



Using automatic merge tools in conflict resolution

How to use KDiff3 for automatically merging records that were marked as conflicts during the application upgrade

Introduction.....	2
Merge strategies.....	2
Three-way merging.....	2
How to use KDiff3	3
What is KDiff3?.....	3
Merge verification.....	11
Procedure for merging format records	11
Special Considerations for using 3-way-merge	14
Considerations when using KDiff3.....	14
Considerations when using the 2-way merge in Service Manager.....	17
For more information.....	18

Introduction

An integral part of a successful upgrade is conflict resolution. While conflict resolution can be very time consuming, estimating that 1 person can solve around 100 conflicts per week, the success of the upgrade hinges on making the correct decisions for the system to work properly after the upgrade. This document introduces how to use the new 3-way merge functionality that was first introduced with upgrade patch 3 of Service Manager 9.30 for more educated decisions and easier merging of functionality between the customer version of a record and the out-of-box counterpart. Use this document in conjunction with other Service Manager Upgrade documentation. Detailed steps on how to perform an application upgrade, are documented the *Service Manager Upgrade Guide*.

Merge strategies

First, there are 2 basic strategies when going through a customer upgrade:

1. Use as much out-of-box functionality as possible (recommended by HP)
2. Stay as close to the customer version as possible

An additional third option is to merge in new out-of-box functionality while keeping the customer look and feel

The following are some pros and cons for each strategy:

Strategy	Pros	Cons
Close to Customer	<ul style="list-style-type: none">• No requirement for End User Training	<ul style="list-style-type: none">• New functionality of new version is not available, or only partially available• Increased amount of conflicts next time
Close to OOB	<ul style="list-style-type: none">• Next upgrade will be less conflicts• All new functionality is available	<ul style="list-style-type: none">• Large requirement for End User Training
Merged functionality	<ul style="list-style-type: none">• Some end user training required for new features, but look and feel stay the same• Fewer conflicts on next upgrade• All new functionality is available	<ul style="list-style-type: none">• During conflict resolution, the decision maker needs to know both the customer tailoring and the new functionality to make the right decision

Based on this basic strategy, there are 3 different merge strategies when you merge single objects:

- Use the customer version without changes
- Use the new OOB version without changes
- Merge functionality of OOB and customer

In this paper, we will concentrate on how to most efficiently merge OOB functionality with customer tailoring using the KDiff3 tool and the 2-way-merge function in the Service Manager upgrade results record.

Three-way merging

When a conflict occurs on an object, it involves three versions of that object:

- The out-of-box version that was originally released
- The current version of your system that was tailored
- The upgrade version that the Upgrade Utility tries to apply to your system

To make an educated decision during conflict resolution, you must compare both your tailored version and the upgrade version with the original out-of-box version to determine what has changed. For each

object marked as "Renamed" in the Upgrade Results list, the Upgrade Utility automatically generates XML files for the three versions of the object.

Version	Location	Description
base	<Upgrade>\3waymerge\work\base	An XML representation of every object that has been signed in the pre-upgrade out-of-box version.
customer	<Upgrade>\3waymerge\work\customer	An XML representation of all objects that were tailored in the customer version and resulted in a conflict during the upgrade.
upgrade	<Upgrade>\3waymerge\work\upgrade	An XML representation of the object provided by the upgrade package of all objects that resulted in a conflict.

Note: <Upgrade> represents the Upgrade path specified when applying an upgrade.

Each of the three folders described above contains a subfolder for each signed table. You can find the XML representations of the objects within the table within these subfolders. You may visually compare the three versions of each object using a three-way compare and merge tool outside Service Manager, and then load the merged version using Service Manager's 2-way merge utility from the Upgrade Results record.

How to use KDiff3

What is KDiff3?

KDiff3 can compare and automatically merge two or three text files or directories. It shows the delta between the input files both line by line and character by character. Visit the KDiff3 Web site (<http://kdiff3.sourceforge.net>) to obtain the KDiff3 software and related information.

KDiff3 is available for Windows and Gnu Linux. For Service Manager running on any other Unix flavors, we recommend that you run KDiff3 on Windows and transfer the files from your Unix server to your Windows client.

To compare and merge objects:

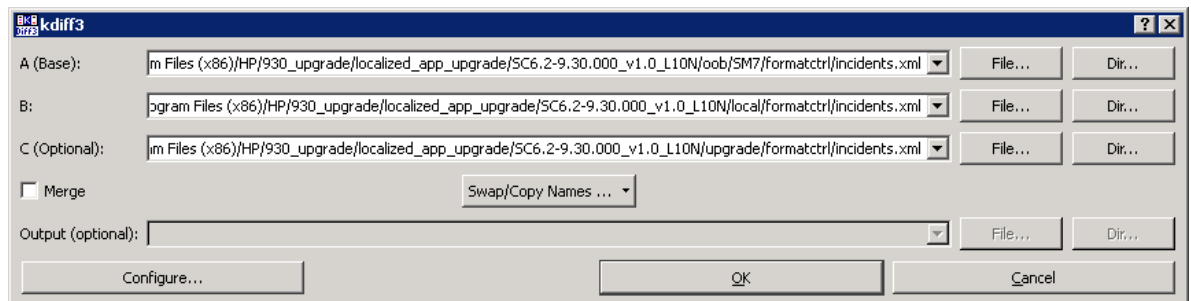
1. Start KDiff3, and specify the paths as follows

