

# Service Manager Logical Name Solution



## Table of contents

Introduction .....	4
Purposes of this document .....	4
Target audience .....	4
Background .....	5
Problem statement .....	5
Loss of CI data integrity .....	5
Inability to have duplicate CI names .....	6
Implications to the SM-UCMDB integration .....	8
Customer situations .....	9
Logical name solution .....	11
Overview .....	11
Design principles .....	12
Solution details .....	12
Reference field .....	12
Display Field .....	13
Client/Server protocol .....	14
Auto Complete .....	14
SM-UCMDB integration .....	16
Release Control integration .....	18

Support matrix .....	19
Upgrade process .....	20
Upgrade process for customers of type A .....	20
Step 1. Upgrade to Service Manager 9.41 .....	21
Step 2. Enable the logical name solution .....	21
Step 3. Copy logical.name data to the display.name field .....	21
Step 4. Map CI related Blob/Clob fields to an alias table .....	23
Step 5. Resolve data conflicts .....	31
Step 6. Enable relationships between the device table and the reference tables .....	31
Step 7. Convert logical.name values from a meaningful string to a sequential number .....	33
Step 8. Update the UCMDB integration .....	33
Step 9. Run a full push in UCMDB .....	39
Step 10. Update CI related reports .....	40
Service Manager reports .....	40
HP Executive Scorecard reports .....	40
Crystal Report reports .....	40
Other reporting solutions .....	40
Step 11. Update formats/links/format controls/JavaScript (including conflicts caused by the logical name solution) .....	41
Format .....	41
Format Control (Mandatory validation, Classic mode only) .....	41
RuleSet (Set Mandatory Fields, Codeless mode only) .....	41
Notifications and HTML templates .....	42
Link (Fill, View Detail/Find, Auto Complete, and new or updated RAD/JavaScript functions) .....	43
Fill .....	43
View Detailed Information/Find .....	45
Auto Complete .....	45
Append Query .....	46
RAD and JavaScript functions .....	47
General Issues and recommended solutions .....	66
Step 12. Update miscellaneous integrations .....	67
General guidelines for inbound integrations .....	67
Change the CI Identifier value to a sequential number .....	68
Modify the SM API to accept CI Display Name .....	68
Use the Service Manager Device API .....	69

General guidelines for outbound integrations .....	69
Changes required for specific integrations .....	70
Release Control (RC) integration .....	73
Application Lifecycle Management (ALM) integration .....	73
Operations Orchestration (OO) integration .....	74
HP Executive Scorecard integration .....	74
IDE metadata changes .....	75
Workflow changes .....	79
Upgrade process of customers of type B or C .....	83
Upgrade steps for customers of type B or C .....	84
Troubleshooting .....	85
Troubleshooting - search .....	85
Issue: Cannot search by CI display name .....	85
Troubleshooting - view/display .....	86
Issue 1: CI Identifier is displayed in detail forms .....	86
Issue 2: CI Identifier is displayed in virtual join forms .....	87
Troubleshooting - Fill/Auto Complete .....	87
Issue 1: Cannot find CIs matching an existing display name .....	88
Issue 2: Auto Complete does not work .....	89
Troubleshooting - Find/View Detailed Information .....	90
Issue: Customized “View Affected Services” functionality does not work .....	91
Troubleshooting - Service Catalog user selections .....	92
Issue: CI Identifier is displayed when the “Record in Table” type is used .....	92
Troubleshooting - field mandatory validation (Classic mode) .....	94
Issue: Incorrect mandatory field validation message .....	94
Troubleshooting - inbound integrations .....	96
Troubleshooting - outbound integrations .....	96
Issue 1: Cannot create a CI .....	96
Issue 2: Cannot create a non-CI record (Change, Incident, and so on) .....	96
Issue 3: Cannot retrieve additional CI information .....	97
Troubleshooting checklist .....	98
More information .....	99

# Introduction

The “logical name” problem has been a long-standing issue for ITSM solutions based on HP Service Manager. This issue is largely attributed to the historical design of the Service Manager data model, in which the `logical.name` field is specified as a unique key in the **device** table and it is also regarded as a meaningful CI name. Therefore, this design results in the following three major problems, which are detailed in ["Problem statement" on the next page](#).

- The loss of CI data integrity across modules in Service Manager
- The inability to have duplicate CI names in the Configuration module
- The complexity of CI reconciliation between HP Service Manager (SM) and Universal CMDB (UCMDB)

After a series of design reviews and validations with customers, HP has identified a solution to the logical name issue, with little impact on customer environments and limited effort required for upgrade. HP has implemented this solution in Service Manager 9.41.

## Purposes of this document

This document aims to help customers adopt the solution successfully after installing or upgrading to Service Manager 9.41. It provides the following information:

- The motivation for solving the problem and the solution implementation details
- The upgrade approach for existing customers, including information on how to update integrations that are impacted by the solution
- Troubleshooting information, which can assist customers in solving issues related to the solution

This document can help ensure that the solution not only works well for all business modules that consume CI data, but also works well for all integrations that exchange CI data between Service Manager and another HP application.

## Target audience

This document is intended for HP customers and partners who want to install or upgrade to Service Manager 9.41. This document can help such customers understand the solution and take necessary actions after adopting the solution.

# Background

This section further explains the problems and customer situations that the logical name solution aims to solve.

## Problem statement

The legacy design for the logical.name field of the **device** table in Service Manager created a long-standing issue, the so-called logical name issue. HP has observed the following facets to this issue, which apply to all versions of HP Service Manager until 9.40.

- ["Loss of CI data integrity" below](#)
- ["Inability to have duplicate CI names " on the next page](#)
- ["Implications to the SM-UCMDB integration " on page 8](#)

## Loss of CI data integrity

The first problem is the CI data integrity issue in the entire Service Manager system, which is largely considered as the biggest concern from a customer point of view.

The most common use case is that, when customers change a CI name in the **device** table, Service Manager does not cascade updates to the reference records (such as Incident, Problem, and Change records) because the logical.name value is saved in each reference record. As a result, CI data integrity is lost across modules in the Service Manager system.

The following is a list of typical scenarios where CI name changes are necessary.

- Very frequently, reorganizations or acquisitions of companies require CI name changes.
- Sometimes, when a CI is rolled out into the infrastructure, its name needs to be changed from an original state (which is specified during procurement) to a new state (which is much more recognizable for operation). This scenario usually involves Change Management, and relevant information must be passed to the CMDB team to update the CI name.
- Some customers like to suffix a CI name with a CI location. Therefore, when a CI is moved from one site

to another, its name needs to be changed.

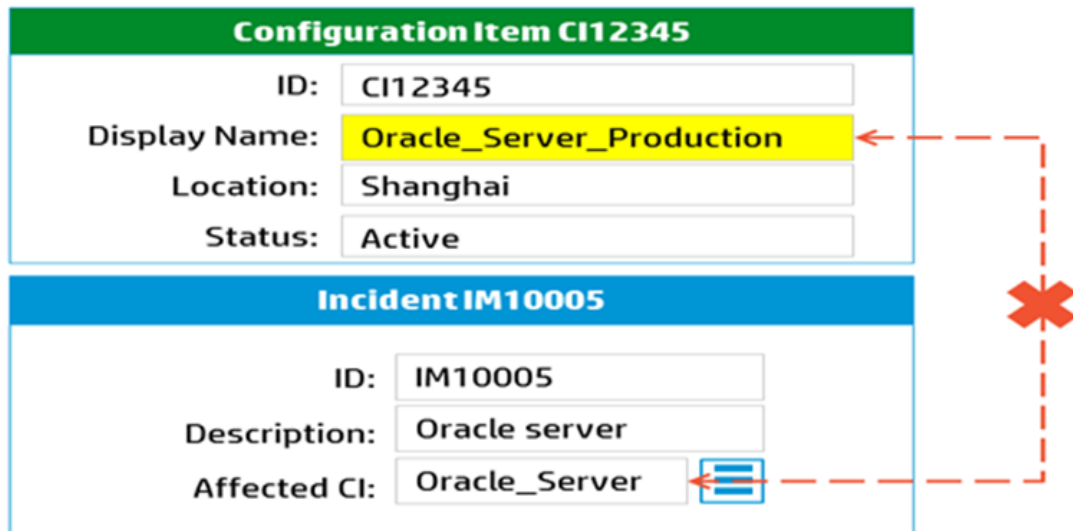
- Discrepancies discovered by audits are another trigger for CI name changes.

The following figure illustrates an example of the loss of CI data integrity. With the CI name being changed from Oracle\_Server to Oracle\_Server\_Production, the CI name on the incident form is still "Oracle\_Server". The potential consequences are as follows:

- From an ITIL process perspective, the linkage between the Configuration process and the other ITIL processes, such as the Incident process, Problem process, and Change process, is broken.
- Orphaned data is left in the system, which is useless and cannot serve business purposes.
- In some senses, this issue sometimes results in problems with configuration audit and history traceability.

Therefore, it is imperative that CI data integrity should be maintained across the entire Service Manager system.

**FIGURE 1-1: CI name is changed from Oracle\_Server to Oracle\_Server\_Production**



## Inability to have duplicate CI names

Another issue is that Service Manager does not allow duplicate CI names. This is because the logical.name field was designed as a unique key in the Service Manager **device** table. If an existing CI is not physically removed from the **device** table, no new CIs that have an identical name are allowed to be added to the

Configuration module. Practically, this issue blocks several typical scenarios for the use of duplicate CI names.

- Frequently, customers like to give an identical CI name for the same type of CIs in different locations. The following figure shows an example. In a global company, there are two Apache Tomcat instances in different locations: one is in San Francisco, and the other in Shanghai.

**FIGURE 1-2: Two CIs have the same name (“Apache Tomcat”)**

Configuration Item CI67890	
ID:	CI67890
CI Identifier :	Apache Tomcat
Location:	San Francisco
Status:	Active
Configuration Item CI12345	
ID:	CI12345
CI Identifier :	Apache Tomcat
Location:	Shanghai
Status:	Active



- Duplicate CI names may occur independently of their CI type. For example, an “Intranet” business application and an “Intranet” business service.
- In a multi-tenancy environment, one CI name might appear many times for various CIs.
- Customers like to reuse an existing CI name for a new device that is replacing a retired one, and this case happens on a regular basis as a consequence of asset lifecycles.

Per HP’s observation, currently a large number of customers are taking advantage of prefixes or suffixes in their CI naming convention to work around this issue. For example, customer name or company name is used as a prefix in CI names, while location or country is used as a suffix. However, even with such a naming convention, there still remains some inefficiency when duplicate CI names are handled. For example, a few integrations with external systems such as UCMDB allow CIs to have duplicate CI names. As a conclusion, having the ability for the Service Manager Configuration module to allow duplicate CI names is critical.

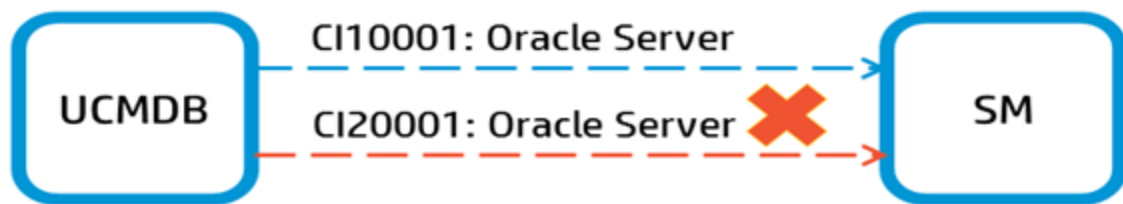
## Implications to the SM-UCMDB integration

The logical name issue also increases the complexity of the integration between Service Manager and Universal CMDB. This section summarizes the integration problems resulting from the legacy design of the `logical.name` field.

### No duplicate CI names are allowed to be pushed to SM

UCMDB allows CIs to have duplicate names; however, Service Manager does not. When CIs are being pushed from UCMDB to Service Manager, the inability to have duplicate CI names appears. As an example, the following figure illustrates that two Oracle servers are being pushed to Service Manager, and the second Oracle server cannot be created in Service Manager because the `logical.name` field is designed as a unique key of the **device** table. Although the out-of-box system allows a suffix to be added to CI names in Service Manager to handle this situation, it consequently brings in redundant information and increases operation cost on the customer side.

**FIGURE 1-3: Two Oracle Server instances are being pushed from UCMDB to SM**



### Data inconsistency between Service Manager and UCMDB

Data inconsistency between Service Manager and UCMDB is another impact of the logical name issue. This usually occurs after data is exchanged reciprocally between the two applications, through either push or population. The main reasons are as follows:

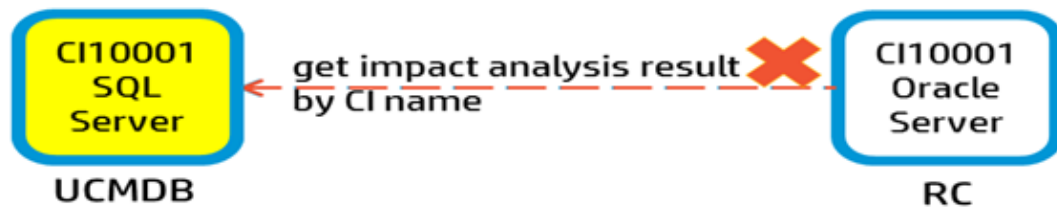
- The `logical.name` field is often used as one of the factors in the rule for CI reconciliation between Service Manager and UCMDB.
- The value of the `logical.name` field in Service Manager could be changed by appending a suffix when the “duplicate CI names” scenario appears as mentioned previously; however, the value of the mapping field `display_label` in UCMDB remains as is. Therefore, the reconciliation turns out to be a real problem.



## Failure to get impact analysis result in Release Control

Another relevant case is the Release Control integration. The logical.name field is defined as the search-by field for Release Control to get the impact analysis result from UCMDB. However, if a CI name is changed in UCMDB, it is no longer possible for Release Control to get the correct impact analysis result. The following figure shows an example: once the CI name is changed from Oracle Server to SQL Server in UCMDB, the impact analysis result for Oracle Server in Release Control becomes incorrect. Therefore, the Release Control behavior of retrieving impact assessment based on the logical.name field longer works.

**FIGURE 1-4: Oracle Server in RC is using CI name to retrieve impact analysis result from UCMDB**



## Customer situations

In designing the logical name solution, HP reviewed a large number of customer situations, to ensure that the solution can adapt to the broad spectrum of individual customer cases.

A customer survey has revealed that there are three typical ways in which the logical name issue has been circumvented.

- One group of customers is leveraging a CI naming convention to avoid duplicating CI names. However, this approach cannot solve the CI data integrity issue.

**Note:** This group of customers are referred to as "type A" customers in this document.

- Another group of customers implemented an official workaround provided by HP through the [Temporary workaround of CI Logical Name and Display Name - storing redundant data](#) white paper. This workaround introduces a display name field to all relevant modules (not only the Configuration module, but also the other reference modules such as Incident, Problem, and Change). Additionally, the required database triggers are created, so that when a CI name is changed in the Configuration module, the corresponding database triggers are activated to cascade the updates to the reference records. However, this workaround brings in redundant display name fields in the entire system, and implies a considerable implementation cost. In addition, this workaround does not lead to a relational

database. Considering all this, HP has not implemented the workaround in the out-of-box system.

**Note:** This group of customers are referred to as "type B" customers in this document.

- Other customers implemented their own solution by applying customizations on top of the HP workaround described previously; however, the same limitations and shortcomings remain.

**Note:** This group of customers are referred to as "type C" customers in this document.

# Logical name solution

To help you adopt the optimal logical name solution successfully, this section describes the solution details.

Overview .....	11
Design principles .....	12
Solution details .....	12

## Overview

HP introduced the logical name solution in the Service Manager 9.41 release to completely solve the long-standing issue. The main idea of this new solution is to apply a relational data model mechanism for both the **device** table and the reference tables (probsummary, cm3r, and so on), so as to not only keep the CI data integrity across modules in Service Manager but also remove the need for redundant data in the entire system. As a consequence, once a CI name is changed, the reference records (such as Change and Incident records) can reflect the change and display the correct CI name immediately.

The following provides a high-level description of the solution.

- In the **device** table, the logical.name field is specified as a unique key with a sequential number (for example, CI00001), and the logical name value cannot be changed after being assigned; additionally, administrators can configure the Configuration environment record to allow users to manually assign a CI Identifier when they create a new CI.
- In the **device** table, a new display.name field is added to display CI names. This field is not a unique key so that it can allow duplicate CI names.
- In the reference tables, the logical.name field is used as a reference field referring to the **device** table; the display.name field is not copied to the reference tables.
- On the forms of the reference modules, the CI name field is populated with the value of the display.name field from the **device** table, and the value of the logical.name field is not shown.
- In the user interface, an enhanced Auto Complete function is provided to improve the user experience for selecting CIs (which can have duplicate display names now).

- For the SM-UCMDB integration, the UCMDB display\_label field is mapped to the SM CI display.name field for data synchronization.
- For the Release Control integration, the ucldb.id (global.id) field instead of the logical name field is used to retrieve appropriate impact assessment from UCMDB.

## Design principles

The following design principles are applied during the development of this solution.

- Solve the data integrity issue by using a relational database model
- Minimize customer impact during upgrade
- Allow extensibility for other master data, for example, assignment, and company
- Optimize the maintenance mechanism for the SM-UCMDB integration

## Solution details

This section describes the technical details of the logical name solution, which mainly focuses on two aspects:

- The changes that have been made in the system to completely address the issues mentioned previously
- HP's recommendation on how administrators and users should use the solution

## Reference field

The logical name solution leverages the logical.name field as a reference identifier to set up a relationship between the **device** table and the reference tables.

The following figure shows an example of the Incident module. In the data policy, the “affected.item” field of the **probsummary** table is a reference field referring to the **device** table, which keeps the close relationship between the Incident module and the Configuration module.

**Note:** The **device** table is referred to as the “referenced table” of the reference fields in the reference modules (Incident, Change, and so on).

**FIGURE 2-1: The “affected.item” field in the probsummary table is defined as a reference identifier**

**Data Policy**

Name:

\* probsummary

SQL Base Name:

\* probsummary

Unique Key:

number

Default Format:

probsummary

☐ Prohibit Default Access

☐ System Table

☐ Enable Data Reference Check

Description:

Incident Database

Field Settings

Validations

Field Name	Caption	Field Type	Referenced Table	Usage Type
adj.resolution.time	Adj Resolution			Data
affected	Affected			System
affected.item	Affected Servi		device	Application

**Supported widgets for reference fields**

The following widgets are supported for a reference field:

- Web client: Table (both QBE and detail form), Comfill (including Auto Complete), "Select" type input in a dynamic form, Label, and Text (read-only)
- Windows client: QBE table, table on detail form, and Comfill (not including Auto Complete), Label, and Text (read-only)
- Mobility client: Table, Comfill (including Auto Complete) , Label, and Text (read-only)

**Display Field**

In the out-of-box system, Display Field is set to the display.name field of the **device** table in the data policy. Note that if customers have added a custom field to store CI names, they are able to set Display

Field to that custom field in the data policy; moreover, Display Field must be of the character type, and must have a "No Nulls" index and no " Unique" index.

**FIGURE 2-2: Display Field in the data policy for the device table**

Data Policy

Name:

\*

device

SQL Base Name:

\*

device2

Unique Key:

logical.name

Display Field:

\*

Configuration Item Name - display.name

Default Format:

☐ Prohibit Default Access

☐ System Table

☒ Enable Data Reference Check

Description:

## Client/Server protocol

In addition to the relational data model, a special protocol between the SM client and the SM server is also applied so that the CI names are displayed appropriately in the user interface. The client is able to retrieve both the CI Identifier and display name.

The following table lists the reference field, display field, database value, and display value of a sample CI (CI10005).

Reference field	Display field	CI DB value in the probsummary table	CI display value on the Incident form
affected.item	display.name	CI10005	Applications

## Auto Complete

To differentiate CIs with duplicate names, the Auto Complete feature is enhanced in the system. The following figure shows how the Auto Complete feature displays CI details to the user.

**FIGURE 2-3: Auto Complete for Affected Service in the Incident module**

Affected Service	* CHAR				
Affected CI	Display Name	CI Identifier	Environment	Type	Config Ad...
	CHAR-Applications	Applications		Business S...	Application
	CHAR-Education	Education		Business S...	Service De...
	CHAR-E-mail / Webmail (Africa)	E-mail / W...		Business S...	E-mail / W...
Outage Start	CHAR-E-mail / Webmail (Asia)	E-mail / W...		Business S...	E-mail / W...
Outage End	CHAR-E-mail / Webmail (Austr...	E-mail / W...		Business S...	E-mail / W...
Service Contract	CHAR-E-mail / Webmail (Euro...	E-mail / W...		Business S...	E-mail / W...
Next SLA Expiration	CHAR-E-mail / Webmail (Nort...	E-mail / W...		Business S...	E-mail / W...
	CHAR-E-mail / Webmail (Sout...	E-mail / W...		Business S...	E-mail / W...

