted for a password. *Optional.*

Example:

```
<Servers>
  <Node ip="10.1.2.11" cluster_member="true" hostname="hostname1"
        ssh_user="admin" ssh_identity="/home/hpsa/.ssh/id_rsa"/>
  <Node ip="10.1.2.12" cluster_member="true" hostname="hostname2"/>
  <Node ip="10.1.2.13" cluster_member="true" hostname="hostname3"
        ssh_user="hpsa" os="windows"/>
  <Node ip="192.168.2.71" cluster_member="false" hostname="prod"
        os="unix"/>
</Servers>
```

NOTE Servers can also be added, modified, and deleted using the Deployment Manager's GUI; see the section "Preferences" on page 52.

Checksum Files

You can specify the directories for which to run the checksum operation by modifying the <Checksum> XML element. In general, you should make sure that the Deployment Manager only calculates checksums operations for static files. It does not make sense to calculate checksums for files that keep changing; for instance, log files.

NOTE In the default configuration the Deployment Manager excludes several directories from the checksum operation. For instance, the $ACTIVATOR_OPT/examples and $ACTIVATOR_OPT/solutions are excluded by default.

Directories can be included in the checksum operation by adding <Directory> elements – the directory name is specified in the "name" attribute. If you wish to exclude files or directories from the checksum calculation you can specify this using the <Exclude> element; the file or directory name to be excluded is specified in the "name" attribute.

It is possible to use environment variables, such as $ACTIVATOR_OPT and $JBOSS_HOME, in the "name" attributes for both the <Directory> element and the <Exclude> element. For a complete list of supported environment variables, see page 37.

In addition, the <Exclude> element allows the use of wildcards and recursive wildcards (**). So, if you specify the **/*.exe pattern in the "name" element all files with the ".exe" extension will be excluded from the checksum calculation.

Example:

```
<Checksum>
  <Directory name="$ACTIVATOR_ETC"/>
  <Directory name="$ACTIVATOR_OPT">
    <Exclude name="$ACTIVATOR_ETC"/>
    <Exclude name="$ACTIVATOR_VAR"/>
    <Exclude name="**/*.bat"/>
    <Exclude name="**/*.exe"/>
    <Exclude name="$ACTIVATOR_OPT/bin/designer"/>
  </Directory>
  <Directory name="$ACTIVATOR_VAR/repository"/>
  <Directory name="$JBOSS_HOME">
    <Exclude name="$JBOSS_HOME/server/default/data"/>
```

```
    <Exclude name="$JBOSS_HOME/server/default/log"/>
    <Exclude name="$JBOSS_HOME/server/default/tmp"/>
    <Exclude name="$JBOSS_HOME/server/default/work"/>
  </Directory>
```

NOTE The checksum configuration in the `dm.xml` is different from the example shown here.

Service Activator Configuration Files

The list of configuration files that can be copied from one server to another by the Deployment Manager can also be configured in the `dm.xml` file by adding `<File>` elements to the `<Config-File>` XML element.

It is possible to use environment variables, such as `$ACTIVATOR_OPT` and `$JBOSS_HOME`, in the `<File>` elements. For a complete list of supported environment variables, see page 37.

Example:

```
<Config-Files>
  <File>$ACTIVATOR_ETC/config/dm.xml</File>
  <File>$ACTIVATOR_ETC/config/mwfm.xml</File>
  <File>$ACTIVATOR_ETC/config/resmgr.xml</File>
  <File>$ACTIVATOR_ETC/config/role_mappings.xml</File>
  <File>$JBOSS_HOME/server/default/conf/login-config.xml</File>
  <File>$JBOSS_DEPLOY/hpovact.sar/activator.war/WEB-INF/web.xml</File>
  <File>$JBOSS_DEPLOY/jboss-web.deployer/server.xml</File>
</Config-Files>
```

File Extensions

External scripts executed by the Deployment Manager as part of a deploy or undeploy operation (see Section "`<Scripts>`" on page 36) must be executable files. This means that on UNIX the scripts must have the executable flag set, whereas on Windows the executable status is (to a large degree) determined by the file extension – for instance, files ending with ".bat" or ".exe" can be executed without any problems.

In order to be able to run "UNIX-like" scripts on Windows (e.g. Bash or Perl scripts) you can install Cygwin on your Windows machine (or other utilities) and specify in the Deployment Manager's configuration a mapping between file extensions and the interpreter to be used for files with that extension.

Example:

```
<File-Extensions>
  <Extension script_type="sh">C:\cygwin\bin\bash.exe -l</Extension>
  <Extension script_type="pl">D:\ActiveState\perl\bin\perl.exe -w</Extension>
</File-Extensions>
```

NOTE The `<File-Extensions>` element only has effect on Windows. On UNIX it will be completely ignored.

4 Checksums and Conflicts

Understanding the checksum files and conflict files is key to understanding how the Deployment Manager manages patches and customizations. This chapter will explain the Deployment Manager's use of the checksum files and conflict files in more detail.

Checksum Files

The Deployment Manager uses checksum files to save information about deployed files, workflows, inventory trees, plug-ins, etc. in a structured way.

NOTE — The Deployment Manager's use of checksum files to keep track of deployed components should not be confused with the "Generate Local/Remote Checksum Files" and "Compare Two Checksum Files" operations.

Each time a solution, patch, or customization is deployed the Deployment Manager will calculate MD5 checksums of all deployed components and store these in a file called `checksums_N.xml` (where *N* is a sequence number starting with 1) located in the `$SOLUTION_HOME/install` directory. Note that when installing patches and customization, the Deployment Manager will *not* store the checksums in `$PATCH_HOME/install` directory. In fact, the `$PATCH_HOME/install` directory is not used at all by the Deployment Manager.

The checksum files are generally divided into two parts:

- The first part of the checksum files contain general information about the solution, patch, or customization, such as:
  - The name of the solution (i.e. the name of "base solution"; not the name of the patch or customization)
  - A reference to the deployment file that was used for this deploy operation
  - The time the solution was deployed
  - The version of the solution (major, minor, and revision)
  - The home directory (a.k.a. "base directory") of the solution, patch, or customization
  - The name of the directory containing backups of components that were overwritten during the deploy operation (if any)
- The second part of the checksum files lists the actual components that have been deployed by the Deployment Manager, including their MD5 checksums and (optionally) a reference to a backup of the component that was overwritten. The checksum lists are divided into different areas denoted by the following XML elements:
  - **`<Files>`** – contains all files that were deployed into the runtime system (except for the files that were deployed using the `<Other>` element in the deployment descriptor)
  - **`<Plugins>`** – contains all plugins that were deployed to the system database

- **<CTs>** – contains all compound tasks that were deployed to the system database
- **<Inventory-Trees>** – contains all inventory trees that were deployed to the system database
- **<Workflows>** – contains all workflows that were deployed to the system database
- **<Other>** – contains all files that were deployed using the `<Other>` element in the deployment descriptor
- **<Solution-Files>** – contains all files that were deployed from a patch or customization base directory to the solution's base directory

The full syntax of the checksum files is defined by the DTD file `install.dtd` located in `$ACTIVATOR_ETC/config`.

An example of a checksum file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Install SYSTEM "install.dtd">
<Install>
   <Solution>VPN_Example</Solution>
   <Deploy-File>1\deploy.xml</Deploy-File>
   <Deploy-Time millis="1246825806752">2009-07-05 22:30:06</Deploy-Time>
   <Version type="solution">
      <Label>VPN_Example</Label>
      <Major>5</Major>
      <Minor>0</Minor>
      <Revision>1</Revision>
   </Version>
   <Home-Dir>C:\HP\OpenView\ServiceActivator\solutions\VPN_Example</Home-Dir>
   <Backup-Dir>backup</Backup-Dir>
   <Checksums>
      <Files>
         <File md5="e01b39b112051fc3c4cc21e65559f7a3">
C:\HP\OpenView\ServiceActivator\3rd-party\inventory\classes\com\hp\activator\ex
ample\Contact$SearchBean.class
         </File>
         ...
      </Files>
      <Plugins>
         <Plugin md5="dc47952e0d143ff1bfd434b04d733c86" namespace="GLOBAL">
            DemoRouter
         </Plugin>
      </Plugins>
      <Workflows>
         <Workflow md5="b9c3328a33f53001bfc7b5476d2edc40"
                   solution="VPN_Example">
            VE_AddSite
         </Workflow>
         <Workflow md5="28b06b2c2ef316f931249569e40f2379"
                   solution="VPN_Example">
            VE_Controller
         </Workflow>
         ...
      </Workflows>
   </Checksums>
</Install>
```

When a solution is deployed, the Deployment Manager will create the checksum file called `checksums_1.xml` in the `$SOLUTION_HOME/install` directory. For all sub-sequent patches and customizations, the Deployment Manager will create new `checksums_`*`N`*`.xml` files where *N* is increased by one each time. In addition, the Deployment Manager will create directories under `$SOLUTION_HOME/install` using the same sequence numbers as for the checksum files where a copy of the deployment descriptors (that were used for that particular deploy operation) will be saved. The Deployment Manager will actually not read these deployment descriptors again; they are intended to be a service to the user so that he can always find out

exactly how the deployment descriptor looked at the time when the solution, patch, or customization was deployed.

All this means that if a solution has been deployed, followed by a patch and two customizations (i.e. four deploy operations in total), the `$SOLUTION_HOME/install` directory will contain the following:

```
$SOLUTION_HOME/install/
                       |-checksums_1.xml
                       |-checksums_2.xml
                       |-checksums_3.xml
                       |-checksums_4.xml
                       |-1/
                          |-deploy.xml
                       |-2/
                          |-deploy-patch.xml
                       |-3/
                          |-deploy-cust1.xml
                       |-4/
                          |-deploy-cust2.xml
```

The Deployment Manager uses the checksum files for the following things:

- Verification of deployed solutions (see the Section “Verify Deployed Solution” on page 86)
  - To do this the Deployment Manager first consolidates all checksum files into a single checksum file (kept in memory) where later components (and their MD5 checksums) will always take precedence over earlier deployed components with the same names. Using this consolidated checksum file, the Deployment Manager can traverse all deployed components in the runtime system, calculate their MD5 checksums and compare to the MD5 checksums in the consolidated checksum file. In this way, the Deployment Manager can generate a list of components that have been modified since they were deployed as well as a list of missing components. If both lists are empty, the Deployment Manager can be certain that the solution components are intact.
- Ensuring that solutions are undeployed in the correct order
  - The Deployment Manager refuses to undeploy a patch or customization unless it last in the deployment history. Similarly, if is not possible to undeploy a solution if one or more patches or customizations have been deployed on top of the solution.
- Detection of conflicts
  - More about this later in this chapter.

Solution developers may also use the checksum files to check which components were deployed and when, or to recover backups of some of the overwritten components and merge them into the new components.

Conflict Files

This section will describe the use of conflict files through some scenarios.

Scenario 1

Imagine the following situation:

You have created a solution for HP Service Activator that has been deployed at a customer’s location. Some months later you create a service pack for the solution and send it to the customer. However, when the customer uses the Deployment Manager to deploy the service pack it causes the whole solution to fall apart. Luckily, the Deployment Manager is capable of undeploying the service pack again, so the (old version of the) solution can quickly get up and running again.

After some investigation, you find out that the reason that the service pack caused the whole solution to malfunction was that the customer had made some modification in some of the deployed components and these modifications turned out to be incompatible with the service pack.

If there had been a way to detect this when the service pack was deployed, you would have been saved from a lot of trouble and embarrassment.

The Deployment Manager *can* actually detect this. Whenever a file or database component is about to be overwritten during the deployment of a patch (or customization) the Deployment Manager will compare the MD5 checksums of the components that is about to be overwritten with the MD5 checksums of the components that are saved in the `checksum_X.xml` files. If the MD5 checksums are *not* identical, the Deployment Manager will inform the user about the detected conflict, and in addition all conflicts will be written to the file `conflict.xml` in the `$SOLUTION_HOME/install` directory.

By looking into the conflict file the solution developer will get a good idea about where to check for potential issues, which will normally lead to a quicker resolution of issues.

NOTE

A conflict free patch deployment does in no way *guarantee* that the solution will work as expected after the patch has been deployed; other changes may have been done to the runtime system that lay outside the solution's domain.

Scenario 2

Imagine the following situation:

Someone (i.e. not you) have created a solution for HP Service Activator that you have deployed at a customer's location. You have created a customization for the solution and deployed your customization using the Deployment Manager. Now, some months later, a service pack is released for the solution. You know that the service pack cannot be installed on the customized version of the solution, so you use the Deployment Manager to undeploy your customization and now you can (without any issues) deploy the service pack.

However, even with the new service pack, the customer still needs your customizations. So you need to modify your customization so that it can be deployed on the solution with the service pack installed.

Also in this case the Deployment Manager can assist you. You can deploy your customization in so-called "check mode" (which means that your customization will actually not be deployed) and make the Deployment Manager act as if the service pack was not installed by instructing it to check against an earlier version (see the Section "Deploy Customization" on page 73).

This will cause the Deployment Manager to generated a *partially* consolidated checksum file (in memory) of the files `checksums_1.xml`, `checksums_2.xml`, …, `checksums_M.xml` where $1 \leq M < N$ (N is the highest sequence number in use) and M is the sequence number of the checksum file corresponding to the version you chose to check against (in this example, M is 1 because you are checking against the base solution). This partially consolidated checksum file will be used to produce a conflict file that lists all conflict between the service pack and your customization; hence, it becomes easier for you to find out where you (may) need to make some modifications to your customization in order to make it compatible with the service pack.

Conflict File Syntax

The syntax of the conflict file is quite simple. It contains a "type" attribute that can be either "patch" or "customization" depending on whether a patch deployment or solution deployment operation created the conflict file. In addition, the file contains a reference to the base directory of the patch or customization as well as the name of the directory containing backups of components that were overwritten as part of the patch or solution deployment operation. The conflicts are listed in the remaining part of the conflict file, and they are divided into different sections denoted by the following XML elements:

- **<Files>** – contains all conflicting files (except for the files that were deployed using the <Other> element in the deployment descriptor)
- **<Plugins>** – contains all conflicting plug-ins
- **<CTs>** – contains all conflicting compound tasks
- **<Inventory-Trees>** – contains all conflicting inventory trees
- **<Workflows>** – contains all conflicting workflows
- **<Other>** – contains all conflicting files that were deployed using the <Other> element in the deployment descriptor

Each conflict in the list is accompanied by an attribute called “action” that can have the values “overwrite” (if the user chose to overwrite the conflicting component) or “keep” (if the user chose *not* to overwrite the conflicting component). Finally, for all conflict with the “action” attribute set to “overwrite” an additional <Backup> element will contain a reference to the location of the backup that was created of the component that caused the conflict.

The full syntax of the conflict files is defined by the DTD file `conflicts.dtd` located in `$ACTIVATOR_ETC/config`.

An example of a conflict file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Conflicts SYSTEM "conflicts.dtd">
<Conflicts type="customization">
  <Home-Dir>C:\HP\OpenView\ServiceActivator\solutions\VE_Secure</Home-Dir>
  <Backup-Dir>backup_3</Backup-Dir>
  <Files>
    <File action="keep">
      <Src>
C:\HP\OpenView\ServiceActivator\solutions\VE_Secure\UI\images\inventory-
gui\tree\interface-reserved.gif
      </Src>
      <Dest>
C:\HP\jboss\server\default\deploy\hpovact.sar\activator.war\images\inventory-
gui\tree\interface-reserved.gif
      </Dest>
    </File>
  </Files>
  <Workflows>
    <Workflow action="overwrite">
      <Src>
C:\HP\OpenView\ServiceActivator\solutions\VE_Secure\etc\workflows\VE_Controller
.xml
      </Src>
      <Dest solution="VPN_Example">VE_Controller</Dest>
      <Backup>etc\workflows\VE_Controller.xml</Backup>
    </Workflow>
  </Workflows>
</Conflicts>
```

5 Using the Deployment Manager

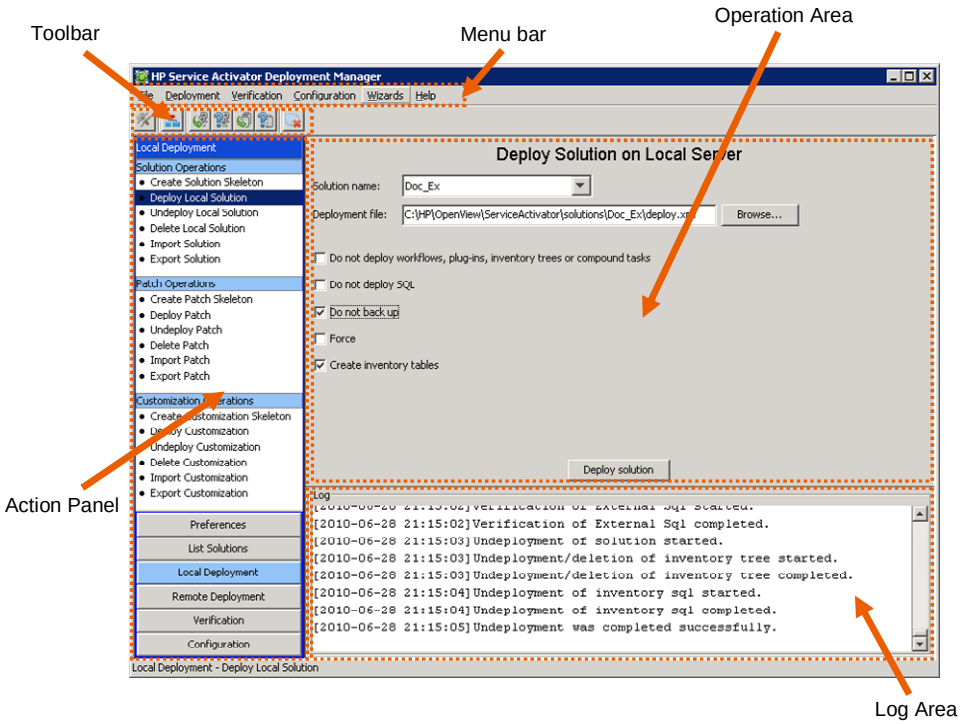
The HP Service Activator Deployment Manager is a tool that allows you to easily deploy and undeploy solutions, patches, and customization on a Service Activator installation. It can be used to manage solutions on a single server or on multiple servers running HP Service Activator (except for the patch and customization management features). This chapter will describe how to use the Deployment Manager.

TIP	The `$ACTIVATOR_OPT/example/vpn_example` directory contains a sample solution (`VPN_Ex.zip`) as well as a sample patch (`VE_5.1.1.zip`) and a sample customization (`VE_Cust.zip`). You should study these to get a better understanding of the Deployment Manager and patch/customization processes.

Navigating the Deployment Manager

The Deployment Manager's graphical user interface is shown in Figure 5-1. It consists of five main areas: The menu bar and the toolbar (at the top), the action panel (at the left), the operation area (at the center), and the log area (at the bottom).

Figure 5-1 The Deployment Manager



The Menu Bar

The Deployment Manager's menu bar consists of the following menus:

1. `File`

   The `File` menu contains the menu items `Preferences...`, `List Solutions...`, and `Exit`. If you have changed the server configuration or the default SSH settings under "Preferences" the information will be saved to the `dm.xml` file when exiting the application.

2. `Deployment`

   From the `Deployment` menu you can access all operations related to solution, patch, and customization deployment. The operations are subdivided into "local" and "remote" operations and "patch" and "customization" operations.

3. `Verification`

   The `Verification` menu provides access to the three operations related to checksum calculation and comparison. In addition there is a menu item that gives access to the "verify deployed solution" operation.

4. `Configuration`

   The `Configuration` menu contains a single menu item called `Copy configuration files...`

5. `Wizards`

   Four wizards are available from the `Wizards` menu. Two of the wizards can be used to copy a solution to other servers whereas the other two wizards can be used to compare servers.