A(Base): <Upgrade>\3waymerge\work\base

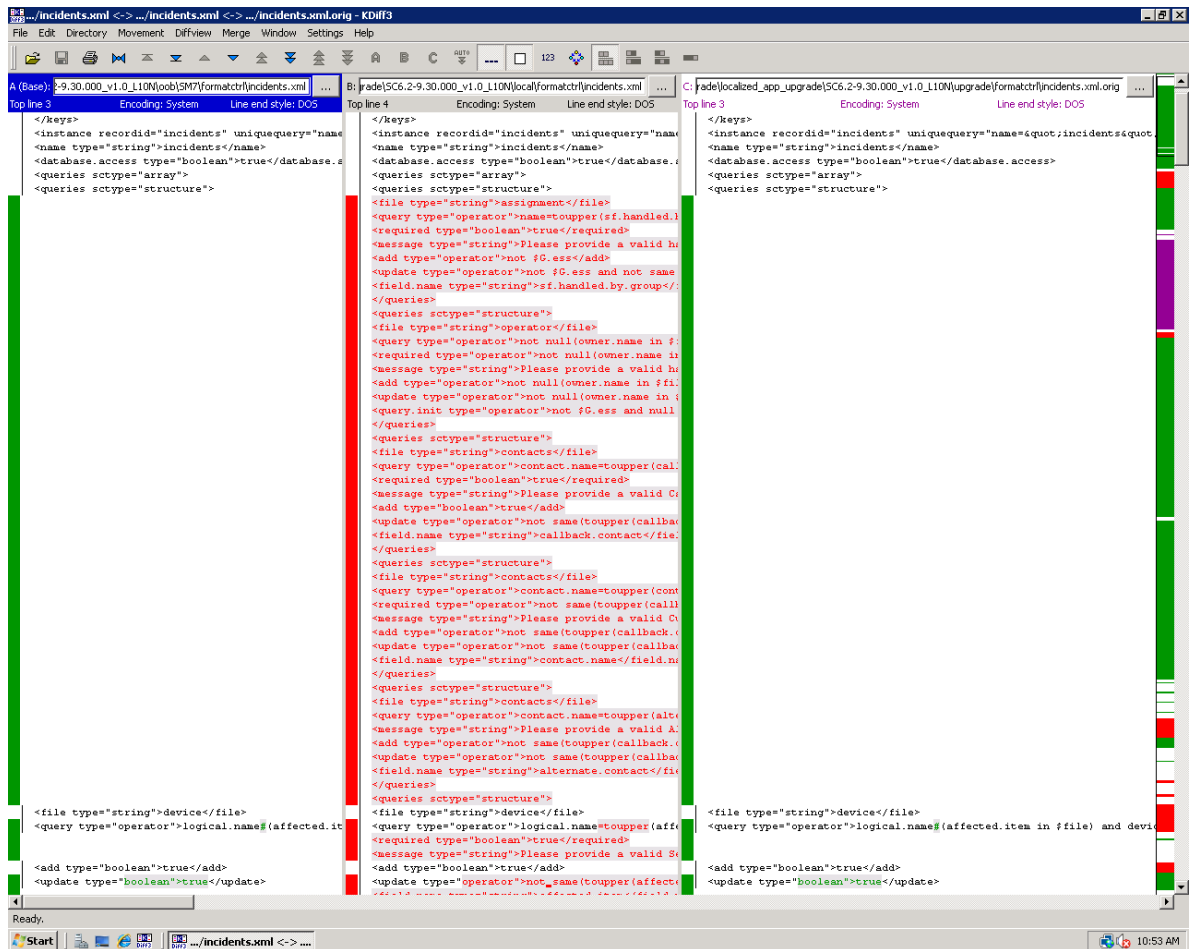
B: <Upgrade>\3waymerge\work\customer


C: <Upgrade>\3waymerge\work\upgrade

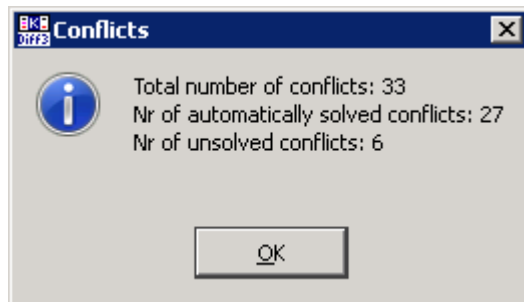
2. Navigate to the XML file for the object that you want to merge.