**Note:** Administrators can customize the list of CI fields to display for Auto Complete in the **Auto Complete Table Columns** tab in the data policy for the **device** table. See the following figure for an example. If no attributes are configured on the **Auto Complete Table Columns** tab, the system will show the Display Field by default.

**FIGURE 2-4: Auto Complete Table Columns for CI selection**

Name:

\* device

SQL Base Name:

\* device2

Unique Key:

logical.name

Display Field:

\* Configuration Item Name - display.na

Default Format:

☐ Prohibit Default Access

☐ System Table

☒ Enable Data Reference Check

Description:

Applications

IR Specifications

Manage Queues

Auto Complete Table Columns

Columns to display in the Auto Complete list:

Display Name - sm.device.display.name

CI Identifier - logicalName

Environment - environment

Type - type

Auto Complete appends an additional query to the query defined in the link:

- If the field has a referenced table defined, the additional query is “REFERENCED TABLE DISPLAY FIELD # WHAT USER INPUTS”.

Page 15 of 99

- If the field has no referenced table defined, the additional query is “REFERENCED TABLE Target Field # WHAT USER INPUTS”.

**Note:** The additional query is forcibly added to the link query. Therefore, if another condition is configured regarding what the user has typed in the Comfill box, unexpected search results are returned. However, the user can still use the search form to locate a matching record by using other fields as well as the display name field.

### Limitation

By default, Auto Complete is not supported for variable fields that are linked to the **device** table. For example, the Affected CIs field in the `pbm.problem.subform.affectedci.qbe` subform is such a variable field (Input=\$affected.cis). Additionally, Fill does not work well for such variable fields either.

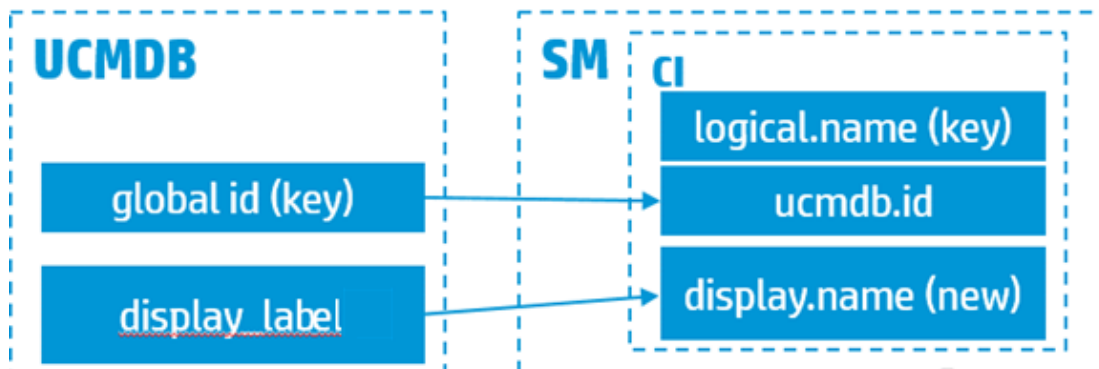
## SM-UCMDB integration

Since the `display.name` field replaces the `logical.name` field in the **device** table to represent CI names, CIs created from UCMDB can have duplicate CI names and be pushed to Service Manager without the need to be renamed.

For CI types that cannot be discovered in UCMDB, you may want to create CIs in Service Manager and then synchronize them to UCMDB through population. Keep in mind that CIs created from the Service Manager side must have a unique CI name so that UCMDB does not regard them as the same CI and hence does not merge them during population.

Also, the out-of-box field mapping is updated in the SM-UCMDB integration. That is, the UCMDB `display_label` field is now mapped to the SM `display.name` field, as shown in the following figure.

**FIGURE 2-5: Field mapping between SM and UCMDB**





Take the Switch CI type as an example. Previously, a Switch CI's name was concatenated from the names of the attributes (Interface, MACAddress, and so on) and mapped to the CIIdentifier field in Service Manager. Now, the mapping is simplified. See the following table.

Does the CI have interfaces?	Old mapping (CIIdentifier)	New mapping	Mapping change
Yes	{interface1["MACAddress"]_*node["Name"]}.  Example: AC-03-05-00-8B-8D_BC-03-05-00-8B-8D_NODENAME	Node.display_label	<ul style="list-style-type: none"> <li>CIIdentifier now removed</li> <li>display_label now mapped to DisplayName</li> </ul>
No	node["Name"]  Example: NODENAME	Node.display_label	

### Disabling duplicate CI name validation if you are not using UCMDB

If you are not using the SM-UCMDB integration and are managing CIs only in Service Manager, you need to manually disable the duplicate display name validation by removing the following expression from the “device” formatctrl record:

```
jscall("DisplayName.checkDuplicatedDisplayName", $file)
```

To do so, follow these steps:

1. Click **Tailoring > Format Control**.
2. Open the **device** record, and remove the validation expression and message from the **Validations**

tab. See the following figure.

Validation	<input type="checkbox"/> Use pop-up validation messages
Message	<input type="checkbox"/> Display all validation messages

Select a folder for this Interaction.
not (null(istatus in \$file))
Please provide a Status.
not (null(assignment in \$file))
Please provide a Config Admin Group.
not (null(name in \$file1))
Invalid Assignment Group
<b>jscall("DisplayName.checkDuplicatedDisplayName", \$file)</b>
<b>Duplicate display name in current CI type.</b>
jscall("DisplayName.checkNotContainQuotIdentifier", logical.name in \$file)
CI Identifier can't contain double quotes.

## Release Control integration

In the entire ecosystem consisting of Service Manager (SM), UCMDB, and Release Control (RC), SM passes the global.id value from UCMDB to RC. Since the global.id field is regarded as a unique CI identifier, as illustrated in the following figure, it is now used to get the correct impact analysis result from UCMDB.

**FIGURE 2-6: Oracle Server in RC is using Global ID to retrieve impact analysis result from UCMDB**



# Support matrix

The following is a list of required product versions to enable the full logical name solution.

- Service Manager – version 9.41 or later (including the Applications, Server, Windows client, Web client, Mobility client, and Service Request Catalog (SRC))
- UCMDB – all 10.x releases
- Release Control (RC) – RC 9.21 patch 3

# Upgrade process

HP is making every effort to not only simplify the upgrade procedure but also provide a seamless upgrade path, covering most of typical customer situations. This section describes the typical upgrade scenarios, which are categorized by customer situation.

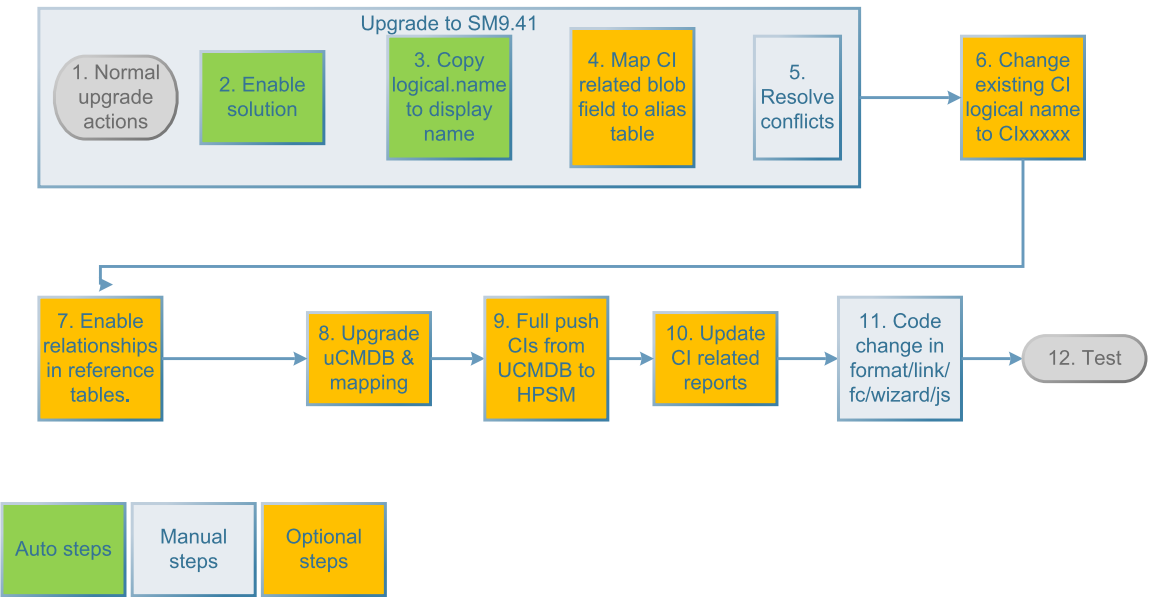
Upgrade process for customers of type A .....20

Upgrade process of customers of type B or C .....83

## Upgrade process for customers of type A

For customers of type A, who are using a CI naming convention to address the logical name issue, the upgrade steps are illustrated in the following figure.

**FIGURE 3-1: Upgrade process for customers of type A (using a CI naming convention)**



Pay attention to the following points:

- In the “Code Change in link/fc/wizard/js” upgrade activity, customers should pay close attention to potential code changes in link/fc/wizard/js, where a few references to the logical.name field most likely need to be changed to the display.name field according to business needs.

- To minimize the downtime resulting from the “Map CI related blob field to alias table” upgrade activity, HP provides instructions in ["Step 4. Map CI related Blob/Clob fields to an alias table" on page 23.](#)

## Step 1. Upgrade to Service Manager 9.41

Run the Service Manager 9.41 Upgrade Utility or Applications Patch Manager to upgrade to Service Manager 9.41 (SM 9.41).

For details, see the Service Manager 9.41 *Applications Upgrade Guide* for your specific Service Manager version or the *Applications Patch Manager Guide*.

## Step 2. Enable the logical name solution

This solution is automatically enabled and cannot be disabled once you have upgraded to the SM 9.41 applications. To use this solution, you need to upgrade both the Service Manager applications and binaries (server and clients) to version 9.41. If the applications version is 9.40 or earlier, this solution is not available.

The solution is enabled in the out-of-box SM 9.41 system through the following implementations:

- Relationships are created between each reference module (Incident, Change, Problem, and so on) and the Configuration module by setting the **device** table as the referenced table for the CI related reference fields (affected.item in cm3r, for example)
- In the data policy of the **device** table, the Display Field is set to the **display.name** field of the **device** table.

## Step 3. Copy logical.name data to the display.name field

During your upgrade to the 9.41 applications, the system automatically copies data from the logical.name field to the new display.name field in the **device** table, using the following query:

```
update DEVICE2M1 set DISPLAY_NAME=LOGICAL_NAME where DISPLAY_NAME is NULL
```

If your system already has a field named display.name defined in the **device** table and the field has only Null values, data is copied from the logical.name field to display.name; otherwise no data is copied. This is to avoid overwriting your existing data in the display.name field.

**Note:** After the Display Field value is updated in the device data policy, the system adds an alias named “sm.device.display.name” to the specified display field automatically. If the alias is removed manually, related functions will not work correctly.

### Step 3.1 Set your own display name field as the display field

If your production system already has a CI display name field defined in the **device** table, you can choose to set your own display name field as the display field after the applications upgrade. To do so, follow these steps:

1. Execute the **datadict** command in the Service Manager command line.
2. Search for the **device** data policy record.
3. Select your own CI display name field from the Display Field drop-down list.

**Note:** Only character fields with a No Nulls index appear in the drop-down list and therefore can be set as the display field. The display field cannot have a unique index.

Name: device  
 SQL Base Name: device2  
 Unique Key: logical.name  
 Display Field: Configuration Item Name - display.name  
 Default Format:  
☐ Prohibit Default Access ☐ System Table

If your own CI display name field allows a null value, SM will prevent you from adding a No Nulls index to the field. In that case, if you still want to use your own CI display name field, you need to update all the null values first, and then add the index to the field. You need to search for device records whose display name is null and then mass update these records to assign a display name to them.

- If the number of such records is more than 5000, a mass update will take considerable time depending on your system capability. In this case, you can refer to the **batchUpdateCIDisplayNameByLogicalName** method in the DisplayName script in Script Library, and then write a similar script to update the records instead of performing a mass

update.

- Sometimes a batch update will fail if the size of your own display name field is smaller than that of the logical.name field. You need to change the field size to the size of the logical.name field.

To set your own display name field ("ci.display.name" for example) through a mass update, follow these steps:

- a. Search for CIs whose ci.display.name value is empty.

A list of CI records is displayed.

- b. Click **Mass Update**.
- c. Click **Complex Update**.
- d. Set the value to be same as the logical name field.

For example: `ci.display.name in $file=logical.name in $file`

Alternatively, you can set the display name value by using a RAD expression.

- e. Add a No Nulls index to the ci.display.name field.
- f. Go to the datadict for the **device** table, and select **ci.display.name** as the Display Field.

4. Save the data policy record.

**Note:** After you update the display field value, the system automatically adds an alias named "sm.device.display.name" to the new display field, and remove the alias "sm.device.display.name" from the previous display field. Keep in mind that if you manually remove the alias, CI related functions will not work correctly.

## Step 4. Map CI related Blob/Clob fields to an alias table

**Note:** This is an optional step after you upgrade to Service Manager 9.41.

In the out-of-box Service Manager 9.41 system, some CI related fields are Blob fields. The following table provides a list of these fields. After you upgrade to SM9.41, these fields have a referenced table defined and are not mapped to an alias table. Your production system may also contain other custom CI related fields that are Blob/Clob fields.

Table	Field
cm3r	affected.services
cm3r	assets
cm3t	asset
probsummary	affected.services
requestTask	CIListCMDB
requestTask	CIListContext
rootcause	affected.ci

After upgrading to Service Manager 9.41, if you do not map CI related Blob/Clob fields to an alias table, the search behavior for such fields is as follows:

- When you select a CI in the field (for example, in the **assets** field in Change) by clicking Fill or through auto-complete and then click **Search**, the search returns the correct result. This is because the CI Identifier value can be retrieved when you click Fill or select an auto-complete entry and hence the CI record can be located by CI Identifier.
- When you manually enter a partial or full CI display name in the field and then click **Search**, the search fails with the following error:

“The join condition \" ta01.asset = ta02.logical.name \" cannot be converted into an SQL expression. One or more fields might be a Blob/Clob/Text/Image field.”

To solve this problem, you need to map the field to an alias table.

Mapping fields to alias tables and moving the data to the alias tables may result in long system downtime. To minimize system downtime, you can implement the dbdict changes in two separate phases:

- **Preparation phase:** In this phase, you create new alias tables and copy the data to them, but also keep the old structure. The application still uses the old structure, and data is kept in sync by triggers. You can do this some days before you make the real dbdict changes, without causing system downtime.



- **Change execution phase:** In this phase, you rename the fields to force the application to use the new structure and drop the triggers. This needs a short downtime for renaming of the fields.

The steps of mapping the fields to an alias table are described in the following, using the assets field in the cm3r dbdict as an example. Use similar steps for other fields and tables.

### Preparation phase

1. Log in to the database.

For example, if it is an Oracle database, log in to the database with SQL\*Plus or with Oracle SQL Developer.

2. Create a restore point in the database so that the database can easily recover from an error.

For example, before the dbdict is changed, set a restore point in an Oracle database using the following command:

```
CREATE RESTORE POINT before_change;
```

In case of an error while changing the dbdict, you can restore the Oracle database using the following command:

```
FLASHBACK DATABASE TO RESTORE POINT before_change;
```

**Note:** You must have SYSDBA permissions to perform this step.

3. Create new alias tables for the dbdict records.

The following steps use the assets field in cm3r as an example.

- a. Add new array assets.new to structure "middle".

The screenshot shows a window titled "field.window" with a "Tools" bar containing "Cancel" and "Add" buttons. Below the bar is a toolbar with various icons, including the HP logo. The main area contains a form with the following fields:

<b>Name:</b>	assets.new	<b>Structure:</b>	middle
<b>Type:</b>	array	<b>Level:</b>	2
		<b>Index:</b>	103

At the bottom of the form are two buttons: a blue square button with a white plus sign, and a blue square button with a white 'X' inside a circle.

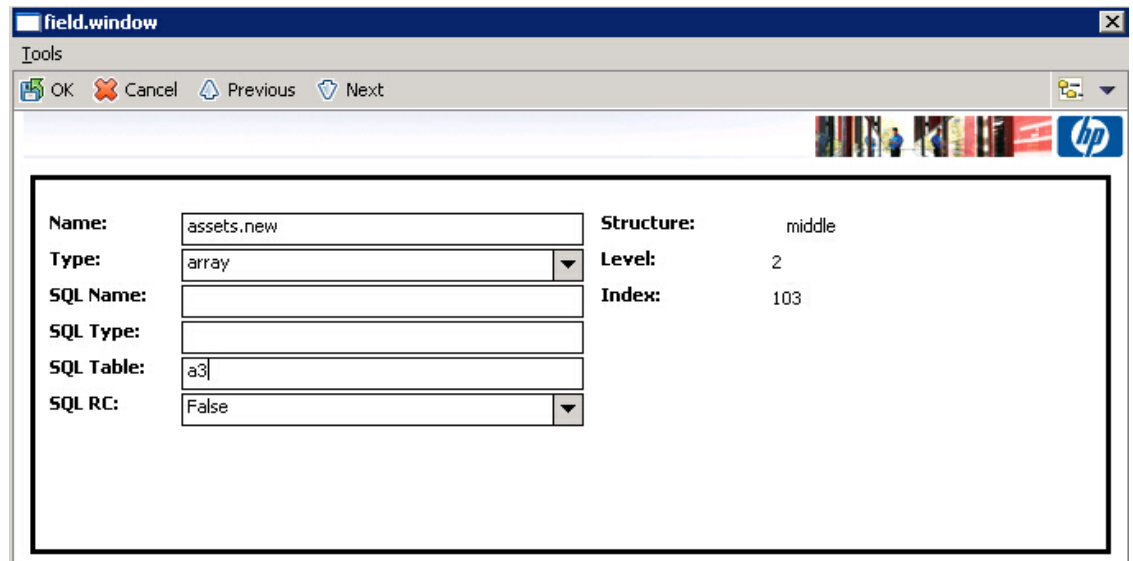
- b. Add array element assets.new with the character data type.

The screenshot shows the same "field.window" window. The "Tools" bar now only contains the "Add" button. A yellow warning icon and the text "Enter data type of array's element." are displayed above the main form area. The form fields are updated as follows:

<b>Name:</b>	assets.new	<b>Structure:</b>	middle
<b>Type:</b>	character	<b>Level:</b>	3
		<b>Index:</b>	1

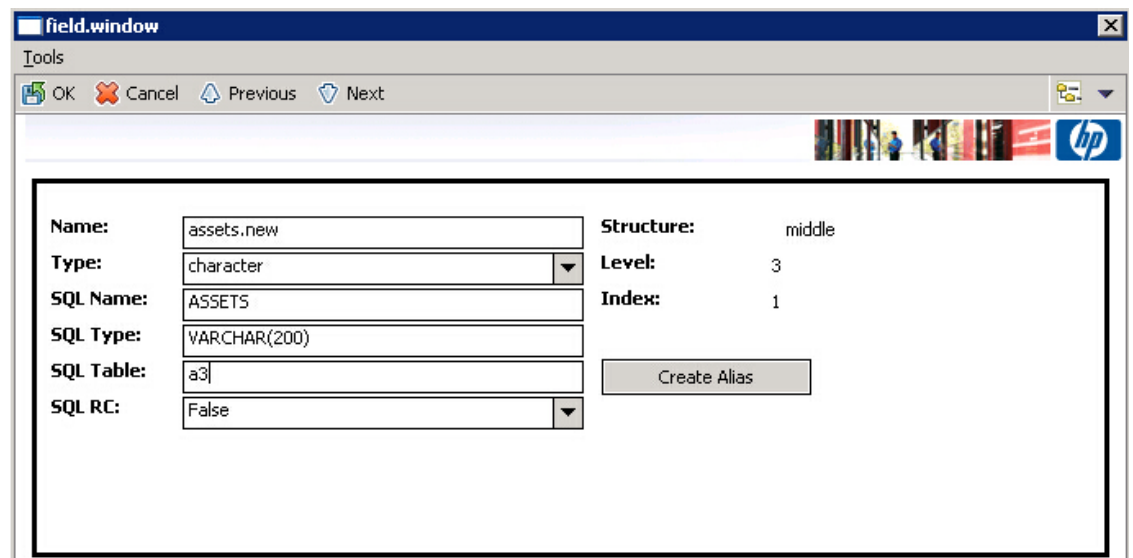
At the bottom of the form are two buttons: a blue square button with a white plus sign, and a blue square button with a white 'X' inside a circle.