6. `Help`

   From this menu you can access the `About Deployment Manager` menu item

The Toolbar

The toolbar contains the 6 buttons listed below:

If you click this button, the three configuration operations ("Configure Database Connection", "Configure Servers", and "Configure Secure Shell") will become available in the action panel.

Clicking this button will provide access to the "List Solutions" operations.

Launches the *Copy Solution to Nodes in Cluster* wizard that guides you through a list of actions for copying a solution to one or more cluster nodes.

Launches the *Compare Nodes in Cluster* wizard that assists you in comparing nodes in a cluster.

Launches the *Copy Solution to Remote Server* wizard that can (for instance) be used to copy a solution from a test server to a production server.

Launches the *Compare to Remote Server* wizard that can (for instance) be used to compare a test server to a production server.

Clears the log area.

The Action Panel

The action panel consists of buttons at the bottom and – depending on the selected button – a number of operations at the top. Table 5-1 lists the buttons their operations.

Table 5-1 Operations available from the Action Panel

Button	Operation
Preferences	*System Database Connection*
	Additional Database Connections
	Servers
	Secure Shell
List Solutions	*List Cluster Members*
	List Other Servers
Local Deployment	*Create Solution Skeleton*
	Deploy Solution
	Undeploy Solution
	Delete Local Solution
	Import Solution
	Export Solution
	Create Patch Skeleton
	Deploy Patch
	Undeploy Patch
	Delete Patch
	Import Patch
	Export Patch
	Create Customization Skeleton
	Deploy Customization
	Undeploy Customization
	Delete Customization
	Import Customization
	Export Customization
Remote Deployment	*Copy Solution to Remote Server*
	Deploy Remote Solution
	Undeploy Remote Solution
	Delete Remote Solution
Verification	*Generate Local Checksum File*
	Generate Remote Checksum File
	Compare Two Checksum Files

	Verify Deployed Solution
Configuration	*Copy Configuration Files*

All operations will be described in more detail later in this chapter.

The Operation Area

The operation area is where the Deployment Manager's main operations are controlled. The content of the operation area depends on the selected action (see the description of the "Action Panel" above).

The Log Area

The log area displays information about the operations carried out in the Deployment Manager. Everything displayed in the log area will also be available in the following two locations:

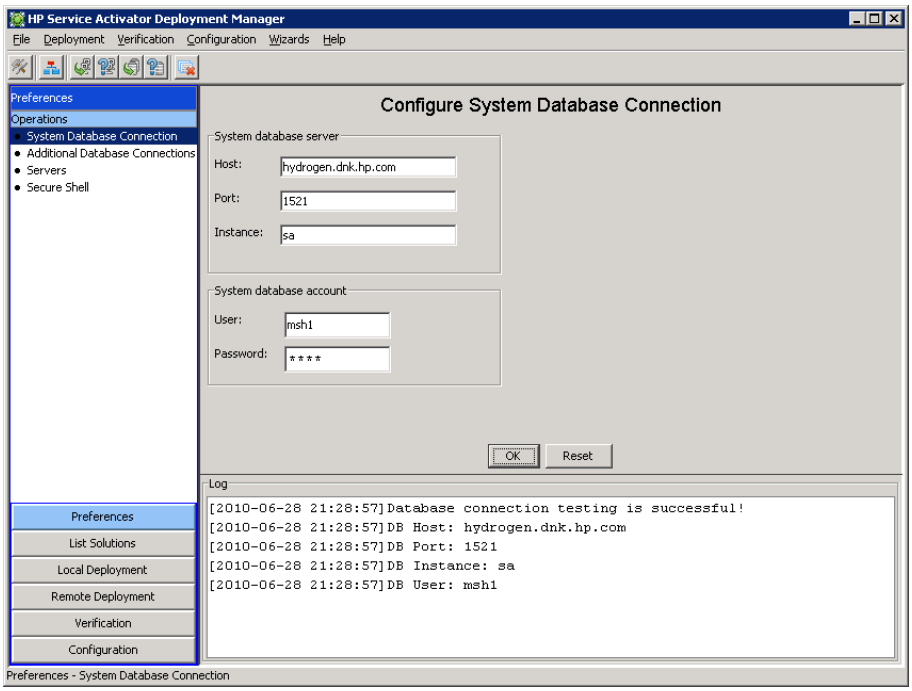
- `$ACTIVATOR_VAR/log/Dmlog.txt` – this log file contains log statements that are not related to a particular solution.
- `$ACTIVATOR_OPT/solutions/`*`SolutionName`*`/log/log.txt` – this log file contains log statements that are specific to a solution (e.g. log information generated during deployment or undeployment of a solution).

Preferences

System Database Connection

The user interface for configuring the database connection parameters is shown in Figure 5-2. The database hostname, port number, and instance name will be pre-populated based on the contents of the default HP Service Activator datasource configuration file (`$JBOSS_DEPLOY/mwfm-default-ds.xml`).

Figure 5-2 **The System Database Connection UI**



Most, but not all, Deployment Manager operations require access to the Service Activator system database. Hence, in most cases you should configure the database username and password as the first step after launching the Deployment Manager.

NOTE

For security reasons, a default installation of the Service Activator does not store database passwords in the file system. This means that the username and password must be specified every time the Deployment Manager is invoked. However, the database username and password will be cached in memory until the application is exited.

In development environments, having to enter the database username and password can become inconvenient. Like the other Service Activator tools (Workflow Designer, Service Builder, Inventory Builder, Inventory Tree Deployer, and Inventory Tree Designer), the Deployment Manager can read the database username and password from the file `$ACTIVATOR_ETC/config/dbAccess.cfg` if the file exists. See the file `$ACTIVATOR_ETC/config/dbAccess_example.cfg` for information about the exact syntax.

NOTE

For security reasons the file `$ACTIVATOR_ETC/config/dbAccess.cfg` must *never* be present in production systems.

Additional Database Connections

The user interface for configuring additional database connections is shown in Figure 5-3 and Figure 5-4.

Figure 5-3 The Additional Database Connections UI

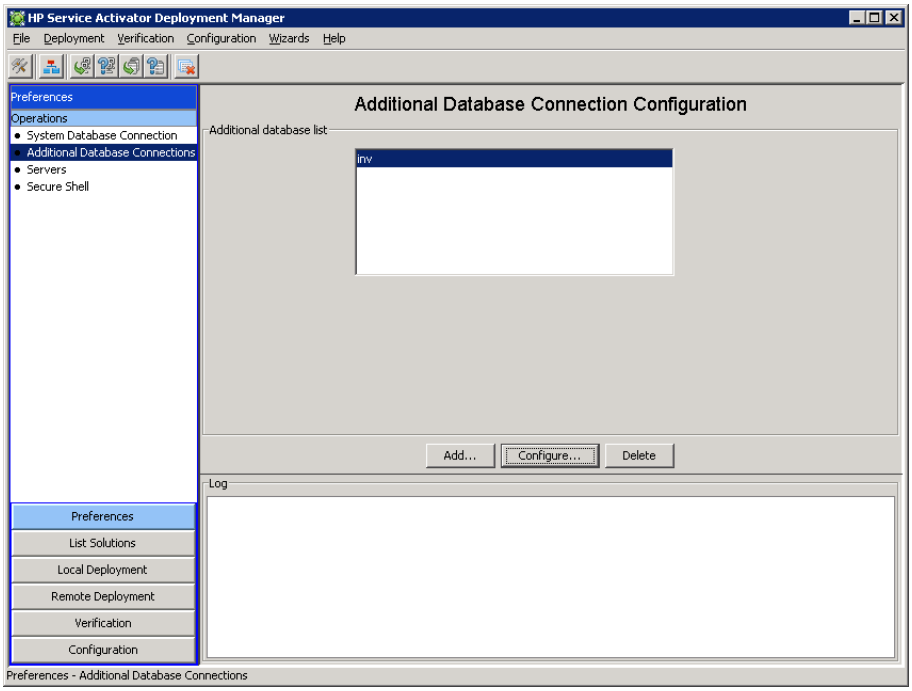
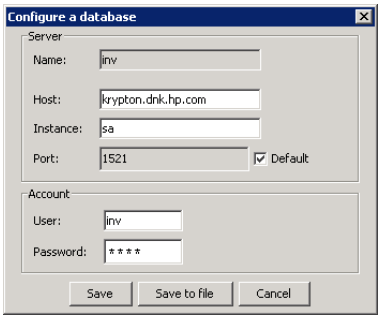


Figure 5-4 Configure Parameters for Additional Database Connection



In order to create a new addition database connection, the user can click the `[Add...]` button and enter the following information:

- **Name** – this is used in the deployment descriptor to identify the database (see the description of the “db” attribute in Chapter 2).
- **Host** – the hostname of the database server
- **Instance** – the database instance name
- **Port** – the TCP port that is used to connect to the database (default: 1521)
- **User** – the database user name
- **Password** – the database password

The user can save the additional database parameters in memory (by clicking the `[Save]` button) or to a file (by clicking the `[Save to file]` button). The file name used for saving the database parameters is `$ACTIVATOR_ETC/config/dbAccess_`*`Name`*`_.cfg`, where *`Name`* is the name that the user has specified.

NOTE	For security reasons, the password is not saved to the file. The user may, however, open the file with a text editor and add the password in clear text. This is convenient during development, but it should *never* be done in production systems.

Servers

Figure 5-5 and Figure 5-6 show the user interface for configuring servers. Server can be added by clicking the `[Add...]` button, deleted by clicking the `[Delete]` button or the settings for a server can be configured by first selecting the server and then clicking the `[Configure...]` button.

As shown in Figure 5-6 it is possible to define an SSH username and SSH identity for each server. If neither of these are specified the Deployment Manager will use the default SSH settings (see the follow section).

Figure 5-5 Servers

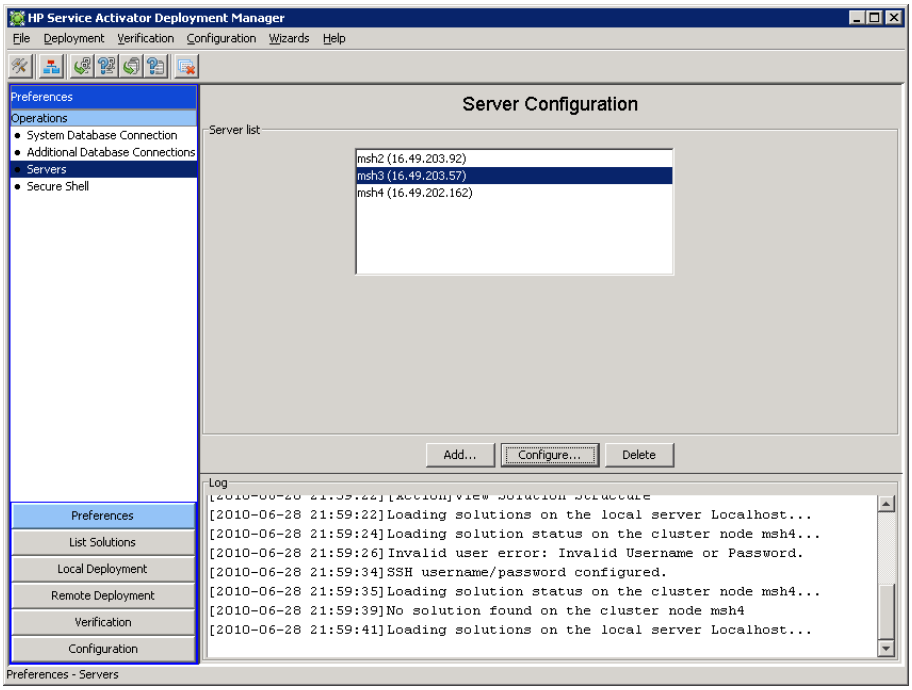
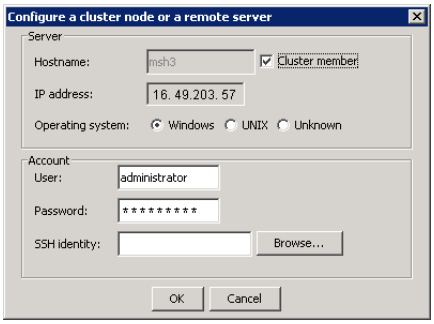


Figure 5-6 Configure parameters for a single server

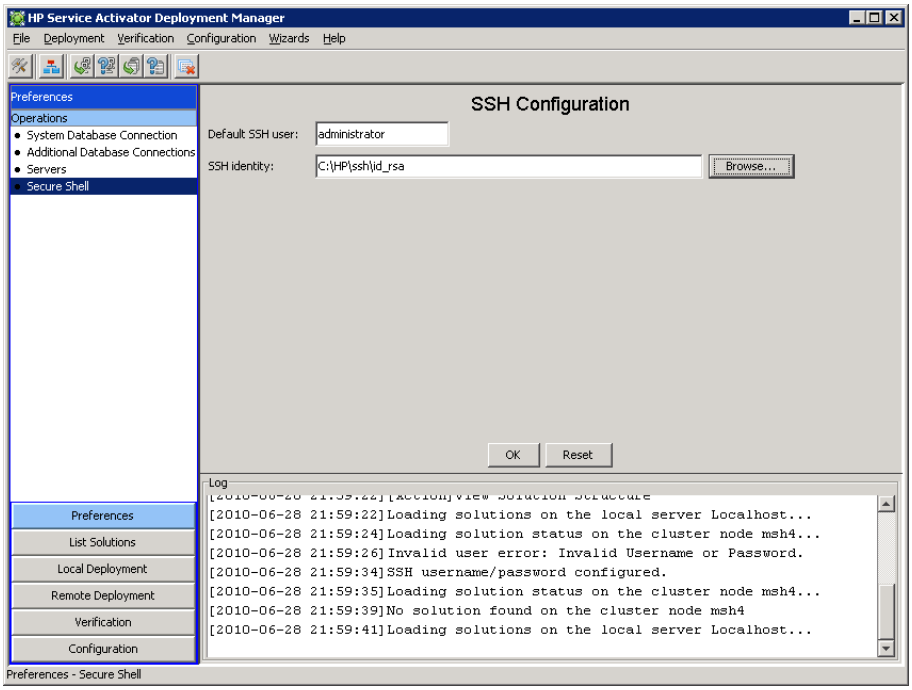


Server configuration changes will take immediate effect in the Deployment Manager. However, the configuration will not be stored in the `dm.xml` until you close the application.

Secure Shell

Figure 5-7 shows the user interface for configuring the default secure shell settings. A secure shell user as well as a secure shell identity must be configured if you wish to operate on remote servers without being prompted for a username and password.

Figure 5-7 Default SSH configuration UI



Secure shell configuration changes will take immediate effect in the Deployment Manager. However, they will not be stored in the `dm.xml` until you close the application.

List Solutions

It is possible from within the Deployment Manager to list solutions that are either “deployed” or “not deployed” on the local server as well as on all servers accessible from the local server. Solutions that are deployed are marked with a green arrow whereas solutions that are not deployed are marked with an orange arrow.

The “List Solutions” operation is divided into two sub-operations (see Figure 5-8 and Figure 5-9):

- **List Cluster Members** – lists all servers that are marked as being “cluster members” and their solutions.
- **List Other Servers** – lists all servers that are not member of the same cluster as the local server.

NOTE The local server will always be displayed as the first server in both list views.

Figure 5-8 **List cluster members user interface**

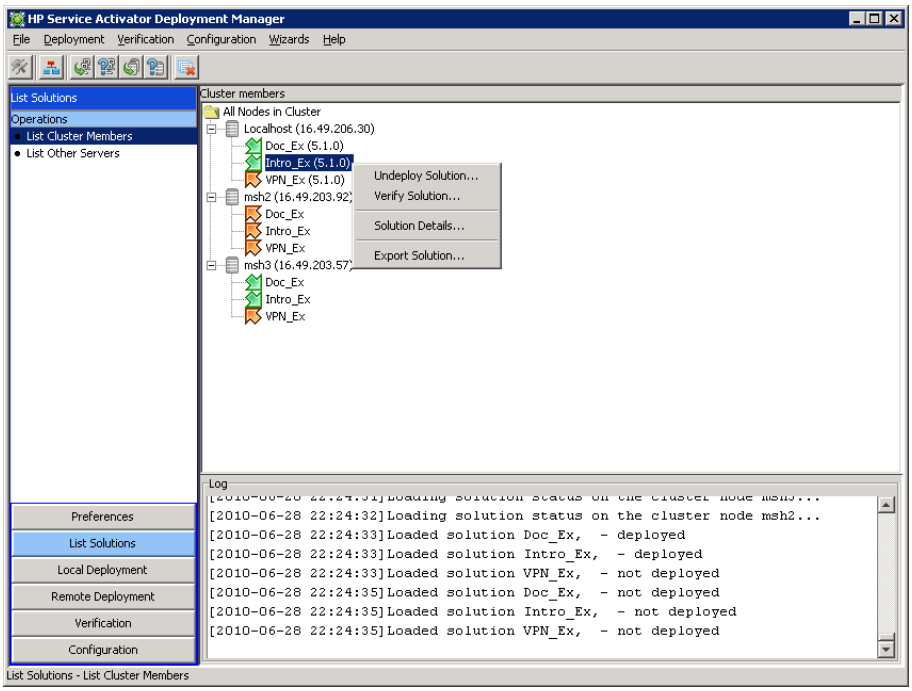
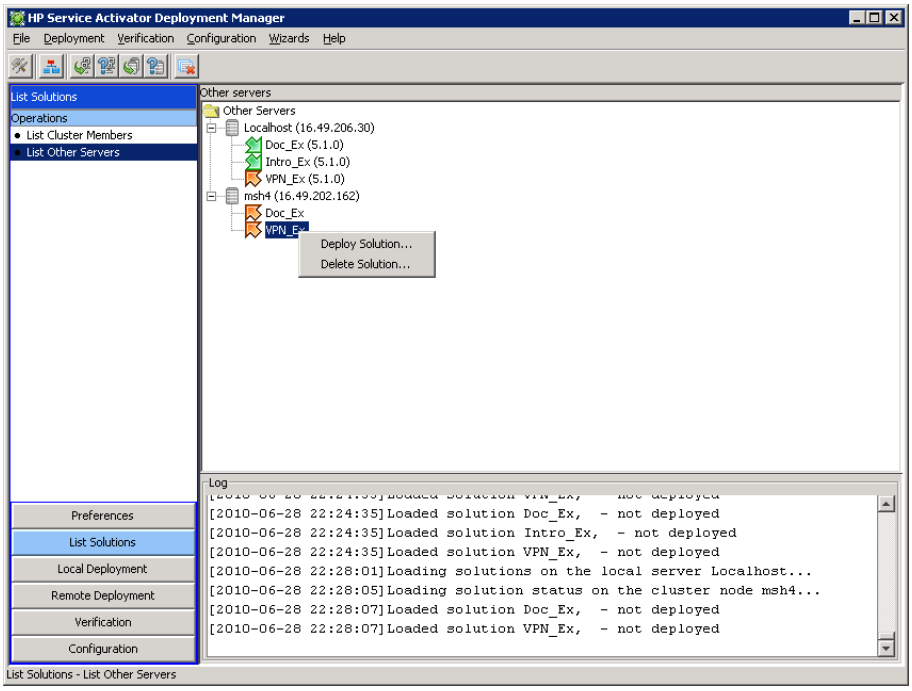


Figure 5-8 shows three solutions that are exist on the local server: “Doc_Ex”, “Intro_Ex”, and “VPN_Ex”. Only the “Doc_Ex” and the “Intro_Ex” have been deployed.

The Deployment Manager can currently not show remote solution versions. Hence, solutions listed on remote servers will always be displayed without a version number.

