

Best Practices for Promoting Service Manager® Tailoring Changes from Development to Production Environments



Introduction	2
Requirements	2
How to use this document	2
Service Manager tools for applying customizations to the production environment	2
Development Auditing utility	2
Revision control	4
Unload Script utility	5
Differential Upgrade utility	7
Patch records	7
Signature records	8
Manual Processing	9
Best Practice recommendations	9
Process	9
Comparison of the tools to use for promoting customizations	11
HP best practice recommendation	11
For more information	13

Introduction

A complete Service Manager environment consists of at least three different instances of Service Manager:

- A development instance, which is used to customize Service Manager to the exact needs of the customer;
- A test instance, which is used to test these changes in a copy of the production system before promoting the customizations into production; and
- The production instance itself.

Service Manager offers the system developer different tools to make the transitions from development to test, and from test to production, easier. This document introduces all the available tools, and discusses which tool is best in which environment.

Requirements

This document assumes that the reader is familiar with Service Manager and its tailoring tools. The system developer is responsible for documenting all tailoring changes for later reference, such as during a subsequent upgrade.

How to use this document

This document is divided into two sections:

- The first section describes the different tools and their use.
- The second section suggests best practices for promoting tailoring changes from development into production.

Service Manager tools for applying customizations to the production environment

Development Auditing utility

The Auditing utility (devaudit) tracks changes to certain records during the development phase of the implementation. Whether you make a few changes or extensively customize your system, it is critical to keep a record of your changes to ensure that you load the correct version when you move to production. Though the Auditing tool helps you find modified records, HP strongly recommends that you record each change more extensively either using tools outside of Service Manager or using the Revision control feature.

The Auditing utility tracks changes to the following files:

- screlconfig
- displayevent
- displayoption
- displayscreen
- joindefs
- eventfilter
- eventmap
- eventregister

- formatctrl
- Object
- Process
- States
- application
- code
- datadict
- enclapplication
- format
- link
- menu
- querystored
- scmessage
- scripts
- triggers
- validity
- wizard

To access the Auditing utility, click on **Menu Navigation – Tailoring - Audit**. Then click **Turn Auditing On/Off** to toggle Auditing on and off.

Auditing should always be disabled on a production or test system. Auditing should be enabled on development systems when it is being used to promote from development to production. The correct Auditing settings for a development system are:

<input checked="" type="checkbox"/>	Do you want to audit development changes?
<input type="checkbox"/>	Do you want to keep backups of Changes?
<input type="checkbox"/>	Do you want to create auditdelete records on unload?

To view changes that Auditing has recorded, click **View Audit History** in the Audit Menu. A search screen is displayed. Run a true query to show a list of all the changes made to the files listed above. The records have the following structure:

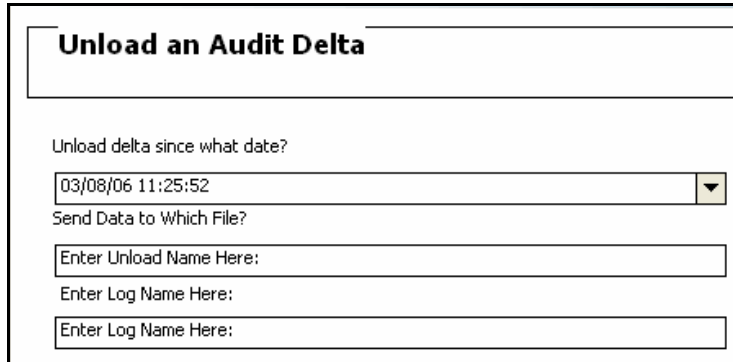
Audit History	
Audit ID:	8862450
Filename:	format
Keys:	IM.template.update.g
Event Type:	update
Date:	11/21/05 09:31:04 ▼
Operator:	falcon
Last Record:	false

To view all changes since a specific date, or changes made by a specific operator, either enter the date in the Date field, prefixed with a greater-than operator (>) or a lesser-than operator (<) (for example, enter >**01/01/01**); or enter the name in the Operator field. Then run the search. If one of the resulting records should not be added to the Auditing delta unload, you can either change the search criteria of these records (such as changing the date to an earlier date), or delete the record

altogether if it does not need to be documented as changed. Be aware that the same record may be included in the `devaudit` file several times, since each change to a record is noted in `devaudit`.