3. In the merge view, you can see the differences between the three versions to determine which changes between the customized version and the new out-of-box version were customer introduced:



4. Click the Merge button  to run an automatic merge.
5. A prompt will inform you whether the merge was successful or requires manual intervention:



6. In the example of the incidents formatctrl record, 6 conflicts could not be auto-merged.
7. To manually merge differences that cannot be automatically merged, search for both "merge conflicts" and "no src" lines that indicate conflicts that have to be resolved

```

Output: [am Files (-86)]HP930_upgrade\localized_app_upgrade\SC6.2-9.30.000_v1.0_110N\Upgrade\formatctrl\incidents.xml.orig Encoding for saving: Codec from C: System Line end style: DOS (A, B, C)
B <query type="operator">logical.name=toupper(logical.name in $file) and device.type="bisservice" and sf.active=true</query>
B <required type="boolean">true</required>
B <message type="string">Please provide a valid Affected CI.</message>
B <add type="operator">not null(logical.name in $file)</add>
B <update type="operator">not null(logical.name in $file) and not same(toupper(logical.name in $file), toupper(logical.name in $file0))</update>
B <field.name type="string">logical.name</field.name>
B </queries>
C <queries sctype="structure">
? <Merge Conflicts>
B <required type="operator">not null(category in $file)</required>
B <message type="string">Please provide a valid Subcategory.</message>
B <add type="operator">not null(category in $file)</add>
B <update type="operator">not null(category in $file)</update>
B <field.name type="string">subcategory</field.name>
B </queries>
B <queries sctype="structure">
B <file type="string">product.type</file>
B <query type="operator">category=category in $file and subcategory=subcategory in $file and product.type=product.type in $file and active=true</query>
B <required type="operator">not null(category in $file) and not null(subcategory in $file) and not null(subcategory in $file)</required>
B <message type="string">Please provide a valid Type.</message>
B <add type="operator">not null(category in $file) and not null(subcategory in $file) and not null(subcategory in $file)</add>
B <update type="operator">not null(category in $file) and not null(subcategory in $file) and not null(subcategory in $file)</update>
B <field.name type="string">product.type</field.name>
B </queries>
B <queries sctype="structure">
B <file type="string">location</file>
B <query type="operator">location=location in $file</query>
B <required type="boolean">false</required>
B </queries>
Number of remaining unsolved conflicts: 6 (of which 0 are whitespace)
Start | .../incidents.xml <-> ... 10:57 AM

```

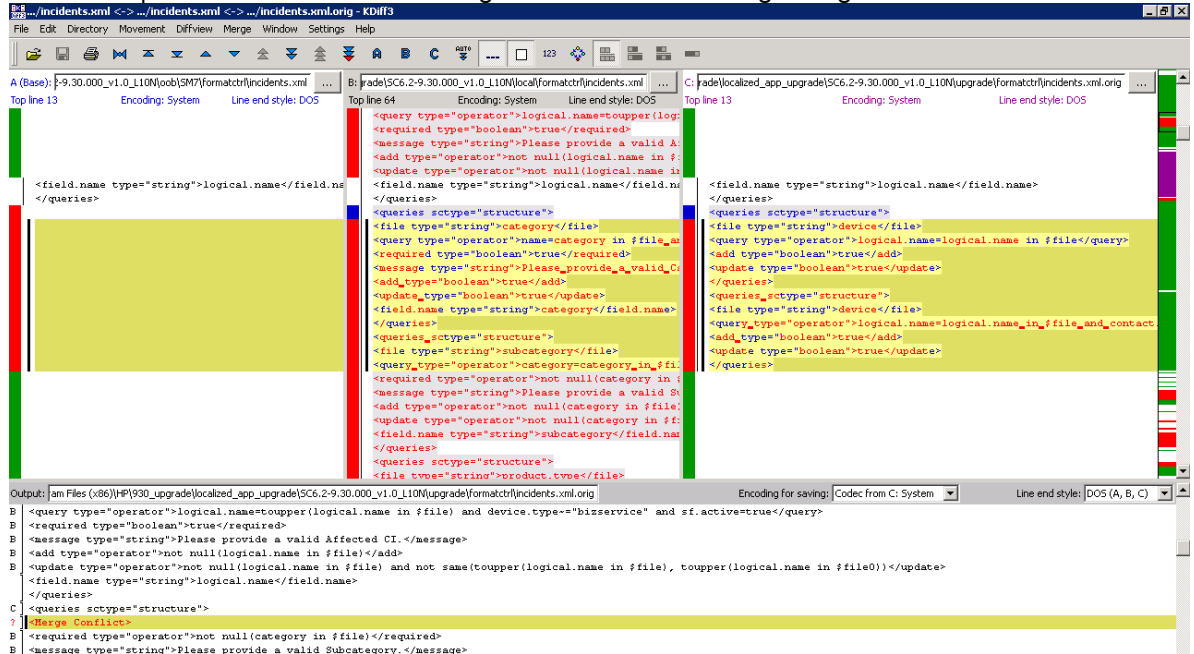
When right-clicking on the <Merge Conflict> line, a selection of which XML line to use is presented:

```

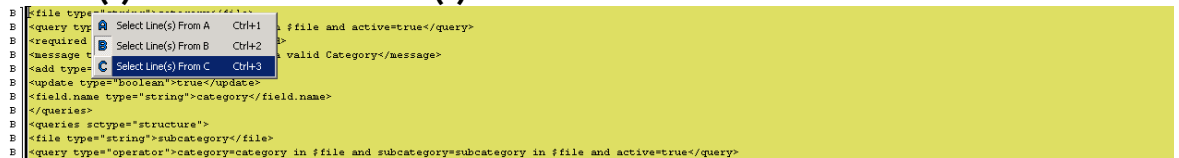
C <queries sctype="structure">
? <Merge Conflicts>
B <required type="operator">not null(category in $file)</required>
B <message type="string">Please provide a valid Subcategory.</message>
B <add type="operator">not null(category in $file)</add>
B <update type="operator">not null(category in $file)</update>
B <field.name type="string">subcategory</field.name>
B </queries>

```

The compare windows above the merge window show the originating lines:



In this case, the customer added a FormatControl query, and the new out-of-box record added another query line. To successfully merge both the customizations and the new out-of-box functionality, both lines from B and the lines from C need to be included. To do so, choose **Select Line(s) from B** and **Select line(s) from C**.