Figure 5-9 **List other servers user interface**



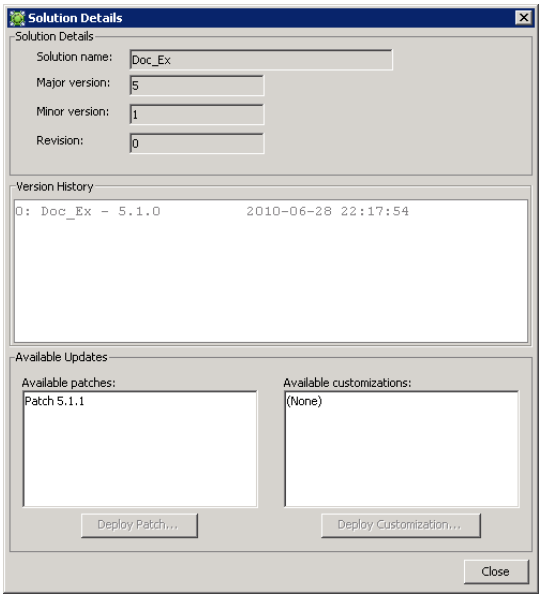
The first time you open a branch representing a remote server in any of the two “List Solutions” view, the Deployment Manager will attempt to log into the remote server (via SSH) and query the list of solutions on that server. This list of solutions will be cached in the Deployment Manager. You can force Deployment Manager to refresh the solution list by right-clicking on the server and selecting `Reload Data` from the context menu.

Other context menus are available within the “List Solutions” view. Right-clicking on a solution that is deployed will bring up a popup menu with the following menu items: `Undeploy Solution...` and `Solution Details...`. Selecting the `Undeploy Solution...` menu item will cause the Deployment Manager to go directly to the “Undeploy Local (or Remote) Solution” operation, depending on the selected server. Selecting the `Solution Details...` menu item will bring up a windows displaying additional information about the solution; see Figure 5-10)

Right-clicking on a solution that is not deployed will bring up a popup menu with the following menu items: `Deploy Solution...` and `Delete Solution...`. Selecting one of these menu items will bring you directly to one of the “Deploy Solution” or “Delete Solution” operations.

Finally, the context menu for local solutions contains `Verify Solution...` and `Export Solution...` menu items that allow you to go directly to the “Verify Solution” or “Export Solution” operations.

Figure 5-10 Solution Details



The “Solution Details” window will display the entire version history; i.e. one entry per deployed solution, patch, and customization. In addition, the patches and customizations that can be deployed onto the currently deployed solution will be listed in the “Available Updates” area. If there are any patches or customizations available, it is possible to select a patch or customization to go directly to the “Deploy Patch” or “Deploy Customization” operation by clicking the `[Deploy Patch...]` or `[Deploy Customization...]` button, respectively.

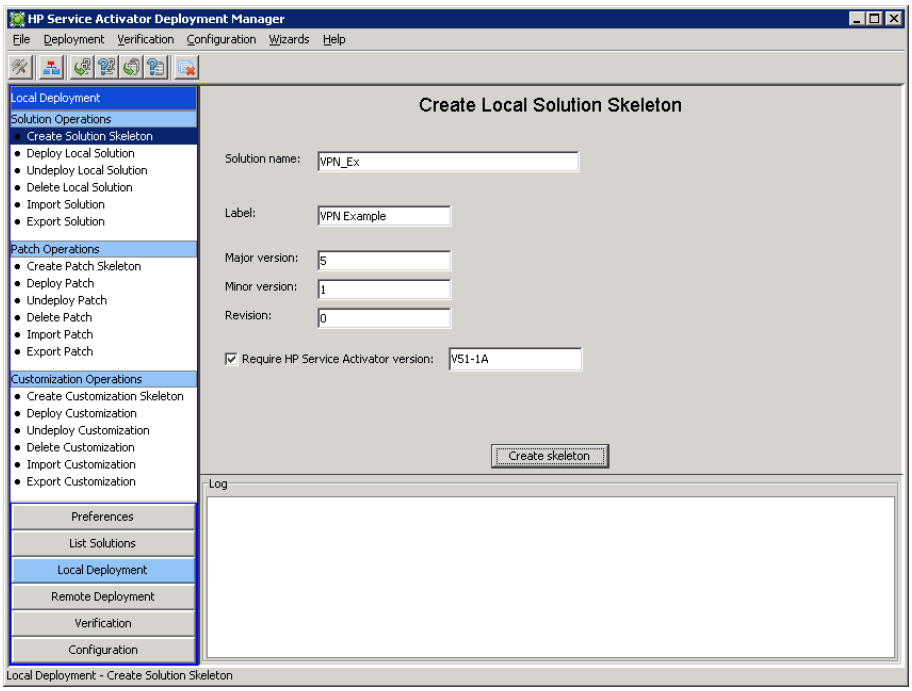
Local Solution Deployment Operations

This section describes all operations that are related to creating, importing, exporting, deploying, undeploying, and deleting solutions on the local server.

Create Solution Skeleton

The “Create Solution Skeleton” operation is used to create a solution skeleton (i.e. the directory structure described in the section “Solution Directory Structure” on page 20) as well as a `version.xml` file. This is shown in Figure 5-11. You may also choose to specify the required version of HP Service Activator for deploying this solution. To execute the operation you must click the `[Create skeleton]` button.

Figure 5-11 Creating a solution skeleton

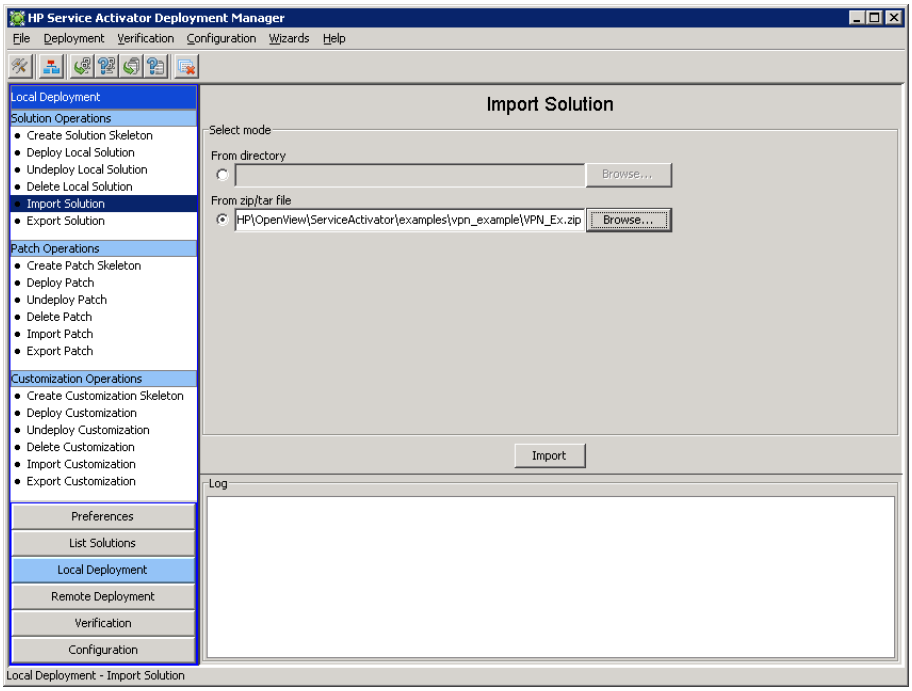


Import Solution

The “Import Solution” operation is used to import an existing solution; either from another directory or directly from a ZIP or TAR file.

The screenshot shown in Figure 5-12 shows the available options. You can select one of the two options by clicking on the radio button. To execute the operation you must click the `[Import]` button.

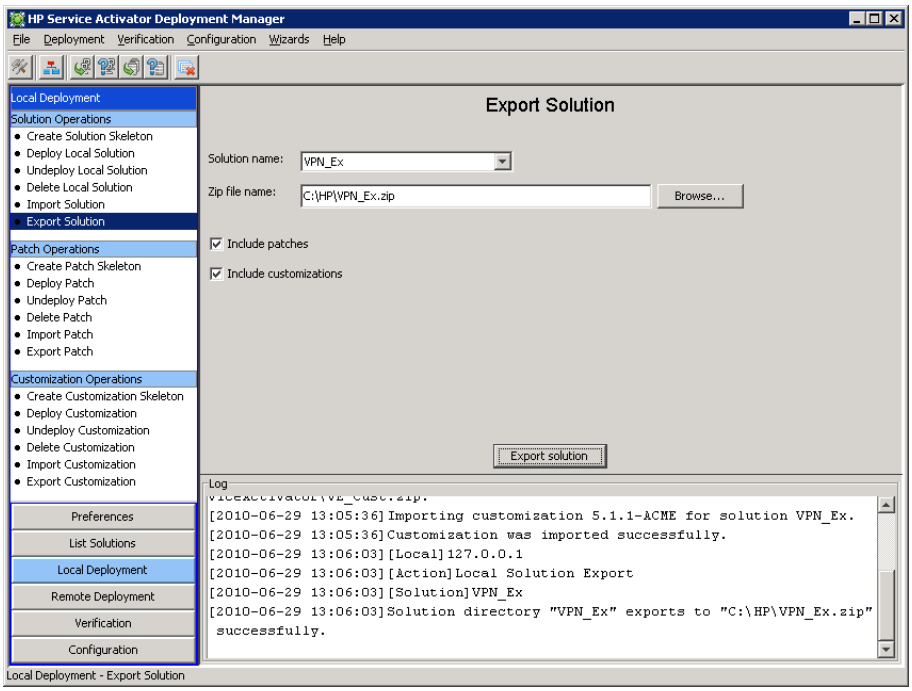
Figure 5-12 Importing a solution



Export Solution

The “Export Solution” operation is used to export a solution to a ZIP file. The screenshot shown in Figure 5-13 shows the available options for the “Export Solution” operation. You can select a solution from the dropdown list and then specify the name of the ZIP file into which the solution will be exported. Finally, if there are available patches and customizations for the solution you may include these by checking the “Include patches” and “Include customizations” checkboxes. To execute the export operation you must click the `[Export Solution]` button.

Figure 5-13 **Exporting a solution**



Deploy Solution

Two screenshots showing the user interface for deploying solutions on the local server are shown in Figure 5-14 and Figure 5-15.

You must first select a solution from the dropdown list of available solutions. Afterwards, you must select a deployment descriptor. If the file `deploy.xml` exists in the solution directory, the Deployment Manager will automatically suggest using that. You may override this by clicking the `[Browse...]` button and selecting another deployment descriptor or by manually entering the name of a deployment descriptor file.

After having selected the solution name as well as the deployment file you have the following five options:

- **Do not deploy workflows, plug-ins, inventory trees or compound tasks**. If this option is checked the Deployment Manager will not deploy workflows, plug-ins, inventory trees or compound tasks into the system database. This option is useful if you want to add the local server into a cluster that has already deployed this solution on the other nodes in the cluster.
- **Do not deploy SQL**. If this option is checked the Deployment Manager will not execute the deploy SQL scripts (located in the `$SOLUTION_HOME/etc/sql` directory) against the system database.
- **Do not back up**. By selecting this checkbox you can drive the Deployment Manager to skip making backups of files and components that are overwritten during the deploy operation. This option should be used with great care since there is a risk of loosing files and other Service Activator components (unless you have used other means to back up your files).
- **Force**. Normally, if an error occurs during a deploy operation, the Deployment Manager will attempt to roll back, thereby leaving the system intact. However, if you check this option the Deployment Manager will continue in a best-effort manner even if errors occur during the deployment operation. This option can be useful during development if, for instance, you know that the files that are overwritten belong to a previous version of the solution.

- **Create inventory tables**. If you set this option the Deployment Manager will create the inventory tables in the database (provided that inventory resource definition files exist in the inventory directory and they have been specified in the deployment descriptor).

During the deploy operation you will see a progress bar (see Figure 5-15). In the event of deployment errors an error dialog will pop up.

Figure 5-14 **The Deploy Local Solution user interface**

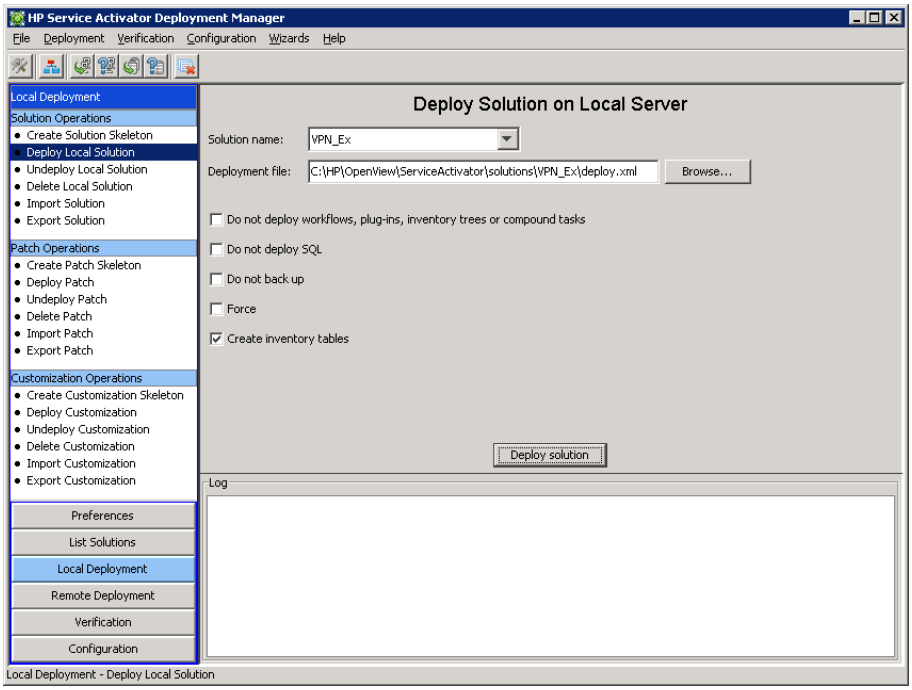
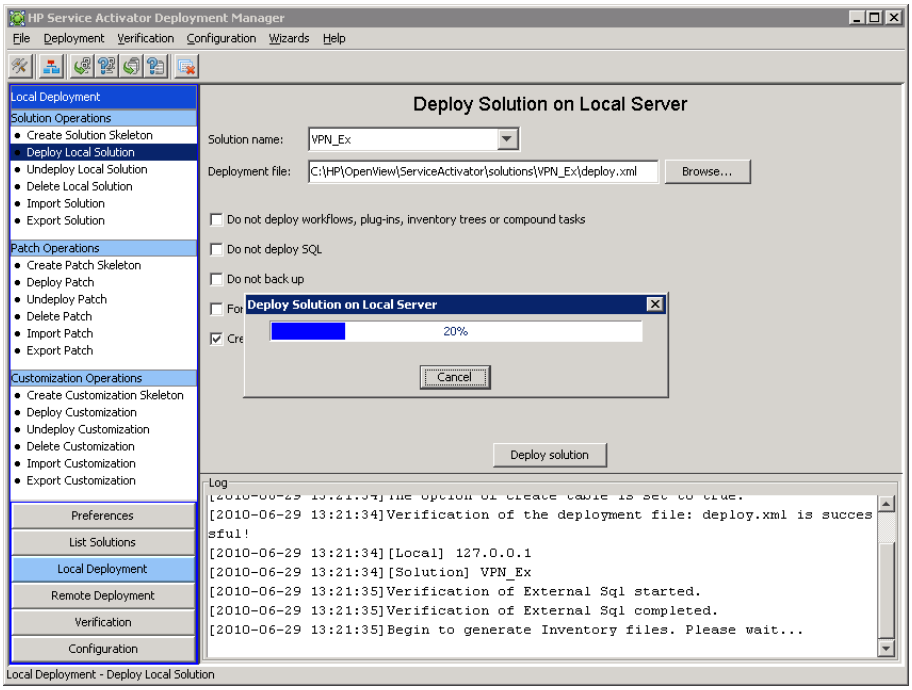


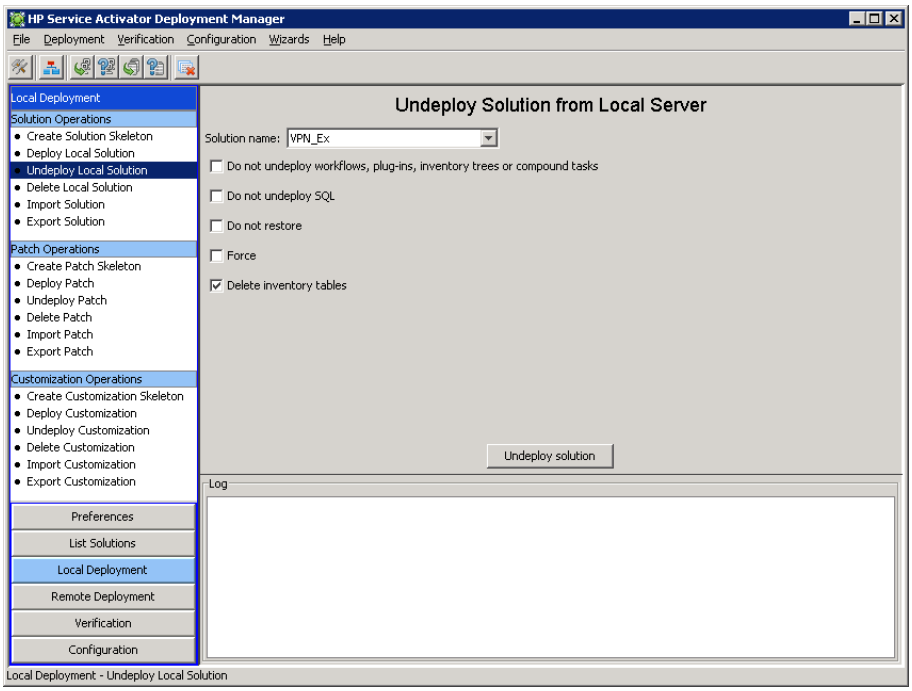
Figure 5-15 **The solution is being deployed on local server**



Undeploy Local Solution

Figure 5-16 shows the user interface for undeploying a solution from the local server. You must first select the solution to be undeployed. The Deployment Manager will know, by reading the checksum file(s), how to undeploy a solution; hence, it does not use any deployment file during undeployment of a solution.

Figure 5-16 The Undeploy Local Solution user interface



When the solution name has been selected you have the following five options:

- **Do not undeploy workflows, plug-ins, inventory trees or compound tasks.** If this option is checked the Deployment Manager will not undeploy workflows, plug-ins, inventory trees or compound tasks from the system database. This option is useful if you want to add the local server into a cluster that has already deployed this solution on the other nodes in the cluster.
- **Do not undeploy SQL**. If this option is checked the Deployment Manager will not execute the undeploy SQL scripts (located in the `$SOLUTION_HOME/etc/sql` directory) against the system database.
- **Do not restore**. If you select this checkbox the Deployment Manager will not attempt to restore the files and components from the backup directory during the undeploy operation. Normally, you should not select this checkbox.
- **Force**. By checking this option you can force the undeploy process to continue in a best-effort manner even if errors occur. This option is particularly useful during the solution development phase.
- **Delete inventory tables**. If you select this option the Deployment Manager will delete the inventory tables as part of the undeploy process.

IMPORTANT The "Delete inventory tables" option must be used with *utmost* care. The Deployment Manager provides no means to restore lost inventory data!