Important: Before unloading a change you made during the development phase, check the audit files for the correct date of this delta. You need the date shown in the Audit History form to enter into the Unload form.

Enter the delta date, the unload filename, and log filename into the **Unload an Audit Delta** screen. Click **Proceed** to unload all modified records into the unload file.



Unload an Audit Delta

Unload delta since what date?

03/08/06 11:25:52

Send Data to Which File?

Enter Unload Name Here:

Enter Log Name Here:

Enter Log Name Here:

This unload can then be loaded into the test system, and eventually into the production system.

After the development phase is completed and all related changes have been transferred into the production system, you can purge the development audit data. You are prompted to specify the start date for records to be purged. After the data is purged, it cannot be restored (unless an unload of the `devaudit` file was performed beforehand).

Revision control

Revision control provides developers and administrators a tool for reverting to a previous version of a file or form. The Service Manager Revision Tracking utility allows a developer to

- Create a snapshot of a record,
- Add detailed information and comments to the snapshot, and
- Replace the current version of the record with a working version of the record at any time if you find an error while creating or modifying forms.

Note: Every revision uses as much disk space as the original record, plus a few bytes for comments.

Revision control does not replace the Auditing utility, but is used in conjunction with it to track, record, and save changes to your system. The Auditing utility provides a record of changes to ensure that you load the correct version when you move to production. Revision control documents these changes and enables you to create working snapshots.

Service Manager handles revisions as part of the Document Engine. Revisions are available in all utilities that use the Document Engine as base code, including Database Manager, Format Control, Link Editor, Forms Designer, and the RAD Editor.

The system stores revisions in a separate file whose name is specified in either the Object record for the file or in the `datadict` record. The system creates this revision file via an option on the Data Policy screen or the Object screen. Administrators specify the maximum number of revisions to store for each record in a file. If you do not specify a number, an unlimited number is stored.

Purge scripts help administrators with revision maintenance. The `sc.revision.purge.hanging` script purges all revisions that no longer have a parent record because it was deleted or renamed. The `sc.revision.purge` script purges all revisions from the system. To access these scripts, click

System Administration – Base System Administration - Miscellaneous, and then click either **Purge Hanging Revisions** or **Purge All Revision Records**.

Administrators need to determine in advance the files in which revisions are tracked, and then perform a few setup steps. They also need to purge revisions prior to migrating to a production system.

Unload Script utility

The Unload Script utility enables system administrators to create Service Manager unload files automatically. Unload scripts enhance the standard unload creation process in many ways. You use the Purge / Archive utility to unload and purge records from one file; and, if you use datamaps, you can use Purge / Archive to unload and purge related records as well.

With unload scripts you can save records from multiple non-related tables into a single unload file, and specify a query for each source table that filters the records added to the unload file. You can purge records during the unload process, and specify which formats to protect during a purge process. If records related to a source file should be archived or purged as well, you can add related records from the data map file to the unload file. As with the Purge / Archive utility, you can schedule the unload script to run as a background process.

The Unload Script utility is available in the Tailoring section. By default, Service Manager includes a collection of unload scripts that you can use for common unload tasks. You can also use the default unload scripts as templates to create your own customized unload scripts.

To create and use unload scripts effectively, you must be familiar with the Service Manager Database Manager. You can create a query on any field in the file, but querying keyed fields improves system performance and response time.