Note: The resulting XML from this type of merge is most likely missing tags and needs to be verified.

8. Continue with all other Merge Conflicts by deciding which line(s) to include in the final XML.
9. Once all merge conflicts are resolved select all lines of the merge result in the output pane and copy them.

The XML in this example prior to verification is as follows (due to the size of the XML, we will only show the queries part of the record):

```
<model name="formatctrl">
<keys>
<name sctype="string">incidents</name>
</keys>
<instance recordid="incidents" uniquequery="name=&quot;incidents&quot;">
<name type="string">incidents</name>
<database.access type="boolean">true</database.access>
<queries sctype="array">
<queries sctype="structure">
<file type="string">assignment</file>
<query type="operator">name=toupper(xx.handled.by.group in $file) and xx.active=true</query>
<required type="boolean">true</required>
<message type="string">Please provide a valid handled by group</message>
<add type="operator">not $G.ess</add>
<update type="operator">not $G.ess and not same(toupper(xx.handled.by.group in $file),
toupper(xx.handled.by.group in $file0))</update>
```

```

<field.name type="string">xx.handled.by.group</field.name>
</queries>
<queries sctype="structure">
<file type="string">operator</file>
<query type="operator">not null(owner.name in $file) and contact.name=toupper(owner.name in
$file) or null(owner.name in $file) and name=operator()</query>
<required type="operator">not null(owner.name in $file) and (owner.name in
$file)~#"linker"</required>
<message type="string">Please provide a valid handled by person.</message>
<add type="operator">not null(owner.name in $file)</add>
<update type="operator">not null(owner.name in $file) and not same(toupper(owner.name in $file),
toupper(owner.name in $file0))</update>
<query.init type="operator">not $G.ess and null(owner.name in $file) or null(xx.handled.by.group in
$file)</query.init>
</queries>
<queries sctype="structure">
<file type="string">contacts</file>
<query type="operator">contact.name=toupper(callback.contact in $file) and active=true</query>
<required type="boolean">true</required>
<message type="string">Please provide a valid Caller.</message>
<add type="boolean">true</add>
<update type="operator">not same(toupper(callback.contact in $file), toupper(callback.contact in
$file0))</update>
<field.name type="string">callback.contact</field.name>
</queries>
<queries sctype="structure">
<file type="string">contacts</file>
<query type="operator">contact.name=toupper(contact.name in $file) and active=true</query>
<required type="operator">not same(toupper(callback.contact in $file), toupper(contact.name in
$file))</required>
<message type="string">Please provide a valid Customer.</message>
<add type="operator">not same(toupper(callback.contact in $file), toupper(contact.name in
$file))</add>
<update type="operator">not same(toupper(callback.contact in $file), toupper(contact.name in $file))
and not same(toupper(contact.name in $file), toupper(contact.name in $file0))</update>
<field.name type="string">contact.name</field.name>
</queries>
<queries sctype="structure">
<file type="string">contacts</file>
<query type="operator">contact.name=toupper(alternate.contact in $file) and active=true</query>
<message type="string">Please provide a valid Alternate Contact.</message>
<add type="operator">not same(toupper(callback.contact in $file), toupper(alternate.contact in
$file))</add>
<update type="operator">not same(toupper(callback.contact in $file), toupper(alternate.contact in
$file)) and not same(toupper(alternate.contact in $file), toupper(alternate.contact in $file0))</update>
<field.name type="string">alternate.contact</field.name>
</queries>
<queries sctype="structure">
<file type="string">device</file>
<query type="operator">logical.name=toupper(affected.item in $file) and device.type="bizservice"
and xx.active=true</query>
<required type="boolean">true</required>
<message type="string">Please provide a valid Service.</message>
<add type="boolean">true</add>

```

```

<update type="operator">not same(toupper(affected.item in $file), toupper(affected.item in
$file0))</update>
<field.name type="string">affected.item</field.name>
</queries>
<queries sctype="structure">
<file type="string">device</file>
<query type="operator">logical.name=toupper(logical.name in $file) and device.type~="bizservice"
and xx.active=true</query>
<required type="boolean">true</required>
<message type="string">Please provide a valid Affected CI.</message>
<add type="operator">not null(logical.name in $file)</add>
<update type="operator">not null(logical.name in $file) and not same(toupper(logical.name in $file),
toupper(logical.name in $file0))</update>
<field.name type="string">logical.name</field.name>
</queries>
<queries sctype="structure">
<file type="string">category</file>
<query type="operator">name=category in $file and active=true</query>
<required type="boolean">true</required>
<message type="string">Please provide a valid Category</message>
<add type="boolean">true</add>
<update type="boolean">true</update>
<field.name type="string">category</field.name>
</queries>
<queries sctype="structure">
<file type="string">subcategory</file>
<query type="operator">category=category in $file and subcategory=subcategory in $file and
active=true</query>
<file type="string">device</file>
<query type="operator">logical.name=logical.name in $file</query>
<add type="boolean">true</add>
<update type="boolean">true</update>
</queries>
<queries sctype="structure">
<file type="string">device</file>
<query type="operator">logical.name=logical.name in $file and contact.name=contact.name in
$file</query>
<add type="boolean">true</add>
<update type="boolean">true</update>
</queries>
<required type="operator">not null(category in $file)</required>
<message type="string">Please provide a valid Subcategory.</message>
<add type="operator">not null(category in $file)</add>
<update type="operator">not null(category in $file)</update>
<field.name type="string">subcategory</field.name>
</queries>
<queries sctype="structure">
<file type="string">product.type</file>
<query type="operator">category=category in $file and subcategory=subcategory in $file and
product.type=product.type in $file and active=true</query>
<required type="operator">not null(category in $file) and not null(subcategory in $file) and not
null(subcategory in $file)</required>
<message type="string">Please provide a valid Type.</message>