- c. Map the array element as field ASSETS VARCHAR(200) to an alias table (for example, **CM3RA3**).



The screenshot shows the 'field.window' dialog box with the following configuration:

<b>Name:</b>	assets.new	<b>Structure:</b>	middle
<b>Type:</b>	array	<b>Level:</b>	2
<b>SQL Name:</b>		<b>Index:</b>	103
<b>SQL Type:</b>			
<b>SQL Table:</b>	a3		
<b>SQL RC:</b>	False		



The screenshot shows the 'field.window' dialog box with the following configuration:

<b>Name:</b>	assets.new	<b>Structure:</b>	middle
<b>Type:</b>	character	<b>Level:</b>	3
<b>SQL Name:</b>	ASSETS	<b>Index:</b>	1
<b>SQL Type:</b>	VARCHAR(200)		
<b>SQL Table:</b>	a3		
<b>SQL RC:</b>	False		

A 'Create Alias' button is visible on the right side of the dialog box.

- d. Save the changes.

Service Manager creates the new alias table CM3RA3.

Similarly, create additional alias tables for other fields.

4. Validate the structure changes. To do this, check that the new alias tables are present and have the

desired structure.

For example, for an Oracle database, run the following SQL command to check the CM3RA3 table:

```
DESCRIBE CM3RA3;
```

##### 5. Create triggers for subsequent adds and updates.

The following screenshots illustrate the triggers for synchronizing added or updated data to the new field `assets.new` in `cm3r`.

**Note:** If the `dbdict` contains more Lob fields, you need to add more lines for these fields to the script of each trigger.

Create the “before.change.add.assets” trigger on the `cm3r` table.

Trigger Name:	before.change.add.assets
Table Name:	cm3r
Trigger Type:	1 - Before Add ▼
Application:	
Script:	<pre>1 record.assets_new = record.assets;</pre>

Create the “before.change.update assets” trigger on the `cm3r` table.

Trigger Name:	before.change.update.assets
Table Name:	cm3r
Trigger Type:	3 - Before Update ▼
Application:	
Script:	<pre>1 record.assets_new = record.assets;</pre>

##### 6. Execute a script for each DBDICT record to copy data to the alias tables of the DBDICT record. For each DBDICT, execute the script only once to copy all data. The following describes how to run a script to copy `cm3r` data to the alias tables (`CM3RA3` and so on).

- a. Open Script Library. For example, type `sl` in the Service Manager command line, and press Enter.
- b. Create the following script:

**Name:** Provide a script name ( for example: `CopyCm3rAssetsToAliasTable`)

**Package:** Provide a package name (for example, `Configuration Management`)

- c. Compile and execute a script to copy the current data to the new alias tables.

```
/*

Copy current content of field cm3r.assets and other fields
to new fields mapped to alias tables. Has to be executed once.
Subsequent data changes are handled by the add and update triggers.

CM3RM1.ASSETS      -> CM3RA3.ASSETS_NEW
If there are more Blob/Clob fields in cm3r, insert more lines below the
"filevar.assets_new = filevar.assets;" line, using the same syntax.
*/
var ret = 0;
var filevar = new SCFile("cm3r");
var rc = filevar.doSelect("true");
var count = 0;
var start = new Date();
var _rtecall=system.functions.rtecall;
while (rc == RC_SUCCESS) {
    filevar.assets_new = filevar.assets;
    if (count % 100 === 0) {
        print("Updated : " + count);
    }
    _rtecall("trigger", ret, 0);
    filevar.doUpdate();
    _rtecall("trigger", ret, 1);
    rc = filevar.getNext();
    count++;
}
var end = new Date();
print("Duration [msec]: " + (end - start) + " for " + count + " change
records");
```

### Change execution phase

In this phase, you rename the new and original fields to force the application to use the new structure. In this phase, Service Manager must run in single user mode (for example, configure a servlet with a “secret” port to prevent other users from logging in).

Before you proceed to this phase, make sure you have saved the corresponding dbdict records or have created a database restore point. The following steps use the cm3r dbdict as an example.

1. Perform cm3r structure changes.
  - a. Open Database Dictionary. For example, type `dbdict` in the Service Manager command line and press Enter.
  - b. Search for `=cm3r`.
  - c. Rename `cm3r.assets` to `cm3r.assets.old`.
  - d. Rename `cm3r.assets.new` to `cm3r.assets`.
  - e. (Optional) Add key (Nulls & Duplicates) on the `assets` field.
  - f. Repeat the steps for other fields.
  - g. Save the `cm3r dbdict`.
2. Change the triggers.

You change the triggers to keep the data up to date in the original fields in the master tables. You need to keep the triggers until all reporting users have migrated their reporting jobs to the new structure.

```
-- -----
-- cm3r Before Add Trigger 'before.change.add.assets'
-- -----
Trigger Name: before.change.add.assets
Table Name:   cm3r
Trigger Type: 1 - Before Add
Application:  <empty>
Script:
record.assets_old = record.assets;

-- -----
-- cm3r Before Update Trigger 'before.change.update.assets'
-- -----
Trigger Name: before.change.update.assets
Table Name:   cm3r
Trigger Type: 3 - Before Update
Application:  <empty>
Script:
record.assets_old = record.assets;
```

3. Roll back the changes if necessary.

If backout is needed, the fastest way is to restore to the database restore point you previously set.

## Step 5. Resolve data conflicts

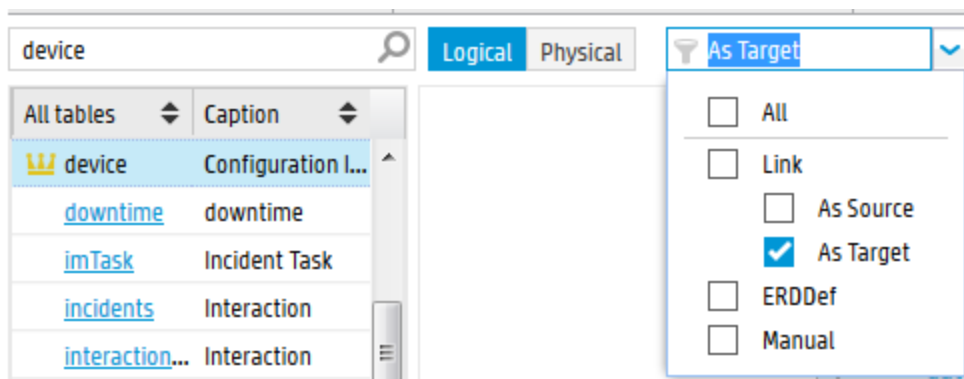
During the upgrade, resolve the data conflicts as described in the Service Manager 9.41 *Applications Upgrade Guide* or *Applications Patch Manager Guide*.

## Step 6. Enable relationships between the device table and the reference tables

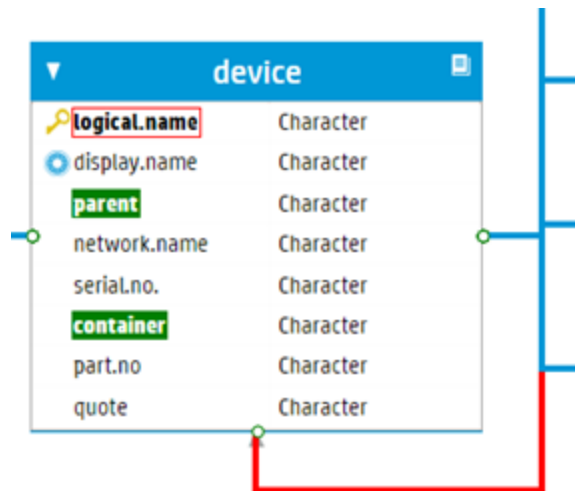
**Note:** This is an optional step.

Reference tables refer to tables that reference the **device** table, such as the **probsummary** and **cm3r** tables. In the out-of-box system, relationships between the **device** table and reference tables (such as relationships from probsummary to device, and from cm3r to device) are already created. However, if you have some new tables that need to reference the **device** table, you need to perform the following steps.

1. Go to **Tailoring > SQL Utilities > Entity Relationship Diagram**.
2. Enter **device** in the search box , and open **Configuration Item-Device**.
3. Specify the filter as “Link As Target”.

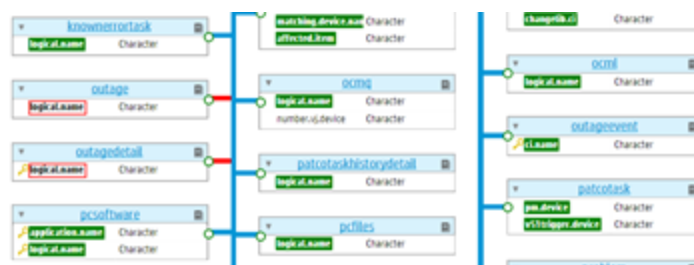


4. Locate the **device** table in the diagram, and click the key (logical.name).



5. Locate the red connector lines between **device** and other tables, and fix the problems.

See the following figure for an example.



In this example, the logical.name fields highlighted for the **outage** table and the **outagedetail** table indicate that the each of them has a link with the **device** table but has not the **device** table enabled as its reference. You need to make sure it is set correctly.

You need to set the reference as follows:

- Click the book icon to open the datadict of the reference table.
- Search for **logical.name** and set the reference as **device**.

**Note:** Service Manager does not support setting a reference for the following types of fields:



- Variable
- Fields in an array of structure. For example, matching.device.name of the knownerror table (no workaround is available for the SM 9.41 release)

The reference field must be of the same type as the field in the primary key (or the first unique key if there is no primary key) of the referenced table. The primary key or the first unique key of a referenced table must be composed of a single field. If the key is combined by multiple fields, the reference configuration fails. Additionally, if the reference field is an array, the type of the element in the array must also match the type of the primary key or first unique key of the referenced table.

## Step 7. Convert logical.name values from a meaningful string to a sequential number

This is an optional step.

To minimize the impact to your production environment, the system will not change the values of logical.name in your production environment. Only when the user adds a new CI, a sequential number is generated by the “getnumber()” function, based on the settings in the Sequential Number File whose Class is “device”.

If you want to convert all existing logical.name values to a sequential number, perform the following steps:

1. Update the CI logical.name values to a sequential number by running database update queries.
2. Update the related records for incidents/probsummary/cm3r and so on by running database update queries. Also, you can check the entity relationship diagrams associated with the **device** table.
3. Modify the **Last Number** value in the Sequential Number File for **device**, to make sure the new CIs can get an ID.

## Step 8. Update the UCMDB integration

**Note:** If you do not use the UCMDB integration, skip this step.

## Deprecated UCMDB integration web services

The following web service objects and their DEM rules are deprecated and removed in Service Manager 9.41.

Deprecated web service	Replaced with
ucmdbApplication	ucmdbRunningSoftware
ucmdbComputer	ucmdbNode
ucmdbNetwork	ucmdbNode
ucmdbPrinter	ucmdbNode

If they existed in your system before you upgrade to Service Manager 9.41, they will remain in your system after upgrade.

- If you no longer want to use them, run the **Copy Data** script. For details, see the Service Manager 9.41 *Universal CMDB Integration Guide* documents.
- If you still want to use them, be sure to expose the **sm.device.display.name** (caption: DisplayName) field in them, as described in one of the following steps.

## How to update the UCMDB integration

**Note:** The following steps apply to both an XSLT-based adapter and enhanced generic adapter, unless otherwise noted. For more information about the SM-CMDB integration, see the Service Manager 9.41 *Universal CMDB Integration Guide* documents, which you can download from the following HP Software Support (SSO) web site:

<https://softwaresupport.hp.com/group/softwaresupport/search-result/-/facetsearch/document/KM01294561>

If you use the UCMDB integration, perform the following steps:

1. Deploy a Service Manager 9.41 adapter appropriate for your specific UCMDB version.

**Note:** The Service Manager 9.41 adapters are needed only when you have upgraded your Service Manager applications, server and clients to version 9.41. If your server and clients have upgraded

to version 9.41 and your applications version is still earlier than 9.41, use an old adapter appropriate for your applications version.

To verify your Service Manager applications version, follow these steps:

- a. Log in to Service Manager as a system administrator.
- b. Enter `db` in the command line, and press Enter.
- c. Enter `scversion` in the Table field, and then click **Search**.
- d. Click **Search** again.
- e. Check the Application Version value that is displayed.

The following table lists the adapters supported for Service Manager 9.41. You can download them from the HP Live Network:

- a. Visit <https://hpln.hp.com/node/11331/contentfiles/?dir=26314>.
- b. Click **SM Adapter for SM 9.41** to download the **adapter\_for\_SM9.41\_0.zip** file.
- c. Extract the compressed files to your local drive. The adapters for different UCMDB versions are located in different folders, as listed in the following table.

**Note:** A known issue (QCCR1E122493) exists in UCMDB versions earlier than 10.21: If two CIs with the same display name are pushed to SM and then updated in SM with a new display name, when you run a population job to synchronize the CIs back to UCMDB, the two CIs are merged in UCMDB and therefore have the same `ucmdb.id` value in SM.

UCMDB Version	Folder Name	Adapter(s)	Note	Solution to Known Issue QCCR1E122493
10.01	CP12	<ul style="list-style-type: none"> <li>◦ ServiceManagerAdapter9-41.zip</li> </ul>	<ul style="list-style-type: none"> <li>◦ XSLT-based adapter</li> </ul>	Will be fixed in a future release

UCMDB Version	Folder Name	Adapter(s)	Note	Solution to Known Issue  QCCR1E122493
10.02	CP14	<ul style="list-style-type: none"> <li>ServiceManagerAdapter9-41.zip</li> </ul>	<ul style="list-style-type: none"> <li>XSLT-based adapter</li> </ul>	Will be fixed in a future release
10.20	CP15	<ul style="list-style-type: none"> <li>ServiceManagerAdapter9-41.zip</li> <li>ServiceManagerEnhancedAdapter9-41.zip</li> </ul>	<ul style="list-style-type: none"> <li>XSLT-based adapter</li> <li>Generic adapter</li> </ul>	Upgrade to version 10.21
10.21	CP16	<ul style="list-style-type: none"> <li>ServiceManagerAdapter9-41.zip</li> <li>ServiceManagerEnhancedAdapter9-41.zip</li> </ul>	<ul style="list-style-type: none"> <li>XSLT-based adapter</li> <li>Generic adapter</li> </ul>	Already fixed

2. Change your own mapping files based on the following recommendations:
  - a. Remove the logical.name field from your CI Push mapping files.
  - b. Map the CI display label field to the Display Name field in Service Manager.
3. If you are still going to synchronize CI names from UCMDB to the logical.name field in SM, make sure each value is unique across the entire SM system. This is because CI name duplication rules are removed in SM 9.41, and therefore Service Manager no longer automatically renames duplicate CIs when they are pushed to SM.
4. Remove existing DEM reconciliation rules from your system. If you want to add new rules, refer to the Service Manager 9.41 *Universal CMDB Integration Guide* for information on how to add new reconciliation rules.
5. Understand that global.id generated by UCMDB is now the only field used to identify CIs throughout UCMDB and SM.

- UCMDB generates the unique global id and then pushes it to SM as a single reconciliation key.
  - As a best practice, you can manage discoverable CIs in UCMDB and non-discoverable CIs in SM. When CIs without a global id are populated from SM to UCMDB, UCMDB can push the global id back to SM. For this reason, it is recommended to run authorized data population to UCMDB first and then run data push from UCMDB to SM.
6. Expose **sm.device.display.name** (caption: DisplayName) in your web service object records of the ucmbdIntegration web service.
  7. For CI relationship push mapping files, make no updates.
  8. For CI population mapping files, you need to perform the following steps:
    - a. Map the UCMDB CI name field to the SM DisplayName field, instead of to the SM CIIdentifier field.
    - b. Map the sm\_id field to the SM CIIdentifier field, instead of to the SM CName field.
    - c. Add the global.id field to the mapping files:

Example for an XSLT-based adapter:

```
<attribute name="global_id" type="String"><xsl:value-of
select="UCMDBId"/></attribute>
```

Example for a generic adapter:

```
<target_mapping datatype="STRING" name="global_id" value="bizservice
['UCMDBId']"/>
```

9. (For an enhanced generic adapter only) Update the Relationship and Running Software population mapping files.
  - Remove all upstream and downstream “name” mapping entries from all relationship population mapping files. The following are two example “name” mapping entries:
 

```
<target_mapping name="name" datatype="STRING" value="cirelationship1to1
['upstreamci.logical.name']"/>
```

```
<target_mapping name="name" datatype="STRING" value="cirelationship1to1
['downstreamci.logical.name']"/>
```
  - Remove the “name” mapping entry (the entry in bold) from the container node in the running

software population mapping file.

```
<for-each-source-entity count-index="i" source-
  entities="runningsoftware.computer">

  <target_entity name="Node">

    <target_mapping name="name" datatype="STRING"
      value="runningsoftware.computer[i]['CIIdentifier']"/>

    <target_mapping name="global_id" datatype="STRING"
      value="runningsoftware.computer[i]['UCMDBId']"/>

  </target_entity>

</for-each-source-entity>
```

10. (For an XSLT adapter only) Make no updates for Federation.

11. (For an enhanced generic adapter only) Make the following updates for Federation.

- Update the smFedConf.xml file to change one groovy method to **convertArrayOrString**, as shown below:

```
<property ucmdbName="reference_number" smName="IncidentID"
  valueConverter="convertArrayOrString"/>
```

- Add the following two functions to the SMFederationConverter.groovy file.

```
/**
 * Convert arraylist to a string
 *
 * @param to be converted
 * @return converted string value
 */
public static String convertArrayString(def arrStr, DataAdapterLogger log){
  def strList=''
  for (s in arrStr) {
    strList+=''+s+''+',';
  }
  if (strList.size()>0)
    strList=strList.substring(0, strList.length()-1);
  return '{' + strList + '}';
}
public static String convertArrayOrString(def str, DataAdapterLogger log){
  if (str instanceof String)
    return convertString(str, log);
  else
```

```
    return convertArrayString(str, log);
}
```

12. (UCMDB 10.21 only) Configure the UCMDB JMX console, as described in the following steps.

**Note:** This feature is available only in UCMDB 10.21 or later. For this reason, a known issue exists in UCMDB versions earlier than 10.21. For more information, see step 1.

- a. Go to the JMX Console: `http://<server>:<port>/jmx-console/`
- b. Enter **UCMDB:service=Settings Services** in the search box, and press Enter.
- c. Click **setSettingValue** to go to this section.
- d. In the **customerID** field, enter your customer ID.
- e. In the **name** field, enter **reconciliation.match.attributes**.
- f. In the **value** field, enter **global\_id**.
- g. Click **Invoke**.

If the configuration is successfully completed, the following message is displayed:

```
Setting [reconciliation.match.attributes] value changed successfully to
[global_id]
```

## Step 9. Run a full push in UCMDB

**Note:** If you do not use the SM-UCMDB integration, skip this step.

Because the field mapping between UCMDB and SM is changed, you need to run a full push to synchronize the CIs from UCMDB to SM. To do so, follow these steps:

1. Update the DEM Rules to update CI directly instead of creating a new change/incident records.
2. Run a full push from UCMDB to SM.
3. Revert your changes made to the DEM rules.

## Step 10. Update CI related reports

CI related reports are categorized into four types by reporting solution.

### Service Manager reports

Service Manager reports refer to reports that are generated by the built-in Service Manager Reports functionality.

The logical name solution has no impact on Service Manager reports. After an upgrade to SM 9.41, your existing Service Manager reports can continue to work. Additionally, Service Manager reports against the **device** table are grouped by the logical.name field and sorted by the display.name field.

- MySM has been obsoleted since SM9.40. MySM reports can be migrated to Service Manager reports by using a migration tool. For more information, see the Service Manager online help.
- Legacy dashboards (Chart by ...) do not support the logical name solution in SM 9.41. Additionally, Report Exerciser does not display CI display names and is obsoleted starting with SM 9.41.

### HP Executive Scorecard reports

After an upgrade from SM9.40 to SM9.41, you need to apply manual updates for the reports to continue to work. For details, see ["HP Executive Scorecard integration" on page 74](#).