For example, the unload script that unloads all records necessary to move the Unload Script utility to another system would look as follows:

Unload Script: <input type="text" value="Unload Script Utility"/>		
<input checked="" type="checkbox"/> Unload?	<input type="checkbox"/> Purge?	<input type="checkbox"/> Show Protected Formats
<input checked="" type="checkbox"/> Show Unload Records		
Filename	Query	Datamap
RAD	application.name#"us.unload"	true
displayoption	screen.id="unload.script"	false
displayscreen	screen.id="unload.script"	false
format	name#"unload.qbe"	false
format	name="unload.script"	true
format	name="us.unload.get.name"	true
unload	name="Unload Script Utility" or name="SC 2000"	false

The following fields can be used in the Unload Script utility:

Field	Description
Unload Script	A name or label that identifies your unload script. This name can include spaces.
Unload?	Creates an external unload file from the records queried by this script.
Purge?	Deletes the records queried by this script.
Show Unload Records	Displays the Filename, Query, and Data Map fields, which enable you to specify the tables and queries to unload or purge records.
Filename	The name of the table used to unload or purge records. Visible only if you selected

Field	Description
	Show Unload Records.
Query	The SQL query that selects records for unloading or purging. You can use Service Manager operators and variables in this field. Visible only if you selected Show Unload Records.
Datamap	Boolean field that specifies whether Service Manager uses the data map file to unload or purge associated records. You must have previously defined a data map for the listed file in order to use this feature. By default, Service Manager reads a blank entry in this field as False. Visible only if you have selected the Show Unload Records option.
Show Protected Formats	Displays the Protected Formats field. Specify the forms to protect from purging.
Protected Formats	The names of the forms not to unload or purge when running this script.

To execute an unload script, click **Proceed**.

HP SERVICE MANAGER UNLOAD

This feature exports information from the unload script to an external file.

External File Name:

Append to File

When loading records into an existing dbdict

Use existing dbdict

Use dbdict of loaded record (replace old dbdict)

When loading records

Add new records and update existing records

Add new records only

Note: The Unload Script utility unloads tables using the binary unload file format.

The following information can be entered in this dialog window. Click **Proceed** to unload all records that meet the query criteria into the unload file.

Field	Description
External File Name	The name of the unload file that you want to create. This name can include path information, but must use characters that are valid for the target operating system. By default, Service Manager saves the unload file in <code>RUN</code> folder of the server. If users have enabled client-side unloading, then the Service Manager saves the unload file on their Windows® desktop.
Append to File?	Select this option to add records to an existing unload file.
When loading records into an existing database dictionary — Use existing database dictionary	Select this option to use the table and field definitions stored in the database dictionary of the target system.
When loading records into an existing database dictionary — Use database dictionary of loaded record	Select this option to use the table and field definitions stored in the unload file. This option will overwrite the existing database dictionary, if any, for the loaded file and records.
When loading records —	Select this option to update existing records in the target system with records

Field	Description
Add new records and update existing records	from the unload file.
When loading records — Add new records only	Select this option to ignore existing records and add only new records from the unload file.

Differential Upgrade utility

The Service Manager Differential Upgrade utility enables you to compare the specified files of two Service Manager systems to create a single unload file. The resulting unload file contains all the necessary records to make the files identical between the two systems.

The Differential Upgrade utility simplifies the way you move changes from a development environment into a test or production environment. You can also use this utility to move files between any two systems, such as between development and test, or between unit testing and acceptance testing.

Applying a Differential Upgrade from a development system to a production system requires the following steps:

1. Create a Patch record.
2. Create signatures for the production (target) system.
3. Move signatures to the development (source) system.
4. Create the Differential Upgrade from the development (source) system.
5. Load the Differential Upgrade onto the production (target) system.

The Differential Upgrade utility, just like the upgrade utility that upgrades applications from one version to another, relies on patch records and signatures. For more information on how to use the Differential Upgrade utility, please refer to the Service Manager online help topics under “Application Development - Differential Upgrade utility.”