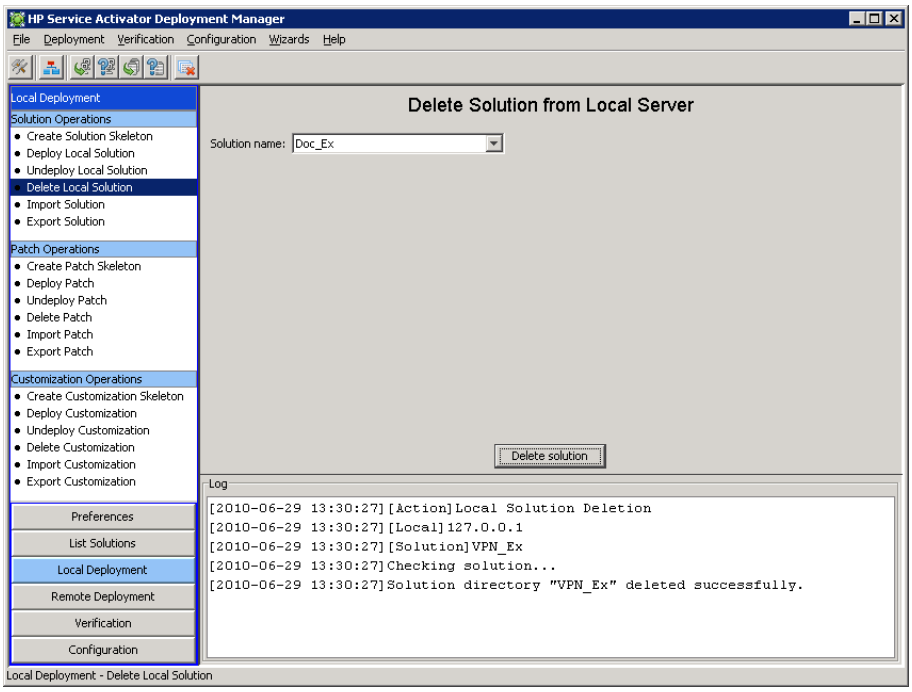
Delete Local Solution

There are two ways to delete a solution on the local server. You can either delete the entire solution directory using standard operating system tools (you should ensure that the solution has been undeployed first) or you can use the Deployment Manager to delete the solution.

NOTE — It is *strongly* recommended that you use the Deployment Manager to delete solutions. This ensures that you do not accidentally delete the solution directory of a deployed solution (incl. the backup directory).

Figure 5-17 shows the user interface for deleting a solution on the local server. You simply need to select the solution to be deleted from a dropdown list (the list will *not* include solutions that are currently deployed) and then click `[Delete solution]`. On Windows systems you must also ensure that no programs are accessing any files or directories in the solution directory; otherwise the delete operation will be incomplete.

Figure 5-17 Deleting a solution directory on the local server



Patch Operations

This section describes all operations that are related to creating, importing, exporting, deploying, undeploying, and deleting patches on the local server. Patch operations cannot be performed on remote servers.

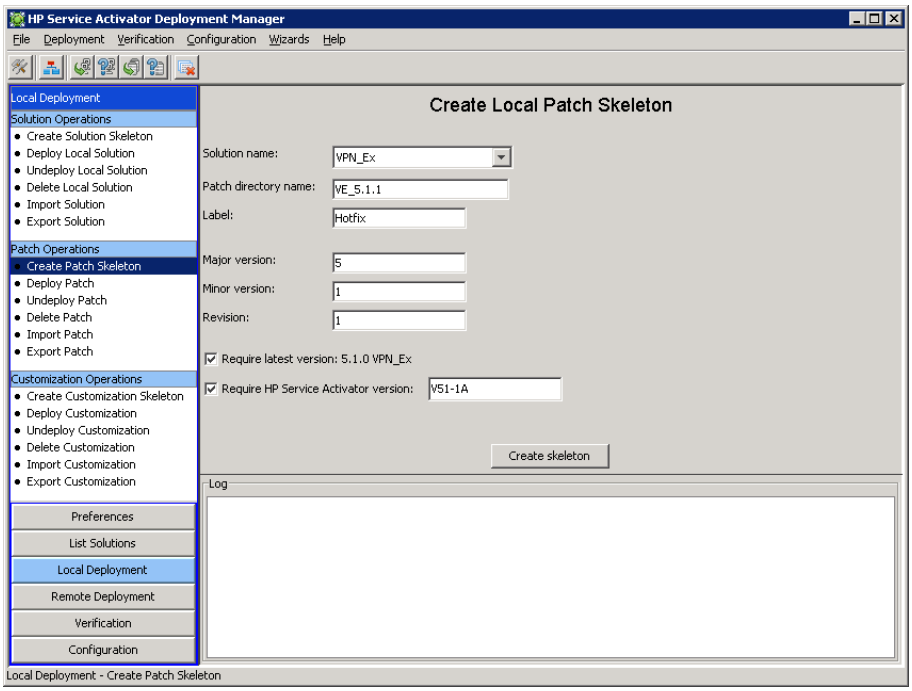
Create Patch Skeleton

The user interface for creating a patch skeleton is shown in Figure 5-18. You must first select the solution for which you want to create a patch. After this you can enter a patch directory name. The directory name will typically contain the major version, minor version, and revision of the patch, but it can be anything. The label of a patch can be used to indicate what kind of patch it is (e.g. “hotfix” or “service pack”). You need to enter the patch’s major version, minor version, and revision; these are typically numbers, but they may also contain letters.

If you want the Deployment Manager to add a `<Requires>` element to the `version.xml` file (see the Section “Creating a Version File” on page 26) you can check the “Require latest version” checkbox. Finally, if you want to add a `<Requires-Core>` element to the `version.xml` file you may check the “Require HP Service Activator version” checkbox and enter the required Service Activator version in the text field.

NOTE	If you create a patch skeleton for a solution that is not deployed, the “Require latest version” checkbox will be disabled.

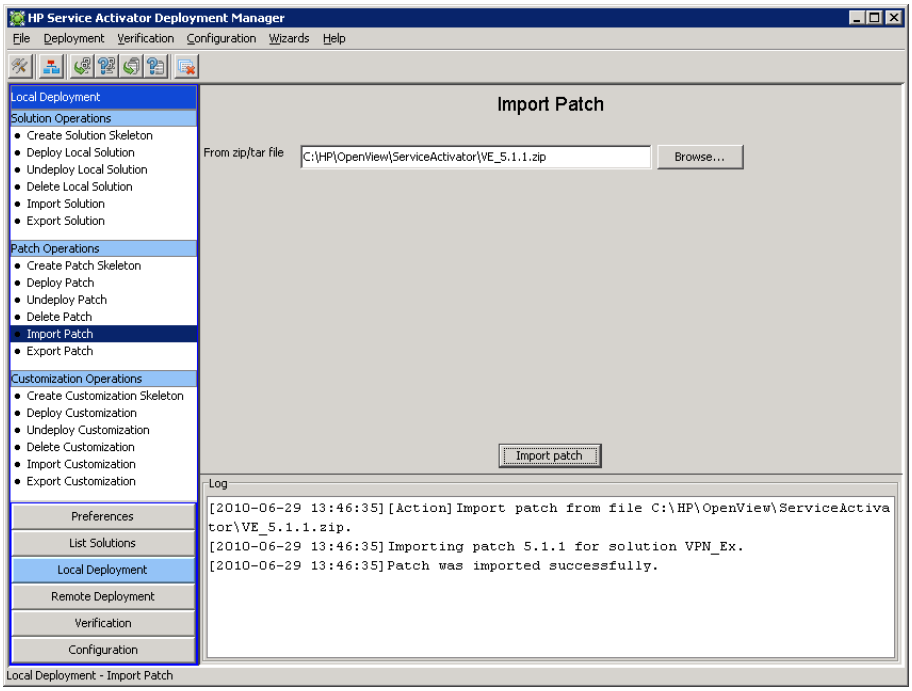
Figure 5-18 Creating a patch skeleton



Import Patch

The user interface for importing a patch is shown in Figure 5-19. You must first select the ZIP or TAR file that contains the patch and then click the `[Import patch]` button.

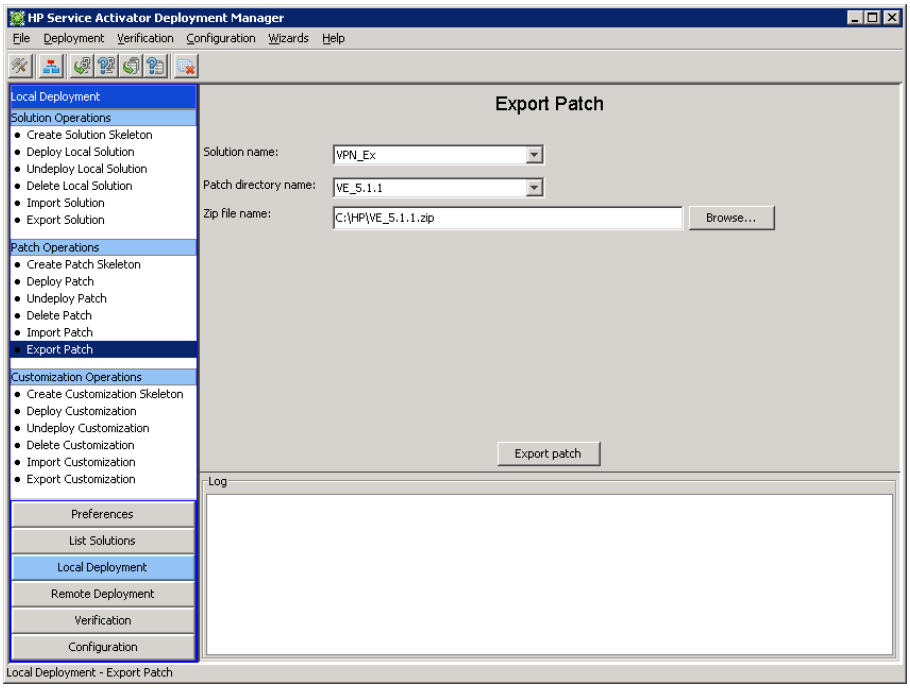
Figure 5-19 Importing a patch



Export Patch

The user interface for exporting a patch to a ZIP file is shown in Figure 5-20. First, you need to select a solution name from the dropdown list and then you can select a patch directory. Finally, enter the name of the ZIP file into which the patch will be exported and click the `[Export patch]` button.

Figure 5-20 Exporting a patch



Deploy Patch

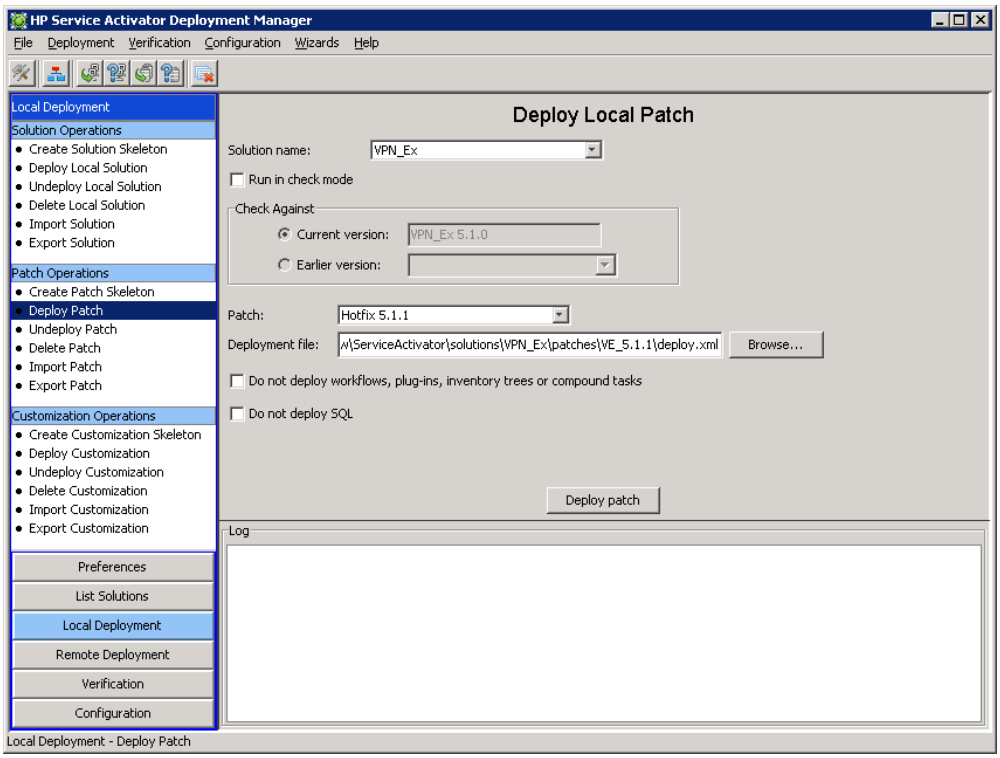
IMPORTANT	It is *highly* recommended that you run a "Verify Deployed Solution" operation before deploying a patch.

Figure 5-21 shows the Deployment Manager's user interface for deploying a patch. You must first select a solution from the dropdown list of available solutions. After that, you must select a patch from the dropdown list as well as a deployment descriptor. If the file `deploy.xml` exists in the patch's base directory, the Deployment Manager will automatically suggest using that. You may override this by clicking the `[Browse...]` button and selecting another deployment descriptor or by manually entering the name of a deployment descriptor file.

After having selected the solution name, the patch, as well as the deployment file you have the following two options:

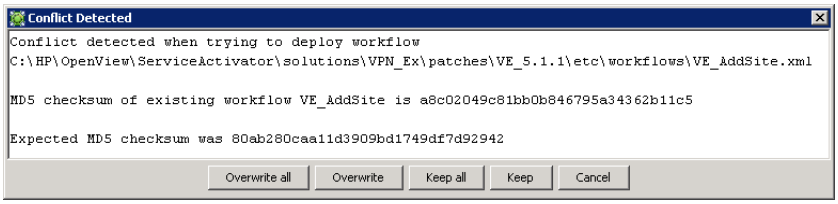
- **Do not deploy workflows, plug-ins, inventory trees or compound tasks.** If this option is checked the Deployment Manager will not deploy workflows, plug-ins, inventory trees or compound tasks into the system database.
- **Do not deploy SQL.** If this option is checked the Deployment Manager will not execute the deploy SQL scripts (located in the `$PATCH_HOME/etc/sql` directory) against the system database.

Figure 5-21 Deploying a patch



If, during the deployment of the patch, the Deployment Manager detects a conflict (see the Section "Conflict Files" on page 45) a popup window will be shown; see Figure 5-22.

Figure 5-22 **Conflict detection during patch deployment**

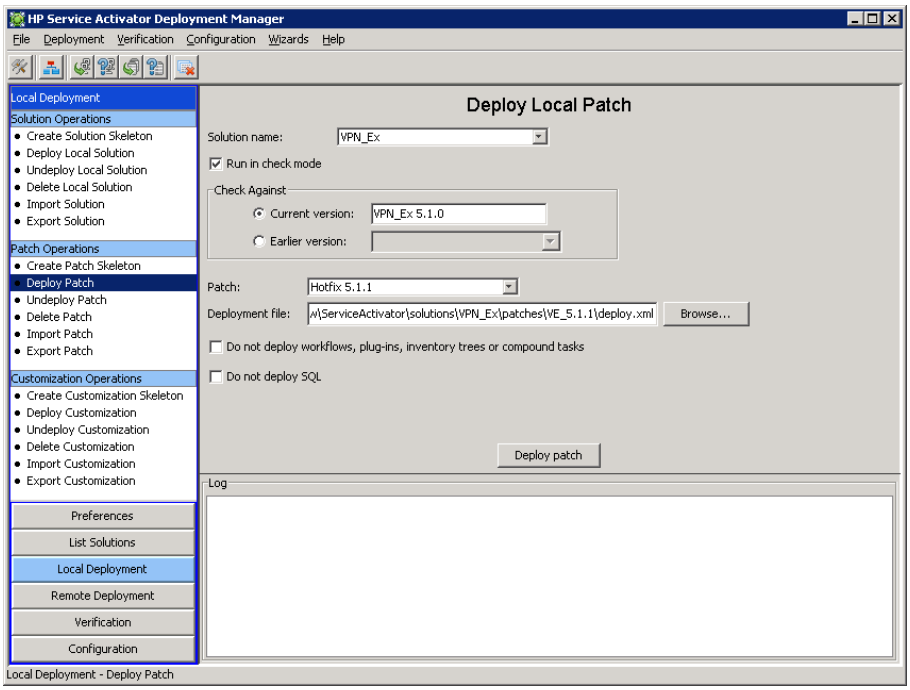


When the conflict window is shown, you can click on one of the following five buttons:

- **`[Overwrite all]`** Click this button, if you wish to overwrite this conflicting component – as well as all subsequent conflicting components (if any) – in the runtime system. A backup of all overwritten components will be created.
- **`[Overwrite]`** Click this button, if you wish to overwrite this conflicting component (and only this) in the runtime system. A backup of the overwritten component will be created.
- **`[Keep all]`** Click this button, if you do *not* wish to overwrite any of the conflicting components in the runtime system. Since nothing is overwritten, the Deployment Manager will not create backups of these components.
- **`[Keep]`** Click this button, if you do *not* wish to overwrite this (and only this) conflicting component in the runtime system. Since nothing is overwritten, the Deployment Manager will not create a backup of this component.
- **`[Cancel]`** If this button is clicked, the Deployment Manager will will abort the patch deployment operation and attempt to roll back so that the runtime system is left intact.

If you wish to check for conflicts without actually deploying the patch, you can run the patch deployment operation in so-called “check mode”. This is done by checking the “Run in check mode” checkbox located below the solution name; see Figure 5-23.

Figure 5-23 **Deploying a patch in “check mode”**



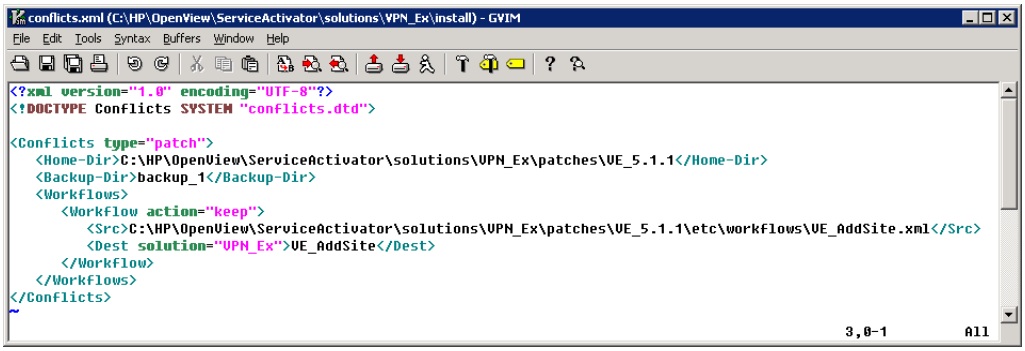
If the “Run in check mode” checkbox is selected, the following options become available on the user interface:

- **Check against current version.** If this option is selected, the Deployment Manager checks the patch against the currently deployed version of the solution. I.e. the Deployment Manager will generate a consolidated checksum file in memory (see the Section “Checksum Files” on page 43) and use that to compare to the components that are deployed in the runtime system.
- **Check against an earlier version.** If this option is selected, the Deployment Manager will check the patch against an earlier version of the solution. To do this, the Deployment Manager generates a partially consolidated checksum file (See the Section “Conflict Files” on page 45) and uses this to compare to the components that are deployed in the runtime system.

In both cases all detected conflicts will be written to the file `conflicts.xml` in the `$SOLUTION_HOME/install` directory. The user will *not* be prompted every time a conflict is detected but instead the Deployment Manager will assume that the option “keep” is chosen for every conflict (see also Figure 5-22).

An example of a conflict file is shown in Figure 5-24. In this example there was only one conflict; namely the workflow “VE_AddSite”.