```



```

<add type="operator">not null(category in $file) and not null(subcategory in $file) and not
null(subcategory in $file)</add>
<update type="operator">not null(category in $file) and not null(subcategory in $file) and not
null(subcategory in $file)</update>
<field.name type="string">product.type</field.name>
</queries>
<queries sctype="structure">
<file type="string">location</file>
<query type="operator">location=location in $file</query>
<required type="boolean">>false</required>
<message type="string">Please provide a valid Location.</message>
<add type="operator">not null(location in $file)</add>
<update type="operator">not $G.ess and not same(toupper(location in $file), toupper(location in
$file0))</update>
<field.name type="string">location</field.name>
</queries>
<queries sctype="structure">
<file type="string">xximrecovery</file>
<query type="operator">record.id=incident.id in $file and recovered=true</query>
<add type="operator">open in $file="Closed"</add>
<update type="operator">open in $file="Closed"</update>
</queries>
</queries>
</instance>
</model>

```

10. Verify that the XML syntax is correct by using an XML-typing editor. All tags have to have an end tag, and the nesting of the tags has to be verified.

When looking at the queries section in an XML typing editor, the following query line fails the syntax check. The correction is entered below in red:

```

<queries sctype="structure">
<file type="string">subcategory</file>
<query type="operator">category=category in $file and subcategory=subcategory in $file and
active=true</query>
<file type="string">device</file>
<query type="operator">logical.name=logical.name in $file</query>
<add type="boolean">>true</add>
<update type="boolean">>true</update>
</queries>
<queries sctype="structure">
<file type="string">device</file>
<query type="operator">logical.name=logical.name in $file and contact.name=contact.name in
$file</query>
<add type="boolean">>true</add>
<update type="boolean">>true</update>
</queries>
<required type="operator">not null(category in $file)</required>
<message type="string">Please provide a valid Subcategory.</message>
<add type="operator">not null(category in $file)</add>
<update type="operator">not null(category in $file)</update>
<field.name type="string">subcategory</field.name>
</queries>

```

```

<queries sctype="structure">
<file type="string">device</file>
<query type="operator">logical.name=logical.name in $file</query>
<add type="boolean">>true</add>
<update type="boolean">>true</update>
</queries>
<queries sctype="structure">
<file type="string">device</file>
<query type="operator">logical.name=logical.name in $file and contact.name=contact.name in $file</query>
<add type="boolean">>true</add>
<update type="boolean">>true</update>
</queries>
<queries sctype="structure">
<file type="string">subcategory</file>
<query type="operator">category=category in $file and subcategory=subcategory in $file and active=true</query>
<required type="operator">not null(category in $file)</required>
<message type="string">Please provide a valid Subcategory.</message>
<add type="operator">not null(category in $file)</add>
<update type="operator">not null(category in $file)</update>
<field.name type="string">subcategory</field.name>
</queries>

```

11. Search for and select the record in the Upgrade Results list that corresponds to the object that you are merging, then click Merge from the More Actions menu to start the Merge tool embedded in the Upgrade Utility.

Upgrade Results	
Object Type:	formatctrl
Object Name:	incidents
Result:	-

12. You will be shown the out-of-box and customer version of the record you are reconciling:

On the right hand side, right mouse-click and choose **Select all**. Then select **Paste** to insert the merged XML. Click **Save** to save your changes to the XML representation of the record.

13. Verify that the merge worked successfully by going into the Service Manager tool used to edit this record (for example, **fc** for format control, **link** for link records), selecting the record and ensure that all lines within the record appear correct and complete. If they are not correct and complete, use the **Revert** option in the Upgrade Results to return to the customer tailored version of the record prior to re-resolving the conflict appropriately.

Merge verification

KDiff3 as a three-way merge tool can assist in comparing and merging objects during conflict resolution. It is important to note that KDiff3 is a text based comparison tool, and it is not aware of the XML syntax and tagging. The KDiff3 three-way merge utility compares XML objects as text line by line and does not compare based on the XML tags. Therefore, it may try to add a line that already exists further down, or not add a beginning tag or ending tag during the merge.

When using KDiff3 to automatically merge different versions of a Service Manager record, the user is responsible to ensure syntactical and contextual integrity of the merged XML file. We recommend using an XML editor to verify start and end tags as well as the XML structure prior to using the merged XML to update the Service Manager record. After applying the merged XML, also verify that the merged record is syntactically correct and complete using the native Service Manager tools and perform sufficient testing to make sure the modified objects are working correctly.