Patch records

A patch record specifies which files the Differential Upgrade utility should compare, and creates a query that limits the comparison to certain records within files. It is very important to include all files that were modified during customizations that you promote to production.

It is also important that a query limit the number of records to upgrade, because the more files in the patch record, the longer the Differential Upgrade process takes. Even though Service Manager has default patch records, you may prefer to create customized patch records before you start the Differential Upgrade process. One method of limiting the number of records included in the patch record is to use the `sysmodtime` field that many `dbdicts` contain. If the file does not contain a `sysmodtime` field, you can either add it before starting the customization work on the development system (this field is filled automatically by the binaries with each update of the record); or use another limiting field such as `update.time`.

In a Differential Upgrade comparison, you must have one patch record for each system to be compared. Moreover, the patch record must point to identical files in each target system and generate the same queries.

In the patch record dialog, you can specify that the Differential Upgrade should perform Add Only processing on a specific file. If you choose this option, Service Manager adds only new records to the Differential Upgrade unload file and ignores changes to existing records in the development system.

There must be a patch record for each system in the comparison. Each patch record must point to the same files and records. You can either create it once on each system; or create it on one system, unload it, and load it into the other system.

An example patch record for moving a Service Management implementation from development to production could include the following:

Patch Name			ServiceManagement Implementation
Base Objects			
Application Clusters			false
Format Records			false
Ubdict	Add Only?	Query	
Object	<input type="checkbox"/>	file.name="incidents"	
Processes	<input type="checkbox"/>	process#"HP.cc"	
States	<input type="checkbox"/>	state#"HP.sm"	
activitytype	<input type="checkbox"/>	true	
capability	<input type="checkbox"/>	true	
datadict	<input type="checkbox"/>	name="incidents"	
dbdict	<input type="checkbox"/>	name="incidents"	
displayevent	<input type="checkbox"/>	screen.id#"cc"	
displayoption	<input type="checkbox"/>	screen.id#"cc"	
displayscreen	<input type="checkbox"/>	screen.id#"cc"	
environment	<input type="checkbox"/>	format.name="environment.sm"	
eventmap	<input type="checkbox"/>	evmap#"HP_service"	
eventregister	<input type="checkbox"/>	evtype#"HPsm"	
extaccess	<input type="checkbox"/>	service.name="ServiceDesk"	
format	<input type="checkbox"/>	name#"HP.SM"	
formatctrl	<input type="checkbox"/>	name#"HP.SM" or name="incidents"	
link	<input type="checkbox"/>	name#"HP.SM" or name="incidents"	
macro	<input type="checkbox"/>	filename="incidents"	
notification	<input type="checkbox"/>	name#"SM"	
number	<input type="checkbox"/>	name="incidents"	
querystored	<input type="checkbox"/>	filename="incidents"	
scripts	<input type="checkbox"/>	sysmodtime>'01/01/06'	
smenv	<input type="checkbox"/>	true	

Signature records

A signature for a Service Manager record is a numerical representation of the record. Any change to the contents of the record causes the signature of that record to change, based on the definitions in the signaturemake file

Important: Never change a record in the signaturemake file. You must create Signature records for any record that you compare in the Differential Upgrade process.

A sample Signature record would look like the following:

Type	Object
Name	incidents
Date	03/08/06 14:39:41
Version	SM Promotion
Type	Release
Signature	1507952360

To move Signature records to the development system, first ensure that the `signatures` file on the development system is empty. Then load the `signatures` file that was created on the production system into the development system using the standard Service Manager Import/Load utility.