### Crystal Report reports

Reports generated by Crystal Report show logical.name data instead of display.name data, and can be sorted only by logical.name instead of display.name.

### Other reporting solutions

You need to update the CI related part of your reporting solution. Because the CI web service now uses a new field called Display Name, you can use that field for reporting purposes.



## Step 11. Update formats/links/format controls/JavaScript (including conflicts caused by the logical name solution)

This step assumes that you use the “logical.name” field as the unique ID and the “display.name” field as the display field in the **device** table.

## Format

You need to tailor formats that are based on the **device** table by adding a new field with an Input of “display.name” (whose alias is sm.device.display.name) for Search and Display purposes. We recommend that you use the alias “sm.device.display.name” instead of field name “display.name” for tailoring, because the “Display Field” can be changed from “display.name” to another field, while the “Display Field” alias “sm.device.display.name” will never be changed for the **device** table.

## Format Control (Mandatory validation, Classic mode only)

For those fields that have a referenced table configured in the datadict, you need to change your mandatory validation code to make the Logical Name solution work in a more friendly way.

Suppose your original code is not `(null(affected.item in $file))`, as shown in the following figure:

[illegible]

You need to change the original code as shown in the following figure:

true	true				jscall("DisplayName.validateRefField", \$file, "affected.item")
affected.item					\$reference.field.validation.msg

**Note:** If there is no mandatory validation for fields that have a referenced table configured in the datadict and you do not use Fill on these fields, user input in these fields is automatically cleared when the record is saved.

## RuleSet (Set Mandatory Fields, Codeless mode only)

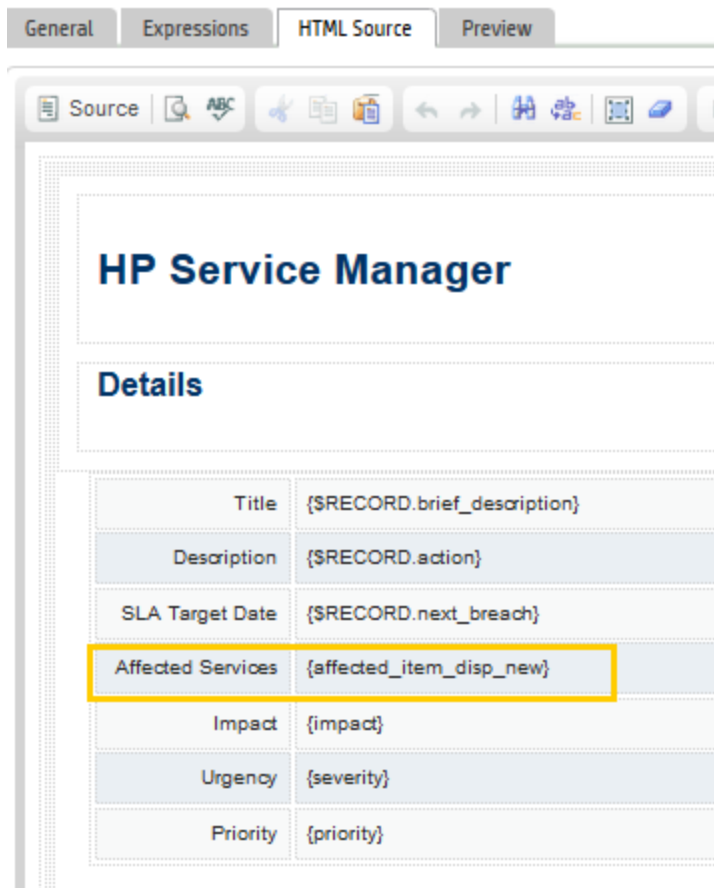
Make sure to configure a Set Mandatory Fields ruleset for the reference field according to your business needs. If such a ruleset is not configured, when a user enters a value in such a field without clicking Fill, the user input is automatically cleared when the record is saved.

## Notifications and HTML templates

For HTML templates, you need to call the “get.display.value” function on the **Expressions** tab to get the CI display name and assign the value to a variable, as shown in the following line using the affected.item field as an example:

```
var affected_item_disp_new= system.functions.get_display_value($RECORD,
    "affected.item",-1,true);
```

On the **HTML Source** tab, you can see the configuration, as shown in the following figure.



For notifications, you need to wrap the logical.name field with the ‘get.display.value’ function to show the CI display name in the message and email/mail subject line argument lists. See the following for an example.

```
{get.display.value($L.file,"logical.name",-1,true), $lo.ufname}
```

Msg Class	Msg No.	Arguments	Condition	Format	Notify Method
cm3	12	{number in \$L.file,current.phase ii	category in \$L.file=="Rele		msg
cm3	12	{number in \$L.file,current.phase ii	category in \$L.file!="Rel		email
cm3	12	{number in \$L.file,current.phase ii	category in \$L.file=="Rele		email
cm3	12	{number in \$L.file,current.phase ii	category in \$L.file=="Rele		email
cm3	12	{number in \$L.file,current.phase ii	category in \$L.file=="Sub		msg
cm3	12	{number in \$L.file,current.phase ii	category in \$L.file=="Sub		email
cm3	12	{number in \$L.file,current.phase ii	category in \$L.file=="Sub		email
cm3	12	{number in \$L.file,current.phase ii	category in \$L.file=="Sub		email
alert	7	{number in \$L.file,current.phase ii	approval.status in \$L.file:		email
cm3	359	{get.display.value(\$L.file,"logical	category in \$L.file="Subsi		email
alert	7	{number in \$L.file,current.phase ii	approval.status in \$L.file:		email

Link (Fill, View Detail/Find, Auto Complete, and new or updated RAD/JavaScript functions)

The Logical Name solution performs “Fill by Display Name” and “View Detail by ID (first unique key)” on reference fields.

## Fill

The system performs the “Fill” action based on the query defined in the link line to filter data against the target table. Previously, you normally defined the query by ID.

The logical name solution introduced a **Skip Query Rewriting** option to the link line form. Different guidelines apply when this option is enabled or disabled.

When writing a link query for a reference field, follow these guidelines.

- If the “Skip Query Rewriting” option is not enabled, update an expression such as “logical.name#affected.item in \$File” to the following:

```
logical.name#\"+affected.item in $File+\""
```

See the following figure for an example.

**FIGURE 3-1: Link query for a reference field when Skip Query Rewriting is not enabled**

Field (From/Source):	File (To/Target):	Format (To/Target):	Field (To/Target):
affected.item	device		logical.name
Comment:			
Query:	\$query		
QBE Format:	affected.item.qbe	Structured Array Name:	

☐ Skip Query Rewriting

Expressions JavaScript

```

1 if ($fill.skip.master=true) then ($fill.skip=true;cleanup($fill.skip.master))
2 $query="device.type=\"bizservice\""
3 if (not (null(affected.item in $File))) then $query+=(" and logical.name#\""+affected.item in $File+"\"")

```

- If the “Skip Query Rewriting” option is enabled, you can write a query like the following:

```

if (not (null(get.display.value($File, "affected.item")))) then $query+=(" and
display.name#\""+get.display.value($File, "affected.item")+\"")

```

See the following figure for an example.

**FIGURE 3-2: Link query for a reference field when Skip Query Rewriting is enabled**

Field (From/Source):	File (To/Target):	Format (To/Target):	Field (To/Target):
affected.item	device		logical.name
Comment:			
Query:	\$query		
QBE Format:	affected.item.qbe	Structured Array Name:	

☒ Skip Query Rewriting

Expressions JavaScript

```

1 if ($fill.skip.master=true) then ($fill.skip=true;cleanup($fill.skip.master))
2 $query="device.type=\"bizservice\""
3 $L.display.value=get.display.value($File,"affected.item")
4 if (not (null($L.display.value))) then $query+=(" and display.name#\""+$L.display.value+"\"")

```

As a system administrator, you need to understand how the Fill function works for a reference field (take the first figure above for example):

- The “affected.item in \$File” will be null when the user types something in the format, and the system gets what the user has typed by the “get.display.value” function.
- If the “Skip Query Rewriting” option is not enabled, the system assigns what the user has typed to “affected.item in \$File” when the user clicks the Fill button, and then rewrites at runtime the evaluated “\$query” from, for example, “logical.name# DISPLAY VALUE” to “display.name# DISPLAY VALUE”. This makes the original link query still work for the reference field.
- If the “Skip Query Rewriting” option is enabled, the system does not perform the additional operations that it does when the “Skip Query Rewriting” option is not enabled.

## View Detailed Information/Find

The system runs an exact matching query by ID against the referenced table instead of using the query defined in related link line for the reference field. However, in some special cases, this will cause problems. You need to fix the corresponding code if you encounter the following case or a similar one:

The “View Affected Services” functionality in the out-of-box Service Desk and Incidents modules allows the user to view all Affected Services of the current CI. If such kind of functionality is implemented by calling RAD Application “us.link”, you need to pass one more parameter (**Skip Exact Find?** (cond.input) as true) to it, and make sure the “Skip Query Rewriting” option is enabled. The system then uses the query defined in the related link line when performing the “Find” action.

RAD Application:

us.link	Condition:
Parameter Names	Parameter Values
record	\$L.file
name	"logical.name"
second.record	\$L.link
prompt	"find"
cond.input	true

Post RAD Expressions

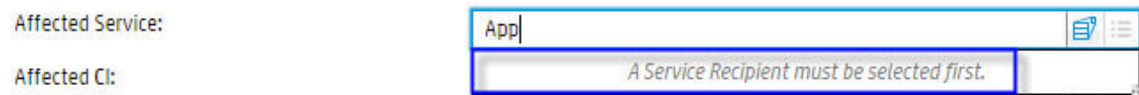
## Auto Complete

The Logical Name solution extends the legacy Auto Complete functionality for reference fields. When selecting CIs, users can view more CI information than before, and the list of CI attributes that are displayed for Auto Complete is customizable. For more information, see ["Solution details" on page 12](#).

**Note:** If no CI attributes are configured on the **Auto Complete Table Columns** tab, the system will show the Display Field by default.

### Showing the link line message for Auto Complete

If you want the system to display the message defined in a link line immediately after the user input triggers Auto Complete (see the following figure for an example), you need to assign the message to the “\$autocomplete.msg” variable in the link line definition.



The following figure shows an example of how you can do so.

Field (From/Source):	File (To/Target):	Format (To/Target):	Field (To/Target):
affected.item	device		logical.name
Comment:			
Query:	\$query		
QBE Format:	affected.item.qbe	Structured Array Name:	
<input type="checkbox"/> Skip Query Rewriting			
<div> <div>Expressions</div> <div>Javascript</div> </div> <pre> 1 if null(contact.name in \$File) then (contact.name in \$File=callback.contact in \$File);\$fill.skip.master=true 2 if (null(contact.name in \$File) and option()=8) then (\$L.void=rtecall("msg", \$L.result, scasg(20, "BPPH"));   \$autocomplete.msg=scasg(20, "BPPH");\$fill.skip=true) 3 \$L.contact.success.flg=rtecall("rmt", \$L.rc1, \$L.contact.file, "contacts", _1) </pre>			

## Customizing your dynamic links for Auto Complete

In the out-of-box SM9.41 system, the Auto Complete feature retrieves links from the **link** table when the user input in a field triggers Auto Complete. However, your production environment may have been tailored such that it uses a dynamic link (which is generated by a process, for example) for a Fill operation. After upgrading to SM 9.41, your dynamic link still works for the Fill operation; however, it no longer works for Auto Complete. To solve this problem, you need to customize your dynamic link by using the LinkUtilOverride JavaScript:

```
lib.LinkUtilOverride.getCustomizedLink(record, format, field, optionValue);
```

To use this script, follow these steps:

1. Enter `sl` in the Service Manager command line, and press Enter.
2. In the Name field, enter `LinkUtilOverride`.
3. Click **Search**.

The script is displayed. It provides only a framework and requires you to manually implement the code. You can review the comments in the script for instructions.

4. Implement the script code for your dynamic link.

For Auto Complete, Service Manager will retrieve the link first from the return value of this script, and then from the **link** table if the script returns Null.

## Append Query

The Append Query functionality is available for application profiles and security roles. This functionality is used to append a query to a normal query. When using this functionality on a table that references the **device** table, be sure to use CI Identifier (logical.name) in the Append Query expression After upgrading to SM9.41, if your legacy CI Identifier values are changed from display names to sequential numbers, you must manually change the CI display name in Append Query to the CI identifier.

For example, you added "affected.item#"Adobe"" in the Append Query part for a Change record and the CI Identifier value for Adobe is changed to Clxxxxx after upgrade to SM9.41, then you must change the Append Query expression to "affected.item#"Clxxxxx"".

## RAD and JavaScript functions

The logical name solution introduced the following RAD functions, JavaScript function, and ScriptLibrary utilities, which you probably need to use for tailoring.

### **RAD function: get.display.value**

A RAD function that returns the display value of a field in a file.

#### **Function**

get.display.value

#### **Format**

get.display.value(\$file, \$fieldname, [index], [bForceFromDB])

#### **Parameters**

The following parameters are valid for this function.

Parameter	Description
\$file	The \$file variable is the file from which you want to retrieve the display value.
\$fieldname	The name of a field in the file.  Note: You cannot use a variable for this parameter. For example, \$L.display=get.display.value(\$L.file, "\$affected.cis", -1, true) is not supported.
[index]	This parameter is optional.  If the field specified in \$fieldname is of the array type, the index is the position number of an element in the array where the element is located.  If the index is not specified or is -1, the entire array is retrieved instead.
[bForceFromDB]	This parameter is optional.  This is a Boolean type variable, which can be true or false. If it is true, this function will bypass the internal display value cache associated with \$file, and retrieve the display value directly from the RDBMS.

#### **Factors**

If an internal display value cache does not already exist, a null value is returned; however, if `bForceFromDB` is set to true, the Service Manager server looks for the display value directly from the RDBMS according to the current field value.

For an array type field, if `bForceFromDB` is set to true, this function combines the value in the display value cache (if any) and the value fetched directly from the RDBMS, and returns the combined value.

For a string type field, if `bForceFromDB` is set to true, while the Service Manager server cannot find any value from the RDBMS, the value in the display value cache (if any) is returned instead.

There are two ways to create an internal display value cache associated with \$file:

- Use the `set.display.value` function explicitly.
- Any time when users enter or change the display value from the client side and the client submits the request to the server side, the internal display value cache is updated. This is automatic and not under the user's control.

### Return values

Can be null, a string type display value or an array of string type display values.

### Example

```
function testDisplayValue()
{
    var sql = 'select * from cm3r where number="C10008"';
    var file = new SCFile('cm3r');
    var rc = file.doSelect(sql);

    print( "Field value of assets=" + file.assets );
    print( "Field value of logical.name=" + file.logical_name );

    var darray = new SCDatum();
    darray.push( "aaa" );
    darray.push( "bbb" );
    print( "Value of darray=" + darray );

    system.functions.set_display_value(file, "assets", darray, -1 );

    var value = system.functions.get_display_value(file, "assets" );
    print( "get.display.value for assets from internal display value cache=" + value
);

    value = system.functions.get_display_value(file, "assets", -1, true );
    print( "get.display.value for assets from RDBMS=" + value );

    value = system.functions.get_display_value(file, "logical.name" );
```



```

        print( "get.display.value for logical.name from internal display value cache=" +
value );

        value = system.functions.get_display_value(file, "logical.name", -1, true );
        print( "get.display.value for logical.name from RDBMS=" + value );
    }
    testDisplayValue();

```

This example returns the following results:

Field value of assets=[C++ object Datum] - {"CI1000984", "CI1000988"}

Field value of logical.name=CI1000984

Value of darray=[C++ object SCDatum] - {"aaa", "bbb"}

get.display.value for assets from the internal display value cache=[C++ object Datum]  
- {"aaa", "bbb"}

get.display.value for assets from the RDBMS=[C++ object Datum] - {"adv-Windows-101",  
"adv-Windows-102", "aaa", "bbb"}

get.display.value for logical.name from the internal display value cache=null

get.display.value for logical.name from the RDBMS=adv-Windows-101

### **RAD function: cursor.field.display.content**

A RAD function that returns a string containing the display value of the field where the cursor was last positioned when the user performed an action such as clicking Fill or Find, or double-clicking on a QBE list. If the action opens a new form, cursor.field.display.content will return the focus on the pop up form.

#### **Function**

cursor.field.display.content

#### **Format**

cursor.field.display.content()

#### **Parameters**

None

#### **Factors**

If the field is an array, the content of the current element is returned.

#### **Return values**

Returns the display value of the current field. For example, "Adobe Reader".

**RAD function: cursor.field.display.content.set**

A RAD function that sets the display value for the field where the cursor is currently positioned.

**Function**

```
cursor.field.display.content.set
```

**Format**

```
cursor.field.display.content.set($display.value)
```

**Parameters**

\$display.value is the display value to be set.

**Factors**

If the field is of the array type, the display value is set to the content of the element where the cursor is currently positioned.

**Return values**

The display value that you specify.

**Example**

```
cursor.field.display.content.set("Adobe Reader")
```

**RAD function: display.value.copy**

A RAD function that copies the internal display value cache from one file variable to another.

**Function**

```
display.value.copy
```

**Format**

```
display.value.copy($targetfile, $sourcefile)
```

**Parameters**

The following parameters are valid for this function.

Parameter	Description
\$targetfile	The file to which the internal display value cache is copied

Parameter	Description
\$sourcefile	The file from which the internal display value cache is copied

**Factors**

The source or target file variable must be a valid file variable.

The old internal display value cache on the target file variable (if any) is released and a new internal display value cache is copied from the source file variable.

**Return values**

0: Success

-1: Failure

**Example**

Refer to the example for “RAD function: same”.

**RAD function: same**

A RAD function that compares two values. If the values are equal or both values are null, the result is true; otherwise, the result is false.

**Note:** This is an old function that is updated in Service Manager 9.41.

**Function**

same

**Format**

same(value1,value2)

**Parameters**

value1 is the first value to be compared, and value2 is the second value to be compared.

**Factors**

Use this function to compare arrays and structures, nulls, files, or empty strings.

When using the same function to compare arrays or structures, you need to preface the variable names with denu11, as shown in the following example:

```
not same(denu11($filea), denu11($fileb))
```

**Note:** When using the same function to compare two files, the files must have the same schema, data, display value, and so on to return a result of True.

### Return values

True: same

False: not same

### Example

same(1,NULL) returns false.

same(NULL,NULL) returns true.

same({},{}) returns true.

same({1},{1}) returns true.

same("",NULL) returns false.

```
function testdisplayvaluecopy()
{
    var sql = 'select * from cm3r where number="C10001"';
    var file = new SCFile('cm3r');

    var file1 = new SCFile('cm3r');
    var rc = system.functions.fduplicate( file, file1 );

    rc = file.doSelect(sql);
    rc = system.functions.same( file, file1 );
    print("file is same as file1 after file.doSelect: ", rc );

    rc = file1.doSelect(sql);
    rc = system.functions.same( file, file1 );
    print("file is same as file1 after file1.doSelect: ", rc );
    var vRet = system.functions.set_display_value(file1, "assets", "testbystone", 1);

    rc = system.functions.same( file, file1 );
    print("file is same as file1 after set.display.value: ", rc );

    vRet = system.functions.display_value_copy(file, file1);

    rc = system.functions.same( file, file1 );
    print("file is same as file1 after display.value.copy: ", rc );
}
testdisplayvaluecopy();
```

The result of this example is as follows:

file is same as file1 after file.doSelect: false

file is same as file1 after file1.doSelect: true

```
file is same as file1 after set.display.value: false
```

```
file is same as file1 after display.value.copy: true
```

### **RAD function: set.display.value**

A RAD function that sets the display value for a specified field of a file variable. The display value can then be retrieved by using the `get.display.value($file,$fieldname)` syntax.

**Note:** This function does not change the display name value in the database. The value set by this function works for the application logic but is discarded if the record is saved. Only the `logical.name` value is saved in a reference table such as Incident (`probsummary`).

#### **Function**

`set.display.value`

#### **Format**

`set.display.value($file,$fieldname,$dispvalue,[index])`

#### **Parameters**

Parameter	Description
<code>\$file</code>	The file variable for which the display value is set
<code>\$fieldname</code>	The field name in the file
<code>\$dispvalue</code>	The display value to be set
<code>[index]</code>	<p>This parameter is optional.</p> <p>If the field specified in <code>\$fieldname</code> is of the array type, the index is the position number of an element in the array where the element is located; if the index is not specified or is -1, <code>\$dispvalue</code> must be an array type value as well.</p>

#### **Factors**

This function sets the display value in an internal display value cache associated with the `$file` variable. If an internal display value cache does not already exist, this function creates a cache.