Figure 5-24 **Conflict file example**

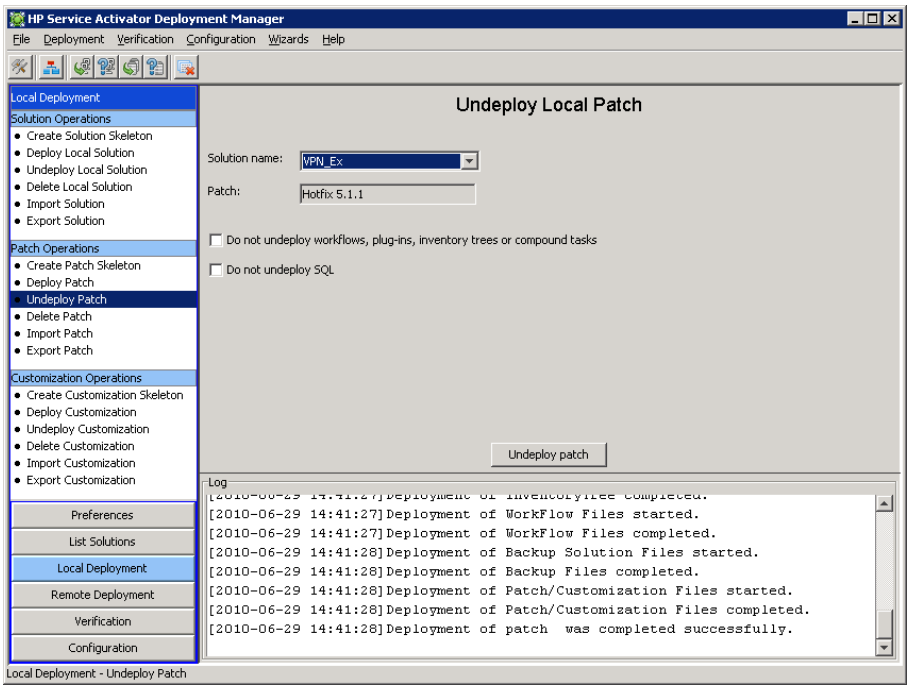


Undeploy Patch

Figure 5-25 shows the user interface for undeploying a patch. You must first select the solution for which you intend to undeploy a patch. This will cause the Deployment Manager to populate the “patch” text area with the latest installed patch. The Deployment Manager knows, by reading the latest checksum file, how to undeploy the patch; hence, it does not use the deployment file when the patch is undeployed.

NOTE — If there are no patches installed for the selected solution, or if the latest deployment for this solution was a customization deployment, the “patch” text area will remain empty and it will not be possible to run a patch undeploy operation.

Figure 5-25 Undeploying a patch



When the solution name has been selected you have the following two options:

- **Do not undeploy workflows, plug-ins, inventory trees or compound tasks.** If this option is checked the Deployment Manager will not undeploy workflows, plug-ins, inventory trees or compound tasks from the system database.
- **Do not undeploy SQL**. If this option is checked the Deployment Manager will not execute the undeploy SQL scripts (located in the `$SOLUTION_HOME/etc/sql` directory) against the system database.

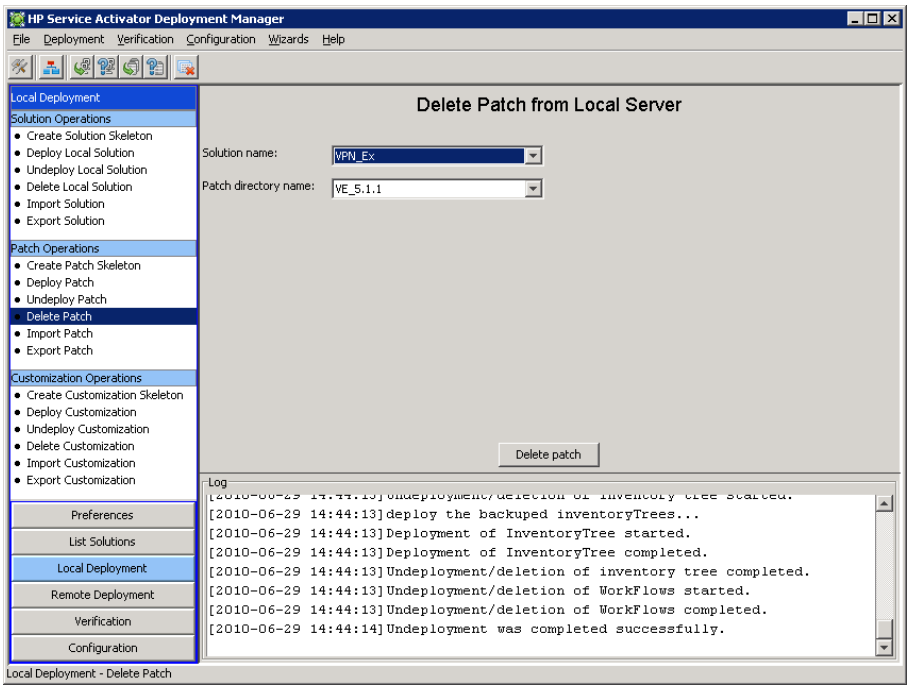
Finally, click the `[Undeploy patch]` button to undeploy the selected patch.

Delete Patch

The user interface for deleting a patch is shown in Figure 5-26. First you need to select the solution name from a dropdown box. This will cause the second dropdown box to be populated with the patch directory names that are available for the selected solution. Finally, click the `[Delete patch]` button to delete the entire patch directory.

NOTE This operation should be used with great care since the Deployment Manager does not provide an option to restore deleted patches.

Figure 5-26 **Deleting a patch**



Customization Operations

This section describes all operations that are related to creating, importing, exporting, deploying, undeploying, and deleting customizations on the local server. Customization operations cannot be performed on remote servers.

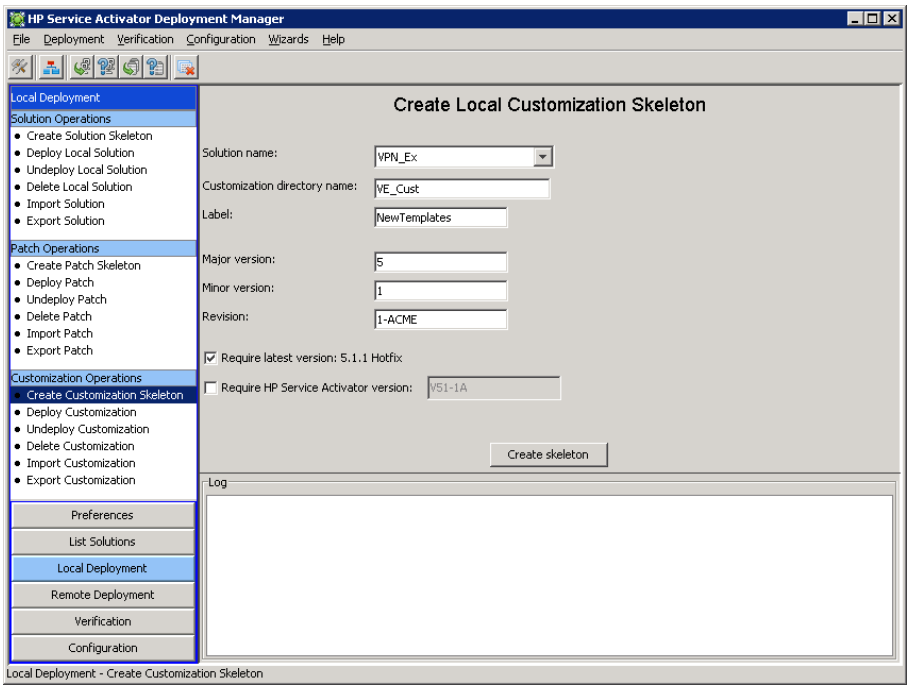
Create Customization Skeleton

The user interface for creating a customization skeleton is shown in Figure 5-27. You must first select the solution for which you want to create a customization. After this you can enter a customization directory name. There directory name should only contain alphanumerical characters and underscores. The label of a customization can, for instance, be used to indicate what the customization does (i.e. it's main functionality) or the name of the customer for whom this customization will be made. Finally, you need to enter the customization's major version, minor version, and revision; these are typically numbers, but they may also contain letters.

If you want the Deployment Manager to add a `<Requires>` section to the `version.xml` file (see the Section "Creating a Version File" on page 26) you can check the "Require latest version" checkbox. Finally, if you want to add a `<Requires-Core>` element to the `version.xml` file you may check the "Require HP Service Activator version" checkbox and enter the required Service Activator version in the text field.

NOTE If you create a customization skeleton for a solution that is not deployed, the "Require latest version" checkbox will be disabled.

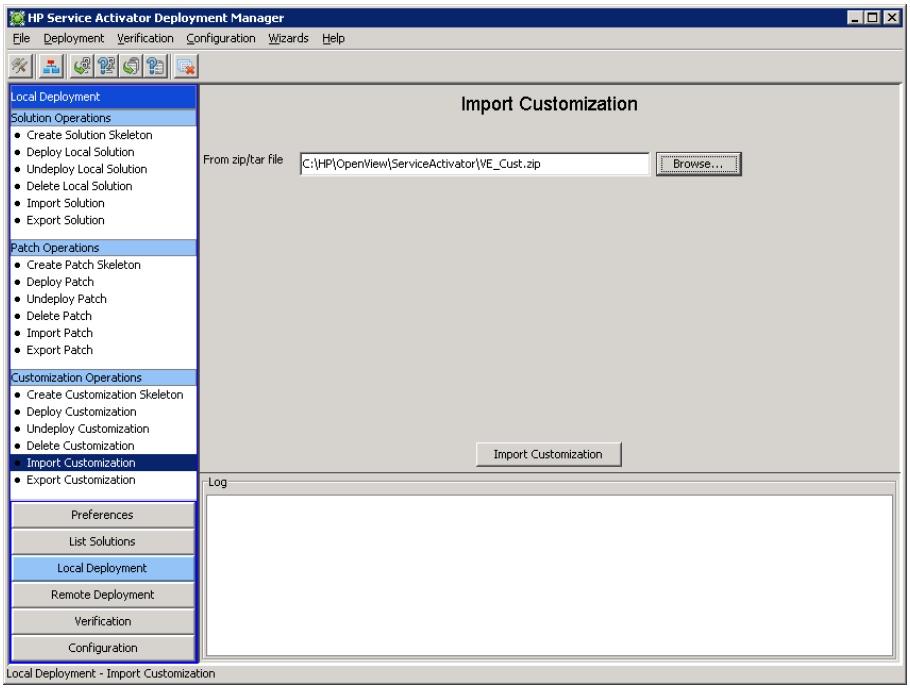
Figure 5-27 Creating customization skeleton



Import Customization

The user interface for importing a customization is shown in Figure 5-28. You must first select the ZIP or TAR file that contains the patch and then click the `[Import customization]` button.

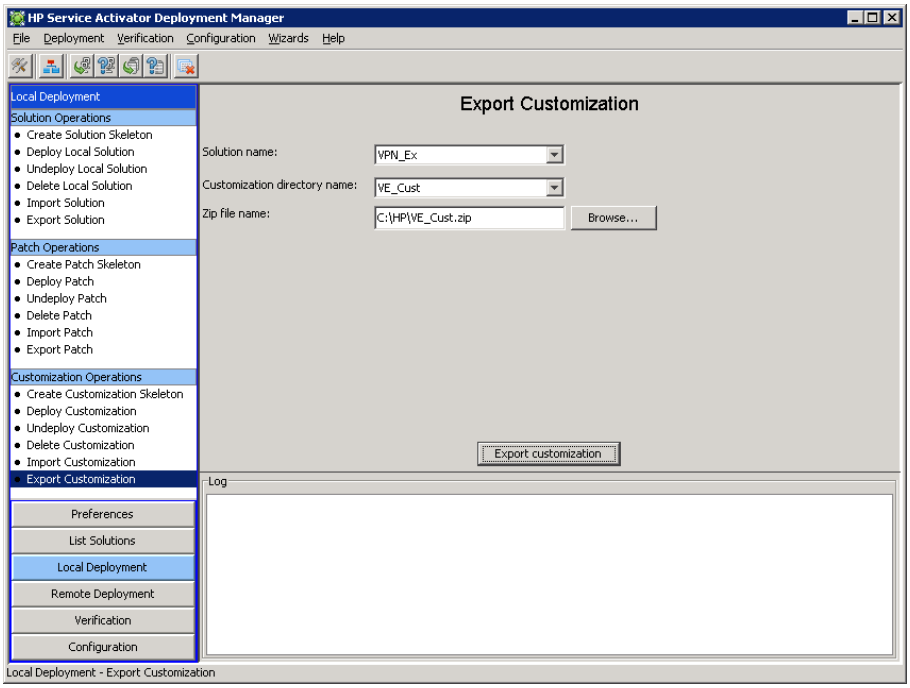
Figure 5-28 Importing a customization



Export Customization

The user interface for exporting a customization to a ZIP file is shown in Figure 5-32. First, you need to select a solution name from the dropdown list and then you can select a customization directory. Finally, enter the name of the ZIP file into which the customization will be exported and click the `[Export customization]` button.

Figure 5-29 **Exporting a customization**



Deploy Customization

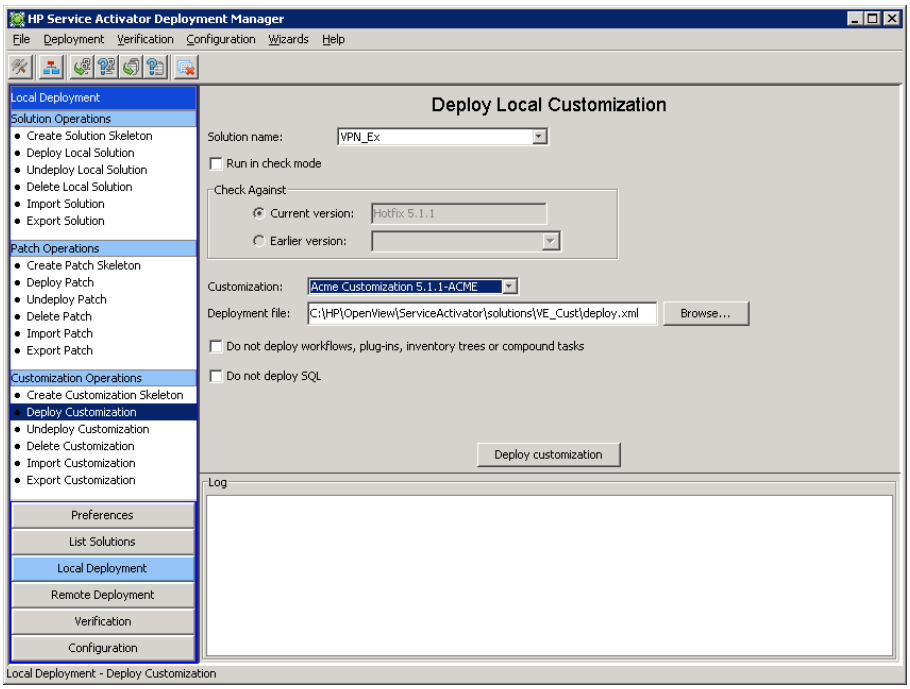
IMPORTANT — It is *highly* recommended that you run a "Verify Deployed Solution" operation before deploying a customization.

Figure 5-30 shows the Deployment Manager's user interface for deploying a customization. You must first select a solution from the dropdown list of available solutions. After that, you must select a customization from the dropdown list as well as a deployment file. If the file `deploy.xml` exists in the customization's base directory, the Deployment Manager will automatically suggest using that. You may override this by clicking the `[Browse...]` button and selecting another deployment descriptor or by manually entering the name of a deployment descriptor file.

After having selected the solution name, the customization, as well as the deployment file you have the following two options:

- **Do not deploy workflows, plug-ins, inventory trees or compound tasks.** If this option is checked the Deployment Manager will not deploy workflows, plug-ins, inventory trees or compound tasks into the system database.
- **Do not deploy SQL.** If this option is checked the Deployment Manager will not execute the deploy SQL scripts (located in the `$PATCH_HOME/etc/sql` directory) against the system database.

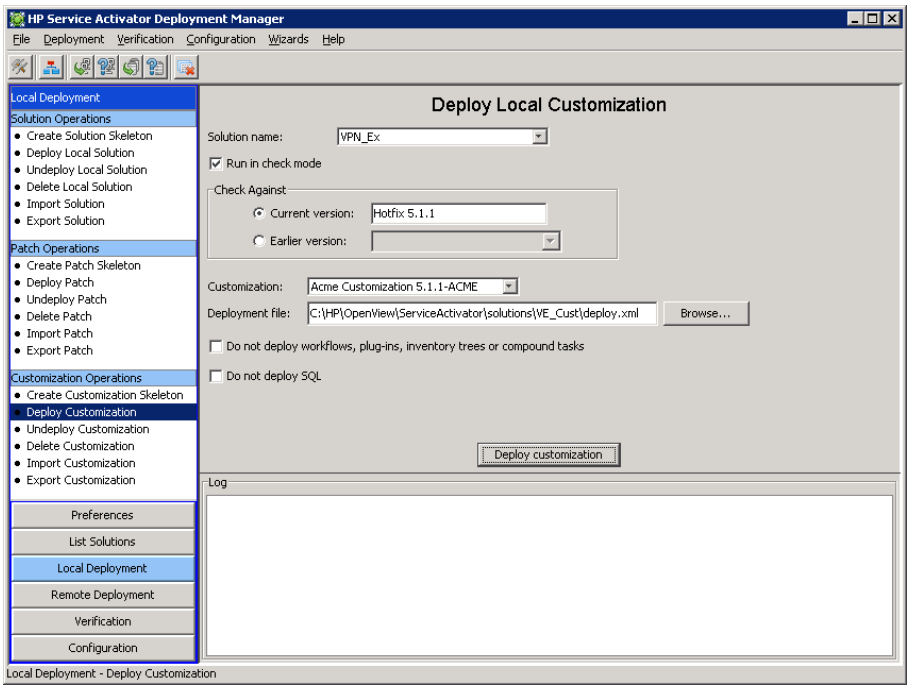
Figure 5-30 Deploying a customization



Similar to when deploying a patch, the Deployment Manager will prompt the user if a conflict (see the Section “Conflict Files” on page 45) is detected while the customization is being deployed; see Figure 5-22. The Deployment Manager will offer the same choices as when conflicts are detected during deployment of patches.

If you wish to check for conflicts without actually deploying the customization, you can run the customization deployment operation in “check mode”. This is done by checking the “Run in check mode” checkbox located below the solution name; see Figure 5-31.

Figure 5-31 **Deploying a customization in “check mode”**



If the “Run in check mode” checkbox is selected, the following options become available on the user interface:

- **Check against current version.** If this option is selected, the Deployment Manager checks the customization against the currently deployed version of the solution. I.e. the Deployment Manager will generate a consolidated checksum file in memory and use that to compare to the components that are deployed in the runtime system.
- **Check against an earlier version.** If this option is selected, the Deployment Manager will check the customization against an earlier version of the solution. To do this, the Deployment Manager generates a partially consolidated checksum file and uses this to compare to the components that are deployed in the runtime system.

In both cases all detected conflicts will be written to the file `conflicts.xml` in the `$SOLUTION_HOME/install` directory. The user will not be prompted every time a conflict is detected but instead the Deployment Manager will assume that the option “keep” is chosen for every conflict.

Figure 5-32 shows an example of a conflict file containing no conflicts.