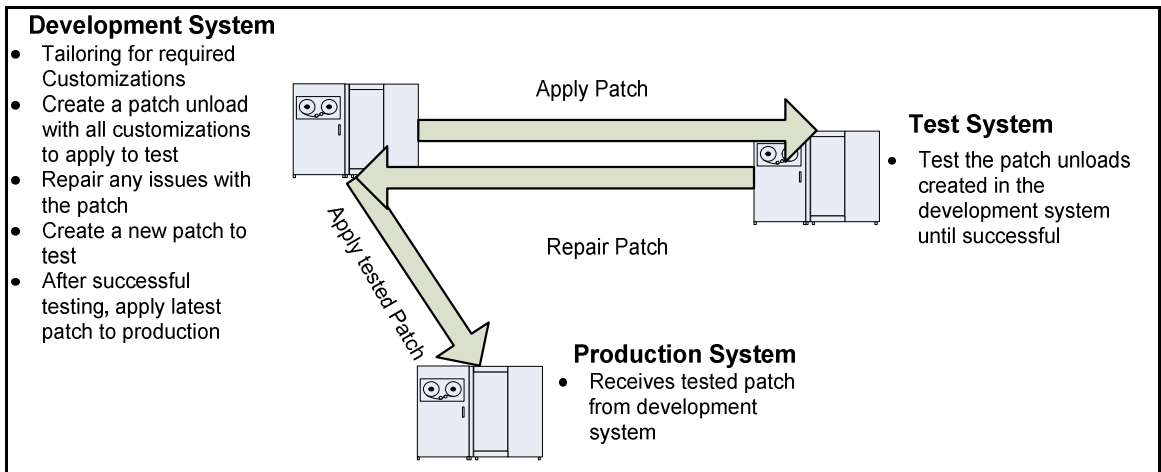
Manual Processing

Manual processing involves writing down each change as it is being made, and then unloading records one-by-one manually via the Service Manager Database Manager. HP strongly recommends that you document each change made when you tailor the system, regardless of how the promotion will be done. Manually unloading each record takes a lot of time and is more prone to errors.

Best Practice recommendations

Process

It is very important that you have a well defined and documented process for proceeding from development through test into production. A well defined process will promote user satisfaction and prevent delays in the development cycle. A process workflow may look like the following:



All changes are made in the development system. Making changes in the test system or directly to the production system will cause the systems to become out-of-sync and more difficult to maintain. Applying changes to a system that is out of sync with the system on which the changes were based will most likely result in broken functionality. After the customizations are completed on the development system, a single unload will be created to apply to the test system.

Important: If you create and apply multiple unloads, you increase the possibility that records can be overwritten or lost.

The test system is used to apply the previously created unloads and test their functionality. If problems are found during these tests, they can be fixed in the development system, where a new unload will be created. This cycle continues until the test is successful.

The tested unload can then be applied to the production system. Some companies require training before an unload is released to production. The tested unload can be applied to a training system first, and then to production. The benefit of having a single tested unload is that it can be applied to as many systems as necessary.

Part of the development process should include documenting all changes. Changes can be documented in an audit log that is stored outside of Service Manager, such as in an Excel® spreadsheet. Below is a short example of a possible audit log:

Change Number	1	
Request Description	Add company-specific fields to Incident Management.	
Task Number	1	2
Task Description	Add company-specific fields to the dbdict.	Add company-specific fields to the formats.
Change Requestor	Max Manager	Max Manager
Change Owner	Jennifer Falcon	Jennifer Falcon
Record modified	probsummary dbdict	IM.company.fields, IM.template.open, IM.template.update, IM.template.close, apm.quick
Date Modified	3/19/2006	3/19/2006
Change Description	Added company.name, company.location, company.contact, character fields and company.products character array to probsummary dbdict.	Added subformat IM.company.fields to all IM formats.
Unload Name	IM.company_dbdict.unl	IM.company.unl
Patch Name	IM_implementation.unl	IM_implementation.unl
Test plan	Enter company-specific information into all 4 formats and save.	Enter company-specific information into all 4 formats and save.

Another possibility is to use Service Manager Change Management for keeping track of all changes done during the development process.

Important: It is very important to keep track of every change done to the development system, so that no change is lost when creating the patch unload.