#### **Return values**

Null or the old display value, which can be an array of strings or a string.

#### **Example**

Refer to the example for “RAD function: `get.display.value`”.

**RAD function: clean.display.value**

A RAD function that cleans up the internal display value cache associated with a file variable.

**Function**

```
clean.display.value
```

**Format**

```
clean.display.value($file, [$fieldname])
```

**Parameters**

The following functions are valid for this function.

Parameter	Description
\$file	The file variable whose internal display value cache is to be cleaned up.
\$fieldname	The field whose display value is to be cleaned up. If this parameter is not specified, the entire internal display value cache of \$file is cleaned up.

**Return values**

Null or the old display value.

**Example**

```
clean.display.value($file)
```

```
clean.display.value($file, "logical.name")
```

**RAD function: cleanup**

A RAD function that frees the storage associated with a variable.

**Note:** This is an old function that is updated in Service Manager 9.41.

**Function**

```
cleanup
```

**Format**

```
cleanup($filename, [$fieldname], [index])
```

**Parameters**

The following parameters are valid for this function.

Parameter	Description
\$filename	The variable that you want to clean up.  If only \$filename is specified, this variable can be a file variable or a field.
[\$fieldname]	Optional.  When specified, indicates the name of a field whose field value and display value (if any) are to be cleaned up.  <b>Note:</b> Fields that have a referenced table defined have a display value.
[index]	Optional.  If the field specified in \$fieldname is of the array type, the value and display value (if any) of the element that has the specified index or position number in that array are cleaned up.

**Factors**

If only \$filename is specified when calling this function, this function returns the same result as it would in versions earlier than 9.41.

This function does not return a value. It cleans up (empties) the variable or field specified by setting the value to NULL.

You can use this function on any variable type (local, thread, or global), but it is generally used on local and thread variables to clean up values before a user logs out.

This is the preferred way to clean up both variables and fields in records as opposed to

\$x=NULL or field in \$file=NULL.

Unless you run this function, variables remain in memory until their defining environment ends.

For local variables, this is the parent RAD program. For thread variables, this is the parent RAD thread. For global variables, this is the user session.

**Note:** This RAD function is not available as a JavaScript system.functions call.

**Return values**

None

**Example**

```
cleanup($file)
```

```
cleanup($file, "logical.name")
cleanup($file, "assets", 0)
```

### **RAD function: dbdict.helper("field.type")**

A RAD utility function that helps retrieve the field type property efficiently.

#### **Function**

```
dbdict.helper("field.type")
```

#### **Format**

```
dbdict.helper("field.type",$L.file/$filename,fieldname)
```

#### **Parameters**

The following parameters are valid for this function.

Parameter	Description
\$L.file/\$filename	The variable or name of the file to which the field belongs
fieldname	The name of the field whose field type property is to be retrieved.

#### **Factors**

Always consider using a file variable first for better performance.

#### **Return values**

A number value that represents the field type. For information about the numeric representations of data types, see the "Data types" section in the Service Manager Programming Guide.

#### **Example**

```
function testDBDictHelper()
{
    var file = new SCFile( 'cm3r' );
    var fieldType = system.functions.dbdict_helper("field.type", file, "assets" );
    print( "fieldType of assets in cm3r=", fieldType );

    var fieldType = system.functions.dbdict_helper("field.type", file, "type" );
    print( "fieldType of type in cm3r=", fieldType );

    fieldType = system.functions.dbdict_helper("field.type", "cm3r", "assets" );
    print( "fieldType of assets in cm3r by filename=", fieldType );

    fieldType = system.functions.dbdict_helper("field.type", "cm3r", "type" );
```



```

        print( "fieldType of type in cm3r by filename=", fieldType );

        print( "db type=", system.functions.dbdict_helper("db.type" ) );

        var isAlias = system.functions.dbdict_helper("is.alias", file, "assets" );
        print( "assets is an alias:", isAlias );

        isAlias = system.functions.dbdict_helper("is.alias", file, "number.vj" );
        print( "number.vj is an alias:", isAlias );

        isAlias = system.functions.dbdict_helper("is.alias", file, "number.vj.ooflow" );
        print( "number.vj.ooflow is an alias:", isAlias );

        isAlias = system.functions.dbdict_helper("is.alias", "cm3r", "number.vj.ooflow"
    );
        print( "number.vj.ooflow is an alias by filename:", isAlias );

        isAlias = system.functions.dbdict_helper("is.alias", file, "number.string" );
        print( "number.string is an alias:", isAlias );

        isAlias = system.functions.dbdict_helper("is.alias", "cm3r", "middle,type" );
        print( "middle,type is an alias by filename:", isAlias );

        isAlias = system.functions.dbdict_helper("is.alias", "cm3r", "number.string" );
        print( "number.string is an alias by filename:", isAlias );

        isAlias = system.functions.dbdict_helper("is.alias", file, "middle,type" );
        print( "middle,type is an alias:", isAlias );

        var fileArray = system.functions.dbdict_helper( "joinfile.names", "joincomputer"
    );
        print( "sub file array of joincomputer is: ", fileArray );

        var uniquekey = system.functions.dbdict_helper( "unique.key", file );
        print( "uniquekey of cm3r is: ", uniquekey );
    }

```

The Result of this example:

fieldType of assets in cm3r= 8

fieldType of type in cm3r= 2

fieldType of assets in cm3r by filename= 8

fieldType of type in cm3r by filename= 2

db type= sqlserver

assets is an alias: false

number.vj is an alias: true

number.vj.ooflow is an alias: true

number.vj.ooflow is an alias by filename: true

number.string is an alias: false

middle.type is an alias by filename: false

number.string is an alias by filename: false

middle.type is an alias: false

sub file array of joincomputer is: [C++ object Datum] - {"device", "computer"}

uniquekey of cm3r is: [C++ object Datum] - {"number"}

uniquekey of cm3r by filename is: [C++ object Datum] - {"number"}

### **RAD function: dbdict.helper("db.type")**

A RAD utility function that helps retrieve the current RDBMS type.

#### **Function**

dbdict.helper("db.type")

#### **Format**

dbdict.helper("db.type")

#### **Parameters**

None

#### **Return values**

A string type value that represents the current RDBMS type, which can be "sqlserver", "oracle", or "db2universal".

### **RAD function: dbdict.helper("is.alias")**

A RAD utility function that helps determine whether a field is an alias or not.

#### **Function**

dbdict.helper("is.alias")

#### **Format**

dbdict.helper("is.alias", \$L.file/\$filename, fieldname)

**Parameters**

The following parameters are valid for this function.

Parameter	Description
<code>\$L.file/\$filename</code>	The variable or name of the file to which the field belongs
<code>fieldname</code>	The name of the field to be checked

**Factors**

Always consider using a file variable first for better performance.

**Return values**

True: The field is an alias.

False: The field is not an alias.

**RAD function: `dbdict.helper("joinfile.names")`**

A RAD utility function that helps retrieve an array of names of the sub files that belong to a join file.

**Function**

```
dbdict.helper("joinfile.names")
```

**Format**

```
dbdict.helper("joinfile.names",$joinfilename)
```

**Parameters**

`$joinfilename` is the name of the join file whose sub file names array is to be retrieved.

**Factors**

The join file must be already defined in `joindefs`.

**Return values**

Null or an array of strings containing the sub file names.

**RAD function: `dbdict.helper("unique.key")`**

A RAD utility function that helps retrieve the first unique key or the primary key of a file efficiently.

**Function**

```
dbdict.helper("unique.key")
```

**Format**

```
dbdict.helper("unique.key",$L.file/$filename)
```

**Parameters**

\$L.file/\$filename is the variable or name of the file whose first unique key or primary key is to be retrieved.

**Factors**

Only the normal file type is supported. Other types of tables are not supported.

This function returns the first unique key or the primary key definition from the datadict file instead of the dbdict file; The application layer will keep consistency between datadict and dbdict for the first unique key or the primary key.

**Return values**

An array of strings containing the names of all fields that constitute the first unique key or the primary key.

**RAD function: policyread**

A RAD function that reads the data policy from a datadict table.

**Note:** This is an old function that is updated in Service Manager 9.41.

**Function**

policyread

**Format**

```
$L.returnvalue = policyread( $L.file/$filename, fieldname, fieldsetting)
```

**Parameters**

The following parameters are valid for the policyread function.

Parameter	Description
\$L.returnvalue	The policy value returned after it is read from the database. It can be a string, a boolean, or null.
\$L.file/\$filename	This can be either a file variable or a file name:  \$L.file - The file handle of the table for which the data policy information is to be

Parameter	Description
	<p>read from the database. It can be a join table, a normal table or an adhoc table.</p> <p><b>Note:</b> Other kinds of tables (merge files, etc.) are not supported.</p> <p>\$filename – The name of the table for which the data policy information is to be read from the database. It can be a normal table only.</p> <p><b>Note:</b> All other kinds of tables are not supported.</p> <p>The table name is the value in the Name field in the datadict record.</p>
fieldname	<p>The name of the field in the specified table for which the data policy information is to be read from the database.</p> <p>The field name is the value in the Field Name field in the datadict record.</p>
fieldsetting	<p>The field setting on the specified field for which the data policy information is to be read from the database.</p> <p>If no field setting is passed in when calling policyread function, the function will return true if the field exists in data policy, otherwise will return false.</p> <p>This is one of the fields on Field Settings tab of the datadict record.</p> <p>Valid field settings are: "invisible", "readonly", "mandatory", "captions", "avail", "encrypt", "defaults", "globallist", "matchfields", "matchfiles", and "validations", "types", "reference.table".</p> <p>In addition, there two special settings: "display.field", "autocomplete.show.field", if one of these settings is specified, the field name will be ignored.</p>

### Factors

If the fieldsetting is not in the list of Field Settings in the datadict table, the server will throw an error message. In any other case, this function returns a valid value or null.

### Return values

String, Boolean or null value.

### Example

#### Example 1

This example demonstrates how you could use a JavaScript to access the data policy on a simple file, such as the operator table.

```
// Access DataPolicy definition on operator file.
var operatorFile = new SCFile( "operator" );
var rteReturn;

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"invisible" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:invisible) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"readonly" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:readonly) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"mandatory" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:mandatory) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"captions" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:captions) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change", "avail"
);
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:avail) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"encrypt" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:encrypt) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"defaults" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:defaults) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"globallist" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:globallist) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
```

```

"matchfields" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:matchfields) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"matchfiles" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:matchfiles) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile , "profile.change",
"validations" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as defined in
(file:datadict, field:validations) " + rteReturn );

```

### Example 2

This example demonstrates how you could use a JavaScript to access the data policy on a complex file, such as a join of the incidents and contacts tables.

```

// Access DataPolicy definitions on join file incontacts (incidents and contacts)

var incident_contacts_file = new SCFile( "incontacts" );
var rteReturn;

// Access policy definition on file(contacts) and field(city) and datadict field
defaults
rteReturn = system.functions.policyread( incident_contacts_file,
"file.contacts,city", "defaults" );
print( "DataPolicy on (joinfile:incontacts, subfile:contacts, field:city), as
defined in(file:datadict, field:defaults)" + rteReturn );

// Access policy definition on file(incidents) and field(phone) and datadict field
defaults
rteReturn = system.functions.policyread( incident_contacts_file,
"file.incidents,phone", "defaults" );
print( "DataPolicy on (joinfile:incontacts, subfile:incidents, field:phone), as
defined in(file:datadict, field:defaults)" + rteReturn );

// Access policy definition on file(incidents) and field(phone), and datadata
field readonly
rteReturn = system.functions.policyread( incident_contacts_file,
"file.incidents,phone", "readonly" );
print( "DataPolicy on (joinfile:incontacts, subfile:incidents, field:phone), as
defined in(file:datadict, field:readonly)" + rteReturn );

```

### Example 3

```

function testPolicy()
{

```

```

var file = new SCFile( 'cm3r' );

var ret = system.functions.policyread(file, "assets", "reference.table" );
print( "The reference.table attribute for assets field in cm3r file is: ", ret );

ret = system.functions.policyread(file, "assets", "" );
print( "assets field in cm3r exists in data policy: ", ret );

ret = system.functions.policyread(file, "nofield", "" );
print( "nofield field in cm3r exists in data policy: ", ret );

ret = system.functions.policyread(file, "", "display.field" );
print( "display.field for file cm3r is: ", ret );

ret = system.functions.policyread("device", "", "display.field" );
print( "display.field for file device is: ", ret );
}

```

The Result of example 3:

The reference.table attribute for assets field in cm3r file is: device

assets field in cm3r exists in data policy: true

nofield field in cm3r exists in data policy: false

display.field for file cm3r is: null

display.field for file device is: display.name

### JavaScript method: XML.removeAttribute()

This method removes an XML attribute from the current element. It requires that you use other XML get methods to navigate through an XML document.

#### Syntax

```
XML.removeAttribute( AttributeName );
```

#### Arguments

The following argument is valid for this method.

Argument	Data type	Description
AttributeName	String	This argument specifies the text string you want the script to use as the XML attribute name. The string argument must contain characters valid for an XML element( for example, the string cannot include the characters< or >).



## Return values

An XML object or null.

If successful, the method returns the updated XML element. Otherwise, it return null.

## ScriptLibrary utilities

The following table lists the ScriptLibrary functions that you probably need to use as logical name utilities.

Function	Description
lib.DisplayName.hasDisplayName (tableName)	Check if a file has display field enabled.  @param {String} tableName - SM file name  @return {boolean} true/false
lib.DisplayName.getRefTable (field, tableName)	Get a field's referenced table name, if the parsed tableName is a jointable name, the system will return the referenced table name of the first valid joined table's field.  @param {String} field – field name of a table  @param {String} tableName – SM file name  @return {String} referenced table name or null
lib.DisplayName. getRefTableDisplayField (tableName)	Get the referenced table's display field, if the parsed tableName is a jointable name, the system will return the first valid joined table's display field name.  @param {String} tableName - SM file name  @return {String} referenced table's display field name or null
lib.DisplayName. getDisplayNameById (tableName,id)	Convert id to display value.  @param {String} tableName – SM file name  @param {String} id – record id value  @return {String} display value/id value if display value is null
lib.DisplayName. getDisplayNamesByIds (tableName,ids,keys,displays)	Convert array ids to array display values.  @param {String} tableName – SM file name  @param {Array} ids – record id value  @param {Array} keys – SM variable, will be returned with id values

Function	Description
	<p>@param {Array} displays – SM variable, will be returned with display values/id values if display value is null</p> <p>@return {Array} keys</p> <p>@return {Array} displays</p>
lib.DisplayName.convertNameTold (name,smrecord,field)	<p>This is used for the Integration utility.</p> <p>@param {String/Array} name – Display Name to be converted to ID</p> <p>@param {String/file} smrecord – SM file name or SM file record</p> <p>@param {String} field – field name of a table</p> <p>@return {String/Array} rtn – ID converted from Display Name</p> <p>Note: The system will get the referenced table by the smrecord and field parameters, and then convert the display name to id against the referenced table.</p> <p>if both smrecord and field are null, the system regards “name” as CI Display Name and converts it to CI Identifier.</p>

## General Issues and recommended solutions

The Logical Name solution does not support variable fields. You need to fix such kind of issues on a case by case basis as it is impossible for the system to completely address these issues. You need to use a real dbdict field instead of a variable as a reference field.

Issue	Solution
A reference field displays ID instead of Display Name.	<ol style="list-style-type: none"> <li>1. Check the widget in the format, and make sure it is a Comfill or a virtual join subform.</li> <li>2. Check the field's referenced table in the datadict to make sure it is configured correctly.</li> <li>3. If the field is a variable, you need to initialize the Value List/Display List for it in Forms Designer.</li> </ol>
Fill, Auto Complete, or both do not work well for a reference field.	<ol style="list-style-type: none"> <li>1. Check the field's referenced table in the datadict to make sure it is configured correctly.</li> </ol> <p>The referenced table must be same as the Target File Name defined</p>

Issue	Solution
	<p>in the related link.</p> <p>2. If the field is a variable, replace or re-implement it with a real dbdict field.</p>
Searching by display name does not work for a reference field.	<p>1. Check the field type in the dbdict. The Array/Structure type must not have a SQL Type of TEXT/IMAGE/LOB, and must have an alias table specified as its SQL Table.</p> <p>2. Check the field's referenced table in the datadict to see if it is configured correctly.</p> <p>3. If the field is a variable, replace or re-implement it with a real dbdict field.</p>

## Step 12. Update miscellaneous integrations

Besides the SM-UCMDB integration, the logical name solution also has impact on some other integrations. In the out-of-box Service Manager 9.41 system, required code changes are already implemented to address this impact. However, some integrations may also require code changes be made on the other product side, which may not be possible at the time of the Service Manager 9.41 release. Additionally, integrations in your production environment are normally tailored or customized, and therefore need manual updates. For these reasons, you may still need to make manually adjustments in your production environment.

**Note:** For information about how to update the UCMDB integration, see ["Step 8. Update the UCMDB integration" on page 33](#).

**Caution:** To enable the logical name solution, you must upgrade the Service Manager applications, server, and clients to version 9.41. If you upgrade only the platform (server and clients), your existing integrations require no manual changes and can continue to work as before.

## General guidelines for inbound integrations

For integrations that invoke SM APIs to create or update records (for example, Change/Incident/Problem records) in SM, be sure to specify the CI Identifier (logical.name) value of the affected Service/CI instead of the CI Display Name in your request.

After upgrading to SM9.41, you can solve your integration problems caused by the logical name solution in one of the following ways:

["Change the CI Identifier value to a sequential number" below](#)

["Modify the SM API to accept CI Display Name" below](#)

["Use the Service Manager Device API" on the next page](#)

## Change the CI Identifier value to a sequential number

Since it is optional to update your legacy CI Identifier values to a sequential number, determine if the legacy CI Identifier values in your system have been updated to a sequential number. If not, your integration is not impacted, and hence you do not need to make any manual changes; if yes, manually change the CI Identifier value specified in your integration configuration to a sequential number.

For example, Cloud Service Automation (CSA) calls OO flows to create Change records in SM, and you may have specified a constant value of "Applications" as a default Service value in the OO flow.

- After upgrading to SM 9.41, if the CI still uses "Applications" as its CI Identifier, your integration is not impacted, and therefore you do not need to make any manual changes.
- After upgrading to SM 9.41, if the CI Identifier (logical.name) value has been updated to Cxxxxx, you need to change "Applications" in the OO flow to "Cxxxxx" accordingly. Or, if you are passing the Service value from a parameter, and the value passed to the OO flow is a CI Name (for example, "John's Computer") while the CI in SM has a CI Identifier value of CI00056, you need to pass CI00056 to the OO flow in order to create a Change record in SM.

## Modify the SM API to accept CI Display Name

You can also modify the SM API to accept CI Display Name. To do so, add the following expressions to the web service definition:

```
logical.name in $L.file=jscall("DisplayName.convertNameToId ", logical.name in
    $L.file, $L.file, "logical.name")
```

```
affected.item in $L.file=jscall("DisplayName.convertNameToId ", affected.item in
    $L.file, $L.file, "affected.item")
```

```
assets in $L.file=jscall("DisplayName.convertNameToId ", assets in $L.file, $L.file,
    "assets")
```

See the following figure for an example.

## External Access Definition

Service Name:

Name:

Object Name:

Allowed Actions

Expressions

Fields

RESTful

```
logical.name in $.file=jsoncall("DisplayName.convertNameTold ", logical.name in $.file, $.file, "logical.name")
affected.item in $.file=jsoncall("DisplayName.convertNameTold ", affected.item in $.file, $.file, "affected.item")
assets in $.file=jsoncall("DisplayName.convertNameTold ", assets in $.file, $.file, "assets")
```

## Use the Service Manager Device API

You can modify your integration to first send a request for a CI's CI Identifier value to the Service Manager Device API (for example, ConfigurationManagement), and then use the retrieved CI Identifier value for creating or updating your records in SM.

## General guidelines for outbound integrations

The following are guidelines for integrations from Service Manager to another HP application.


If the other product system needs additional CI information, such as the CI display name and UCMDB ID, you need to add these fields to the web service definition, using the “[current table field]..[referenced table field]” format. For example, if you want to retrieve the display name and UCMDB ID of the affected service of a Change record, you need to add the following fields to the web service definition, as shown in the following table and figure.

**Note:** The exposed fields listed in the following table take effect only on the Retrieve operation (not on the Update or Create operation).