Figure 5-32 **Conflict file example with no conflicts**

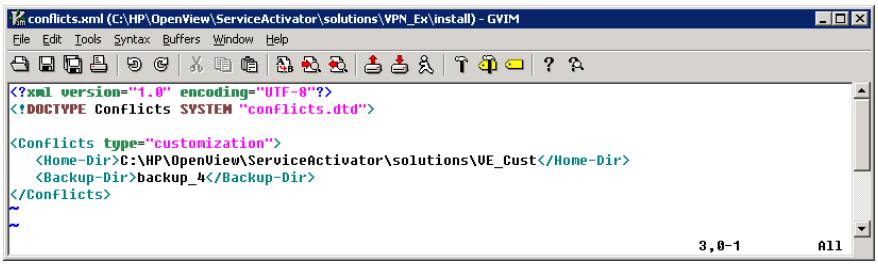
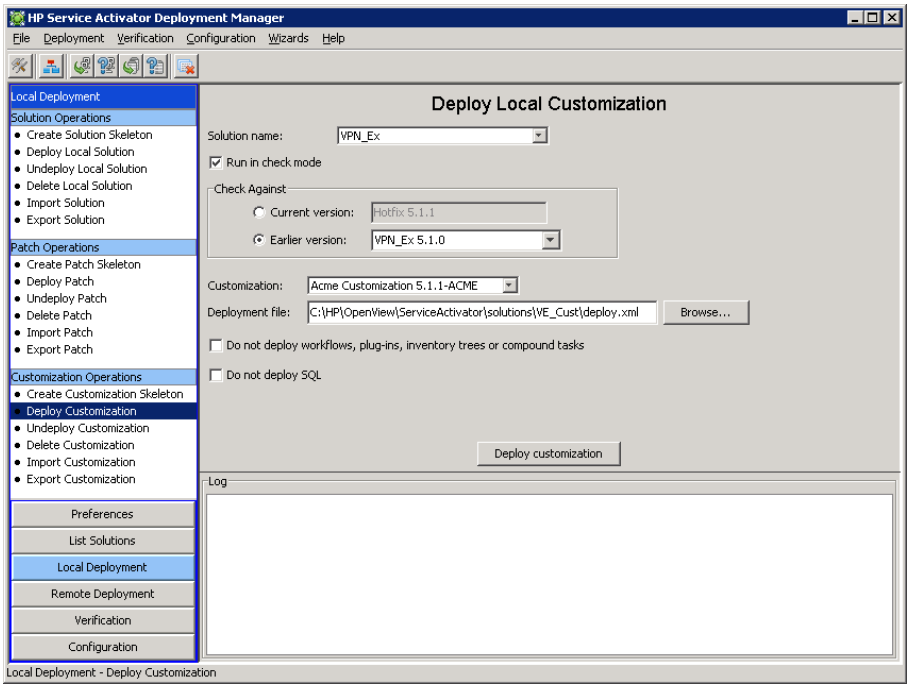


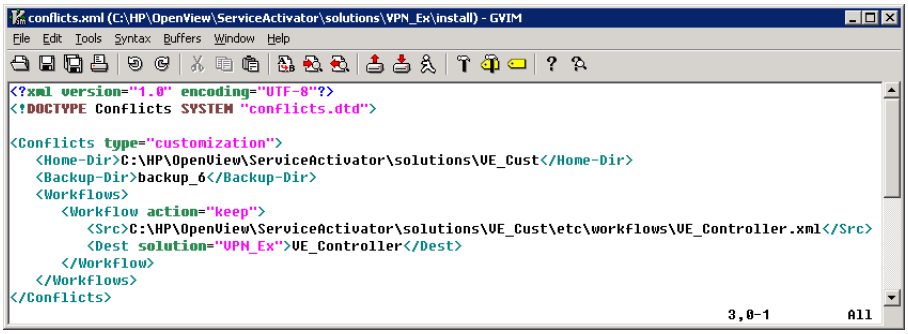
Figure 5-33 shows an example of the deploy customization operation running in “check mode” where the check is done against an earlier version of the solution.

Figure 5-33 **Deploying a customization in “check mode” (check against an earlier version)**



There should be no changes in patch “Hotfix 5.1.1” that collide with the customization “Acme Customization 5.1.1-ACME”, but in this example someone has modified the “VE_Controller” workflow after it was deployed using the Deployment Manager. Hence, the resulting conflict file of the operation is shown in Figure 5-34.

Figure 5-34 **Conflict file example containing two conflicts**



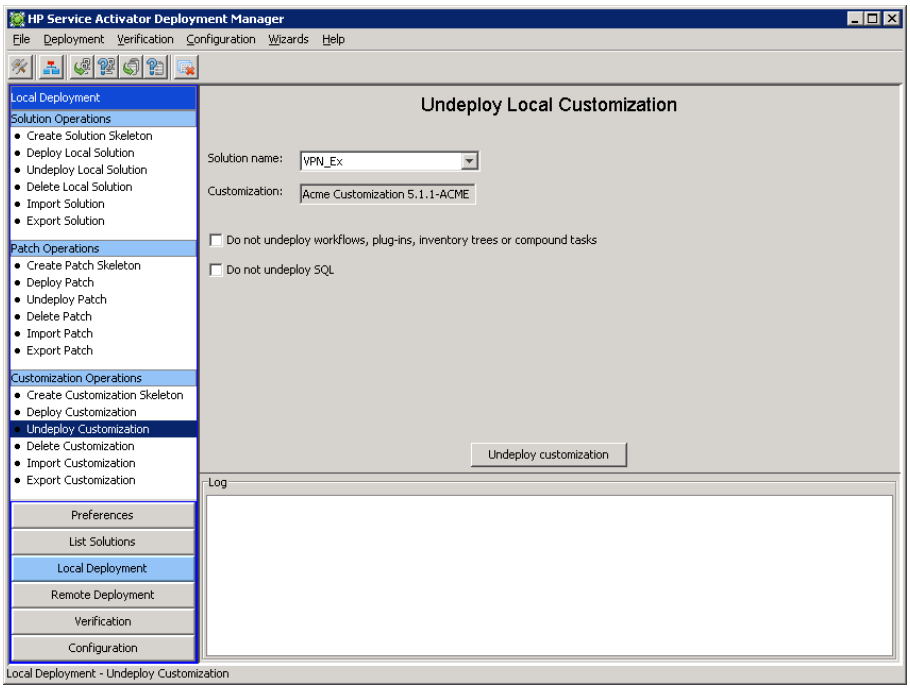
If there had been any collisions between the patch “Hotfix 5.1.1” and the customization “Acme Customization 5.1.1-ACME” these would also be listed in the conflict file.

Undeploy Customization

The user interface for undeploying a customization is shown in Figure 5-35. You must first select the solution for which you intend to undeploy a customization. This will cause the Deployment Manager to populate the “customization” text area with the latest installed customization. The Deployment Manager knows, by reading the latest checksum file, how to undeploy the customization; hence, it does not use the deployment file when the customization is undeployed.

NOTE	If there are no customizations installed for the selected solution, or if the latest deployment for this solution was a patch deployment, the "customization" text area will remain empty and it will not be possible to run a customization undeployment.

Figure 5-35 Undeploying a customization



When the solution name has been selected you have the following two options:

- **Do not undeploy workflows, plug-ins, inventory trees or compound tasks.** If this option is checked the Deployment Manager will not undeploy workflows, plug-ins, inventory trees or compound tasks from the system database.
- **Do not undeploy SQL.** If this option is checked the Deployment Manager will not execute the undeploy SQL scripts (located in the `$SOLUTION_HOME/etc/sql` directory) against the system database.

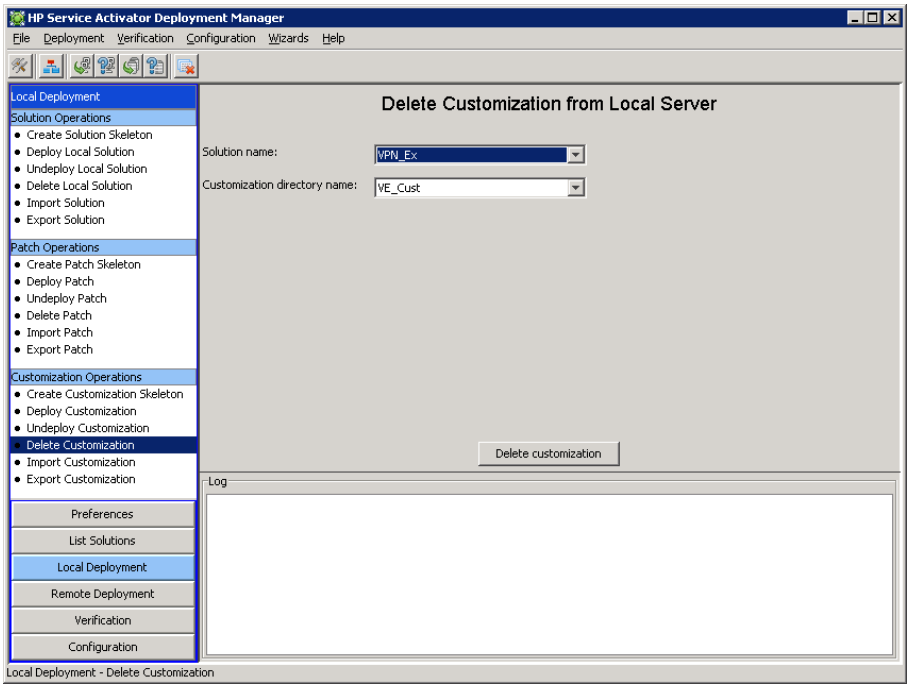
Finally, click the `[Undeploy customization]` button to undeploy the selected customization.

Delete Customization

The user interface for deleting a customization is shown in Figure 5-36. First you need to select the solution name from a dropdown box. This will cause the second dropdown box to be populated with the customization directory names that are available for the selected solution. Finally, click the `[Delete customization]` button to delete the entire customization directory.

NOTE	This operation should be used with great care since the Deployment Manager does not provide an option to restore deleted customizations.

Figure 5-36 Deleting a customization



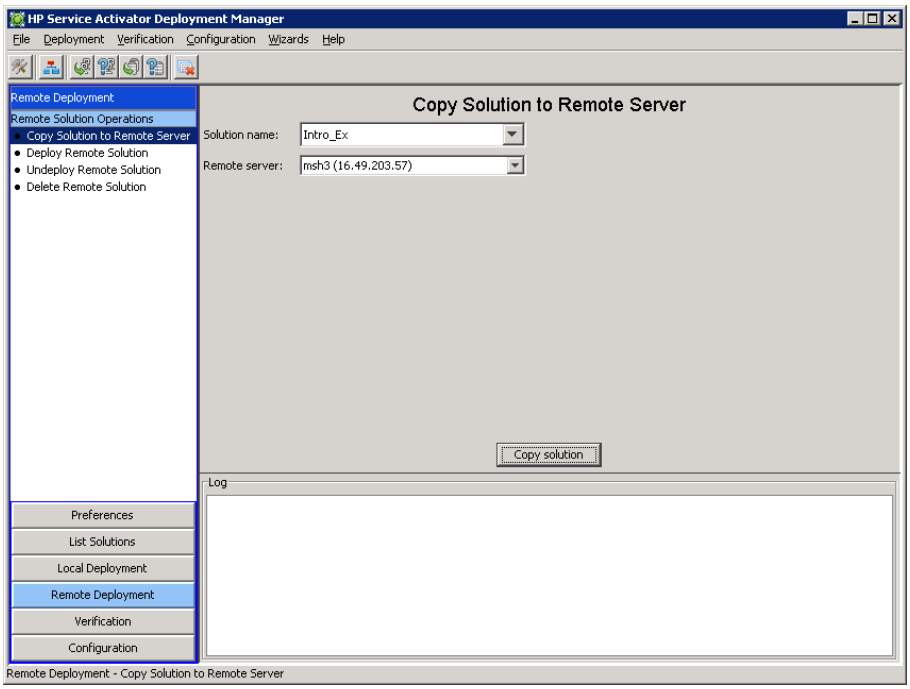
Remote Solution Deployment Operations

This section describes all operations that are related to copying, deploying, undeploying, and deleting solutions on a remote server.

Copy Solution to Remote Server

The Deployment Manager can be used to copy a solution from the local server to a remote server via Secure Copy (SCP). The user interface for this operation is shown in Figure 5-37. To copy a solution, you first need to select the solution from the dropdown list, select the remote server, and finally click the `[Copy solution]` button.

Figure 5-37 **Copying a solution to a remote server**



The Deployment Manager will *not* copy the `install` directory to the target server. This is to ensure that even though a solution is deployed on the local server, it will be marked as *not deployed* on the target server.

Deploy Remote Solution

Deploying a solution on a remote server is similar to deploying a solution on the local server.

However, if the remote server is a member of the same cluster as the local server you will typically only want to deploy solution components on the remote server; *not* in the system database. Hence, you should select the “Do not deploy workflows, plug-ins, inventory trees or compound tasks” as well as the “Do not deploy SQL” checkboxes (these components have typically already been deployed from the local server).

On the other hand, if the remote server is *not* a member of the same cluster as the local server, you will normally want to deploy all solution components; including those components that are deployed into the system database. In this case you should *not* select the “Do not deploy workflows, plug-ins, inventory trees or compound tasks” nor the “Do not deploy SQL” checkboxes. You may also want to create inventory tables on the remote server in this case.

IMPORTANT	Remember to change the database configuration to that of the remote server if you perform a remote deploy operation on a server that is not a part of the same cluster as the local machine. Otherwise the database operations will be performed against the wrong database server.

Since the Deployment Manager knows (from its server configuration) whether a remote server is a cluster member or not, it will automatically set the states of the “Do not deploy workflows, plug-ins, inventory trees, or compound tasks” and “Do not deploy SQL” checkboxes to appropriate default values.

A remote deploy operation does runs in a non-interactive mode. Hence, there is no way to prompt the user if the remote deploy operation (running in “force” mode) is about to overwrite a file or a database component that does not have the “overwrite” attribute set to “true”. Therefore, the remote deploy operation has a checkbox called “Overwrite all” with a self-explanatory effect.

Figure 5-38 shows the user interface for deploying a solution on a remote server. The first step is selecting the remote server from the dropdown list. Then you need to click the `[Get solution list]` button to connect to the remote server and retrieve a list of not deployed solutions on the remote server.

Figure 5-38 Deploy Remote Solution – Select remote server

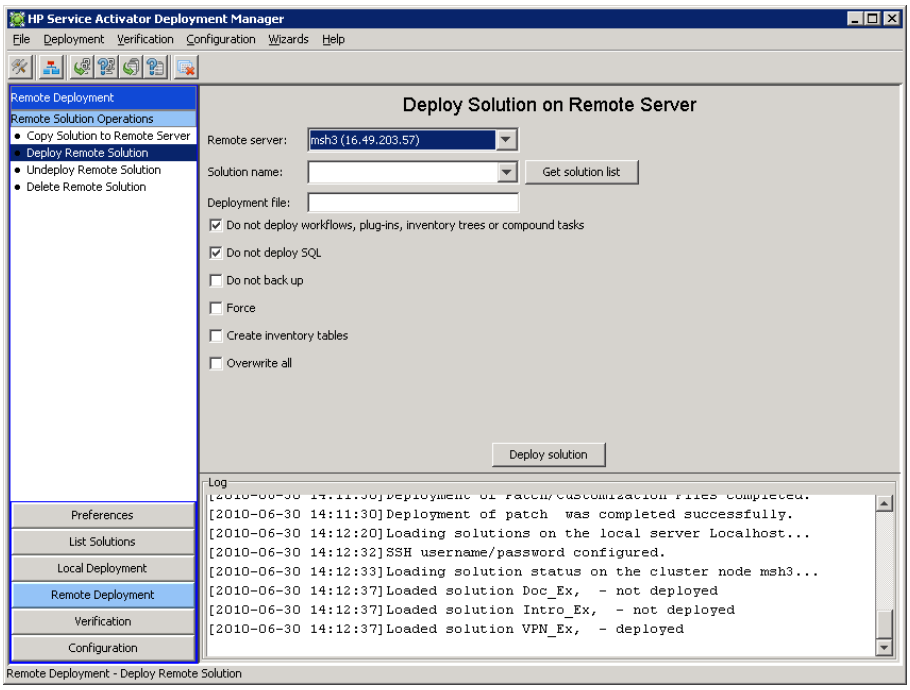
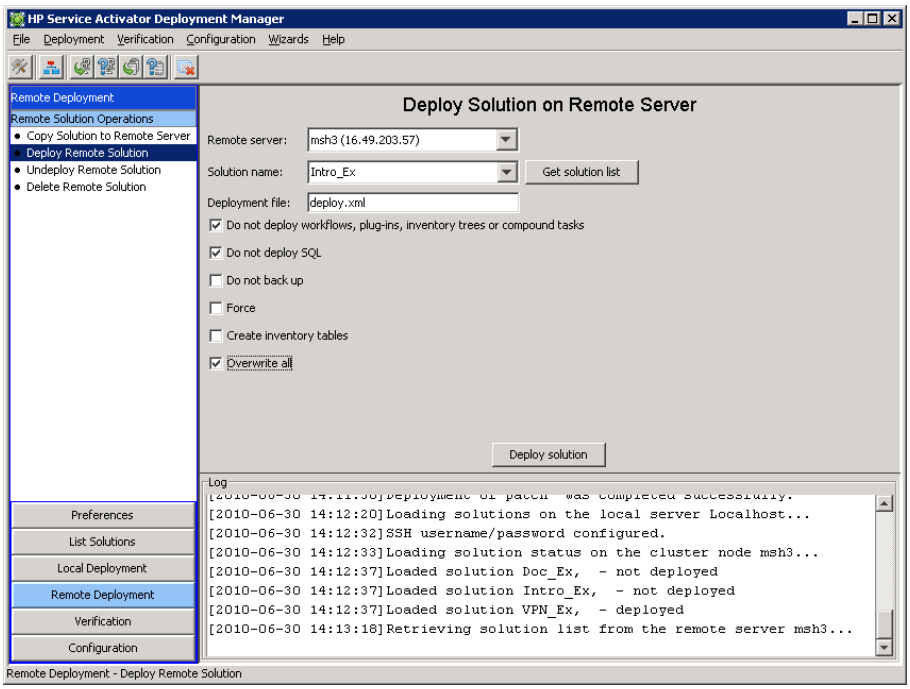


Figure 5-39 shows a screenshot of the Deployment Manager after the list of solutions have been retrieved from the remote server. The name of the deployment file must then be entered manually; the Deployment Manager does not provide a function to retrieve a list of deployment files from the remote server.

Finally, you need to set the five checkboxes shown in Figure 5-39 appropriately and click the `[Deploy solution]` button.

NOTE The behavior of the five checkbox options are described in detail in the section “Deploy Solution” on page 61.

Figure 5-39 **Deploy Remote Solution – Enter name of deployment file**



Undeploy Remote Solution

Undeploying a solution from a remote server is similar to undeploying a solution from the local server. However, as for the “Remote Deploy Solution” operation you need to distinguish between the situation where the remote server is a member of the same cluster as the local server and the situation where the remote server is *not* a cluster member.

If the remote server is a member of the same cluster as the local server you will typically *not* want to undeploy the components that a located in the system database as this would render the solution useless across the entire cluster. Also, you would typically *not* want to delete the inventory tables in this scenario.

Contrary, if the remote server is *not* a member of the same cluster as the local server, the intended behavior is normally to delete all solution components; including those deployed in the Service Activator system database. In this case you should leave the “Do not undeploy workflows, plug-ins, inventory trees or compound tasks” and the “Do not deploy SQL” options unchecked. If you want to delete the inventory tables you must select the “Delete inventory tables” checkbox.

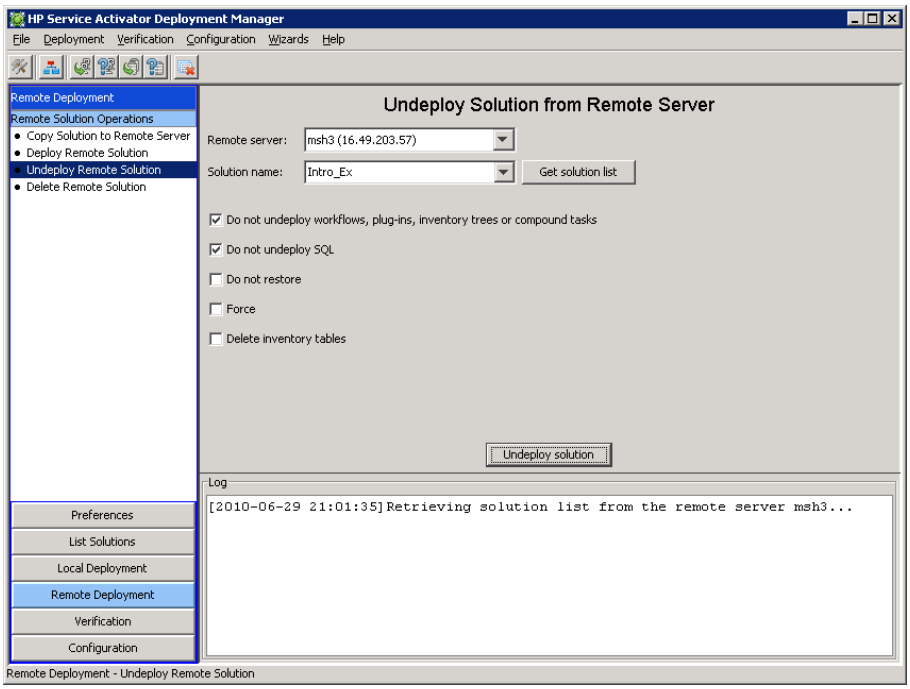
The Deployment Manager sets the default value of the “Do not undeploy workflows, plug-ins, inventory trees or compound tasks” and the “Do not deploy SQL” checkboxes based on the knowledge it has about the remote server; hence, whether or not it is a cluster member.