In summary, the following items are vital in any environment for successful customization of a Service Manager system:

- The process has to be defined and consistently applied.
- All changes have to be done in one system, the development system, and must be thoroughly documented.
- A single unload should contain all records that were customized. In some environments, many developers work on developing customizations. In such environments, a central system should be used to load the individual changes, using revision tracking. That central system can then be used to create the unload.

- All changes must be tested in a central system, the test system, and issues found there must be repaired in the development system.

Comparison of the tools to use for promoting customizations

Each method of promoting customizations has advantages and disadvantages. The following table compares what each method is capable of providing.

	<u>Development Auditing</u>	<u>Unload Script Utility</u>	<u>Differential Up-grade Utility</u>	<u>Manual Processing</u>
Can monitor all files?	No. Hard-coded list of tailoring files	Yes	Yes	Yes
Possible to specify a detailed query?	By date only	Yes	Yes	Yes
Can select and deselect records to unload?	Yes, by removing records from the devaudit file	No	Yes	No
Can do dbdict field merge?	No	No	Yes on fields, but not on keys	Yes, by changing the dbdict load option
Are changes to the record included in unload?	All	Latest	Latest	Latest
Performance Ranking, includes creating and applying patch	Poor	Good	Excellent	Fair
Ranking by ease of use	Good	Excellent	Fair	Poor
Ranking by ease to set up	Excellent	Good	Fair	Poor

HP best practice recommendation

Before you promote customizations from development to production, thoroughly document every change made in the development system that will need to be part of the patch that is promoted to production. The information that you collect in this documentation depends on the environment, but should at a minimum include the following fields, which are mentioned in the log file:

- Change Number
- Request Description
- Task Number
- Task Description
- Change Requestor
- Change Owner
- Record modified
- Date Modified
- Change Description
- Unload Name
- Patch Name

- Test plan

You can document changes either inside of Service Manager Change Management or by using an external program such as Microsoft Excel.

After all changes are documented, tested, and functioning in the development system, the next step is to create a single unload. HP recommends that you:

1. As a preparatory step, remove the field "keys" from the exclude list in the `signaturemake` file for the `dbdict` table, before you create the signatures on both production and development. That way, `dbdicts` whose keys were changed will be unloaded into the file that is loaded into test and then production, making these changes easier to find.
2. Use the Differential Upgrade utility to create a single unload file containing all changes.
3. Manually modify all keys that were changed in any of the `dbdicts` contained in the unload

For more information about the Differential Upgrade utility refer to the Service Manager online help topics under "Application Development – Differential Upgrade utility."

After the single unload is created, perform the following steps:

1. Apply it to a test system that is a recent copy of the production environment.
2. Test the changes thoroughly.
3. Document any reported issues and fix these in the development system after each test iteration.
4. Provide a repaired patch file (while still using a single unload file) to the testers at the beginning of each test iteration.

After the unload is thoroughly tested and accepted, the latest unload can be applied to a production system; or to any other system that needs to be upgraded with the patch, such as a training system. The patch should not be modified after this point, and new issues found after testing is complete should be addressed in the next development cycle.

For more information

Please visit the HP Management Software support Web site at:

<http://www.hp.com/managementsoftware/support>

This web site provides contact information and details about the products, services, and support that HP Management Software offers.

HP Management Software online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valuable support customer, you can benefit by being able to:

- Search for knowledge documents of interest
- Submit and track progress on support cases
- Submit enhancement requests online
- Download software patches
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Note: Most of the support areas require that you register as an HP Passport user and sign in. Many also require an active support contract.

To find more information about support access levels, go to the following URL:

http://www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to the following URL:

<http://www.managementsoftware.hp.com/passport-registration.html>

© 2008 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Service Manager is a registered trademark of Hewlett-Packard Development Company, L.P. Windows and Excel are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both. All other trademarks are the property of their respective owners.

07/24/2008