Procedure for merging format records

Format records are not suitable to be merged using a text-based tool, particularly when merging large format records. You can use KDiff3 to assist in identifying the differences, though, and then apply the necessary changes manually using Forms Designer.

The following procedure uses the **advFind.search.quote** format as an example:

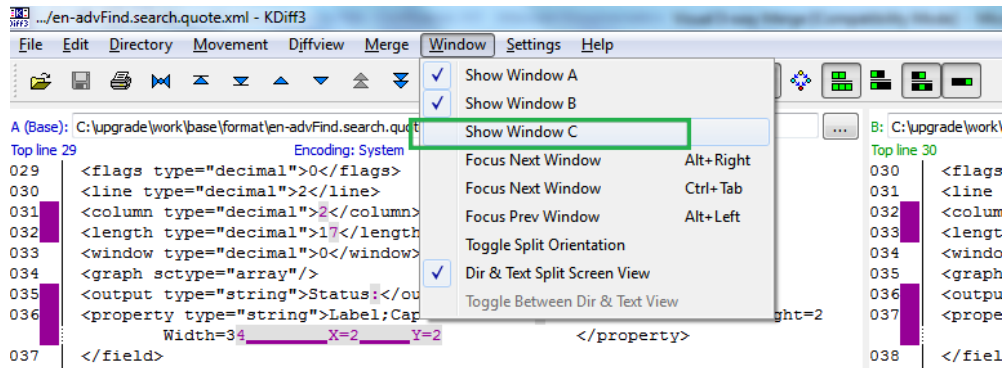
14. Start KDiff3, and specify the paths as follows

A(Base): <Upgrade>\3waymerge\work\base

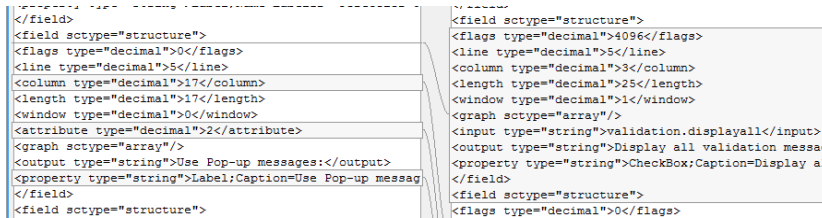
B: <Upgrade>\3waymerge\work\customer

C: <Upgrade>\3waymerge\work\upgrade

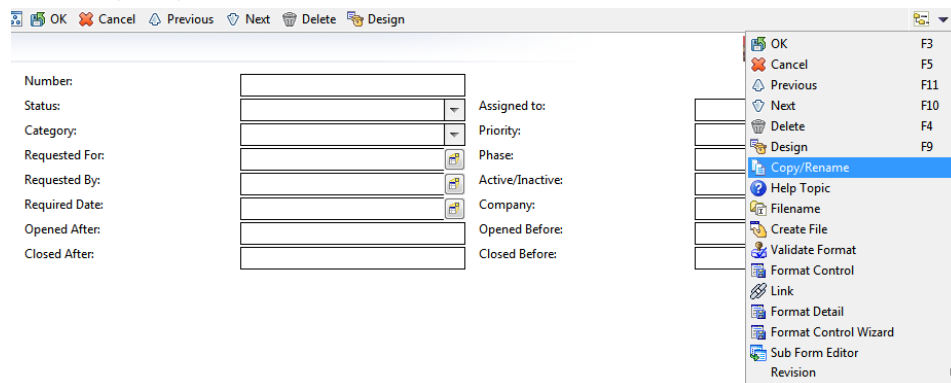
15. Within A, B, and C navigate to the **format** folder and open the **advFind.search.quote.xml** file.
16. It is possible to hide a version of the record, to perform a side-by-side comparison that involves only two versions. To do this, clear the check mark for the version that you want to hide from the **Window** menu.



17. To determine which format version has the least manual changes necessary to merge the two records, compare the B and C versions of the record, and scroll through to see which side has the most additions or changes.
18. Alternatively, you can compare both formats by going into the upgrade results record and clicking on More Actions > Compare (Note: The merge option is not available for format records) to compare the customer version and the NEW version of the format record. Below an example showing information that was added on the customer (right) version of the format:

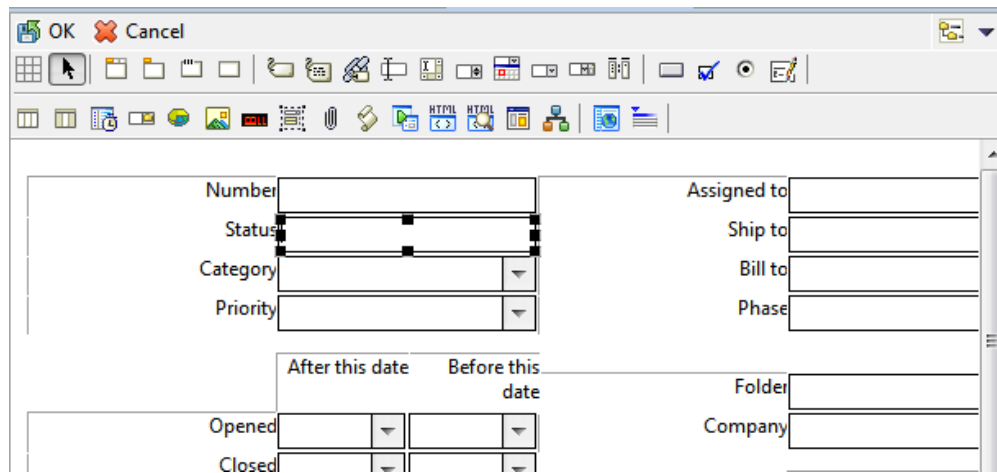


19. In this example, the customer tailored version has fewer changes than the upgrade version. Therefore, we will apply the customer's tailoring changes to the upgrade version, such as changing the type of the status field from a text field to a combo box.
20. To do the manual merge of features on the advFind.search.quote format, log on to Service Manager as an administrator, click on **Tailoring > Forms Designer**, and type **=advFind.search.quote** in the **Name** field. Click Search to retrieve the record.
21. Since the lesser amount of manual changes required in this case is on the customer tailored version with the name of advFind.search.quote, no rename of the form is necessary. Were it the other way around with less changes necessary against the NEW930 version of the format you would rename the customer version to CUSTadvFind.search.quote and rename the NEW930advFind.search.quote to advFind.search.quote prior to continuing with the manual merge. When renaming the formats, ensure to NOT perform the action on format control or link when prompted.



22. Go into design mode on the **advFind.search.quote** format by clicking on **Design**.

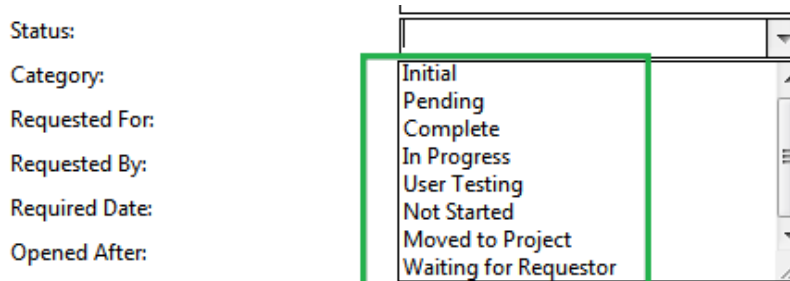
23. Remove the text box next to **Status**.



24. To most efficiently add the drop down box from the NEW930advFind.search.quote format, open another instance of **Forms Designer**, select the **NEW930advFind.search.quote** format, and go into Design mode by clicking the **Design** button.
25. In Design mode, click on the combo box next to the Status label and click **Ctrl + C** to copy the Object.
26. In the Designer session on the advFind.search.quote format, add the Object copied from the advFind.search.quote format by clicking **Ctrl + V** and then position it correctly next to the Status label.

Combo Box	
Input	val.status
Accessible Name	
Accessible Description	
Tab Stop	0
Read-Only	<input type="checkbox"/>
Read-Only Condition	
Mandatory	<input type="checkbox"/>
Mandatory Condition	
Array Length	0
Password	<input type="checkbox"/>
Maximum Chars	0
Maximum Characters Beep	<input type="checkbox"/>
Case Conversion	0
Decimals	None
Parse	<input type="checkbox"/>
Data Changed Event	0
Value List	Initial;Pending;Complete;In Progress;User Testing;Not Started;Moved to Project;Waiting for Requestor
Value List Condition	
Display List	Initial;Pending;Complete;In Progress;User Testing;Not Started;Moved to Project;Waiting for Requestor
Display List Condition	

27. Verify that the **Status** field has been changed correctly.



28. Keeping both Forms Designer sessions open, continue to copy and paste all other required changes from the NEW900advFind.search.quote format into the advFind.search.quote format.

29. Click **Save** twice on the **advFind.search.quote** format to save the changes, and close the forms designer session on the NEW930advFind.search.quote record without saving changes.
30. In the upgrade results record for the advFind.search.quote format, mark the record as Reconciled by selecting **Mark as Reconciled** from the More Actions menu.
31. Go to **Request Management > Quotes > Search Quotes** and test the modified fields to verify that this form is functioning correctly.

Special Considerations for using 3-way-merge

To begin, please always use the latest version of the upgrade utility. Since Service Manager 9.30, we release regular patches of the upgrade utility via the SSO patches site. New versions of the utility will include new functionality as well as fixes for any issues encountered in earlier versions.

Considerations when using KDiff3

KDiff3 is a text based tool that does a line by line comparison. It is not aware of XML tags, and thus will not ensure that each start tag has a proper end tag. In my testing, it often happened that it included an XML tree within another XML tree that should have been parallel. After merging XML documents using KDiff3, it is very important to use an XML editor to verify the XML structure and to fix the XML structure prior to loading the merged XML into Service Manager's 2-way merge utility in the upgrade results.

Additionally, the view in KDiff3 can be confusing when the customer replaced a function for example in the ScriptLibrary record. The merge will most likely try to combine the original function and the replacing function, rather than during the merge create two separate functions that are independent of each other. Again, this is due to the fact that KDiff3 is a pure text based tool with no ability to make "smart" decisions based on the type of record it is comparing and merging.