WARNING	Be very careful before deleting inventory tables. The Deployment Manager provides no mechanisms for restoring lost inventory data.

The user interface for undeploying a solution from a remote server is shown in Figure 5-40. First select the remote server from the dropdown list and then click the `[Get solution list]` button. The Deployment Manager will now connect to the remote server and retrieve the list of deployed solutions. Now, select the solution name from the dropdown list and check/uncheck the five checkboxes if needed. Finally, click the `[Undeploy solution]` button.

NOTE	The behavior of the five checkbox options are described in details in the section “Undeploy Local Solution” on page 63.

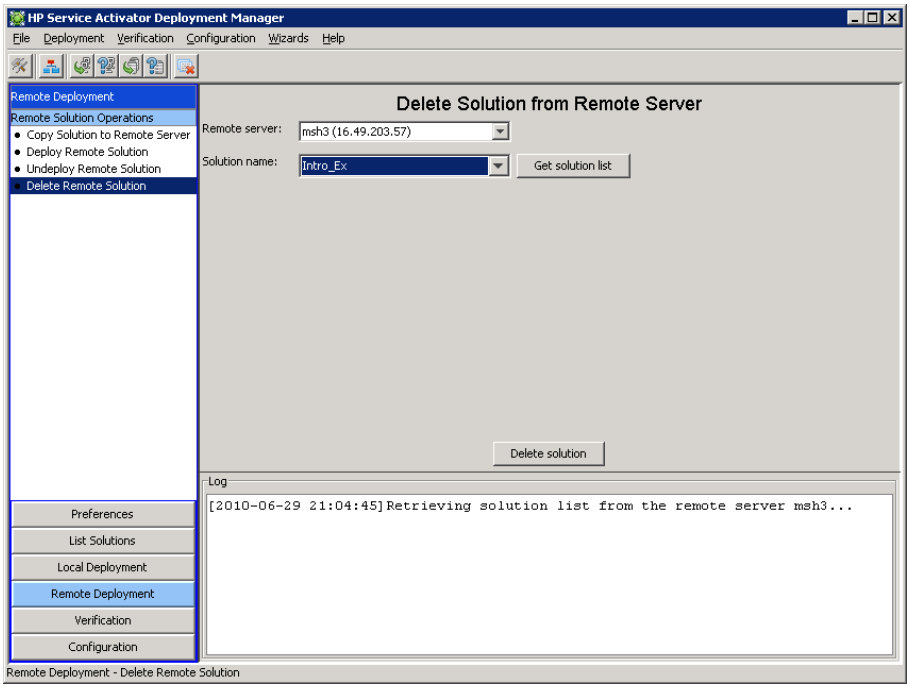
Figure 5-40 Undeploy a solution from a remote server



Delete Remote Solution

Figure 5-41 shows the Deployment Manager's user interface for deleting a solution from a remote server. You first need to select the remote server name and then click the `[Get solution list]` button to retrieve a list of candidates. Then select the solution you wish to delete and click `[Delete solution]`.

Figure 5-41 Delete a solution directory from a remote server



Verification

Generate Local Checksum File

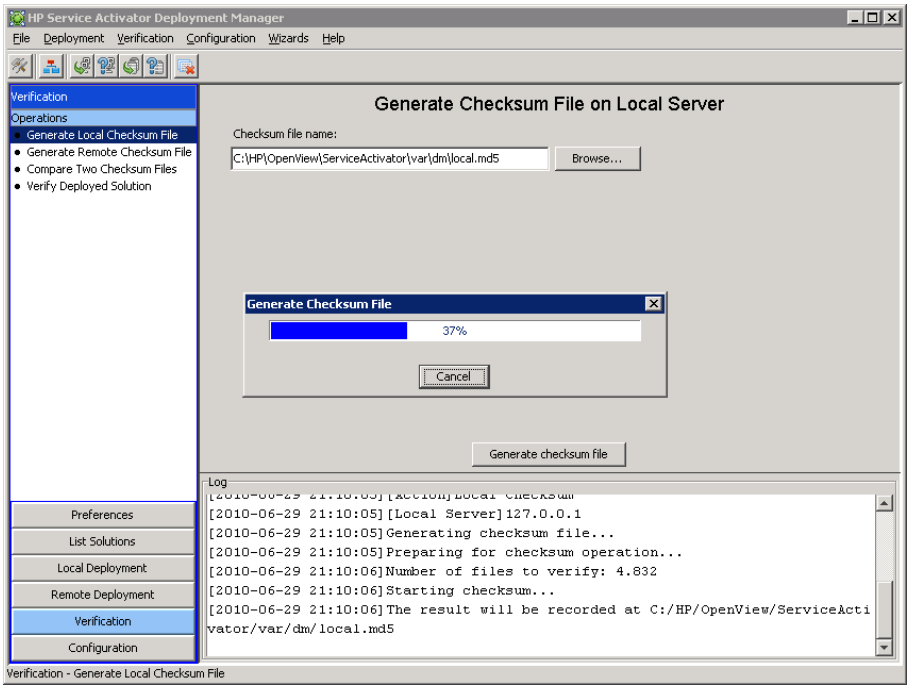
NOTE	The checksum files described in this section should not be confused with the checksum files that are generated by the Deployment Manager when a solution, patch, or customization is deployed.

The Deployment Manager includes a checksum calculation function. This function can be useful in various scenarios:

- To compare a Service Activator installation on one server to a Service Activator installation on another server.
- To compare a Service Activator installation before and after deploying a solution.
- To compare a Service Activator installation with an old version of a solution deployed to a Service Activator installation with a new version of a solution deployed.
- In addition, it can make good sense to generate a checksum file and attach it to an email when submitting a support call.

Figure 5-42 shows a screenshot of the Deployment Manager's user interface for generating checksums. You need to specify a file name to be used to store the calculated checksums followed by clicking the `[Generate checksum file]` button.

Figure 5-42 Generating checksums on the local server



Generate Remote Checksum File

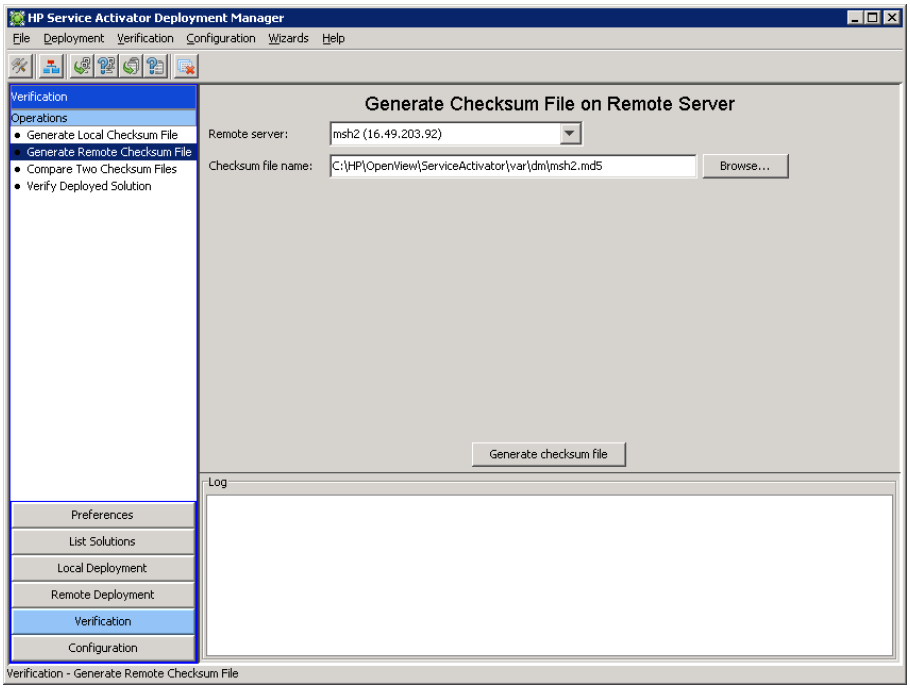
Generating a checksum file on a remote server is identical to generating a checksum file on the local server, with the following two exceptions (see Figure 5-43):

- You need to select the remote server from the dropdown list

- The checksum file that is generated one the remote server is automatically copied by the Deployment Manager to the local server. Hence, the file name that is specified for the checksum file is a file name on the *local* server.

After having selected the remote server as well as specified the checksum file name, click the `[Generate checksum file]` button.

Figure 5-43 Generating checksums on a remote server



Compare Two Checksum Files

You can use the Deployment Manager to compare two checksum files generated on different servers or generated on the same server. Simply enter the names of the two checksum files (see Figure 5-44) and click `[Compare]`.

Figure 5-44 **Comparing two checksum files**

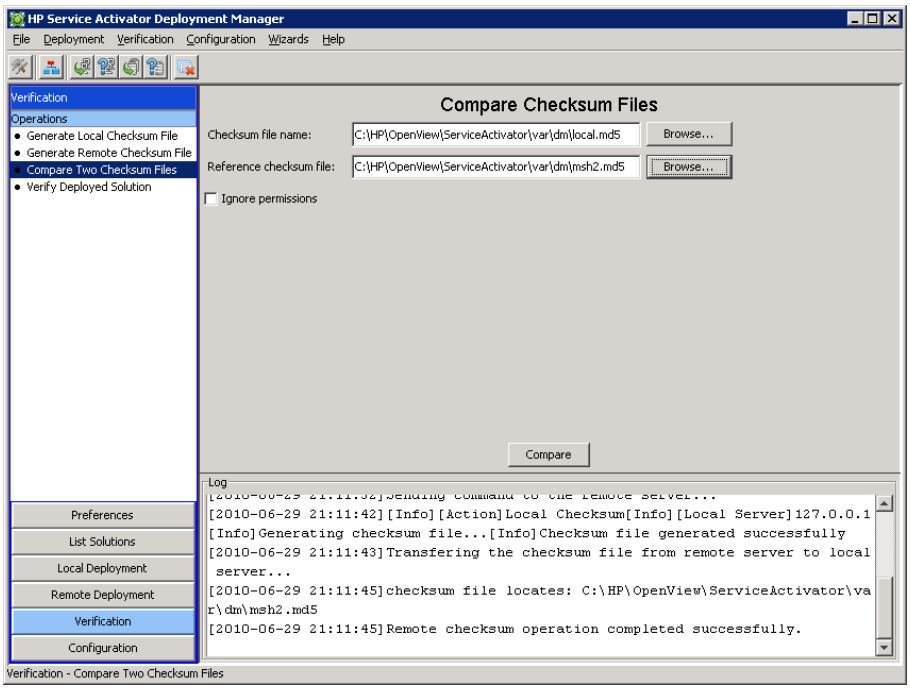
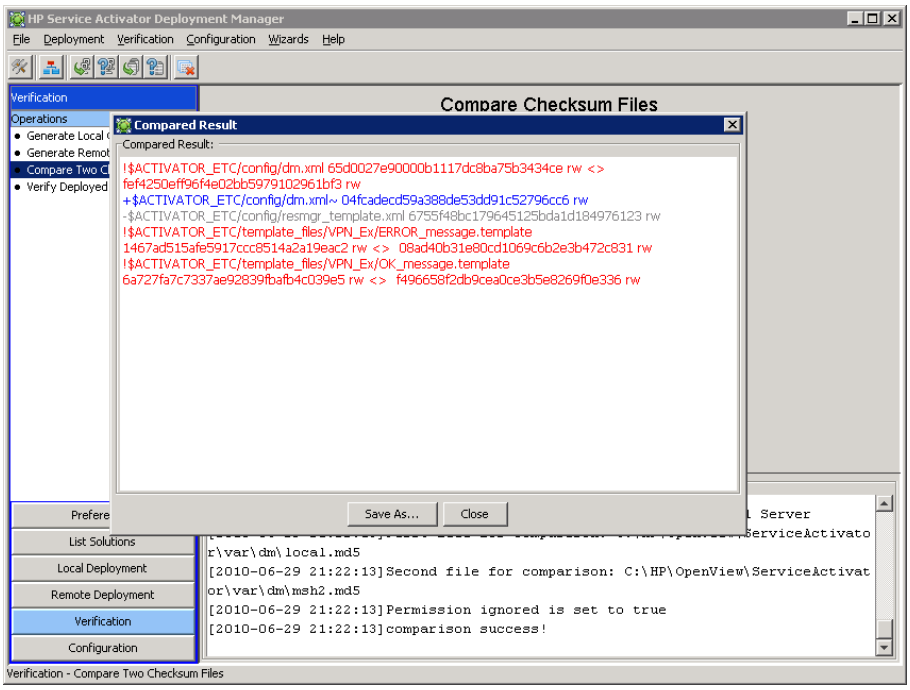


Figure 5-45 shows a screenshot of the Deployment Manager after the compare checksums operation has been executed. Differences between the two checksum files are color coded in the following way:

- **Red** (line prefixed with an exclaimation mark) – means that a file is present in both reference files, but their checksums or permissions were not identical.
- **Blue** (line prefixed with a plus) – means that a file is present in the “checksum file” but not in the “reference checksum file”.
- **Grey** (line prefixed with a minus) – means that a file is present in the “reference checksum file” but not in the “checksum file”.

If you wish to ignore file permissions when comparing the checksum files you must select the “Ignore permissions” checkbox (see Figure 5-44). This is particularly useful if you want to compare a file generated on a UNIX system to a file generated on a Windows system.

Figure 5-45 Result of comparing two checksum files



Verify Deployed Solution

You can use the Deployment Manager to verify that a deployed solution has not been modified since it was deployed. To do this, you can use the “Verify Deployed Solution” operation; see Figure 5-46.

To verify a solution you simply need to select the solution name from a dropdown list and then click `[Verify solution]`.

Figure 5-46 **Verifying a deployed solution**

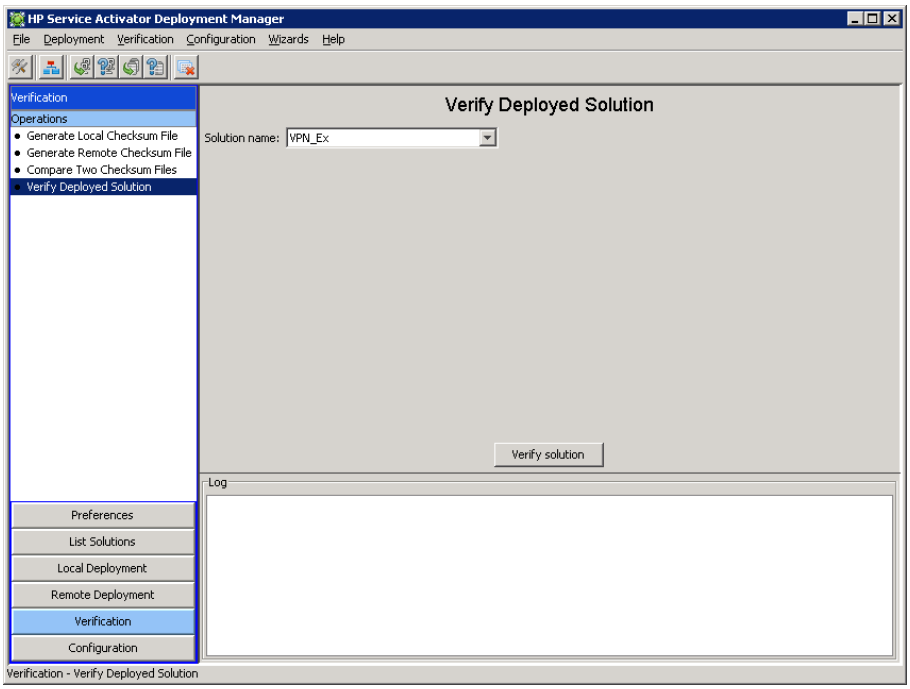


Figure 5-47 shows an example of a verification result where no components of a solution have been modified after the solution was deployed.

If one or more components in a solution have been modified or deleted since the solution was deployed, the Deployment Manager will list all components that were deleted as well as all components that were modified since the solution was deploy. Figure 5-48 shows an example of this.

Figure 5-47 **Verification report for a solution that has not been modified since deployment**

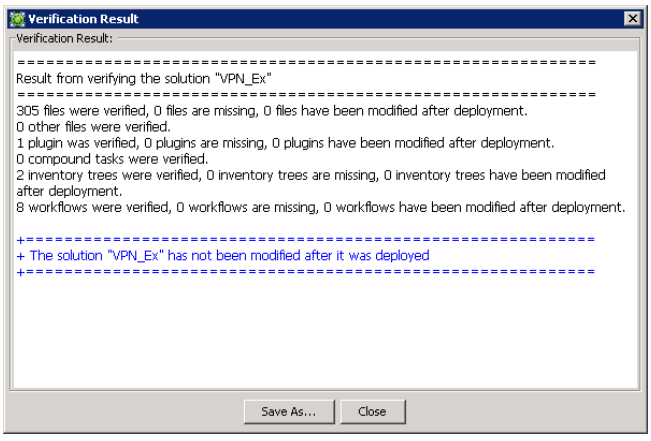
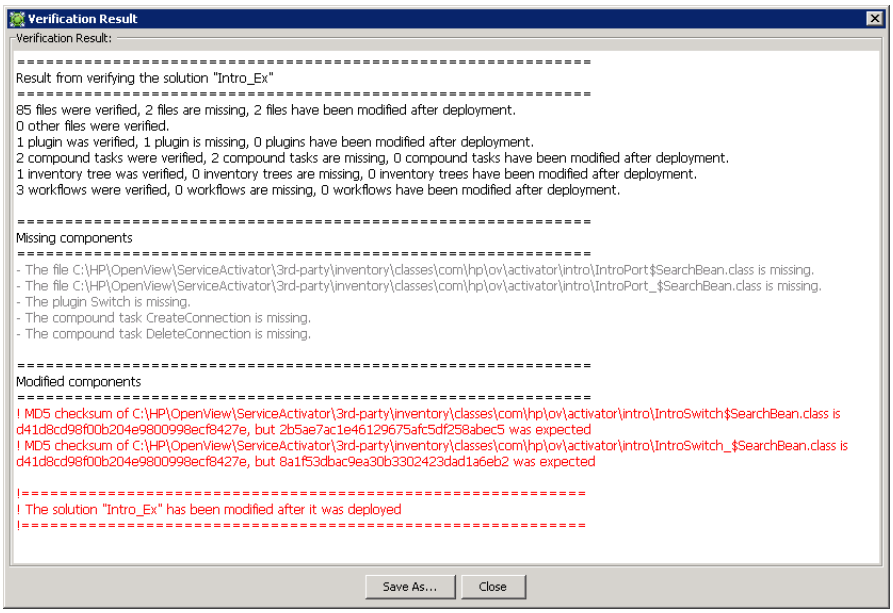


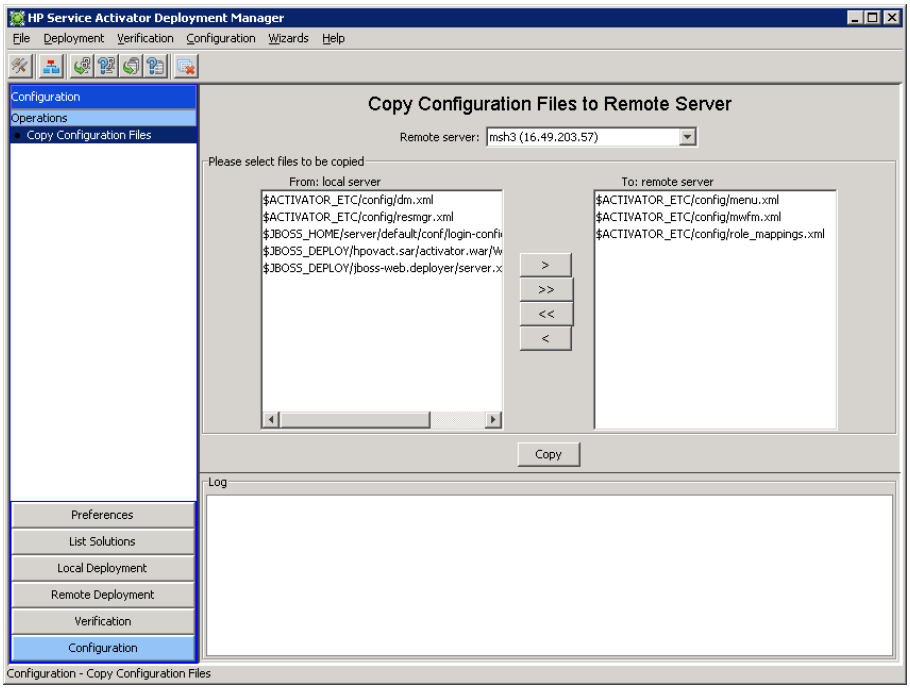
Figure 5-48 **Verification report of a solution that has been modified since deployment**



Copy Configuration Files

Figure 5-49 shows the Deployment Manager's UI for copying files from the local server to a remote server. This wizard is extremely useful when you want to change one or more configuration files in a cluster environment. You can include additional configuration files in the list of candidate reference files by configuring the `dm.xml` file (see the section "Service Activator Configuration Files" on page 42).