Field	Caption	Description
affected.item..sm.device.display.name	ServiceName	Display name of the affected item in the Change record
affected.item..ucmdb.id	ServiceID	UCMDB ID of the affected item in the Change record

## External Access Definition

Service Name:

Name:  

Object Name:

Allowed Actions

Expressions

Fields

RESTful

Field	Caption	Type
affected.item	Service	
rc.analysis,collision.severity	CollisionSeverity	
rc.analysis,impact.severity	ImpactSeverity	
rc.analysis,time,period.conflict	TimePeriodConflict	
rc.analysis,risk.severity	RiskSeverity	
rc.analysis,suggestedStart	SuggestedStartDate	DateTimeType
rc.analysis,suggestedEnd	SuggestedEndDate	DateTimeType
header,assign.dept	AssignmentGroup	
affected.item..sm.device.display.	ServiceName	
affected.item..ucmdb.id	ServiceID	

## Changes required for specific integrations

This section summarizes the solutions for specific integrations that are impacted by the logical name solution. After upgrading to SM9.41, some integrations require manual changes.

Target product	Solution	Manual changes
Operations Manager i (OMi)/Business Service Management (BSM)	<p>This integration is using a BDM mapping API defined in Service Manager (SM). Required code changes are already implemented in the out-of-box SM9.41 system.</p> <p><b>Caution:</b> Only external UCMDB instances are supported for the logical name solution. Embedded</p>	Not required.

Target product	Solution	Manual changes
	UCMDB (BSM RTSM) instances are not supported.	
Release Control (RC)	Both SM and RC need code changes to adapt to the logical name solution. Required changes are already implemented in SM9.41 and RC9.21p3.	Upgrade to RC9.21p3.
Application Lifecycle Management (ALM)	Out-of-box, this integration requires no configuration on the SM side.	Reconfigure the integration in ALM.  For details, see the integration documentation, which also includes additional manual changes that are documented in <a href="#">"Application Lifecycle Management (ALM) integration" on page 73</a> .
Operations Orchestration (OO)	OO flows will handle/parse CI Identifier to interact with Service Manager.	Depend on whether you change the CI Identifier field to a sequential number after upgrading to SM9.41: <ul style="list-style-type: none"> <li>• If not, OO flows will continue to work as before. No manual changes are required.</li> <li>• if yes, update the integration. For details, see <a href="#">"Operations Orchestration (OO) integration" on page 74</a>.</li> </ul>
Universal CMDB (UCMDB) CM	CM will specify a Service (CI Identifier) to open a Change record in Service Manager.	Depend on whether you change the CI Identifier field to a sequential number after upgrading to SM9.41: <ul style="list-style-type: none"> <li>• If not, no manual changes are required.</li> <li>• if yes, change the default Affected Service value specified for creating Change records in SM to a CI Identifier (logical.name) value.</li> </ul>
OMU	OMU is integrated with Service Manager through HP Connect-It (CIT) or SCAuto.  For using SCAuto, required code	<ul style="list-style-type: none"> <li>• If using SCAuto, no manual changes are required.</li> <li>• If using CIT, wait until a future release of CIT that includes required code changes.</li> </ul>

Target product	Solution	Manual changes
	<p>changes are implemented in SM9.41.</p> <p>For using CIT, required code changes will be implemented in a future release of CIT.</p>	
OMW	<p>OMW is integrated with Service Manager through SCAuto. Required code changes are implemented in SM9.41.</p> <p><b>Note:</b> The use of CIT has been obsoleted for this integration.</p>	Not required.
Executive Scorecard	Code changes are required on the Executive Scorecard side but not implemented yet at the time of the SM9.41 release.	<p>Follow the instructions in <a href="#">"HP Executive Scorecard integration" on page 74</a> to update the integration.</p> <p>Alternatively, wait until a future release of Executive Scorecard that contains these changes.</p>
CSA	Cloud Server Automation (CSA) is integrated with Service Manager through OO flows.	<p>Depends on whether you change the CI Identifier field to a sequential number after upgrading to SM9.41:</p> <ul style="list-style-type: none"> <li>• If not, OO flows will continue to work as before. No manual changes are required.</li> <li>• if yes, make the same changes for the OO-SM integration. See <a href="#">"Operations Orchestration (OO) integration" on page 74</a>.</li> </ul>
TeMIP	TeMIP can integrate with SM to create Incident records in SM.	<p>Depend on whether you change the CI Identifier field to a sequential number after upgrading to SM9.41:</p> <ul style="list-style-type: none"> <li>• If not, no manual changes are required.</li> <li>• if yes, change the Affected Service value configured for creating Incident records to the CI Identifier value (CIxxxxx).</li> </ul>



## Release Control (RC) integration

Code changes have been implemented in Service Manager 9.41. You do not need to make any manual updates except that you must upgrade Release Control to version 9.21 p3, which implements required code changes in the RC side.

## Application Lifecycle Management (ALM) integration

After you upgrade to SM 9.41, your existing ALM integration needs the following manual changes because of the `display.name` field introduced in SM 9.41.

1. Add the CI display name related fields to the web service definitions for Change Management and Problem Management.

### Change Management

Field	Caption
<code>affected.item..sm.device.display.name</code>	ServiceName
<code>middle,logical.name..sm.device.display.name</code>	ConfigurationItemName
<code>middle,assets..sm.device.display.name</code>	AssetsName

### Problem Management

Field	Caption
<code>affected.item..sm.device.display.name</code>	ServiceName
<code>logical.name..sm.device.display.name</code>	PrimaryCIName

2. Rebuild the SM adapter.
3. Configure the new field mapping.

When retrieving the change/problem record from SM, you will get a “Service” value like “CI1001030”. Now you can use the new field “ServiceName”, and then you will get the value like before (“Applications”).

4. Change the default value for creating new change/problem records.

For example, previously, you assigned “Applications” to “Service” to create a change/problem record. Now you need to search for the CI Identifier in SM by display name (for example, the display name equals “Applications”), and replace the default “Service” name.

**Caution:** ServiceName and PrimaryCIName do not work for the Create or Update operation.

## Operations Orchestration (OO) integration

After upgrade to SM 9.41, your existing OO-SM integration needs the following changes because of the display.name field introduced in SM 9.41.

1. Add the following expressions to the SM Change Management web service definition.

```
logical.name in $L.file=jscall("DisplayName.convertNameToId", logical.name in
    $L.file, $L.file, "logical.name")

affected.item in $L.file=jscall("DisplayName.convertNameToId", affected.item in
    $L.file, $L.file, "affected.item")

assets in $L.file=jscall("DisplayName.convertNameToId", assets in $L.file,
    $L.file, "assets")
```

2. Add the following expressions to the SM Incident Management web service definition.

```
logical.name in $L.file=jscall("DisplayName.convertNameToId", logical.name in
    $L.file, $L.file, "logical.name")

affected.item in $L.file=jscall("DisplayName.convertNameToId", affected.item in
    $L.file, $L.file, "affected.item")
```

3. Add the following expressions to the SM Problem Management web service definition.

```
logical.name in $L.file=jscall("DisplayName.convertNameToId", logical.name in
    $L.file, $L.file, "logical.name")

affected.item in $L.file=jscall("DisplayName.convertNameToId", affected.item in
    $L.file, $L.file, "affected.item")
```

## HP Executive Scorecard integration

You need to apply the solution described in the following after an upgrade from SM 9.40 to SM 9.41.

**Note:** This solution applies only to an upgrade from SM 9.40 to SM 9.41. It does not apply for upgrades

from a version earlier than SM 9.40.

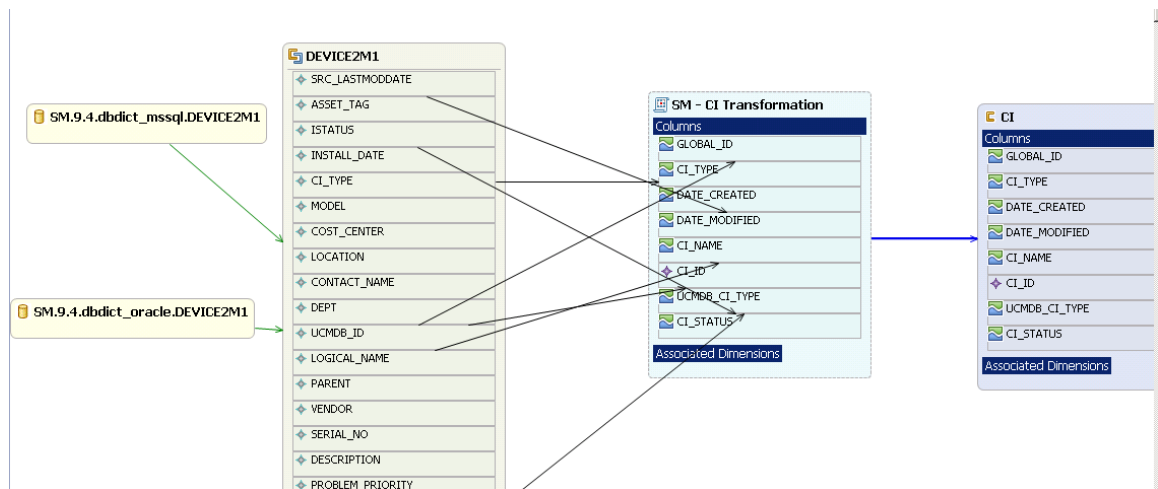
Changes need to be implemented in two modules:

- IDE metadata
- Workflows

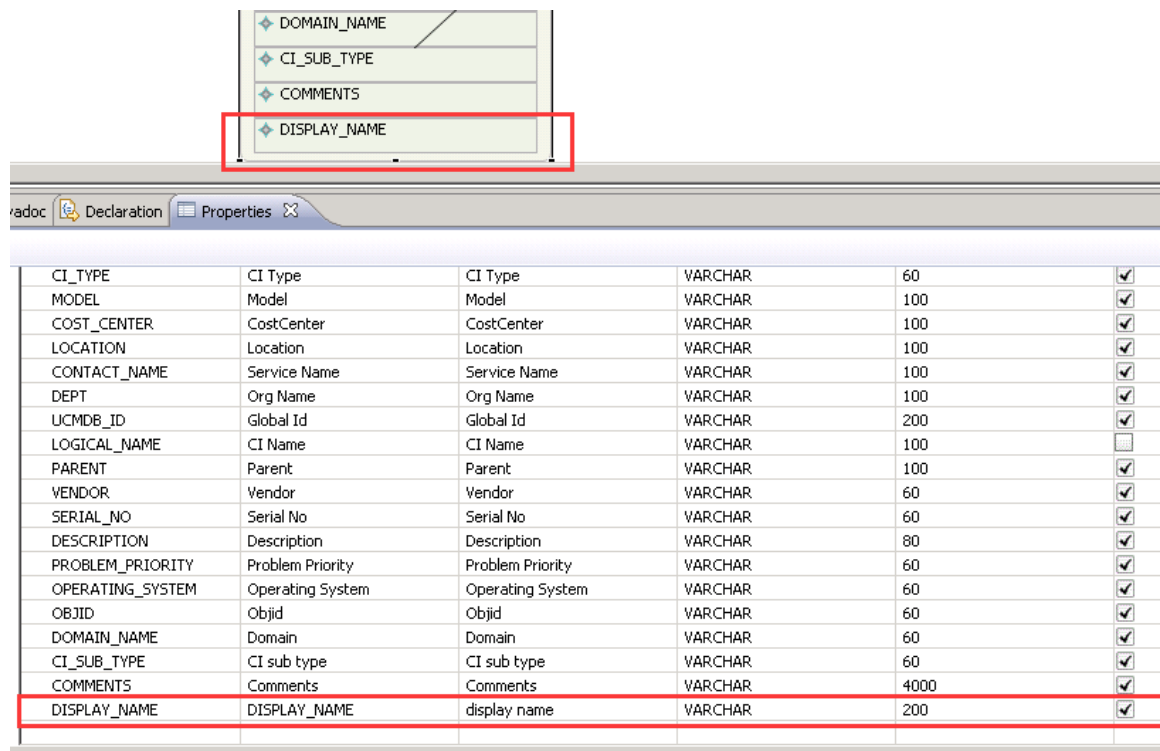
## IDE metadata changes

As we only support the upgrade from SM version 9.40 to 9.41, back up your IDE metadata first because we will remove the source models for older versions in the diagram. Additionally, only CI and SERVICE need IDE metadata changes; APPLICATION and NODE do not need IDE metadata changes but need only SSI workflow changes.

1. Open the engineer diagram for CI, and remove the source models for older versions. Keep only SM 9.4, as shown in the following figure.

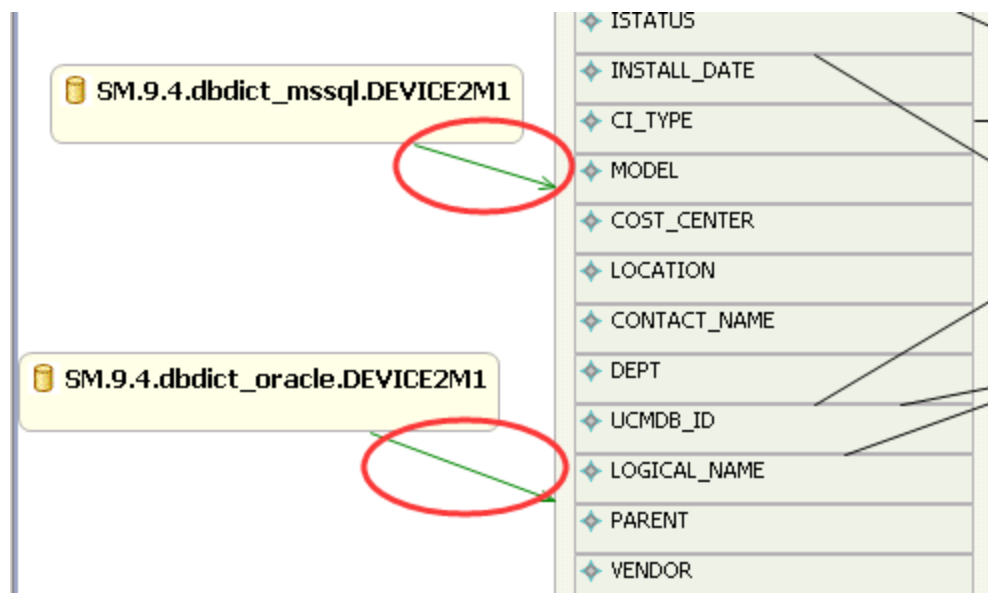


2. Add one column 'DISPLAY\_NAME' at the integration level, set its data type to VARCHAR (200), and allow NULL.



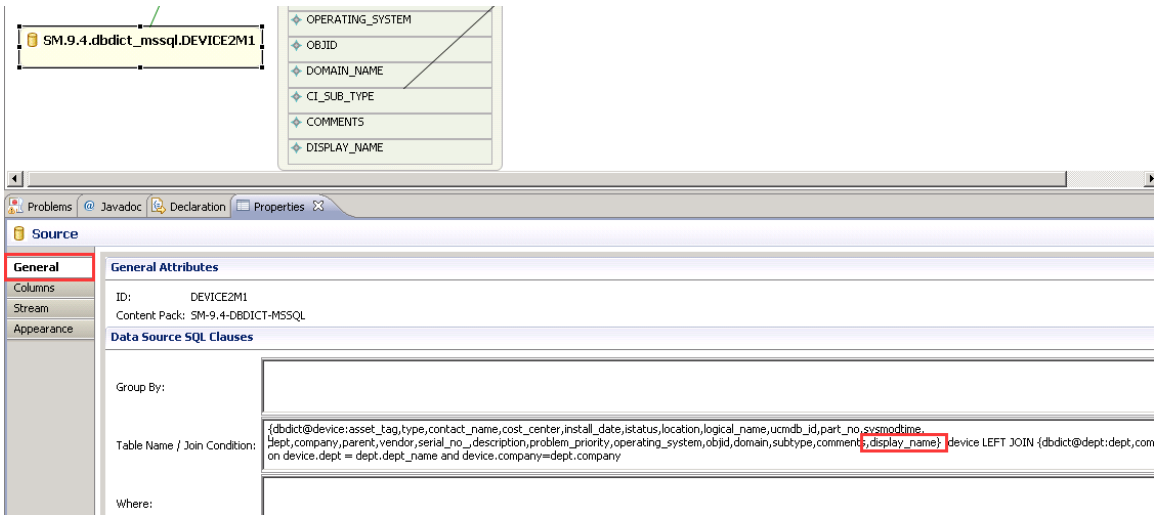
CI_TYPE	CI Type	CI Type	CI Type	CI Type	CI Type	CI Type
MODEL	Model	Model	Model	Model	Model	Model
COST_CENTER	CostCenter	CostCenter	CostCenter	CostCenter	CostCenter	CostCenter
LOCATION	Location	Location	Location	Location	Location	Location
CONTACT_NAME	Service Name	Service Name	Service Name	Service Name	Service Name	Service Name
DEPT	Org Name	Org Name	Org Name	Org Name	Org Name	Org Name
UCMDB_ID	Global Id	Global Id	Global Id	Global Id	Global Id	Global Id
LOGICAL_NAME	CI Name	CI Name	CI Name	CI Name	CI Name	CI Name
PARENT	Parent	Parent	Parent	Parent	Parent	Parent
VENDOR	Vendor	Vendor	Vendor	Vendor	Vendor	Vendor
SERIAL_NO	Serial No	Serial No	Serial No	Serial No	Serial No	Serial No
DESCRIPTION	Description	Description	Description	Description	Description	Description
PROBLEM_PRIORITY	Problem Priority	Problem Priority	Problem Priority	Problem Priority	Problem Priority	Problem Priority
OPERATING_SYSTEM	Operating System	Operating System	Operating System	Operating System	Operating System	Operating System
OBJID	Objid	Objid	Objid	Objid	Objid	Objid
DOMAIN_NAME	Domain	Domain	Domain	Domain	Domain	Domain
CI_SUB_TYPE	CI sub type	CI sub type	CI sub type	CI sub type	CI sub type	CI sub type
COMMENTS	Comments	Comments	Comments	Comments	Comments	Comments
DISPLAY_NAME	DISPLAY_NAME	display name	display name	display name	display name	display name

3. Remove Source to Integration Relation, and reconnect Source and Integration.



4. At the Source level, go to the General pane, and add the 'display\_name' column in the source table

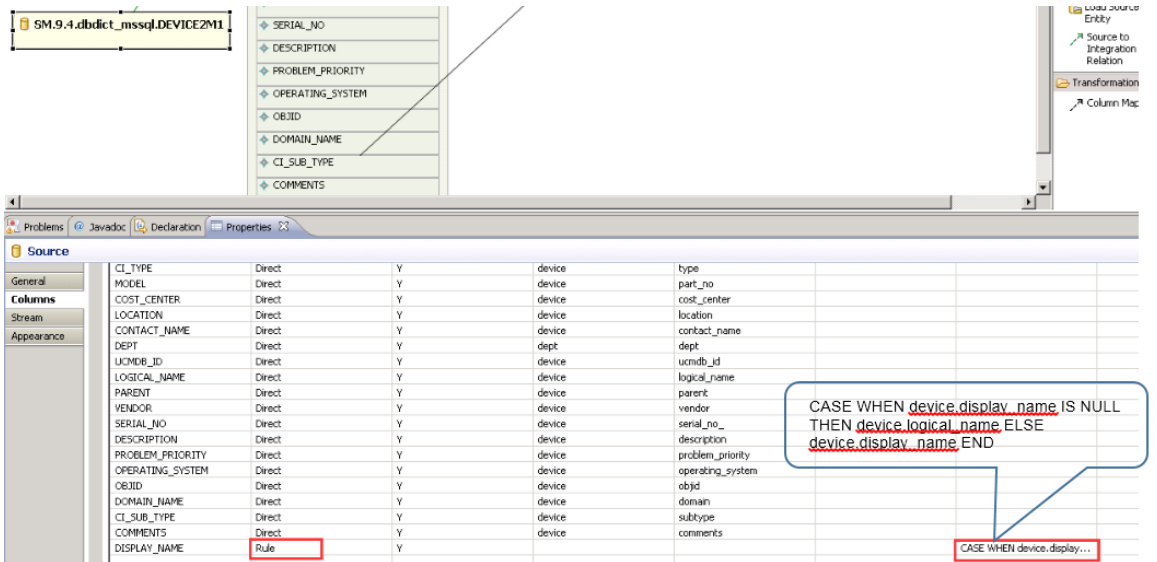
definition part. See the highlighted part on the right side of the following figure (using dbdict\_mssql as an example).