A hint on "no src line" merge issues – this line often indicates that the XML structure in the resulting XML will have an issue, since it is trying to merge two areas that do not match. The no source line indicator shows that either the customer or the new out-of-box version is missing a line or contains a line that does not match anything in the compared version. Take extra caution when encountering this message in the merged text.

And a final tip / trick: If the comparison in KDiff3 is too complicated and does not seem to make sense, use the Service Manager tailoring utility for this type of record to look at the differences. For example, if the comparison of the ScriptLibrary record does not make sense through KDiff3, look at it with the Service Manager ScriptLibrary editor to determine what has changed, as in the example for the BSGFunctions ScriptLibrary record shown below:

In the excerpt from the JavaScript code below, you can see how the KDiff3 merge tool handled the merge of the functions on the left and the manually merged (correct) functions on the right.

```

function
getMembers (ciName,firstLevelOnly,maxLevel,exclud
eCItoCI) {
    if (firstLevelOnly) {
        maxLevel = 1;

        nodeMembers =
getMemberOneLevel (levelOne);
        if (nodeMembers.length ==0)

    }

    if (excludeCItoCI == null) {
        try {
            excludeCItoCI = isBusinessService (ciName);
        }
        catch (e) {
            excludeCItoCI = true;
        }
    }
    if (!excludeCItoCI & & maxLevel > 1)
    {
        maxLevel--;
    }
    return
getAllMembersList ([ciName],maxLevel,excludeCItoC
I);
}

function
getMembers (ciName,firstLevelOnly,maxLevel )
{
    //print("getMembers 1 "+ciName+" and
"+firstLevelOnly+" maxLevel "+maxLevel);
    var allMembers = new Array ();
    var levelOne = getMemberOneNode (ciName);
    if (firstLevelOnly || maxLevel==1)
        return levelOne;
    //do a breadth first search to navigate the
entire tree
    allMembers = levelOne;
    var count =0;
    if (maxLevel <1)
        maxLevel=10;
    maxLevel = maxLevel - 1;
    do {
        nodeMembers =
getMemberOneLevel (levelOne);
        if (nodeMembers.length ==0)
            break;

        var nextLevel = new Array ();
        nextLevel =
appendArray (nextLevel,nodeMembers);

        levelOne = nextLevel;
        allMembers =
appendArray (allMembers,nextLevel);

        ++count;
        //print("level =" +count);
    } while (count<maxLevel)

    return allMembers;
}

```

Note that the function on the left is missing all the new code from the right and that the syntax with the open and close {} is incorrect.

Another issue is at the end of the merged JavaScript:

```

function updateSeenList (seen, array)
//append array to an existing array
function appendArray (appendTo, appendFrom)
{
    for (var i = 0; i < array.length; i++)
    {
        seen[array[i]] = true;
    }
    return seen;
    var appendToLng = appendTo.length;
    for (var i=0;i<appendFrom.length;++i)
    {
        appendTo[appendToLng+i] = appendFrom[i];
    }
    return appendTo;
}

function validListSize (members) {
    return system.library.util.arrayToQueryString (members).length <
MAX_QUERY_STRING ();
}

```


It attempted to merge line by line and by doing so put a function within a function. This was not the result of a merge conflict, but was done by the automatic merge. Compiling the code within the ScriptLibrary brought the issue to light.

When looking at the conflict resolution for this ScriptLibrary record within the Script Library tool, it became obvious that the customer replaced one function and added more helper functions, which could be resolved manually without issues within a few minutes. Thus the hint – it is very important to look at conflicts not just in KDiff3, but using the appropriate tailoring tool as well.

Considerations when using the 2-way merge in Service Manager

The 2-way merge utility integrated with the upgrade results records in the Service Manager Windows client can be used to view changes between the customer version of a record and the new out-of-box version of the record. It can be used in conjunction with the KDiff3 auto-merged results to load the merged XML into Service Manager.

The XML based record in the 2-way merge utility has to be translated back into a Service Manager record. On occasion, issues occur during that process. Issues might include sections missing from the saved XML, eg. the subroutines in FormatControl disappearing, or mis-translations of special characters such as "<" still showing as "<", which will cause syntax errors when trying to save the record using the correct tailoring utility.

To verify that the 2-way merge utility saved the record correctly, always check in the proper tailoring utility immediately after saving the record from the 2-way merge and verify that all data is stored correctly (e.g. are all calculations, validations, subroutines stored in the FormatControl record and does the record save without syntax errors). If any issue is found, use the revert functionality in the upgraderesults table and attempt to fix the issue. If the issue is reproducible, provide customer support with an unload of the NEWxx as well as the current customer record, as well as a copy of the XML text that failed to save correctly.

For more information

Please visit the HP Software support Web site at:

www.hp.com/go/hpssoftwaresupport

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

HP 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 URL:

www.hp.com/go/hpssoftwaresupport/new_access_levels

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

www.hp.com/go/hpssoftwaresupport/passport-registration

Technology for better business outcomes

© Copyright 2011 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.

Linux is a U.S. registered trademark of Linus Torvalds. Microsoft and Windows are U.S. registered trademarks of Microsoft Corporation. UNIX is a registered trademark of The Open Group. JavaScript is a registered trademark of Sun Microsystems, Inc. in the United States and other countries. Oracle is a registered trademark of Oracle Corporation and/or its affiliates