Figure 5-49 **Copying configuration files to a remote server**



In order to copy configuration files from the local server to a remote server you must select the remote server from the dropdown list and then select the files to be copied to the remote server. Finally, click the `[Copy]` button to copy the selected files via Secure Copy.

NOTE	This operation must be used with great care since it overwrites files without warnings and without making backups.

Wizards

The Deployment Manager implements four wizards guiding you through a sequence of actions that is recommended for copying (and deploying) a solution to remote servers as well as comparing servers.

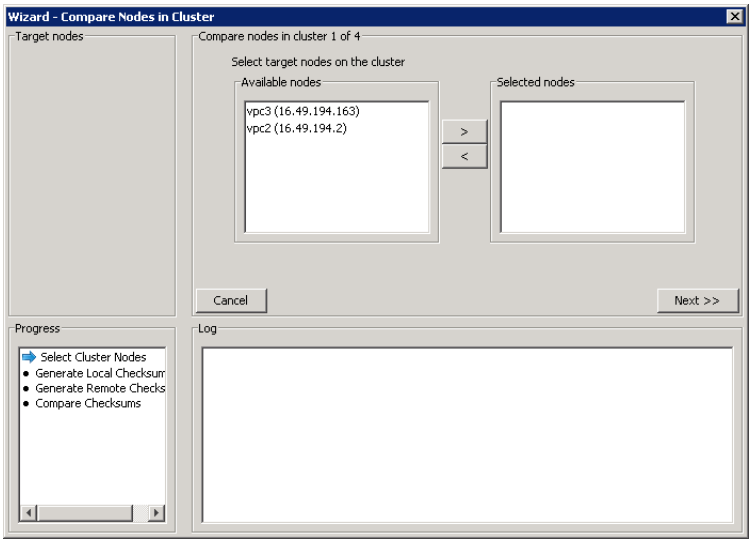
The wizards use default settings for the “remote deploy” operation (described in more detail later in this section); these can not be overwritten. If you wish to use other settings you must use the individual operations instead (see the section “Local Solution Deployment” on page 58).

Compare Nodes in Cluster

The “Compare Nodes in Cluster” wizard can guide you through the steps needed for comparing a checksum file generated on the local server to one or more check sum files generated on one or more nodes in the cluster.

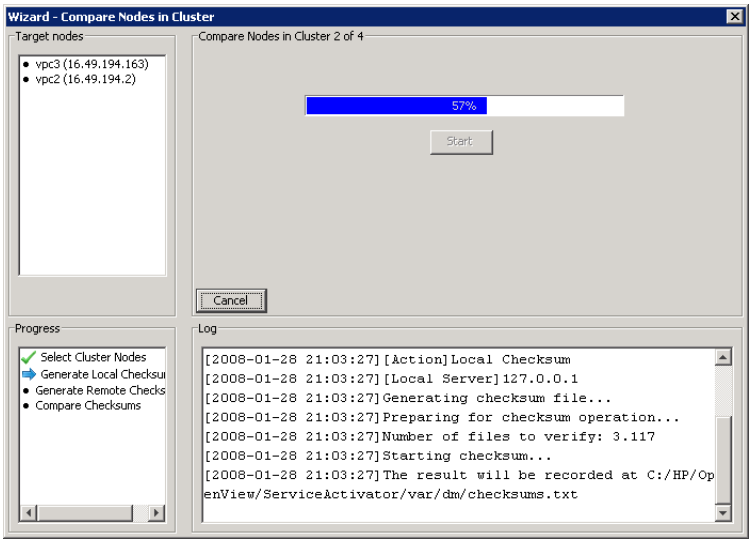
The wizard is invoked by selecting the `Wizards:Compare Nodes in Cluster...` menu item. This will bring up a window as shown in Figure 5-50.

Figure 5-50 Compare Nodes in Cluster wizard – select nodes



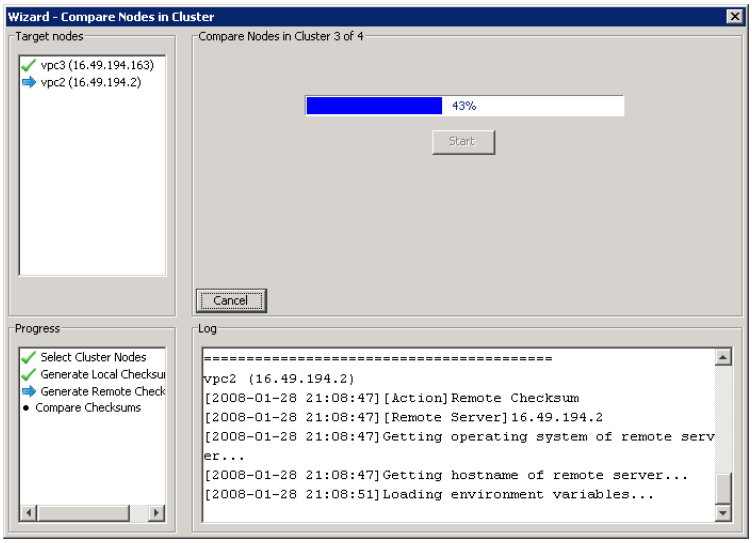
In this windows you can select the nodes that you wish to compare to the local server. When you have done so, click `[Next>>]`. This brings you to a new screen and the arrow in the process area moves to “Generate Local Checksum”; see Figure 5-51. You need to click the `[Start]` button to begin the checksum calculation.

Figure 5-51 **Compare Nodes in Cluster wizard – generate checksums on local server**



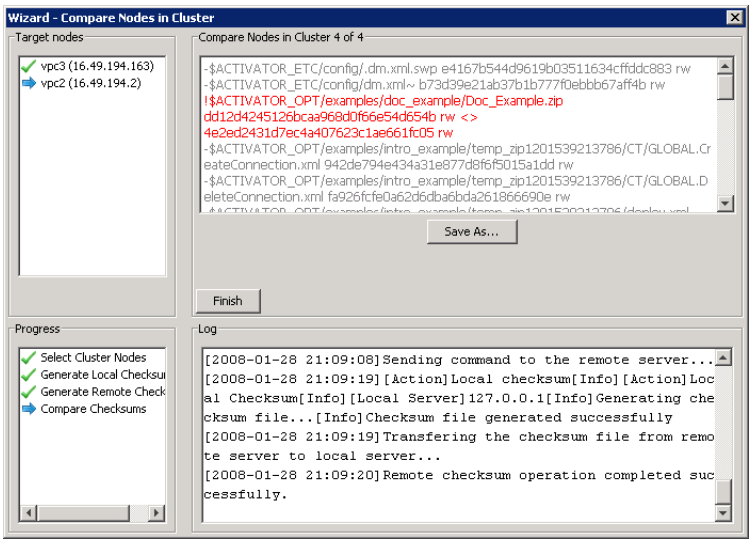
After a while, when the local checksum file has been generated, the wizard moves on to the selected cluster nodes (one at a time). This is indicated in the wizard with the blue arrow now pointing at one of the selected nodes (see Figure 5-52). Again, click the `[Start]` button to begin the checksum calculation on the selected cluster node.

Figure 5-52 **Compare Nodes in Cluster wizard – generate checksums on remote node**



When the checksum generation is finished for each target node, the wizard will bring you to a screen that displays the difference between the checksum file on the local server and the checksum file from the remote server.

Figure 5-53 **Compare Nodes in Cluster wizard – show result**



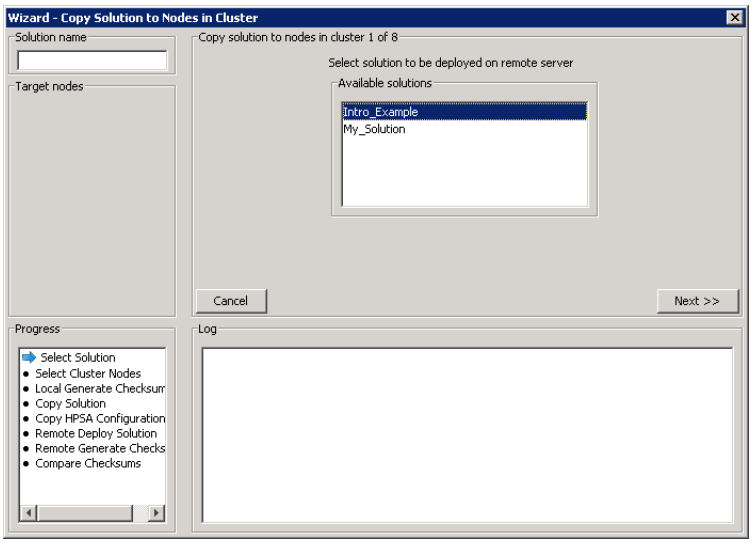
Copy Solution to Nodes in Cluster

The “Copy Solution to Nodes in Cluster” wizard can be used to push a solution from the local server to all other nodes in the cluster. You should deploy the solution in the local server before you use this wizard to push the solution to the other nodes in the cluster.

When the solution is deployed on each target node the Deployment Manager will *not* deploy anything to the database; i.e. the same behavior as when the “Do not deploy workflows, plug-ins, inventory trees, compound tasks or SQL” option is checked in the “Deploy Solution” operation.

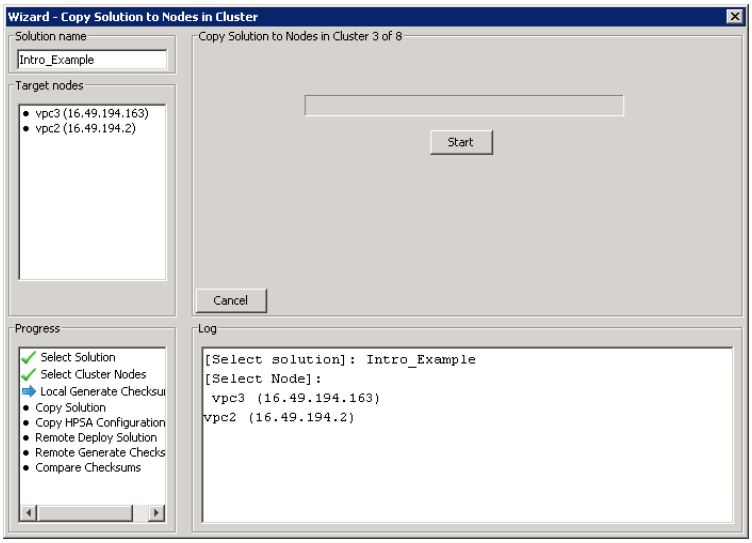
The first step in the wizard is to select the solution to be copied to the other nodes in the cluster. A screenshot illustrating this step is shown in Figure 5-54. Simply select a solution from the list of candidates and click the `[Next>>]` button.

Figure 5-54 **Copy Solution wizard – select solution**



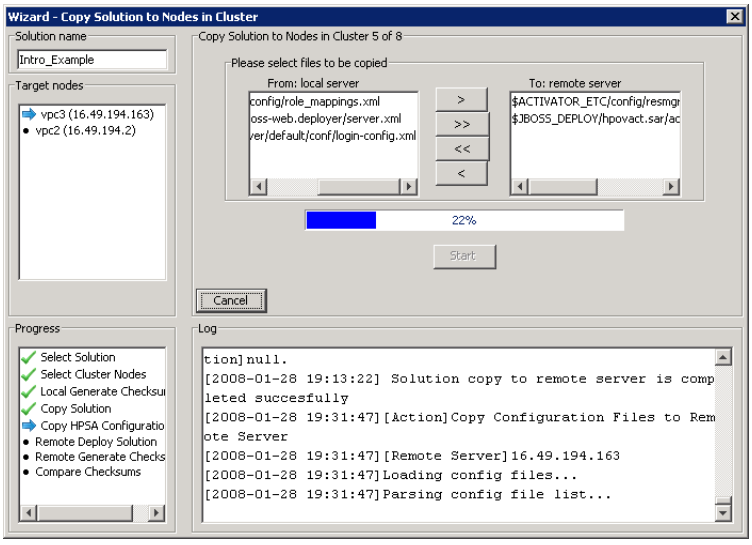
The next step is to select the target nodes. After this has been done, the wizard will generate a checksum file on the local server; see Figure 5-55. This checksum file will be compared to checksums generated on the remote servers for manual verification purposes.

Figure 5-55 **Copy Solution wizard – generate checksum file on local server**



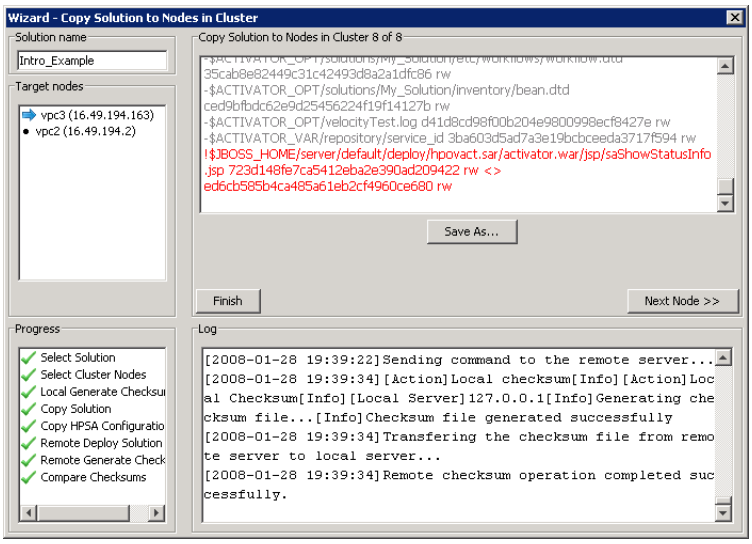
After this the wizard will begin working on the selected target nodes, one by one. In the first step, the selected solution is simply copied to the target node. When this is done, the wizard presents you with a list of configuration files that you can choose to copy to the target node; this is shown in Figure 5-56.

Figure 5-56 **Copy Solution wizard – copy configuration files**



The next two steps are to deploy the solution on the remote server followed by generating a checksum file on the remote server. Finally, when the checksum file has been generated and transferred to the local server, the wizard will compare to the checksum file generated on the local server and display the comparison result in the user interface; see Figure 5-57.

Figure 5-57 **Copy Solution wizard – show comparison result**



Compare to Remote Server

This wizard is very similar to the “Compare Nodes in Cluster” wizard (see page 89). The only difference is that it is only possible to compare to a single server, and you can only choose to compare to a server that is not a member of the same cluster as the local server.

Copy Solution to Remote Server

The “Copy Solution to Remote Server” is similar to the “Copy Solution to Nodes in Cluster” wizard described on page 91. There are two differences:

- You can only select a single target server (one from the list of configured server that is not a member of the same cluster as the local server).
- Workflows, plug-ins, inventory trees, compound tasks and SQL *will* be deployed on the remote server.

Using the Deployment Manager in Text Mode

In addition to the Java Swing graphical user interface, the Deployment Manager also has a command-line interface. This is useful if you need to work with the Deployment Manager over a slow connection or if you want to use the operations provided by the Deployment Manager in your scripts.

The general syntax for running the Deployment Manager from command-line is shown below:

```
deploymentmanager <COMMAND> [ARGUMENTS]
```

The following commands are supported when using the Deployment Manager in text mode:

- **`CreateSolution`** – creates an empty solution skeleton on the local server
- **`DeploySolution`** – deploys a solution on the local server
- **`UndeploySolution`** – undeploys a solution from the local server
- **`DeleteSolution`** – deletes a solution directory on the local server
- **`GenerateChecksums`** – generates a file containing checksum of selected files on the local server

- **`CompareChecksums`** – compares two checksum files are calculates the differences
- **`RemoteCopySolution`** – copies a solution to a remote server
- **`RemoteDeploySolution`** – deploys a solution on a remote server
- **`RemoteUndeploySolution`** – undeploys a solution from a remote server
- **`RemoteDeleteSolution`** – deletes a solution directory on a remote server
- **`CopyConfigFile`** – copies one or more Service Activator configuration files from the local server to a remote server
- **`DeployPatch`** – deploys a patch on the local server
- **`UndeployPatch`** – undeploys a patch from the local server
- **`DeployCustomization`** – deploys a customization on the local server
- **`UndeployCustomization`** – undeploys a customization from the local server
- **`VerifyDeployedSolution`** – verifies all components of a deployed solution
- **`ImportSolution`** – imports a solution from a ZIP file
- **`ExportSolution`** – exports a solution (with or without patches and customizations) to a ZIP file
- **`ImportPatch`** – imports a patch from a ZIP file
- **`ExportPatch`** – exports a patch to a ZIP file
- **`ImportCustomization`** – imports a customization from a ZIP file
- **`ExportCustomization`** – exports a customization to a ZIP file

You can get more information by typing `deploymentmanager -help` at the command prompt.

6 Troubleshooting

Copy solution operation fails

If the Deployment Manager fails when trying to copy a solution to a remote server you should verify that the target server has `scp` in its default `PATH`. For more information read the following:

- http://www.openssh.com/faq.html

Oracle issues

If your Oracle database is configured with too few processes the Deployment Manager will fail during deployment of workflows, inventory trees, plug-ins, compound tasks or SQL. If you encounter such issues you should try to increase the number of Oracle processes.

Example:

```
ALTER SYSTEM SET PROCESSES=500 SCOPE=SPFILE;
COMMIT;
```

You should restart your Oracle database to ensure that the changes take effect.

Directories are not deleted when I undeploy a solution

This is a known limitation. The Deployment Manager has no capabilities for managing directories.

Failure when trying to deploy SQL

If the Deployment Manager encounters an error during a deployment operation it will try to roll back to ensure that the system is intact. This is one of the most fundamental principles onto which the Deployment Manager is built.

This also means that if an error occurs when the Deployment Manager tries to execute an SQL statement the entire deployment operation will fail; hence, you must make sure that your SQL is correct and that it does not cause Oracle to throw any errors. The Deployment Manager does not notice the difference between different types of Oracle errors.