5. At the Source level, go to the Columns pane, and set the value for DISPLAY\_NAME.

After upgrading from version 9.40 to 9.41, Service Manager keeps the data from SM9.40. However, after the data upgrade from 9.40, device2m1.display\_name is NULL. So we need to add a condition for setting this value. To do so, select the **Rule** type and type the following Calculation Rule:

```
CASE WHEN device.display_name IS NULL THEN device.logical_name ELSE
device.display_name END
```



6. Repeat the same steps for dbdict\_oracle.

**Note:** If you are using an Oracle database, follow this step. Otherwise skip this step.

Source table definition:

```
{dbdict@device:asset_tag,type,contact_name,cost_center,install_
date,istatus,location,logical_name,ucmdb_id,part_no,sysmodtime,dept,company,
parent,vendor,serial_no_,description,problem_priority,operating_
system,objid,domain,subtype,comments,display_name} DEVICE LEFT JOIN
{dbdict@dept:dept,company,dept_name} DEPT on DEVICE."dept" = DEPT."dept_name" and
DEVICE."company"=DEPT."company"
```

Calculation Rule for DISPLAY\_NAME:

```
CASE WHEN "DEVICE"."display_name" IS NULL THEN "DEVICE"."logical_name" ELSE
"DEVICE"."display_name" END
```

7. Repeat the same steps for the SERVICE entity. There is no need to add a new column at the integration level. Simply make the same changes as you do for CI at the source level.

Source table definition in the General pane:

dbdict\_mssql:

```
{dbdict@device:problem_priority,ucmdb_id,sysmodtime,logical_
name,istatus,type,display_name} device LEFT JOIN
```

```
{dbdict@bizservice:service_description,logical_name,problem_manager,service_
status} bizservice ON device.logical_name = bizservice.logical_name
```

dbdict\_oracle:

```
{dbdict@device:problem_priority,ucmdb_id,sysmodtime,logical_
name,istatus,type,display_name} DEVICE LEFT JOIN {dbdict@bizservice:service_
description,logical_name,problem_manager,service_status} BIZSERVICE ON
DEVICE."logical_name" = BIZSERVICE."logical_name"
```

Calculation Rule for DISPLAY\_LABEL in the Columns Pane:

dbdict\_mssql:

```
CASE WHEN device.display_name IS NULL THEN device.logical_name ELSE
device.display_name END
```

dbdict\_oracle

```
CASE WHEN "DEVICE"."display_name" IS NULL THEN "DEVICE"."logical_name" ELSE
"DEVICE"."display_name" END
```

8. Save all changes and generate ETL.
9. Copy the generated SM folder to this directory: ..\HPXS\agora\ContentPacks

**Caution:** DO NOT copy SM\_CI\_SSI\_WF.xml and SM\_SERVICE\_SSI\_WF.xml. Remove the two files before you copy the entire SM folder.

10. Redeploy Service Manager. Run the following command under this directory:

```
..\HPXS\agora\DataWarehouse\bin dw_ds_automation.bat -task Redeploy -cp SM
```

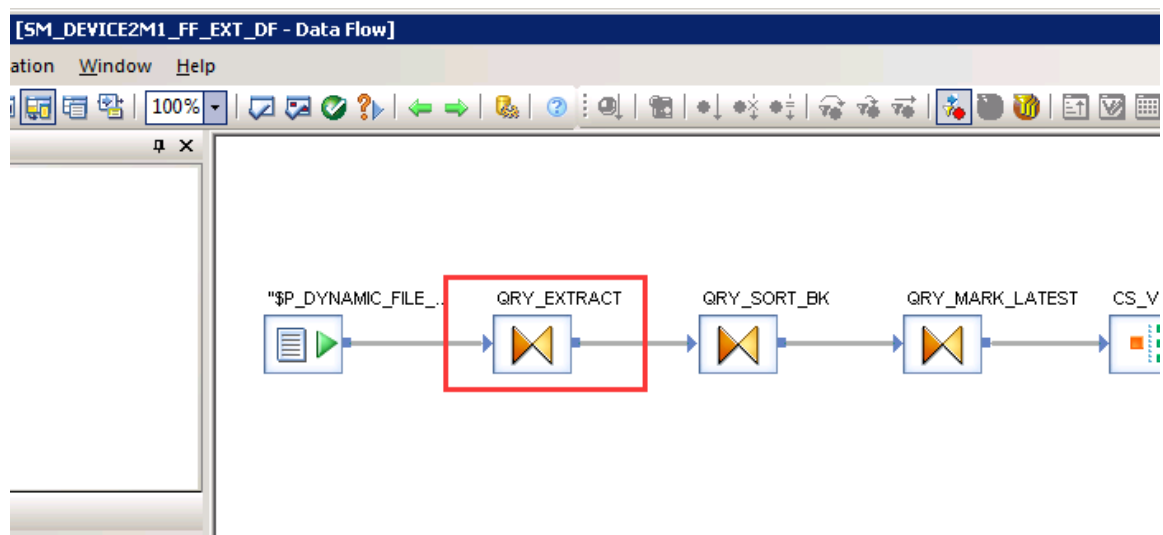
## Workflow changes

Changes are need for the following workflows.

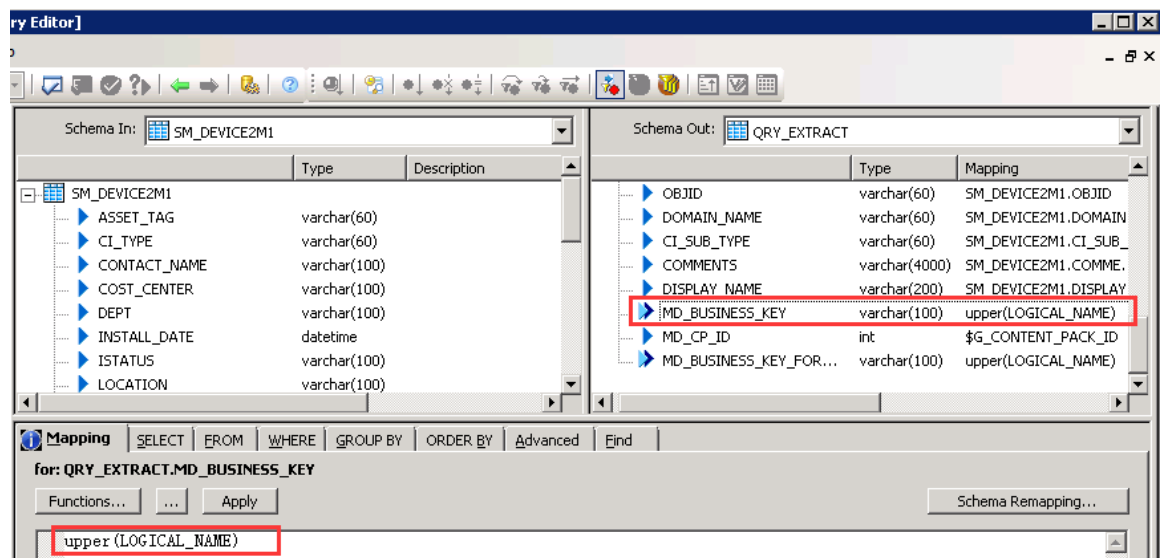
### **SM\_DEVICE2M1\_FF\_EXT\_WF**

We have customization in this workflow, and MD\_BUSINESS\_KEY must be in upper case.

1. Open the query highlighted in the following figure.



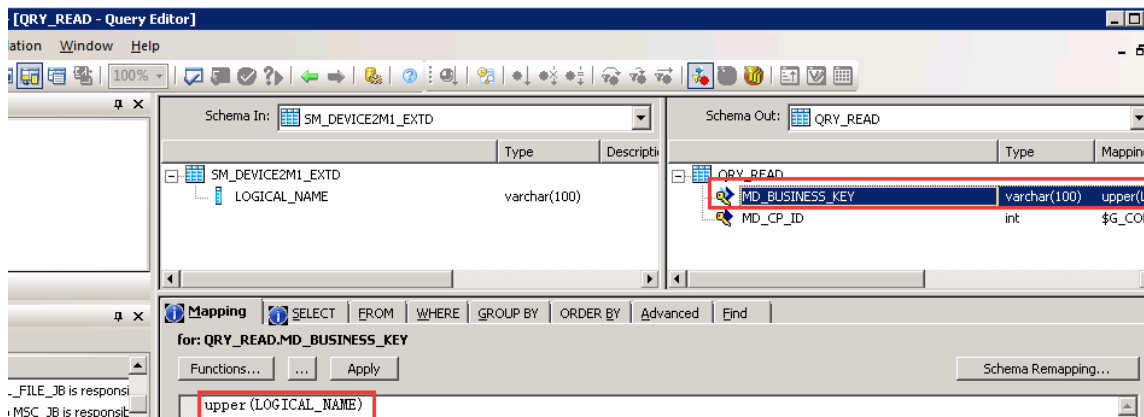
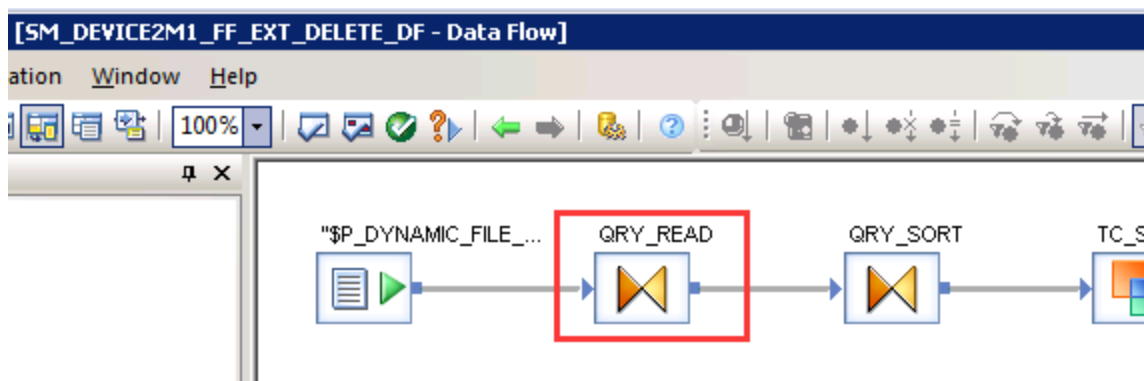
2. Add the **upper** function to MD\_BUSINESS\_KEY.



## SM\_DEVICE2M1\_FF\_EXT\_DELETE\_WF

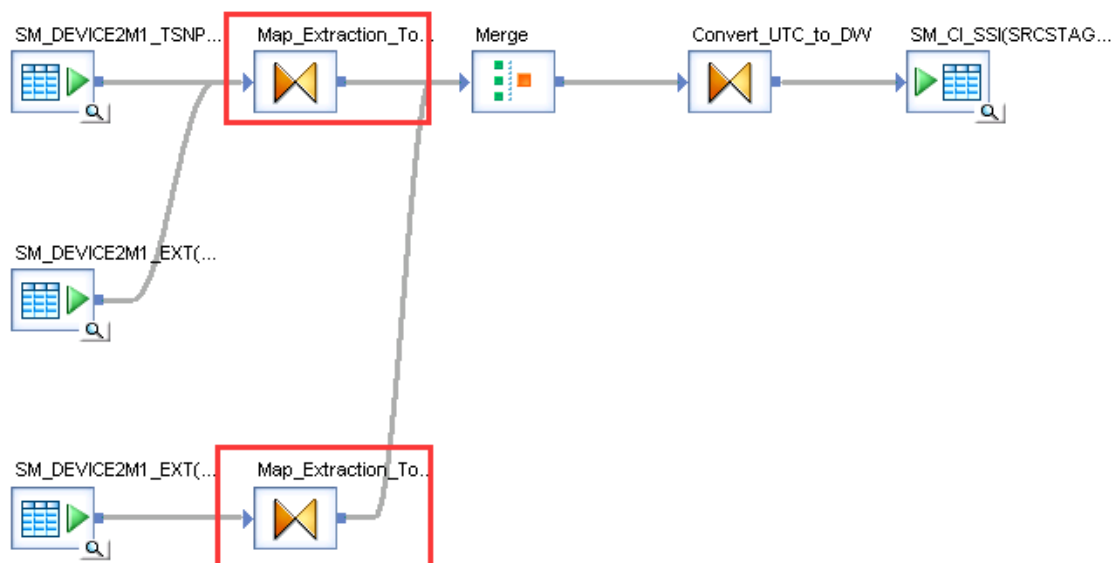
Similarly, we need to add the **upper** function to MD\_BUSINESS\_KEY for this workflow.



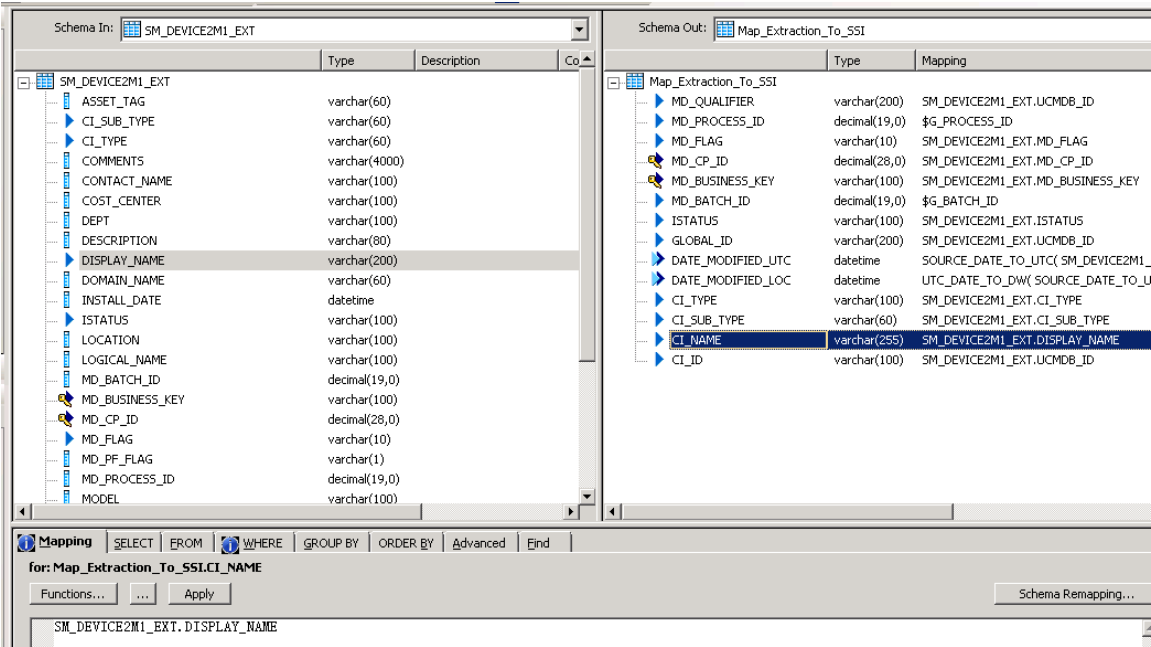


## SM\_CI\_SSI\_WF

You need to change the two queries highlighted in the following figure.

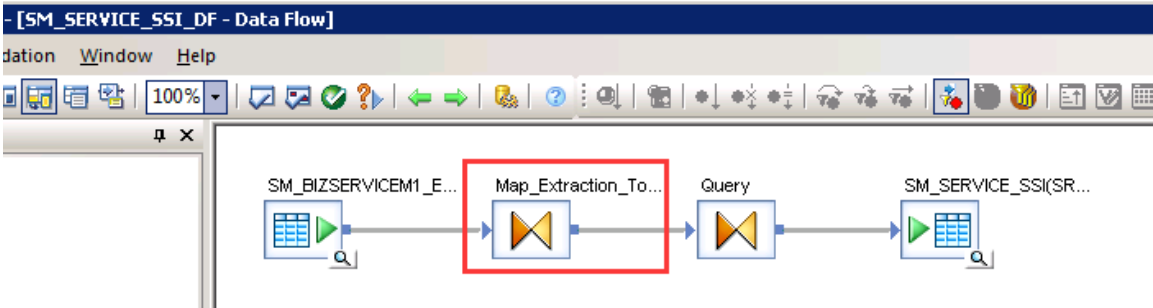


In each query, remap DISPLAY\_NAME to CI\_NAME.

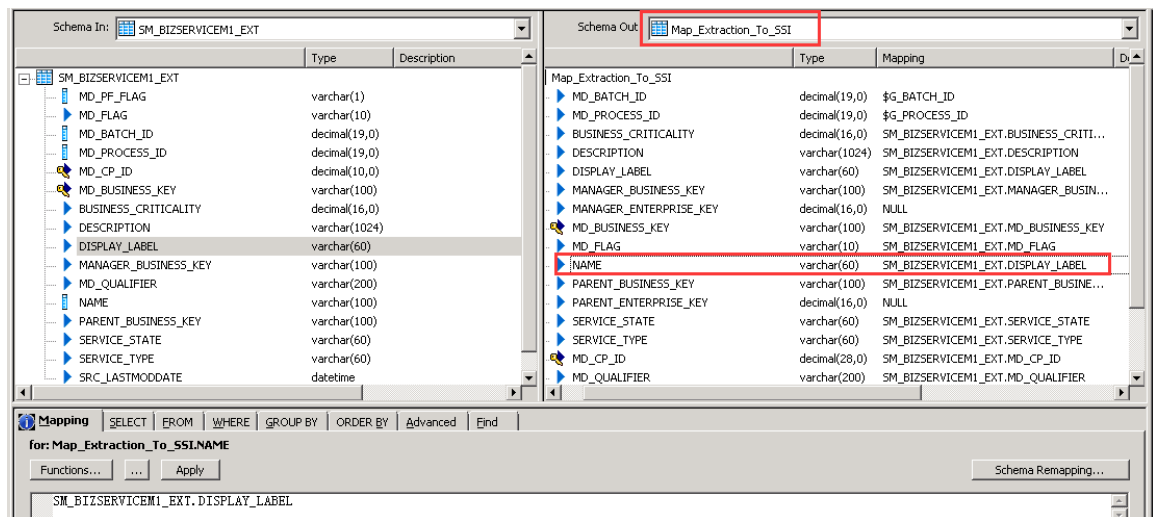


## SM\_SERVICE\_SSI\_WF

1. Open the mapping highlighted in red in the following figure.



## 2. Remap SM\_BIZSERVICEM1\_EXT.DISPLAY\_LABEL to NAME.



## SM\_APPLICATION\_SSI\_WF

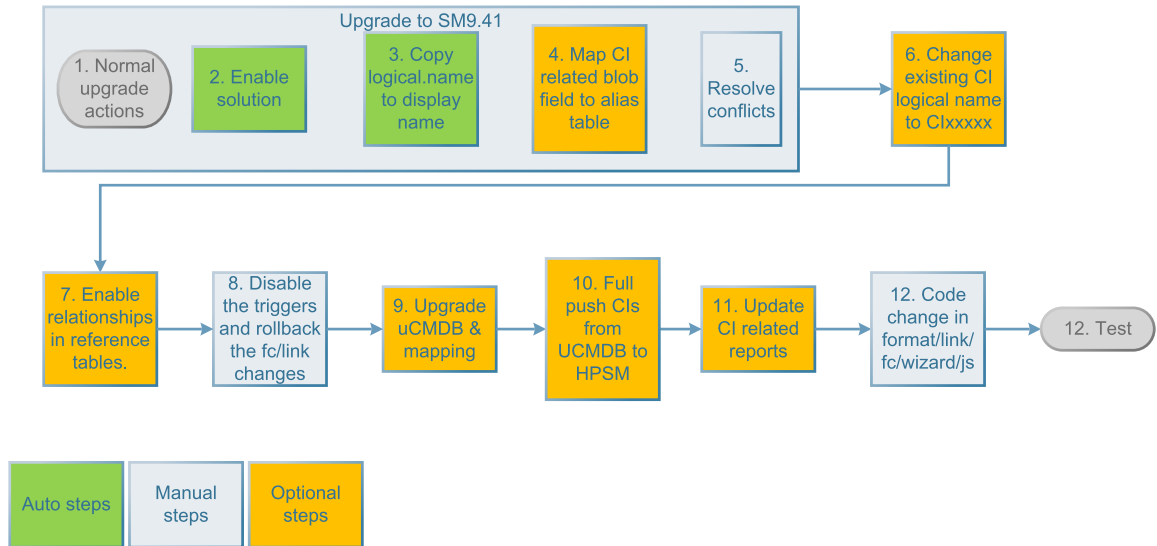
Like for CI, make changes in two queries to remap APPLICATION\_NAME and APPLICATION\_NAME\_ALT to SM\_DEVICE2M1\_EXT.DISPLAY\_NAME instead of SM\_DEVICE2M1\_EXT.LOGICAL\_NAME.

## SM\_NODE\_SSI\_WF

Like for CI, make changes in two queries to remap NODE\_NAME to SM\_DEVICE2M1\_EXT.DISPLAY\_NAME instead of SM\_DEVICE2M1\_EXT.LOGICAL\_NAME.

# Upgrade process of customers of type B or C

For customers of type B or C who implemented the HP workaround for the logical name issue, the upgrade process is illustrated in the following figure.

**FIGURE 3-1: Upgrade process for customers of type B or C (using the HP workaround)**

Besides the attention points for customers of type A, customers of type B or C should pay attention to the following additional points:

- As the display.name field in the **device** table was introduced by the previous workaround, as described in the “Solution details” section, customers should set Display Field to the display.name field at the data policy level.
- Customers should remove the display.name field in the formats of the reference modules, such as the Incident, Problem, and Change modules. Additionally, customers should disable all relevant triggers and roll back the previous fc/link changes introduced by the workaround.

## Upgrade steps for customers of type B or C

Customers of either type B or type C are those who have adopted the logical name workaround provided by HP through a white paper. The detailed upgrade steps for these types of customers are basically the same as those for customers of type A (who have not applied the workaround), except that the following additional step is required.

After upgrade to SM9.41, disable the triggers and roll back the format control/link changes you have made for the workaround according to that white paper. This is step 8 in ["Upgrade process of customers of type B or C" on the previous page](#).

# Troubleshooting

This section provides information that can assist you in troubleshooting issues that are associated with the logical name solution.


- Troubleshooting - search .....85
- Troubleshooting - view/display .....86
- Troubleshooting - Fill/Auto Complete .....87
- Troubleshooting - Find/View Detailed Information .....90
- Troubleshooting - Service Catalog user selections .....92
- Troubleshooting - field mandatory validation (Classic mode) .....94
- Troubleshooting - inbound integrations .....96
- Troubleshooting - outbound integrations .....96
- Troubleshooting checklist .....98

## Troubleshooting - search

### Issue: Cannot search by CI display name


**Description**

No search results are returned when searching by CI display name. For example, in the Change Task search form, if you enter CI Display Name “Adobe” for Affected CI and click **Search**, no change tasks are found.



08/11/15 19:57:06

Field \"ta01.asset\" used in query is not mapped or mapped to Blob/Clob/Text/Image. (cm.search,select)



08/11/15 19:57:06

No Tasks found to satisfy search arguments.

**Root cause**

This is because the Affected CI (cm3t/asset) field is of the LOB type but not mapped to an alias table.

## Solution

In the dbdict record, map the LOB type field to an alias table.

# Troubleshooting - view/display

## Issue 1: CI Identifier is displayed in detail forms

### Description

CI Identifier (logical.name) is displayed instead of CI Display Name in detail formats.

Incorrect:

Primary Affected Service:

A screenshot of a form field labeled 'Primary Affected Service:'. The field contains the text 'CI1001060'. To the right of the text are three icons: a document with a magnifying glass, a list icon, and a refresh icon.

Correct:

Primary Affected Service:

A screenshot of a form field labeled 'Primary Affected Service:'. The field contains the text 'Printing (North America)'. To the right of the text are three icons: a document with a magnifying glass, a list icon, and a refresh icon.

### Root Cause

Any of the following:

- The Referenced Table in the datadict is not configured.
- The widget on the form is not a Comfill, Label, or Text (read-only) control.
- The current field is a variable, instead of a real field that is defined in the dbdict.

## Solution

Check the related datadict to make sure the current field's Referenced Table is configured correctly and the form widget is a Comfill, Label, or Text control, and then log out and log in to system again.

If the current field is a variable, you must use the Value List/Display List function of the widget to convert CI Identifier to Display Name.

## Issue 2: CI Identifier is displayed in virtual join forms

### Description

CI Identifier (logical.name) is displayed instead of CI Display Name in Virtual Join formats.

Incorrect:

CMDB attributes need to be changed for CIs in the list

Field Name	CI	Old Value	New Value	Status
Admin Password	CI1002508		1234	planned

Correct:

CMDB attributes need to be changed for CIs in the list

Field Name	CI	Old Value	New Value	Status
Admin Password	iwfvm05806		1234	planned

### Root Cause

The Referenced Table in the datadict is not configured.

### Solution

Check the datadict of the virtual join target table to make sure the Referenced Table of the corresponding field is configured correctly, and then log out and log in to system again.

## Troubleshooting - Fill/Auto Complete

**Note:** For fields that have a referenced table configured in the datadict, such as “Primary Affected Service” and “Affected CI” in Incident records, the system filters the referenced data by Display Name when performing Fill or Auto Complete.

### Limitation

Fill and Auto Complete do not work well either for Variable fields or for CI related fields in the Advanced Filter. See the following figures for examples.

Affected CI:

**\$ci.list**

No description  
file=sloavail  
form=wizard-wiz.slo.avail.1  
field=\$ci.list

Field in Incident: Primary Affected Service

[Use Multi Level Field Chooser](#)

Comparison: ☐ Not Starts With

Value:

## Issue 1: Cannot find CIs matching an existing display name

### Description

When selecting a CI for a reference field, CI records that match an existing CI Display Name cannot be found.

### Root Cause

Any of the following:

- The Referenced Table in the datadict is not configured correctly.
- The link query is incorrect.

### Solution

1. Check the datadict to make sure the current field's Referenced Table is set to be the same as the target table defined in the related link.
2. Check that the link query definition is correct. There are two ways to write a link query for the logical name solution. See the following figures for examples.

**Skip Query Rewriting** is not enabled:



Field (From/Source):	File (To/Target):	Format (To/Target):
affected.item	device	
Comment:		
Query:	\$query	
QBE Format:	affected.item.qbe	Structured Array Name:
<input checked="" type="checkbox"/> Skip Query Rewriting		
<div>Expressions</div> <div>JavaScript</div> <pre> 1 if (\$fill.skip.master=true) then (\$fill.skip=true;cleanup(\$fill.skip.master)) 2 \$query="device.type=\"bizservice\"" 3 if (not (null(affected.item in \$File))) then \$query+=(" and logical.name#\\"+affected.item in \$File+"\\"") </pre>		

**Skip Query Rewriting** is enabled:

Field (From/Source):	File (To/Target):	Format (To/Target):
affected.item	device	
Comment:		
Query:	\$query	
QBE Format:		
<input checked="" type="checkbox"/> Skip Query Rewriting		
<div>Expressions</div> <div>JavaScript</div> <pre> 1 if (\$fill.skip.master=true) then (\$fill.skip=true;cleanup(\$fill.skip.master)) 2 \$query="device.type=\"bizservice\"" 3 \$L.affected.item.display.value=get.display.value(\$File,"affected.item") 4 if (not (null(\$L.affected.item.display.value))) then \$query+=(" and display.name#\\"+\$L.affected.item.display.value+"\\"") </pre>		

For more information, see ["Link \(Fill, View Detail/Find, Auto Complete, and new or updated RAD/JavaScript functions\)"](#) on page 43.

## Issue 2: Auto Complete does not work

### Description

The Fill function works well but Auto Complete does not.

### Root cause

Any of the following:

- Combo is enabled for the Comfill control.
- The input of the current field is not configured in the field mapping in the link record.
- The Fill function is customized in such a way that it bypasses the out-of-box link mechanism.

### Solution

Fix the issue accordingly:

- Disable Combo for the Comfill control.
- Configure the input of the current field in the field mapping of the link record.
- Re-implement the JavaScript interface of LinkUtilOverride.getCustomizedLink for the Auto Complete feature. Before reimplementing the script, see the description provided in the script.

**ScriptLibrary**

Name:  Package:

```

1  /**
2  * This function is left to customers to customize their dynamic links instead of using links defined in link table for auto complete functionality
3  *
4  * @param record (SM file)
5  * @param format (String)
6  * @param field (String)
7  * @param optionValue (String)
8  * @return The link line definition, or null if not found
9  *
10 * The record against which the master link record is defined
11 * The format against which the form-based link record is defined, if not given, the current format will be used
12 * The source field against which the link line is defined
13 * The content user has typed for current field
14 * @return The link line definition, or null if not found
15 *
16 * var result = {
17 *   // The field the user is filling data
18 *   sourceField: 'affected.item',
19 *   // The table name which system will query data against
20 *   targetFile: 'device',
21 *   // The field of the targetFile which the query will base on
22 *   targetField: 'logical.name',
23 *   targetFormat: 'device.qbe',
24 *   query: 'logical.name$App',
25 *   // If the link has post expressions or javascripts
26 *   hasPostStatement: true/false
27 * }
28 * //Object: Source/Target fields
29 * mapping: {'logical.name': 'affected.item', 'assignment': 'assignment'}
30 *
31 * function getCustomizedLink(record, format, field, optionValue){
32 * //add your customized code for link and return as an object
33 * //var link = GET_LINK_FUNCTION; */
34 * //before return link, please eval pre javascript and pre RAD, below is as an example
35 * //
36 * if(link!=null){
37 *
38 *   return {
39 *     sourceField: link['source fields'][0],
40 *     targetFile: link['target files'][0],
41 *     targetField: link['target fields'][0],
42 *     targetFormat: link['target formats'][0],
43 *     query: query,
44 *     hasPostStatement: hasPostStatement,
45 *     mapping: mapping
46 *   };
47 * }
48 * }
49 *
50 * }

```

For more information, see ["Link \(Fill, View Detail/Find, Auto Complete, and new or updated RAD/JavaScript functions\)" on page 43.](#)

## Troubleshooting - Find/View Detailed Information

**Note:** For fields that have a referenced table configured in the datadict (such as “Primary Affected Service” and “Affected CI” in Incident), the system by default does not use the query defined in the link to perform the Find action any more. Instead, the system dynamically generates a query by using the target table’s unique key field and performs an exact match search against the target table.

### Limitation

Find or View Detailed Information does not work either for Variable fields, or for “CI” related fields in the Advanced Filter.

## Issue: Customized “View Affected Services” functionality does not work

### Description

The customized “View Affected Services” functionality does not work.

The screenshot shows the Service Manager interface with the following fields and values:

- Incident ID: IM10002
- Status: Categorize
- Phase: Categorization
- Primary Affected Service: Printing (North America)
- Affected CI: adv-nam-copier-mar
- CI is operational (no outage): ☐
- Outage Start Time:

The 'More' dropdown menu is open, showing the following options:

- Export/Unload
- Visualize CI
- Send Survey
- View Alert Log
- Solution Matching
- Create Template from Record
- Notify
- Notes
- Copy Record
- Set Reminder
- Change Category
- Create Knowledge
- Search Knowledge
- Generate Maintenance
- View Affected Services

### Root cause

This functionality is implemented by calling the us.link RAD application to find out the related CI records.

### Solution

Pass one more parameter “cond.input” as true when calling the us.link RAD application, which will make the “Find” function still use the query defined in the link record. See the following figure for an example.

us.link		Condition
Parameter Names		Parameter Values
record		\$L.file
name	<b>cond.input: true</b>	"logical.name"
second.record		\$L.link
prompt		"find"
cond.input		true

## Troubleshooting - Service Catalog user selections

Issue: CI Identifier is displayed when the “Record in Table” type is used

### Description

When a user selection is defined with the “Record in Table” type, CI Identifier instead of Display Name is displayed. See the following figures for an example.

### Dynamic Field Validations

You may define a validation rule for a string field that will force the user to select a value that matches a record in another table.



Validation Rule

☐ None
☒ Record in Table

Table Name:
probsummary

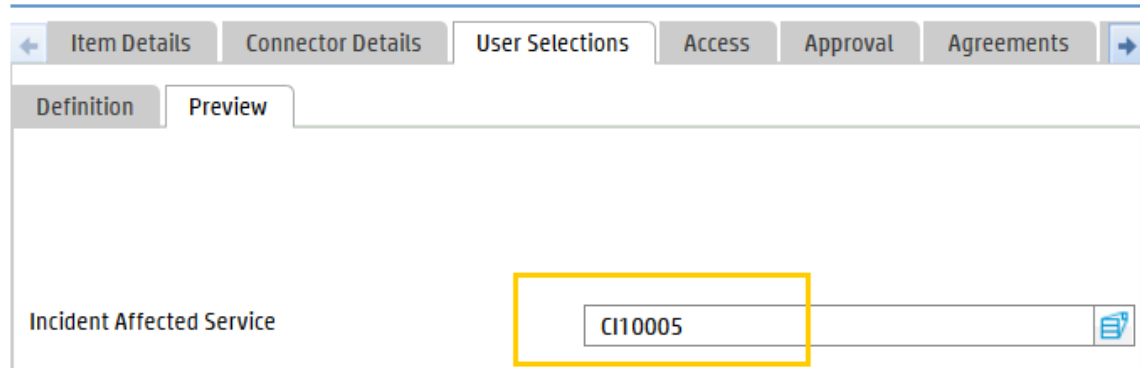
Field Name:
Primary Affected Service

Query:
Edit

Multiple Selections:

Group by Fields:

## Catalog Item Definition



The screenshot shows the 'Catalog Item Definition' interface. At the top, there are tabs: 'Item Details', 'Connector Details', 'User Selections', 'Access', 'Approval', and 'Agreements'. Below these, there are sub-tabs: 'Definition' and 'Preview'. The 'Definition' sub-tab is active. In the main area, the label 'Incident Affected Service' is followed by a text input field containing 'CI10005'. A yellow box highlights the input field. To the right of the input field is a small icon representing a document or list.

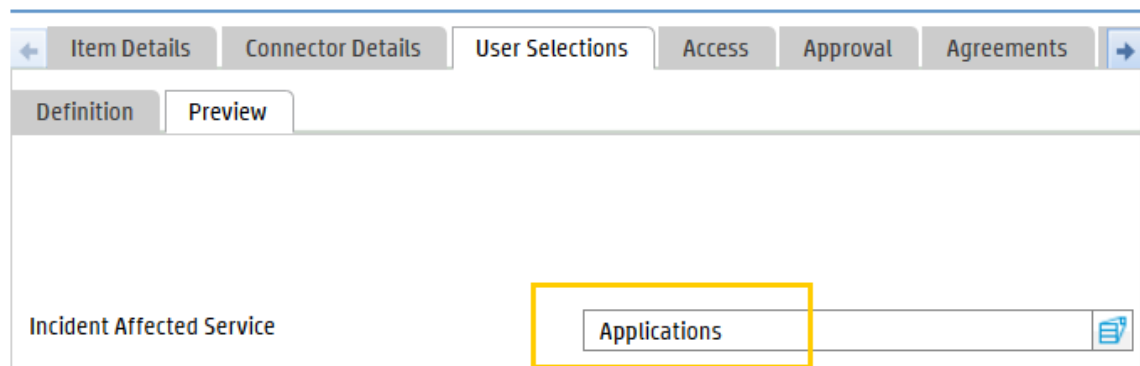
### Root Cause

In the datadict for the **probsummary** table, the Referenced Table for the Affected Service field is not set to **device**.

### Solution

In the datadict, set the Referenced Table for the field to **device**. CI Display Name is then displayed as shown in the following figure.

## Catalog Item Definition



This screenshot is similar to the one above, showing the 'Catalog Item Definition' interface. The 'Incident Affected Service' field now contains the text 'Applications'. A yellow box highlights the input field. The rest of the interface, including the tabs and sub-tabs, remains the same.

# Troubleshooting - field mandatory validation (Classic mode)

## Issue: Incorrect mandatory field validation message

### Description

When the user has entered an incorrect value in a reference field and then attempt to save the record, the system displays a message that resembles the following: "XXX the field is required." For example, if you enter "aaaa" in the Primary Affected Service field in an Incident record, the following message is displayed when you click **Save**:

Primary Affected Service is required.

Incorrect:

#### Incident - IM10002

The screenshot shows the 'Incident - IM10002' form in Classic mode. The form has several fields: Title, Description, Incident ID, Status, Phase, Primary Affected Service, and Major Incident. The Title field contains 'Not able to print due to lack of ink in printer.' and the Description field contains 'Seldom ink exist in printer, there's alway error, when printing.' The Incident ID is 'IM10002'. The Status is 'Categorize' and the Phase is 'Categorization'. The Primary Affected Service field contains 'aaaa' and has a yellow warning icon next to it. The Major Incident field is empty. An 'Information' dialog box is displayed over the form, showing the message 'Primary Affected Service is required.' with an 'OK' button.

Title: \* Not able to print due to lack of ink in printer.

Description: \* Seldom ink exist in printer, there's alway error, when printing.

Incident ID: IM10002

Status: \* Categorize

Phase: Categorization

Primary Affected Service: aaaa

Major Incident: ☐

Information dialog box: Primary Affected Service is required. OK

Correct:

Incident - IM10002

Title:

\* Not able to print due to lack of ink in printer.

Description:

\* Seldom ink exist in printer, there's alway error, when printing.

Incident ID:

Status:

Phase:

Location:

Primary Affected Service:

aaaa

Major Incident:

☐

Information

Value for Primary Affected Service field is not valid. Please select a correct one.

OK

Root cause

The logical name mandatory validation solution for the reference field is not configured.

Solution

Configure the logical name mandatory validation solution in the formatctrl record. See the following table and figure.

Validation	Message
jscall("DisplayName.validateRefField", \$file, "affected.item")	\$reference.field.valiadtion.msg

not (r	not (r				false
logical.name				1011	Configuration Item may not be a group
\$G.fo					evaluate(new in tableAccess in \$G.pm.environment)
folder				1012	Select a folder for this Incident.
	\$G.fo				update in tableAccess in \$G.pm.environment~="never"
folder				1012	Select a folder for this Incident.
true	true				jscall("DisplayName.validateRefField", \$file, "affected.item")
affected.item					\$reference.field.valiadtion.msg
not (r	not (r				not (null(logical.name in \$file2))

# Troubleshooting - inbound integrations

For issues with inbound integrations, see ["General guidelines for inbound integrations" on page 67](#).

# Troubleshooting - outbound integrations

## Issue 1: Cannot create a CI

### Description

The CI is not created successfully.

### Root cause

The CI Display Name field (whose alias is sm.device.display.name) is not configured in the other endpoint product. However, the Display Name field is a No Nulls key.

### Solution

Specify CI Display Name for creating new CI records in the other endpoint product.

## Issue 2: Cannot create a non-CI record (Change, Incident, and so on)

### Description

The record is not created successfully.

### Root cause

A default CI Identifier (a logical.name value) is not configured correctly from the other endpoint product.

### Solution

Configure a CI Identifier instead of CI Display Name for creating a record.



## Issue 3: Cannot retrieve additional CI information

### Description

The other endpoint product system can not retrieve additional CI information such as Display Name from a Service Manager Change record.

### Root cause

Exposed fields in the Change related web service are not configured correctly.

### Solution

Check the web service configuration to see if the CI related fields are configured correctly. You can use the “[current table field]..[referenced table field]” format to retrieve any additional information from the referenced table. See the following table and figure for examples.

Field	Caption
affected.item..sm.device.display.name	ServiceName
affected.item..ucmdb.id	ServiceID

Service Name:  ☐ Released

Name:  ☐ Deprecated

Object Name:

◆ Allowed Actions ◆ Expressions ◆ **Fields** ◆ RESTful

Field	Caption	Type
closureComments	ClosureComments	
emergency	Emergency	
requestedDate	RequestedDate	DateTimeType
affected.item	Service	
rc.analysis,collision.severity	CollisionSeverity	
rc.analysis,impact.severity	ImpactSeverity	
rc.analysis,time.period.conflict	TimePeriodConfl...	
rc.analysis,risk.severity	RiskSeverity	
rc.analysis,suggestedStart	SuggestedStartD...	DateTimeType
rc.analysis,suggestedEnd	SuggestedEndDate	DateTimeType
header,assign.dept	AssignmentGroup	
affected.item..sm.device.display.name	ServiceName	
affected.item..ucmdb.id	ServiceID	

# Troubleshooting checklist

The following is a list of items that you may need to check when troubleshooting logical name solution related issues.

- The Referenced Table configuration in datadict
- The form widgets in Forms Designer
- The link query definition
- The Web Service configurations for integrations

## More information

For more information, please visit the HP Software Support Online website: <https://softwaresupport.hp.com>

This website 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 valued 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 website:

[http://www.hp.com/managementsoftware/access\\_level](http://www.hp.com/managementsoftware/access_level)

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

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

---

© Copyright 2015 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.

HP, Service Manager, Change Configuration and Release Management, and UCMDB are registered trademarks of Hewlett-Packard Development Company, L.P.

- Java™ is a registered trademark of Oracle and/or its affiliates.
- Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.
- Oracle® is a registered US trademark of Oracle Corporation.
- UNIX® is a registered trademark of The Open Group.